```
**********************************************************
   42294 Thu Jul 12 12:22:27 2012
new/exception_lists/cstyle
XXXX Intel X540 support
**********************************************************
```

```
  1 usr/src/cmd/krb5/kadmin/cli/kadmin_ct.c
  2 usr/src/cmd/krb5/kadmin/cli/kadmin.c
  3 usr/src/cmd/krb5/kadmin/cli/kadmin.h
  4 usr/src/cmd/krb5/kadmin/cli/keytab.c
  5 usr/src/cmd/krb5/kadmin/cli/ss_wrapper.c
  6 usr/src/cmd/krb5/kadmin/dbutil/dump.c
  7 usr/src/cmd/krb5/kadmin/dbutil/import_err.h
  8 usr/src/cmd/krb5/kadmin/dbutil/kadm5_create.c
  9 usr/src/cmd/krb5/kadmin/dbutil/kdb5_create.c
 10 usr/src/cmd/krb5/kadmin/dbutil/kdb5_destroy.c
 11 usr/src/cmd/krb5/kadmin/dbutil/kdb5_stash.c
 12 usr/src/cmd/krb5/kadmin/dbutil/kdb5_util.c
 13 usr/src/cmd/krb5/kadmin/dbutil/kdb5_util.h
 14 usr/src/cmd/krb5/kadmin/dbutil/nstrtok.h
 15 usr/src/cmd/krb5/kadmin/dbutil/ovload.c
 16 usr/src/cmd/krb5/kadmin/dbutil/string_table.c
 17 usr/src/cmd/krb5/kadmin/dbutil/string_table.h
 18 usr/src/cmd/krb5/kadmin/dbutil/strtok.c
 19 usr/src/cmd/krb5/kadmin/dbutil/util.c
 20 usr/src/cmd/krb5/kadmin/kpasswd/kpasswd_strings.h
 21 usr/src/cmd/krb5/kadmin/kpasswd/kpasswd.c
 22 usr/src/cmd/krb5/kadmin/kpasswd/kpasswd.h
 23 usr/src/cmd/krb5/kadmin/kpasswd/tty_kpasswd.c
 24 usr/src/cmd/krb5/kadmin/ktutil/ktutil_ct.c
 25 usr/src/cmd/krb5/kadmin/ktutil/ktutil_funcs.c
 26 usr/src/cmd/krb5/kadmin/ktutil/ktutil.c
 27 usr/src/cmd/krb5/kadmin/ktutil/ktutil.h
 28 usr/src/cmd/krb5/kadmin/server/kadm_rpc_svc.c
 29 usr/src/cmd/krb5/kadmin/server/misc.c
 30 usr/src/cmd/krb5/kadmin/server/misc.h
 31 usr/src/cmd/krb5/kadmin/server/ovsec_kadmd.c
 32 usr/src/cmd/krb5/kadmin/server/server_glue_v1.c
 33 usr/src/cmd/krb5/kadmin/server/server_stubs.c
 34 usr/src/cmd/krb5/kdestroy/kdestroy.c
 35 usr/src/cmd/krb5/kinit/kinit.c
 36 usr/src/cmd/krb5/klist/klist.c
 37 usr/src/cmd/krb5/krb5kdc/dispatch.c
 38 usr/src/cmd/krb5/krb5kdc/do_as_req.c
 39 usr/src/cmd/krb5/krb5kdc/do_tgs_req.c
 40 usr/src/cmd/krb5/krb5kdc/extern.c
 41 usr/src/cmd/krb5/krb5kdc/extern.h
 42 usr/src/cmd/krb5/krb5kdc/kdc_preauth.c
 43 usr/src/cmd/krb5/krb5kdc/kdc_util.c
 44 usr/src/cmd/krb5/krb5kdc/kdc_util.h
 45 usr/src/cmd/krb5/krb5kdc/main.c
 46 usr/src/cmd/krb5/krb5kdc/network.c
 47 usr/src/cmd/krb5/krb5kdc/policy.c
 48 usr/src/cmd/krb5/krb5kdc/policy.h
 49 usr/src/cmd/krb5/krb5kdc/replay.c
 50 usr/src/cmd/krb5/krb5kdc/sock2p.c
 51 usr/src/cmd/krb5/ldap_util/kdb5_ldap_list.c
 52 usr/src/cmd/krb5/ldap_util/kdb5_ldap_list.h
 53 usr/src/cmd/krb5/ldap_util/kdb5_ldap_policy.c
 54 usr/src/cmd/krb5/ldap_util/kdb5_ldap_policy.h
 55 usr/src/cmd/krb5/ldap_util/kdb5_ldap_realm.c
 56 usr/src/cmd/krb5/ldap_util/kdb5_ldap_realm.h
 57 usr/src/cmd/krb5/ldap_util/kdb5_ldap_services.c
 58 usr/src/cmd/krb5/ldap_util/kdb5_ldap_services.h
 59 usr/src/cmd/krb5/ldap_util/kdb5_ldap_util.c
 60 usr/src/cmd/krb5/ldap_util/kdb5_ldap_util.h
 61 usr/src/cmd/krb5/slave/kprop.c
```



```
 62 usr/src/cmd/krb5/slave/kprop.h
 63 usr/src/cmd/krb5/slave/kpropd.c
 64 usr/src/common/bzip2/bzlib.h
 65 usr/src/common/bzip2/crctable.c
 66 usr/src/common/bzip2/randtable.c
 67 usr/src/common/bzip2/blocksort.c
 68 usr/src/common/bzip2/compress.c
 69 usr/src/common/bzip2/bzlib.c
 70 usr/src/common/bzip2/decompress.c
 71 usr/src/common/bzip2/bzlib_private.h
 72 usr/src/common/bzip2/huffman.c
 73 usr/src/common/openssl/crypto/krb5/krb5_asn.c
 74 usr/src/common/openssl/crypto/krb5/krb5_asn.h
 75 usr/src/lib/gss_mechs/mech_krb5/crypto/aes/aes_s2k.c
 76 usr/src/lib/gss_mechs/mech_krb5/crypto/cksumtype_to_string.c
 77 usr/src/lib/gss_mechs/mech_krb5/crypto/coll_proof_cksum.c
 78 usr/src/lib/gss_mechs/mech_krb5/crypto/crc32/crc.c
 79 usr/src/lib/gss_mechs/mech_krb5/crypto/des/afsstring2key.c
 80 usr/src/lib/gss_mechs/mech_krb5/crypto/des/string2key.c
 81 usr/src/lib/gss_mechs/mech_krb5/crypto/dk/stringtokey.c
 82 usr/src/lib/gss_mechs/mech_krb5/crypto/enctype_compare.c
 83 usr/src/lib/gss_mechs/mech_krb5/crypto/enctype_to_string.c
 84 usr/src/lib/gss_mechs/mech_krb5/crypto/hash_provider/hash_md5.c
 85 usr/src/lib/gss_mechs/mech_krb5/crypto/hash_provider/hash_sha1.c
 86 usr/src/lib/gss_mechs/mech_krb5/crypto/keyed_checksum_types.c
 87 usr/src/lib/gss_mechs/mech_krb5/crypto/keyed_cksum.c
 88 usr/src/lib/gss_mechs/mech_krb5/crypto/keyhash_provider/hmac_md5.c
 89 usr/src/lib/gss_mechs/mech_krb5/crypto/keyhash_provider/k5_md5des.c
 90 usr/src/lib/gss_mechs/mech_krb5/crypto/keylengths.c
 91 usr/src/lib/gss_mechs/mech_krb5/crypto/make_random_key.c
 92 usr/src/lib/gss_mechs/mech_krb5/crypto/md4/md4.c
 93 usr/src/lib/gss_mechs/mech_krb5/crypto/old_api_glue.c
 94 usr/src/lib/gss_mechs/mech_krb5/crypto/old/des_stringtokey.c
 95 usr/src/lib/gss_mechs/mech_krb5/crypto/pbkdf2.c
 96 usr/src/lib/gss_mechs/mech_krb5/crypto/random_to_key.c
 97 usr/src/lib/gss_mechs/mech_krb5/crypto/state.c
 98 usr/src/lib/gss_mechs/mech_krb5/crypto/string_to_cksumtype.c
 99 usr/src/lib/gss_mechs/mech_krb5/crypto/string_to_enctype.c
100 usr/src/lib/gss_mechs/mech_krb5/crypto/string_to_key.c
101 usr/src/lib/gss_mechs/mech_krb5/crypto/valid_cksumtype.c
102 usr/src/lib/gss_mechs/mech_krb5/crypto/valid_enctype.c
103 usr/src/lib/gss_mechs/mech_krb5/et/com_err.c
104 usr/src/lib/gss_mechs/mech_krb5/et/error_message.c
105 usr/src/lib/gss_mechs/mech_krb5/et/error_table.h
106 usr/src/lib/gss_mechs/mech_krb5/et/internal.h
107 usr/src/lib/gss_mechs/mech_krb5/et/mit-sipb-copyright.h
108 usr/src/lib/gss_mechs/mech_krb5/include/cache-addrinfo.h
109 usr/src/lib/gss_mechs/mech_krb5/include/cm.h
110 usr/src/lib/gss_mechs/mech_krb5/include/com_err.h
111 usr/src/lib/gss_mechs/mech_krb5/include/db-config.h
112 usr/src/lib/gss_mechs/mech_krb5/include/db.h
113 usr/src/lib/gss_mechs/mech_krb5/include/fake-addrinfo.h
114 usr/src/lib/gss_mechs/mech_krb5/include/foreachaddr.h
115 usr/src/lib/gss_mechs/mech_krb5/include/k5-int-pkinit.h
116 usr/src/lib/gss_mechs/mech_krb5/include/k5-utf8.h
117 usr/src/lib/gss_mechs/mech_krb5/include/kdb_kt.h
118 usr/src/lib/gss_mechs/mech_krb5/include/krb5_libinit.h
119 usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm_defs.h
120 usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm_proto.h
121 usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm.h
122 usr/src/lib/gss_mechs/mech_krb5/include/krb5/copyright.h
123 usr/src/lib/gss_mechs/mech_krb5/include/krb5/k5-err.h
124 usr/src/lib/gss_mechs/mech_krb5/include/krb5/k5-plugin.h
125 usr/src/lib/gss_mechs/mech_krb5/include/krb5/kdb_dbc.h
126 usr/src/lib/gss_mechs/mech_krb5/include/krb5/kdb.h
127 usr/src/lib/gss_mechs/mech_krb5/include/locate_plugin.h
```

```
usr/src/lib/gss_mechs/mech_krb5/include/osconf.h
usr/src/lib/gss_mechs/mech_krb5/include/port-sockets.h
usr/src/lib/gss_mechs/mech_krb5/include/preauth_plugin.h
usr/src/lib/gss_mechs/mech_krb5/include/socket-utils.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_decode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_decode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_encode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_encode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1buf.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krb5_decode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krb5_encode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krbasn1.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/ldap_key_seq.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_memory.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_retr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccbase.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cccopy.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccdefops.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccfns.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/fcc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/scc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ser_cc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/error_tables/adm_err.h
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/file/ktfile.h
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt_srvtab.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktadd.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktbase.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktfns.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktfr_entry.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktremove.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/read_servi.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_comp.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_order.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_srch.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/appdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/bld_pr_ext.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/bld_princ.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/chk_trans.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/cleanup.h
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/conv_princ.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_addrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_creds.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_data.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_tick.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/cp_key_cnt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/decode_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/decrypt_tk.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/deltat.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/enc_helper.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/encode_kdc.c
```



```
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/encrypt_tk.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/free_rtree.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/fwd_tgt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gc_frm_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gc_via_tkt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gen_seqnum.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gen_subkey.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/get_creds.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/get_in_tkt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_keytab.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_opt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_pwd.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/init_keyblock.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/int-proto.h
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/kdc_rep_dc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/kerrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_error.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_priv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_rep.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_req_ext.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_req.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_safe.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/pac.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/pr_to_salt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/preauth.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/preauth2.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/princ_comp.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_cred.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_error.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_priv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_rep.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_req_dec.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_req.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_safe.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/recvauth.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/send_tgs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/sendauth.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/set_realm.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/srv_rcache.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/str_conv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/tgtname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/valid_times.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/vic_opt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/walk_rtree.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/accessor.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/an_to_ln.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/ccdefname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/changepw.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/def_realm.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnsglue.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnsglue.h
usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnssrv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/foreachaddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/free_hstrl.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/free_krbhs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/full_ipadr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gen_port.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gen_rname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/genaddrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/get_krbhst.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gmt_mktime.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/hostaddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/hst_realm.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/ktdefname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/kuserok.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/localaddr.c
```

```
usr/src/lib/gss_mechs/mech_krb5/krb5/os/locate_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/lock_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/mk_faddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/net_read.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/net_write.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/os-proto.h
usr/src/lib/gss_mechs/mech_krb5/krb5/os/osconfig.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/port2ip.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/prompter.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/promptusr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/read_msg.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/read_pwd.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/realm_dom.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/realm_iter.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/sendto_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/sn2princ.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/thread_safe.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/unlck_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/ustime.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/write_msg.c
usr/src/lib/gss_mechs/mech_krb5/krb5/posix/daemon.c
usr/src/lib/gss_mechs/mech_krb5/krb5/posix/setenv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_base.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_conv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_io.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_none.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rcdef.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rcfns.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/ser_rc.c
usr/src/lib/gss_mechs/mech_krb5/mech/accept_sec_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/acquire_cred_with_pw.c
usr/src/lib/gss_mechs/mech_krb5/mech/acquire_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/add_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/compare_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/context_time.c
usr/src/lib/gss_mechs/mech_krb5/mech/copy_ccache.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_com_err_status.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_major_status.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_status.c
usr/src/lib/gss_mechs/mech_krb5/mech/export_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/export_sec_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/get_tkt_flags.c
usr/src/lib/gss_mechs/mech_krb5/mech/gss_libinit.h
usr/src/lib/gss_mechs/mech_krb5/mech/import_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/indicate_mechs.c
usr/src/lib/gss_mechs/mech_krb5/mech/init_sec_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/inq_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/inq_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/inq_names.c
usr/src/lib/gss_mechs/mech_krb5/mech/krb5_gss_glue.c
usr/src/lib/gss_mechs/mech_krb5/mech/lucid_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/oid_ops.c
usr/src/lib/gss_mechs/mech_krb5/mech/process_context_token.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_buffer.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_oid_set.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_oid.c
usr/src/lib/gss_mechs/mech_krb5/mech/set_allowable_enctypes.c
usr/src/lib/gss_mechs/mech_krb5/mech/set_ccache.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_buffer_set.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_buffer.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_cksum.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_ctxsetup.c
```



```
usr/src/lib/gss_mechs/mech_krb5/mech/util_dup.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_localhost.c
usr/src/lib/gss_mechs/mech_krb5/mech/utl_nohash_validate.c
usr/src/lib/gss_mechs/mech_krb5/profile/prof_err.h
usr/src/lib/gss_mechs/mech_krb5/profile/prof_get.c
usr/src/lib/gss_mechs/mech_krb5/profile/prof_set.c
usr/src/lib/gss_mechs/mech_krb5/support/errors.c
usr/src/lib/gss_mechs/mech_krb5/support/fake-addrinfo.c
usr/src/lib/gss_mechs/mech_krb5/support/init-addrinfo.c
usr/src/lib/gss_mechs/mech_krb5/support/plugins.c
usr/src/lib/gss_mechs/mech_krb5/support/supp-int.h
usr/src/lib/gss_mechs/mech_krb5/support/threads.c
usr/src/lib/gss_mechs/mech_krb5/support/utf8_conv.c
usr/src/lib/gss_mechs/mech_krb5/support/utf8.c
usr/src/lib/krb5/dyn/dyn_append.c
usr/src/lib/krb5/dyn/dyn_create.c
usr/src/lib/krb5/dyn/dyn_debug.c
usr/src/lib/krb5/dyn/dyn_delete.c
usr/src/lib/krb5/dyn/dyn_initzero.c
usr/src/lib/krb5/dyn/dyn_insert.c
usr/src/lib/krb5/dyn/dyn_paranoid.c
usr/src/lib/krb5/dyn/dyn_put.c
usr/src/lib/krb5/dyn/dyn_realloc.c
usr/src/lib/krb5/dyn/dyn_size.c
usr/src/lib/krb5/kadm5/admin_internal.h
usr/src/lib/krb5/kadm5/admin_xdr.h
usr/src/lib/krb5/kadm5/admin.h
usr/src/lib/krb5/kadm5/alt_prof.c
usr/src/lib/krb5/kadm5/chpass_util_strings.h
usr/src/lib/krb5/kadm5/chpass_util.c
usr/src/lib/krb5/kadm5/clnt/changepw.c
usr/src/lib/krb5/kadm5/clnt/client_handle.c
usr/src/lib/krb5/kadm5/clnt/client_init.c
usr/src/lib/krb5/kadm5/clnt/client_internal.h
usr/src/lib/krb5/kadm5/clnt/client_principal.c
usr/src/lib/krb5/kadm5/clnt/client_rpc.c
usr/src/lib/krb5/kadm5/clnt/clnt_chpass_util.c
usr/src/lib/krb5/kadm5/clnt/clnt_policy.c
usr/src/lib/krb5/kadm5/clnt/clnt_privs.c
usr/src/lib/krb5/kadm5/clnt/logger.c
usr/src/lib/krb5/kadm5/kadm_err.h
usr/src/lib/krb5/kadm5/kadm_rpc_xdr.c
usr/src/lib/krb5/kadm5/kadm_rpc.h
usr/src/lib/krb5/kadm5/misc_free.c
usr/src/lib/krb5/kadm5/server_internal.h
usr/src/lib/krb5/kadm5/srv/adb_xdr.c
usr/src/lib/krb5/kadm5/srv/chgpwd.c
usr/src/lib/krb5/kadm5/srv/logger.c
usr/src/lib/krb5/kadm5/srv/server_acl.c
usr/src/lib/krb5/kadm5/srv/server_acl.h
usr/src/lib/krb5/kadm5/srv/server_dict.c
usr/src/lib/krb5/kadm5/srv/server_handle.c
usr/src/lib/krb5/kadm5/srv/server_init.c
usr/src/lib/krb5/kadm5/srv/server_kdb.c
usr/src/lib/krb5/kadm5/srv/server_misc.c
usr/src/lib/krb5/kadm5/srv/svr_chpass_util.c
usr/src/lib/krb5/kadm5/srv/svr_iters.c
usr/src/lib/krb5/kadm5/srv/svr_misc_free.c
usr/src/lib/krb5/kadm5/srv/svr_policy.c
usr/src/lib/krb5/kadm5/srv/svr_principal.c
usr/src/lib/krb5/kadm5/srv/xdr_alloc.c
usr/src/lib/krb5/kadm5/str_conv.c
usr/src/lib/krb5/kdb/adb_err.h
usr/src/lib/krb5/kdb/decrypt_key.c
usr/src/lib/krb5/kdb/encrypt_key.c
usr/src/lib/krb5/kdb/kdb_cpw.c
```

usr/src/lib/krb5/kdb/kdb_default.c
usr/src/lib/krb5/kdb/kdb5.c
usr/src/lib/krb5/kdb/kdb5.h
usr/src/lib/krb5/kdb/keytab.c
usr/src/lib/krb5/plugins/kdb/db2/adb_openclose.c
usr/src/lib/krb5/plugins/kdb/db2/adb_policy.c
usr/src/lib/krb5/plugins/kdb/db2/db2_exp.c
usr/src/lib/krb5/plugins/kdb/db2/kdb_compat.h
usr/src/lib/krb5/plugins/kdb/db2/kdb_db2.c
usr/src/lib/krb5/plugins/kdb/db2/kdb_db2.h
usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.c
usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_close.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_conv.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_debug.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_delete.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_get.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_open.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_overflow.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_page.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_put.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_search.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_seq.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_split.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_utils.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/btree.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/extern.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/db/db.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/dbm.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/extern.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_bigkey.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_func.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_log2.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_page.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hsearch.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/page.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/search.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-int.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-ndbm.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-queue.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/mpool/mpool.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/mpool/mpool.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/extern.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_close.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_delete.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_get.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_open.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_put.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_search.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_seq.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_utils.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/recno.h
usr/src/lib/krb5/plugins/kdb/db2/pol_xdr.c
usr/src/lib/krb5/plugins/kdb/db2/policy_db.h
usr/src/lib/krb5/plugins/kdb/ldap/ldap_exp.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap_conn.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_create.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_fetch_mkey.c



usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_main.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_misc.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_misc.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal2.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_pwd_policy.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_pwd_policy.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_realm.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_realm.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_rights.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_stash.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_stash.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_clnt.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto_openssl.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto_openssl.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_identity.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_lib.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_matching.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_profile.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_srv.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit.h
usr/src/lib/krb5/ss/copyright.h
usr/src/lib/krb5/ss/data.c
usr/src/lib/krb5/ss/error.c
usr/src/lib/krb5/ss/execute_cmd.c
usr/src/lib/krb5/ss/help.c
usr/src/lib/krb5/ss/invocation.c
usr/src/lib/krb5/ss/list_rqs.c
usr/src/lib/krb5/ss/listen.c
usr/src/lib/krb5/ss/mit-sipb-copyright.h
usr/src/lib/krb5/ss/mk_cmds.c
usr/src/lib/krb5/ss/options.c
usr/src/lib/krb5/ss/pager.c
usr/src/lib/krb5/ss/parse.c
usr/src/lib/krb5/ss/prompt.c
usr/src/lib/krb5/ss/request_tbl.c
usr/src/lib/krb5/ss/requests.c
usr/src/lib/krb5/ss/ss_internal.h
usr/src/lib/krb5/ss/ss.h
usr/src/lib/krb5/ss/std_rqs.c
usr/src/lib/krb5/ss/utils.c
usr/src/lib/libgss/g_glue.c
usr/src/lib/librstp/common/base.h
usr/src/lib/librstp/common/choose.h
usr/src/lib/librstp/common/edge.c
usr/src/lib/librstp/common/edge.h
usr/src/lib/librstp/common/migrate.c
usr/src/lib/librstp/common/migrate.h
usr/src/lib/librstp/common/p2p.c
usr/src/lib/librstp/common/p2p.h
usr/src/lib/librstp/common/pcost.c
usr/src/lib/librstp/common/pcost.h
usr/src/lib/librstp/common/port.c
usr/src/lib/librstp/common/port.h

```
usr/src/lib/librstp/common/portinfo.c
usr/src/lib/librstp/common/portinfo.h
usr/src/lib/librstp/common/rolesel.c
usr/src/lib/librstp/common/rolesel.h
usr/src/lib/librstp/common/roletrns.c
usr/src/lib/librstp/common/roletrns.h
usr/src/lib/librstp/common/statmch.c
usr/src/lib/librstp/common/statmch.h
usr/src/lib/librstp/common/stp_bpdu.h
usr/src/lib/librstp/common/stp_in.c
usr/src/lib/librstp/common/stp_in.h
usr/src/lib/librstp/common/stp_to.h
usr/src/lib/librstp/common/stp_vectors.h
usr/src/lib/librstp/common/stpm.c
usr/src/lib/librstp/common/stpm.h
usr/src/lib/librstp/common/stpmgmt.c
usr/src/lib/librstp/common/sttrans.c
usr/src/lib/librstp/common/sttrans.h
usr/src/lib/librstp/common/times.c
usr/src/lib/librstp/common/times.h
usr/src/lib/librstp/common/topoch.c
usr/src/lib/librstp/common/topoch.h
usr/src/lib/librstp/common/transmit.c
usr/src/lib/librstp/common/transmit.h
usr/src/lib/librstp/common/uid_stp.h
usr/src/lib/librstp/common/vector.c
usr/src/lib/librstp/common/vector.h
usr/src/uts/common/gssapi/gssapi.h
usr/src/uts/common/gssapi/mechs/krb5/crypto/block_size.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/checksum_length.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/cksumtypes.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/combine_keys.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/crc32/crc32.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/decrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/default_state.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/d3_cbc.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/f_cbc.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/f_parity.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/weak_key.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/checksum.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/derive.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/dk_decrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/dk_encrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/arcfour_provider.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/des.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/des3.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/encrypt_length.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/encrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/etypes.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_crc32.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_kmd5.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_ksha1.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hmac.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/descbc.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/k_hmac_md5.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/k5_kmd5des.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/make_checksum.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/mandatory_sumtype.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/nfold.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/old/old_decrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/old/old_encrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/prng.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/raw/raw_decrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/raw/raw_encrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/verify_checksum.c
usr/src/uts/common/gssapi/mechs/krb5/include/aes_s2k.h
```



```
usr/src/uts/common/gssapi/mechs/krb5/include/auth_con.h
usr/src/uts/common/gssapi/mechs/krb5/include/cksumtypes.h
usr/src/uts/common/gssapi/mechs/krb5/include/crc-32.h
usr/src/uts/common/gssapi/mechs/krb5/include/des_int.h
usr/src/uts/common/gssapi/mechs/krb5/include/dk.h
usr/src/uts/common/gssapi/mechs/krb5/include/enc_provider.h
usr/src/uts/common/gssapi/mechs/krb5/include/etypes.h
usr/src/uts/common/gssapi/mechs/krb5/include/gssapi_generic.h
usr/src/uts/common/gssapi/mechs/krb5/include/gssapi_krb5.h
usr/src/uts/common/gssapi/mechs/krb5/include/gssapiP_generic.h
usr/src/uts/common/gssapi/mechs/krb5/include/gssapiP_krb5.h
usr/src/uts/common/gssapi/mechs/krb5/include/hash_provider.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-int.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_16.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_32.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_64.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_16.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_32.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_64.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-thread.h
usr/src/uts/common/gssapi/mechs/krb5/include/keyhash_provider.h
usr/src/uts/common/gssapi/mechs/krb5/include/krb5.h
usr/src/uts/common/gssapi/mechs/krb5/include/old.h
usr/src/uts/common/gssapi/mechs/krb5/include/raw.h
usr/src/uts/common/gssapi/mechs/krb5/include/rsa-md4.h
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_athctr.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_auth.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_cksum.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_key.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_princ.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/init_ctx.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/kfree.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/parse.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_actx.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_adata.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_addr.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_auth.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_cksum.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_ctx.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_key.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_princ.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/serialize.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/unparse.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/os/c_ustime.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/os/init_os_ctx.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/os/timeofday.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/os/toffset.c
usr/src/uts/common/gssapi/mechs/krb5/mech/delete_sec_context.c
usr/src/uts/common/gssapi/mechs/krb5/mech/gssapi_krb5.c
usr/src/uts/common/gssapi/mechs/krb5/mech/import_sec_context.c
usr/src/uts/common/gssapi/mechs/krb5/mech/k5seal.c
usr/src/uts/common/gssapi/mechs/krb5/mech/k5sealv3.c
usr/src/uts/common/gssapi/mechs/krb5/mech/k5unseal.c
usr/src/uts/common/gssapi/mechs/krb5/mech/seal.c
usr/src/uts/common/gssapi/mechs/krb5/mech/ser_sctx.c
usr/src/uts/common/gssapi/mechs/krb5/mech/sign.c
usr/src/uts/common/gssapi/mechs/krb5/mech/unseal.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_crypt.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_ordering.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_seed.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_seqnum.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_set.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_token.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_validate.c
usr/src/uts/common/gssapi/mechs/krb5/mech/val_cred.c
```

usr/src/uts/common/gssapi/mechs/krb5/mech/verify.c
usr/src/uts/common/gssapi/mechs/krb5/mech/wrap_size_limit.c
usr/src/uts/common/io/ixgbe/ixgbe_82598.c
**usr/src/uts/common/io/ixgbe/ixgbe_82598.h**
usr/src/uts/common/io/ixgbe/ixgbe_82599.c
usr/src/uts/common/io/ixgbe/ixgbe_api.c
usr/src/uts/common/io/ixgbe/ixgbe_api.h
usr/src/uts/common/io/ixgbe/ixgbe_common.c
usr/src/uts/common/io/ixgbe/ixgbe_common.h
**usr/src/uts/common/io/ixgbe/ixgbe_mbx.c**
**usr/src/uts/common/io/ixgbe/ixgbe_mbx.h**
usr/src/uts/common/io/ixgbe/ixgbe_osdep.h
usr/src/uts/common/io/ixgbe/ixgbe_phy.c
usr/src/uts/common/io/ixgbe/ixgbe_phy.h
usr/src/uts/common/io/ixgbe/ixgbe_type.h
**usr/src/uts/common/io/ixgbe/ixgbe_x540.c**
**usr/src/uts/common/io/ixgbe/ixgbe_x540.h**
usr/src/uts/intel/io/acpica/debugger/dbcmds.c
usr/src/uts/intel/io/acpica/debugger/dbdisply.c
usr/src/uts/intel/io/acpica/debugger/dbexec.c
usr/src/uts/intel/io/acpica/debugger/dbfileio.c
usr/src/uts/intel/io/acpica/debugger/dbhistry.c
usr/src/uts/intel/io/acpica/debugger/dbinput.c
usr/src/uts/intel/io/acpica/debugger/dbmethod.c
usr/src/uts/intel/io/acpica/debugger/dbnames.c
usr/src/uts/intel/io/acpica/debugger/dbstats.c
usr/src/uts/intel/io/acpica/debugger/dbutils.c
usr/src/uts/intel/io/acpica/debugger/dbxface.c
usr/src/uts/intel/io/acpica/disassembler/dmbuffer.c
usr/src/uts/intel/io/acpica/disassembler/dmnames.c
usr/src/uts/intel/io/acpica/disassembler/dmobject.c
usr/src/uts/intel/io/acpica/disassembler/dmopcode.c
usr/src/uts/intel/io/acpica/disassembler/dmresrc.c
usr/src/uts/intel/io/acpica/disassembler/dmresrcl.c
usr/src/uts/intel/io/acpica/disassembler/dmresrcs.c
usr/src/uts/intel/io/acpica/disassembler/dmutils.c
usr/src/uts/intel/io/acpica/disassembler/dmwalk.c
usr/src/uts/intel/io/acpica/dispatcher/dsargs.c
usr/src/uts/intel/io/acpica/dispatcher/dscontrol.c
usr/src/uts/intel/io/acpica/dispatcher/dsfield.c
usr/src/uts/intel/io/acpica/dispatcher/dsinit.c
usr/src/uts/intel/io/acpica/dispatcher/dsmethod.c
usr/src/uts/intel/io/acpica/dispatcher/dsmthdat.c
usr/src/uts/intel/io/acpica/dispatcher/dsobject.c
usr/src/uts/intel/io/acpica/dispatcher/dsopcode.c
usr/src/uts/intel/io/acpica/dispatcher/dsutils.c
usr/src/uts/intel/io/acpica/dispatcher/dswexec.c
usr/src/uts/intel/io/acpica/dispatcher/dswload.c
usr/src/uts/intel/io/acpica/dispatcher/dswload2.c
usr/src/uts/intel/io/acpica/dispatcher/dswscope.c
usr/src/uts/intel/io/acpica/dispatcher/dswstate.c
usr/src/uts/intel/io/acpica/events/evevent.c
usr/src/uts/intel/io/acpica/events/evglock.c
usr/src/uts/intel/io/acpica/events/evgpe.c
usr/src/uts/intel/io/acpica/events/evgpeblk.c
usr/src/uts/intel/io/acpica/events/evgpeinit.c
usr/src/uts/intel/io/acpica/events/evgpeutil.c
usr/src/uts/intel/io/acpica/events/evmisc.c
usr/src/uts/intel/io/acpica/events/evregion.c
usr/src/uts/intel/io/acpica/events/evrgnini.c
usr/src/uts/intel/io/acpica/events/evsci.c
usr/src/uts/intel/io/acpica/events/evxface.c
usr/src/uts/intel/io/acpica/events/evxfevnt.c
usr/src/uts/intel/io/acpica/events/evxfgpe.c
usr/src/uts/intel/io/acpica/events/evxfregn.c
usr/src/uts/intel/io/acpica/executer/exconfig.c



usr/src/uts/intel/io/acpica/executer/exconvrt.c
usr/src/uts/intel/io/acpica/executer/excreate.c
usr/src/uts/intel/io/acpica/executer/exdebug.c
usr/src/uts/intel/io/acpica/executer/exdump.c
usr/src/uts/intel/io/acpica/executer/exfield.c
usr/src/uts/intel/io/acpica/executer/exfldio.c
usr/src/uts/intel/io/acpica/executer/exmisc.c
usr/src/uts/intel/io/acpica/executer/exmutex.c
usr/src/uts/intel/io/acpica/executer/exnames.c
usr/src/uts/intel/io/acpica/executer/exoparg1.c
usr/src/uts/intel/io/acpica/executer/exoparg2.c
usr/src/uts/intel/io/acpica/executer/exoparg3.c
usr/src/uts/intel/io/acpica/executer/exoparg6.c
usr/src/uts/intel/io/acpica/executer/exprep.c
usr/src/uts/intel/io/acpica/executer/exregion.c
usr/src/uts/intel/io/acpica/executer/exresnte.c
usr/src/uts/intel/io/acpica/executer/exresolv.c
usr/src/uts/intel/io/acpica/executer/exresop.c
usr/src/uts/intel/io/acpica/executer/exstore.c
usr/src/uts/intel/io/acpica/executer/exstoren.c
usr/src/uts/intel/io/acpica/executer/exstorob.c
usr/src/uts/intel/io/acpica/executer/exsystem.c
usr/src/uts/intel/io/acpica/executer/exutils.c
usr/src/uts/intel/io/acpica/hardware/hwacpi.c
usr/src/uts/intel/io/acpica/hardware/hwgpe.c
usr/src/uts/intel/io/acpica/hardware/hwpci.c
usr/src/uts/intel/io/acpica/hardware/hwregs.c
usr/src/uts/intel/io/acpica/hardware/hwsleep.c
usr/src/uts/intel/io/acpica/hardware/hwtimer.c
usr/src/uts/intel/io/acpica/hardware/hwvalid.c
usr/src/uts/intel/io/acpica/hardware/hwxface.c
usr/src/uts/intel/io/acpica/namespace/nsaccess.c
usr/src/uts/intel/io/acpica/namespace/nsalloc.c
usr/src/uts/intel/io/acpica/namespace/nsdump.c
usr/src/uts/intel/io/acpica/namespace/nsdumpdv.c
usr/src/uts/intel/io/acpica/namespace/nseval.c
usr/src/uts/intel/io/acpica/namespace/nsinit.c
usr/src/uts/intel/io/acpica/namespace/nsload.c
usr/src/uts/intel/io/acpica/namespace/nsnames.c
usr/src/uts/intel/io/acpica/namespace/nsobject.c
usr/src/uts/intel/io/acpica/namespace/nsparse.c
usr/src/uts/intel/io/acpica/namespace/nspredef.c
usr/src/uts/intel/io/acpica/namespace/nsrepair.c
usr/src/uts/intel/io/acpica/namespace/nsrepair2.c
usr/src/uts/intel/io/acpica/namespace/nssearch.c
usr/src/uts/intel/io/acpica/namespace/nsutils.c
usr/src/uts/intel/io/acpica/namespace/nswalk.c
usr/src/uts/intel/io/acpica/namespace/nsxfeval.c
usr/src/uts/intel/io/acpica/namespace/nsxfname.c
usr/src/uts/intel/io/acpica/namespace/nsxfobj.c
usr/src/uts/intel/io/acpica/parser/psargs.c
usr/src/uts/intel/io/acpica/parser/psloop.c
usr/src/uts/intel/io/acpica/parser/psopcode.c
usr/src/uts/intel/io/acpica/parser/psparse.c
usr/src/uts/intel/io/acpica/parser/psscope.c
usr/src/uts/intel/io/acpica/parser/pstree.c
usr/src/uts/intel/io/acpica/parser/psutils.c
usr/src/uts/intel/io/acpica/parser/pswalk.c
usr/src/uts/intel/io/acpica/parser/psxface.c
usr/src/uts/intel/io/acpica/resources/rsaddr.c
usr/src/uts/intel/io/acpica/resources/rscalc.c
usr/src/uts/intel/io/acpica/resources/rscreate.c
usr/src/uts/intel/io/acpica/resources/rsdump.c
usr/src/uts/intel/io/acpica/resources/rsinfo.c
usr/src/uts/intel/io/acpica/resources/rsio.c
usr/src/uts/intel/io/acpica/resources/rsirq.c

```
usr/src/uts/intel/io/acpica/resources/rslist.c
usr/src/uts/intel/io/acpica/resources/rsmemory.c
usr/src/uts/intel/io/acpica/resources/rsmisc.c
usr/src/uts/intel/io/acpica/resources/rsutils.c
usr/src/uts/intel/io/acpica/resources/rsxface.c
usr/src/uts/intel/io/acpica/tables/tbfadt.c
usr/src/uts/intel/io/acpica/tables/tbfind.c
usr/src/uts/intel/io/acpica/tables/tbinstal.c
usr/src/uts/intel/io/acpica/tables/tbutils.c
usr/src/uts/intel/io/acpica/tables/tbxface.c
usr/src/uts/intel/io/acpica/tables/tbxfroot.c
usr/src/uts/intel/io/acpica/utilities/utalloc.c
usr/src/uts/intel/io/acpica/utilities/utcache.c
usr/src/uts/intel/io/acpica/utilities/utclib.c
usr/src/uts/intel/io/acpica/utilities/utcopy.c
usr/src/uts/intel/io/acpica/utilities/utdebug.c
usr/src/uts/intel/io/acpica/utilities/utdecode.c
usr/src/uts/intel/io/acpica/utilities/utdelete.c
usr/src/uts/intel/io/acpica/utilities/uteval.c
usr/src/uts/intel/io/acpica/utilities/utglobal.c
usr/src/uts/intel/io/acpica/utilities/utids.c
usr/src/uts/intel/io/acpica/utilities/utinit.c
usr/src/uts/intel/io/acpica/utilities/utlock.c
usr/src/uts/intel/io/acpica/utilities/utmath.c
usr/src/uts/intel/io/acpica/utilities/utmisc.c
usr/src/uts/intel/io/acpica/utilities/utmutex.c
usr/src/uts/intel/io/acpica/utilities/utobject.c
usr/src/uts/intel/io/acpica/utilities/utosi.c
usr/src/uts/intel/io/acpica/utilities/utresrc.c
usr/src/uts/intel/io/acpica/utilities/utstate.c
usr/src/uts/intel/io/acpica/utilities/uttrack.c
usr/src/uts/intel/io/acpica/utilities/utxface.c
usr/src/uts/intel/io/acpica/utilities/utxferror.c
usr/src/uts/intel/sys/acpi/acapps.h
usr/src/uts/intel/sys/acpi/accommon.h
usr/src/uts/intel/sys/acpi/acconfig.h
usr/src/uts/intel/sys/acpi/acdebug.h
usr/src/uts/intel/sys/acpi/acdisasm.h
usr/src/uts/intel/sys/acpi/acdispat.h
usr/src/uts/intel/sys/acpi/acevents.h
usr/src/uts/intel/sys/acpi/acexcep.h
usr/src/uts/intel/sys/acpi/acglobal.h
usr/src/uts/intel/sys/acpi/achware.h
usr/src/uts/intel/sys/acpi/acinterp.h
usr/src/uts/intel/sys/acpi/aclocal.h
usr/src/uts/intel/sys/acpi/acmacros.h
usr/src/uts/intel/sys/acpi/acnames.h
usr/src/uts/intel/sys/acpi/acnamesp.h
usr/src/uts/intel/sys/acpi/acobject.h
usr/src/uts/intel/sys/acpi/acopcode.h
usr/src/uts/intel/sys/acpi/acoutput.h
usr/src/uts/intel/sys/acpi/acparser.h
usr/src/uts/intel/sys/acpi/acpi.h
usr/src/uts/intel/sys/acpi/acpiosxf.h
usr/src/uts/intel/sys/acpi/acpixf.h
usr/src/uts/intel/sys/acpi/acpredef.h
usr/src/uts/intel/sys/acpi/acresrc.h
usr/src/uts/intel/sys/acpi/acrestyp.h
usr/src/uts/intel/sys/acpi/acstruct.h
usr/src/uts/intel/sys/acpi/actables.h
usr/src/uts/intel/sys/acpi/actbl.h
usr/src/uts/intel/sys/acpi/actbl1.h
usr/src/uts/intel/sys/acpi/actbl2.h
usr/src/uts/intel/sys/acpi/actypes.h
usr/src/uts/intel/sys/acpi/acutils.h
usr/src/uts/intel/sys/acpi/amlcode.h
```



```
usr/src/uts/intel/sys/acpi/amlresrc.h
usr/src/uts/intel/sys/acpi/platform/accygwin.h
usr/src/uts/intel/sys/acpi/platform/acefi.h
usr/src/uts/intel/sys/acpi/platform/acenv.h
usr/src/uts/intel/sys/acpi/platform/acfreebsd.h
usr/src/uts/intel/sys/acpi/platform/acgcc.h
usr/src/uts/intel/sys/acpi/platform/acintel.h
usr/src/uts/intel/sys/acpi/platform/aclinux.h
usr/src/uts/intel/sys/acpi/platform/acmsvc.h
usr/src/uts/intel/sys/acpi/platform/acnetbsd.h
usr/src/uts/intel/sys/acpi/platform/acos2.h
usr/src/uts/intel/sys/acpi/platform/acsolaris.h
usr/src/uts/intel/sys/acpi/platform/acwin.h
usr/src/uts/intel/sys/acpi/platform/acwin64.h
```

```
**********************************************************
   42852 Thu Jul 12 12:22:27 2012
new/usr/src/uts/common/Makefile.files
XXXX Intel X540 support
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
#

#
# This Makefile defines all file modules for the directory uts/common
# and its children. These are the source files which may be considered
# common to all SunOS systems.

i386_CORE_OBJS += \
                atomic.o        \
                avintr.o        \
                pic.o

sparc_CORE_OBJS +=

COMMON_CORE_OBJS +=             \
                beep.o          \
                bitset.o        \
                bp_map.o        \
                brand.o         \
                cpucaps.o       \
                cmt.o           \
                cmt_policy.o    \
                cpu.o           \
                cpu_event.o     \
                cpu_intr.o      \
                cpu_pm.o        \
                cpupart.o       \
                cap_util.o      \
                disp.o          \
                group.o         \
                kstat_fr.o      \
                iscsiboot_prop.o        \
                lgrp.o          \
                lgrp_topo.o     \
                mmapobj.o       \
                mutex.o         \
```



```
                page_lock.o     \
                page_retire.o   \
                panic.o         \
                param.o         \
                pg.o            \
                pghw.o          \
                putnext.o       \
                rctl_proc.o     \
                rwlock.o        \
                seg_kmem.o      \
                softint.o       \
                string.o        \
                strtol.o        \
                strtoul.o       \
                strtoll.o       \
                strtoull.o      \
                thread_intr.o   \
                vm_page.o       \
                vm_pagelist.o   \
                zlib_obj.o      \
                clock_tick.o

CORE_OBJS +=    $(COMMON_CORE_OBJS) $($(MACH)_CORE_OBJS)

ZLIB_OBJS =     zutil.o zmod.o zmod_subr.o \
                adler32.o crc32.o deflate.o inffast.o \
                inflate.o inftrees.o trees.o

GENUNIX_OBJS += \
                access.o        \
                acl.o           \
                acl_common.o    \
                adjtime.o       \
                alarm.o         \
                aio_subr.o      \
                auditsys.o      \
                audit_core.o    \
                audit_zone.o    \
                audit_memory.o  \
                autoconf.o      \
                avl.o           \
                bdev_dsort.o    \
                bio.o           \
                bitmap.o        \
                blabel.o        \
                brandsys.o      \
                bz2blocksort.o  \
                bz2compress.o   \
                bz2decompress.o \
                bz2randtable.o  \
                bz2bzlib.o      \
                bz2crctable.o   \
                bz2huffman.o    \
                callb.o         \
                callout.o       \
                chdir.o         \
                chmod.o         \
                chown.o         \
                cladm.o         \
                class.o         \
                clock.o         \
                clock_highres.o \
                clock_realtime.o\
                close.o         \
                compress.o      \
                condvar.o       \
```

```
128                 conf.o          \
129                 console.o       \
130                 contract.o      \
131                 copyops.o       \
132                 core.o          \
133                 corectl.o       \
134                 cred.o          \
135                 cs_stubs.o      \
136                 dacf.o          \
137                 dacf_clnt.o     \
138                 damap.o \
139                 cyclic.o        \
140                 ddi.o           \
141                 ddifm.o         \
142                 ddi_hp_impl.o   \
143                 ddi_hp_ndi.o    \
144                 ddi_intr.o      \
145                 ddi_intr_impl.o \
146                 ddi_intr_irm.o  \
147                 ddi_nodeid.o    \
148                 ddi_timer.o     \
149                 devcfg.o        \
150                 devcache.o      \
151                 device.o        \
152                 devid.o         \
153                 devid_cache.o   \
154                 devid_scsi.o    \
155                 devid_smp.o     \
156                 devpolicy.o     \
157                 disp_lock.o     \
158                 dnlc.o          \
159                 driver.o        \
160                 dumpsubr.o      \
161                 driver_lyr.o    \
162                 dtrace_subr.o   \
163                 errorq.o        \
164                 etheraddr.o     \
165                 evchannels.o    \
166                 exacct.o        \
167                 exacct_core.o   \
168                 exec.o          \
169                 exit.o          \
170                 fbio.o          \
171                 fcntl.o         \
172                 fdbuffer.o      \
173                 fdsync.o        \
174                 fem.o           \
175                 ffs.o           \
176                 fio.o           \
177                 flock.o         \
178                 fm.o            \
179                 fork.o          \
180                 vpm.o           \
181                 fs_reparse.o    \
182                 fs_subr.o       \
183                 fsflush.o       \
184                 ftrace.o        \
185                 getcwd.o        \
186                 getdents.o      \
187                 getloadavg.o    \
188                 getpagesizes.o  \
189                 getpid.o        \
190                 gfs.o           \
191                 rusagesys.o     \
192                 gid.o           \
193                 groups.o        \
```



```
194                 grow.o          \
195                 hat_refmod.o    \
196                 id32.o          \
197                 id_space.o      \
198                 inet_ntop.o     \
199                 instance.o      \
200                 ioctl.o         \
201                 ip_cksum.o      \
202                 issetugid.o     \
203                 ippconf.o       \
204                 kcpc.o          \
205                 kdi.o           \
206                 kiconv.o        \
207                 klpd.o          \
208                 kmem.o          \
209                 ksyms_snapshot.o        \
210                 l_strplumb.o    \
211                 labelsys.o      \
212                 link.o          \
213                 list.o          \
214                 lockstat_subr.o \
215                 log_sysevent.o  \
216                 logsubr.o       \
217                 lookup.o        \
218                 lseek.o         \
219                 ltos.o          \
220                 lwp.o           \
221                 lwp_create.o    \
222                 lwp_info.o      \
223                 lwp_self.o      \
224                 lwp_sobj.o      \
225                 lwp_timer.o     \
226                 lwpsys.o        \
227                 main.o          \
228                 mmapobjsys.o    \
229                 memcntl.o       \
230                 memstr.o        \
231                 lgrpsys.o       \
232                 mkdir.o         \
233                 mknod.o         \
234                 mount.o         \
235                 move.o          \
236                 msacct.o        \
237                 multidata.o     \
238                 nbmlock.o       \
239                 ndifm.o         \
240                 nice.o          \
241                 netstack.o      \
242                 ntptime.o       \
243                 nvpair.o        \
244                 nvpair_alloc_system.o   \
245                 nvpair_alloc_fixed.o    \
246                 fnvpair.o       \
247                 octet.o         \
248                 open.o          \
249                 p_online.o      \
250                 pathconf.o      \
251                 pathname.o      \
252                 pause.o         \
253                 serializer.o    \
254                 pci_intr_lib.o  \
255                 pci_cap.o       \
256                 pcifm.o         \
257                 pgrp.o          \
258                 pgrpsys.o       \
259                 pid.o           \
```

```
260                 pkp_hash.o      \
261                 policy.o        \
262                 poll.o          \
263                 pool.o          \
264                 pool_pset.o     \
265                 port_subr.o     \
266                 ppriv.o         \
267                 printf.o        \
268                 priocntl.o      \
269                 priv.o          \
270                 priv_const.o    \
271                 proc.o          \
272                 procset.o       \
273                 processor_bind.o        \
274                 processor_info.o        \
275                 profil.o        \
276                 project.o       \
277                 qsort.o         \
278                 rctl.o          \
279                 rctlsys.o       \
280                 readlink.o      \
281                 refstr.o        \
282                 rename.o        \
283                 resolvepath.o   \
284                 retire_store.o  \
285                 process.o       \
286                 rlimit.o        \
287                 rmap.o          \
288                 rw.o            \
289                 rwstlock.o      \
290                 sad_conf.o      \
291                 sid.o           \
292                 sidsys.o        \
293                 sched.o         \
294                 schedctl.o      \
295                 sctp_crc32.o    \
296                 seg_dev.o       \
297                 seg_kp.o        \
298                 seg_kpm.o       \
299                 seg_map.o       \
300                 seg_vn.o        \
301                 seg_spt.o       \
302                 semaphore.o     \
303                 sendfile.o      \
304                 session.o       \
305                 share.o         \
306                 shuttle.o       \
307                 sig.o           \
308                 sigaction.o     \
309                 sigaltstack.o   \
310                 signotify.o     \
311                 sigpending.o    \
312                 sigprocmask.o   \
313                 sigqueue.o      \
314                 sigsendset.o    \
315                 sigsuspend.o    \
316                 sigtimedwait.o  \
317                 sleepq.o        \
318                 sock_conf.o     \
319                 space.o         \
320                 sscanf.o        \
321                 stat.o          \
322                 statfs.o        \
323                 statvfs.o       \
324                 stol.o          \
325                 str_conf.o      \
```



```
326                 strcalls.o      \
327                 stream.o        \
328                 streamio.o      \
329                 strext.o        \
330                 strsubr.o       \
331                 strsun.o        \
332                 subr.o          \
333                 sunddi.o        \
334                 sunmdi.o        \
335                 sunndi.o        \
336                 sunpci.o        \
337                 sunpm.o         \
338                 sundlpi.o       \
339                 suntpi.o        \
340                 swap_subr.o     \
341                 swap_vnops.o    \
342                 symlink.o       \
343                 sync.o          \
344                 sysclass.o      \
345                 sysconfig.o     \
346                 sysent.o        \
347                 sysfs.o         \
348                 systeminfo.o    \
349                 task.o          \
350                 taskq.o         \
351                 tasksys.o       \
352                 time.o          \
353                 timer.o         \
354                 times.o         \
355                 timers.o        \
356                 thread.o        \
357                 tlabel.o        \
358                 tnf_res.o       \
359                 turnstile.o     \
360                 tty_common.o    \
361                 u8_textprep.o   \
362                 uadmin.o        \
363                 uconv.o         \
364                 ucredsys.o      \
365                 uid.o           \
366                 umask.o         \
367                 umount.o        \
368                 uname.o         \
369                 unix_bb.o       \
370                 unlink.o        \
371                 urw.o           \
372                 utime.o         \
373                 utssys.o        \
374                 uucopy.o        \
375                 vfs.o           \
376                 vfs_conf.o      \
377                 vmem.o          \
378                 vm_anon.o       \
379                 vm_as.o         \
380                 vm_meter.o      \
381                 vm_pageout.o    \
382                 vm_pvn.o        \
383                 vm_rm.o         \
384                 vm_seg.o        \
385                 vm_subr.o       \
386                 vm_swap.o       \
387                 vm_usage.o      \
388                 vnode.o         \
389                 vuid_queue.o    \
390                 vuid_store.o    \
391                 waitq.o         \
```

```
392                 watchpoint.o    \
393                 yield.o         \
394                 scsi_confdata.o \
395                 xattr.o         \
396                 xattr_common.o  \
397                 xdr_mblk.o      \
398                 xdr_mem.o       \
399                 xdr.o           \
400                 xdr_array.o     \
401                 xdr_refer.o     \
402                 xhat.o          \
403                 zone.o

405 #
406 #       Stubs for the stand-alone linker/loader
407 #
408 sparc_GENSTUBS_OBJS =   \
409         kobj_stubs.o

411 i386_GENSTUBS_OBJS =

413 COMMON_GENSTUBS_OBJS =

415 GENSTUBS_OBJS += $(COMMON_GENSTUBS_OBJS) $($(MACH)_GENSTUBS_OBJS)

417 #
418 #       DTrace and DTrace Providers
419 #
420 DTRACE_OBJS += dtrace.o dtrace_isa.o dtrace_asm.o

422 SDT_OBJS += sdt_subr.o

424 PROFILE_OBJS += profile.o

426 SYSTRACE_OBJS += systrace.o

428 LOCKSTAT_OBJS += lockstat.o

430 FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

432 DCPC_OBJS += dcpc.o

434 #
435 #       Driver (pseudo-driver) Modules
436 #
437 IPP_OBJS +=     ippctl.o

439 AUDIO_OBJS += audio_client.o audio_ddi.o audio_engine.o \
440             audio_fltdata.o audio_format.o audio_ctrl.o \
441             audio_grc3.o audio_output.o audio_input.o \
442             audio_oss.o audio_sun.o

444 AUDIOEMU10K_OBJS += audioemu10k.o

446 AUDIOENS_OBJS += audioens.o

448 AUDIOVIA823X_OBJS += audiovia823x.o

450 AUDIOVIA97_OBJS += audiovia97.o

452 AUDIO1575_OBJS += audio1575.o

454 AUDIO810_OBJS += audio810.o

456 AUDIOCMI_OBJS += audiocmi.o
```



```
458 AUDIOCMIHD_OBJS += audiocmihd.o

460 AUDIOHD_OBJS += audiohd.o

462 AUDIOIXP_OBJS += audioixp.o

464 AUDIOLS_OBJS += audiols.o

466 AUDIOP16X_OBJS += audiop16x.o

468 AUDIOPCI_OBJS += audiopci.o

470 AUDIOSOLO_OBJS += audiosolo.o

472 AUDIOTS_OBJS += audiots.o

474 AC97_OBJS += ac97.o ac97_ad.o ac97_alc.o ac97_cmi.o

476 BLKDEV_OBJS += blkdev.o

478 CARDBUS_OBJS += cardbus.o cardbus_hp.o cardbus_cfg.o

480 CONSKBD_OBJS += conskbd.o

482 CONSMS_OBJS +=  consms.o

484 OLDPTY_OBJS +=  tty_ptyconf.o

486 PTC_OBJS +=     tty_pty.o

488 PTSL_OBJS +=    tty_pts.o

490 PTM_OBJS +=     ptm.o

492 MII_OBJS +=     mii.o mii_cicada.o mii_natsemi.o mii_intel.o mii_qualsemi.o \
493                 mii_marvell.o mii_realtek.o mii_other.o

495 PTS_OBJS +=     pts.o

497 PTY_OBJS +=     ptms_conf.o

499 SAD_OBJS +=     sad.o

501 MD4_OBJS +=     md4.o md4_mod.o

503 MD5_OBJS +=     md5.o md5_mod.o

505 SHA1_OBJS +=    sha1.o sha1_mod.o

507 SHA2_OBJS +=    sha2.o sha2_mod.o

509 IPGPC_OBJS +=   classifierddi.o classifier.o filters.o trie.o table.o \
510                 ba_table.o

512 DSCPMK_OBJS +=  dscpmk.o dscpmkddi.o

514 DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o

516 FLOWACCT_OBJS +=        flowacctddi.o flowacct.o

518 TOKENMT_OBJS += tokenmt.o tokenmtddi.o

520 TSWTCL_OBJS +=  tswtcl.o tswtclddi.o

522 ARP_OBJS +=     arpddi.o
```

```
524 ICMP_OBJS +=    icmpddi.o

526 ICMP6_OBJS +=   icmp6ddi.o

528 RTS_OBJS +=     rtsddi.o

530 IP_ICMP_OBJS =  icmp.o icmp_opt_data.o
531 IP_RTS_OBJS =   rts.o rts_opt_data.o
532 IP_TCP_OBJS =   tcp.o tcp_fusion.o tcp_opt_data.o tcp_sack.o tcp_stats.o \
533                 tcp_misc.o tcp_timers.o tcp_time_wait.o tcp_tpi.o tcp_output.o \
534                 tcp_input.o tcp_socket.o tcp_bind.o tcp_cluster.o tcp_tunables.o
535 IP_UDP_OBJS =   udp.o udp_opt_data.o udp_tunables.o udp_stats.o
536 IP_SCTP_OBJS =  sctp.o sctp_opt_data.o sctp_output.o \
537                 sctp_init.o sctp_input.o sctp_cookie.o \
538                 sctp_conn.o sctp_error.o sctp_snmp.o \
539                 sctp_tunables.o sctp_shutdown.o sctp_common.o \
540                 sctp_timer.o sctp_heartbeat.o sctp_hash.o \
541                 sctp_bind.o sctp_notify.o sctp_asconf.o \
542                 sctp_addr.o tn_ipopt.o tnet.o ip_netinfo.o \
543                 sctp_misc.o
544 IP_ILB_OBJS =   ilb.o ilb_nat.o ilb_conn.o ilb_alg_hash.o ilb_alg_rr.o

546 IP_OBJS +=      igmp.o ipmp.o ip.o ip6.o ip6_asp.o ip6_if.o ip6_ire.o \
547                 ip6_rts.o ip_if.o ip_ire.o ip_listutils.o ip_mroute.o \
548                 ip_multi.o ip2mac.o ip_ndp.o ip_rts.o ip_srcid.o \
549                 ipddi.o ipdrop.o mi.o nd.o tunables.o optcom.o snmpcom.o \
550                 ipsec_loader.o spd.o ipclassifier.o inet_common.o ip_squeue.o \
551                 squeue.o ip_sadb.o ip_ftable.o proto_set.o radix.o ip_dummy.o \
552                 ip_helper_stream.o ip_tunables.o \
553                 ip_output.o ip_input.o ip6_input.o ip6_output.o ip_arp.o \
554                 conn_opt.o ip_attr.o ip_dce.o \
555                 $(IP_ICMP_OBJS) \
556                 $(IP_RTS_OBJS) \
557                 $(IP_TCP_OBJS) \
558                 $(IP_UDP_OBJS) \
559                 $(IP_SCTP_OBJS) \
560                 $(IP_ILB_OBJS)

562 IP6_OBJS +=     ip6ddi.o

564 HOOK_OBJS +=    hook.o

566 NETI_OBJS +=    neti_impl.o neti_mod.o neti_stack.o

568 KEYSOCK_OBJS += keysockddi.o keysock.o keysock_opt_data.o

570 IPNET_OBJS +=   ipnet.o ipnet_bpf.o

572 SPDSOCK_OBJS += spdsockddi.o spdsock.o spdsock_opt_data.o

574 IPSECESP_OBJS += ipsecespddi.o ipsecesp.o

576 IPSECAH_OBJS += ipsecahddi.o ipsecah.o sadb.o

578 SPPP_OBJS +=    sppp.o sppp_dlpi.o sppp_mod.o s_common.o

580 SPPPTUN_OBJS += sppptun.o sppptun_mod.o

582 SPPPASYN_OBJS += spppasyn.o spppasyn_mod.o

584 SPPPCOMP_OBJS += spppcomp.o spppcomp_mod.o deflate.o bsd-comp.o vjcompress.o \
585                 zlib.o

587 TCP_OBJS +=     tcpddi.o

589 TCP6_OBJS +=    tcp6ddi.o
```



```
591 NCA_OBJS +=     ncaddi.o

593 SDP_SOCK_MOD_OBJS += sockmod_sdp.o socksdp.o socksdpsubr.o

595 SCTP_SOCK_MOD_OBJS += sockmod_sctp.o socksctp.o socksctpsubr.o

597 PFP_SOCK_MOD_OBJS += sockmod_pfp.o

599 RDS_SOCK_MOD_OBJS += sockmod_rds.o

601 RDS_OBJS +=     rdsddi.o rdssubr.o rds_opt.o rds_ioctl.o

603 RDSIB_OBJS +=   rdsib.o rdsib_ib.o rdsib_cm.o rdsib_ep.o rdsib_buf.o \
604                 rdsib_debug.o rdsib_sc.o

606 RDSV3_OBJS +=   af_rds.o rdsv3_ddi.o bind.o loop.o threads.o connection.o \
607                 transport.o cong.o sysctl.o message.o rds_recv.o send.o \
608                 stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
609                 ib_recv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
610                 rdsv3_sc.o rdsv3_debug.o rdsv3_impl.o rdma.o rdsv3_af_thr.o

612 ISER_OBJS +=    iser.o iser_cm.o iser_cq.o iser_ib.o iser_idm.o \
613                 iser_resource.o iser_xfer.o

615 UDP_OBJS +=     udpddi.o

617 UDP6_OBJS +=    udp6ddi.o

619 SY_OBJS +=      gentty.o

621 TCO_OBJS +=     ticots.o

623 TCOO_OBJS +=    ticotsord.o

625 TCL_OBJS +=     ticlts.o

627 TL_OBJS +=      tl.o

629 DUMP_OBJS +=    dump.o

631 BPF_OBJS +=     bpf.o bpf_filter.o bpf_mod.o bpf_dlt.o bpf_mac.o

633 CLONE_OBJS +=   clone.o

635 CN_OBJS +=      cons.o

637 DLD_OBJS +=     dld_drv.o dld_proto.o dld_str.o dld_flow.o

639 DLS_OBJS +=     dls.o dls_link.o dls_mod.o dls_stat.o dls_mgmt.o

641 GLD_OBJS +=     gld.o gldutil.o

643 MAC_OBJS +=     mac.o  mac_bcast.o mac_client.o mac_datapath_setup.o mac_flow.o
644                 mac_hio.o mac_mod.o mac_ndd.o mac_provider.o mac_sched.o \
645                 mac_protect.o mac_soft_ring.o mac_stat.o mac_util.o

647 MAC_6TO4_OBJS +=        mac_6to4.o

649 MAC_ETHER_OBJS +=       mac_ether.o

651 MAC_IPV4_OBJS +=        mac_ipv4.o

653 MAC_IPV6_OBJS +=        mac_ipv6.o

655 MAC_WIFI_OBJS +=        mac_wifi.o
```

```
MAC_IB_OBJS +=          mac_ib.o

IPTUN_OBJS +=   iptun_dev.o iptun_ctl.o iptun.o

AGGR_OBJS +=    aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
                aggr_send.o aggr_recv.o aggr_lacp.o

SOFTMAC_OBJS += softmac_main.o softmac_ctl.o softmac_capab.o \
                softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o

NET80211_OBJS += net80211.o net80211_proto.o net80211_input.o \
                 net80211_output.o net80211_node.o net80211_crypto.o \
                 net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
                 net80211_crypto_tkip.o net80211_crypto_ccmp.o  \
                 net80211_ht.o

VNIC_OBJS +=    vnic_ctl.o vnic_dev.o

SIMNET_OBJS +=  simnet.o

IB_OBJS +=      ibnex.o ibnex_ioctl.o ibnex_hca.o

IBCM_OBJS +=    ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
                ibcm_arp.o ibcm_arp_link.o

IBDM_OBJS +=    ibdm.o

IBDMA_OBJS +=   ibdma.o

IBMF_OBJS +=    ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.
                ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
                ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
                ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o

IBTL_OBJS +=    ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
                ibtl_cq.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
                ibtl_mcg.o ibtl_ibnex.o ibtl_srq.o ibtl_part.o

TAVOR_OBJS +=   tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
                tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
                tavor_mr.o tavor_qp.o tavor_qpmod.o tavor_rsrc.o \
                tavor_srq.o tavor_stats.o tavor_umap.o tavor_wr.o

HERMON_OBJS +=  hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
                hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
                hermon_mr.o hermon_qp.o hermon_qpmod.o hermon_rsrc.o \
                hermon_srq.o hermon_stats.o hermon_umap.o hermon_wr.o \
                hermon_fcoib.o hermon_fm.o

DAPLT_OBJS +=   daplt.o

SOL_OFS_OBJS += sol_cma.o sol_ib_cma.o sol_uobj.o \
                sol_ofs_debug_util.o sol_ofs_gen_util.o \
                sol_kverbs.o

SOL_UCMA_OBJS +=        sol_ucma.o

SOL_UVERBS_OBJS +=      sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
                        sol_uverbs_hca.o sol_uverbs_qp.o

SOL_UMAD_OBJS += sol_umad.o

KSTAT_OBJS +=   kstat.o

KSYMS_OBJS +=   ksyms.o
```



```
INSTANCE_OBJS += inst_sync.o

IWSCN_OBJS +=   iwscons.o

LOFI_OBJS +=    lofi.o LzmaDec.o

FSSNAP_OBJS +=  fssnap.o

FSSNAPIF_OBJS += fssnap_if.o

MM_OBJS +=      mem.o

PHYSMEM_OBJS += physmem.o

OPTIONS_OBJS += options.o

WINLOCK_OBJS += winlockio.o

PM_OBJS +=      pm.o
SRN_OBJS +=     srn.o

PSEUDO_OBJS +=  pseudonex.o

RAMDISK_OBJS += ramdisk.o

LLC1_OBJS += llc1.o

USBKBM_OBJS += usbkbm.o

USBWCM_OBJS += usbwcm.o

BOFI_OBJS += bofi.o

HID_OBJS += hid.o

HWA_RC_OBJS += hwarc.o

USBSKEL_OBJS += usbskel.o

USBVC_OBJS += usbvc.o usbvc_v4l2.o

HIDPARSER_OBJS += hidparser.o

USB_AC_OBJS += usb_ac.o

USB_AS_OBJS += usb_as.o

USB_AH_OBJS += usb_ah.o

USBMS_OBJS += usbms.o

USBPRN_OBJS += usbprn.o

UGEN_OBJS += ugen.o

USBSER_OBJS += usbser.o usbser_rseq.o

USBSACM_OBJS += usbsacm.o

USBSER_KEYSPAN_OBJS += usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o

USBS49_FW_OBJS += keyspan_49fw.o

USBSPRL_OBJS += usbser_pl2303.o pl2303_dsd.o
```

```
788 WUSB_CA_OBJS += wusb_ca.o

790 USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o

792 USBECM_OBJS += usbecm.o

794 WC_OBJS += wscons.o vcons.o

796 VCONS_CONF_OBJS += vcons_conf.o

798 SCSI_OBJS +=    scsi_capabilities.o scsi_confsubr.o scsi_control.o \
799                 scsi_data.o scsi_fm.o scsi_hba.o scsi_reset_notify.o \
800                 scsi_resource.o scsi_subr.o scsi_transport.o scsi_watch.o \
801                 smp_transport.o

803 SCSI_VHCI_OBJS +=               scsi_vhci.o mpapi_impl.o scsi_vhci_tpgs.o

805 SCSI_VHCI_F_SYM_OBJS +=         sym.o

807 SCSI_VHCI_F_TPGS_OBJS +=        tpgs.o

809 SCSI_VHCI_F_ASYM_SUN_OBJS +=    asym_sun.o

811 SCSI_VHCI_F_SYM_HDS_OBJS +=     sym_hds.o

813 SCSI_VHCI_F_TAPE_OBJS +=        tape.o

815 SCSI_VHCI_F_TPGS_TAPE_OBJS +=   tpgs_tape.o

817 SGEN_OBJS +=    sgen.o

819 SMP_OBJS +=     smp.o

821 SATA_OBJS +=    sata.o

823 USBA_OBJS +=    hcdi.o  usba.o  usbai.o hubdi.o parser.o genconsole.o \
824                 usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
825                 usba_devdb.o usba10_calls.o usba_ugen.o whcdi.o wa.o
826 USBA_WITHOUT_WUSB_OBJS +=       hcdi.o  usba.o  usbai.o hubdi.o parser.o gencons
827                 usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
828                 usba_devdb.o usba10_calls.o usba_ugen.o

830 USBA10_OBJS +=  usba10.o

832 RSM_OBJS +=     rsm.o   rsmka_pathmanager.o     rsmka_util.o

834 RSMOPS_OBJS +=  rsmops.o

836 S1394_OBJS +=   t1394.o t1394_errmsg.o s1394.o s1394_addr.o s1394_asynch.o \
837                 s1394_bus_reset.o s1394_cmp.o s1394_csr.o s1394_dev_disc.o \
838                 s1394_fa.o s1394_fcp.o \
839                 s1394_hotplug.o s1394_isoch.o s1394_misc.o h1394.o nx1394.o

841 HCI1394_OBJS += hci1394.o hci1394_async.o hci1394_attach.o hci1394_buf.o \
842                 hci1394_csr.o hci1394_detach.o hci1394_extern.o \
843                 hci1394_ioctl.o hci1394_isoch.o hci1394_isr.o \
844                 hci1394_ixl_comp.o hci1394_ixl_isr.o hci1394_ixl_misc.o \
845                 hci1394_ixl_update.o hci1394_misc.o hci1394_ohci.o \
846                 hci1394_q.o hci1394_s1394if.o hci1394_tlabel.o \
847                 hci1394_tlist.o hci1394_vendor.o

849 AV1394_OBJS +=  av1394.o av1394_as.o av1394_async.o av1394_cfgrom.o \
850                 av1394_cmp.o av1394_fcp.o av1394_isoch.o av1394_isoch_chan.o \
851                 av1394_isoch_recv.o av1394_isoch_xmit.o av1394_list.o \
852                 av1394_queue.o
```



```
854 DCAM1394_OBJS += dcam.o dcam_frame.o dcam_param.o dcam_reg.o \
855                 dcam_ring_buff.o

857 SCSA1394_OBJS += hba.o sbp2_driver.o sbp2_bus.o

859 SBP2_OBJS += cfgrom.o sbp2.o

861 PMODEM_OBJS += pmodem.o pmodem_cis.o cis.o cis_callout.o cis_handlers.o cis_para

863 DSW_OBJS +=     dsw.o dsw_dev.o ii_tree.o

865 NCALL_OBJS +=   ncall.o \
866                 ncall_stub.o

868 RDC_OBJS +=     rdc.o \
869                 rdc_dev.o \
870                 rdc_io.o \
871                 rdc_clnt.o \
872                 rdc_prot_xdr.o \
873                 rdc_svc.o \
874                 rdc_bitmap.o \
875                 rdc_health.o \
876                 rdc_subr.o \
877                 rdc_diskq.o

879 RDCSRV_OBJS +=  rdcsrv.o

881 RDCSTUB_OBJS += rdc_stub.o

883 SDBC_OBJS +=    sd_bcache.o \
884                 sd_bio.o \
885                 sd_conf.o \
886                 sd_ft.o \
887                 sd_hash.o \
888                 sd_io.o \
889                 sd_misc.o \
890                 sd_pcu.o \
891                 sd_tdaemon.o \
892                 sd_trace.o \
893                 sd_iob_impl0.o \
894                 sd_iob_impl1.o \
895                 sd_iob_impl2.o \
896                 sd_iob_impl3.o \
897                 sd_iob_impl4.o \
898                 sd_iob_impl5.o \
899                 sd_iob_impl6.o \
900                 sd_iob_impl7.o \
901                 safestore.o \
902                 safestore_ram.o

904 NSCTL_OBJS +=   nsctl.o \
905                 nsc_cache.o \
906                 nsc_disk.o \
907                 nsc_dev.o \
908                 nsc_freeze.o \
909                 nsc_gen.o \
910                 nsc_mem.o \
911                 nsc_ncallio.o \
912                 nsc_power.o \
913                 nsc_resv.o \
914                 nsc_rmspin.o \
915                 nsc_solaris.o \
916                 nsc_trap.o \
917                 nsc_list.o
918 UNISTAT_OBJS += spuni.o \
919                 spcs_s_k.o
```

```
NSKERN_OBJS +=  nsc_ddi.o \
                nsc_proc.o \
                nsc_raw.o \
                nsc_thread.o \
                nskernd.o

SV_OBJS +=      sv.o

PMCS_OBJS += pmcs_attach.o pmcs_ds.o pmcs_intr.o pmcs_nvram.o pmcs_sata.o \
                pmcs_scsa.o pmcs_smhba.o pmcs_subr.o pmcs_fwlog.o

PMCS8001FW_C_OBJS +=    pmcs_fw_hdr.o
PMCS8001FW_OBJS +=              $(PMCS8001FW_C_OBJS) SPCBoot.o ila.o firmware.o

#
#       Build up defines and paths.

ST_OBJS +=      st.o    st_conf.o

EMLXS_OBJS +=   emlxs_clock.o emlxs_dfc.o emlxs_dhchap.o emlxs_diag.o \
                emlxs_download.o emlxs_dump.o emlxs_els.o emlxs_event.o \
                emlxs_fcf.o emlxs_fcp.o emlxs_fct.o emlxs_hba.o emlxs_ip.o \
                emlxs_mbox.o emlxs_mem.o emlxs_msg.o emlxs_node.o \
                emlxs_pkt.o emlxs_sli3.o emlxs_sli4.o emlxs_solaris.o \
                emlxs_thread.o

EMLXS_FW_OBJS +=        emlxs_fw.o

OCE_OBJS +=     oce_buf.o oce_fm.o oce_gld.o oce_hw.o oce_intr.o oce_main.o \
                oce_mbx.o oce_mq.o oce_queue.o oce_rx.o oce_stat.o oce_tx.o \
                oce_utils.o

FCT_OBJS += discovery.o fct.o

QLT_OBJS += 2400.o 2500.o 8100.o qlt.o qlt_dma.o

SRPT_OBJS += srpt_mod.o srpt_ch.o srpt_cm.o srpt_ioc.o srpt_stp.o

FCOE_OBJS += fcoe.o fcoe_eth.o fcoe_fc.o

FCOET_OBJS += fcoet.o fcoet_eth.o fcoet_fc.o

FCOEI_OBJS += fcoei.o fcoei_eth.o fcoei_lv.o

ISCSIT_SHARED_OBJS += \
                iscsit_common.o

ISCSIT_OBJS +=  $(ISCSIT_SHARED_OBJS) \
                iscsit.o iscsit_tgt.o iscsit_sess.o iscsit_login.o \
                iscsit_text.o iscsit_isns.o iscsit_radiusauth.o \
                iscsit_radiuspacket.o iscsit_auth.o iscsit_authclient.o

PPPT_OBJS +=    alua_ic_if.o pppt.o pppt_msg.o pppt_tgt.o

STMF_OBJS += lun_map.o stmf.o

STMF_SBD_OBJS += sbd.o sbd_scsi.o sbd_pgr.o sbd_zvol.o

SYSMSG_OBJS +=  sysmsg.o

SES_OBJS +=     ses.o ses_sen.o ses_safte.o ses_ses.o

TNF_OBJS +=     tnf_buf.o       tnf_trace.o     tnf_writer.o    trace_init.o \
                trace_funcs.o   tnf_probe.o     tnf.o
```



```
LOGINDMUX_OBJS += logindmux.o

DEVINFO_OBJS += devinfo.o

DEVPOLL_OBJS += devpoll.o

DEVPOOL_OBJS += devpool.o

I8042_OBJS +=   i8042.o

KB8042_OBJS +=  \
                at_keyprocess.o \
                kb8042.o        \
                kb8042_keytables.o

MOUSE8042_OBJS += mouse8042.o

FDC_OBJS +=     fdc.o

ASY_OBJS +=     asy.o

ECPP_OBJS +=    ecpp.o

VUIDM3P_OBJS += vuidmice.o vuidm3p.o

VUIDM4P_OBJS += vuidmice.o vuidm4p.o

VUIDM5P_OBJS += vuidmice.o vuidm5p.o

VUIDPS2_OBJS += vuidmice.o vuidps2.o

HPCSVC_OBJS += hpcsvc.o

PCIE_MISC_OBJS += pcie.o pcie_fault.o pcie_hp.o pciehpc.o pcishpc.o pcie_pwr.o p

PCIHPNEXUS_OBJS += pcihp.o

OPENEEPR_OBJS += openprom.o

RANDOM_OBJS += random.o

PSHOT_OBJS += pshot.o

GEN_DRV_OBJS += gen_drv.o

TCLIENT_OBJS += tclient.o

TPHCI_OBJS += tphci.o

TVHCI_OBJS += tvhci.o

EMUL64_OBJS += emul64.o emul64_bsd.o

FCP_OBJS += fcp.o

FCIP_OBJS += fcip.o

FCSM_OBJS += fcsm.o

FCTL_OBJS += fctl.o

FP_OBJS += fp.o

QLC_OBJS += ql_api.o ql_debug.o ql_hba_fru.o ql_init.o ql_iocb.o ql_ioctl.o \
        ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o
```

```
1052 QLC_FW_2200_OBJS += ql_fw_2200.o

1054 QLC_FW_2300_OBJS += ql_fw_2300.o

1056 QLC_FW_2400_OBJS += ql_fw_2400.o

1058 QLC_FW_2500_OBJS += ql_fw_2500.o

1060 QLC_FW_6322_OBJS += ql_fw_6322.o

1062 QLC_FW_8100_OBJS += ql_fw_8100.o

1064 QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o

1066 ZCONS_OBJS += zcons.o

1068 NV_SATA_OBJS += nv_sata.o

1070 SI3124_OBJS += si3124.o

1072 AHCI_OBJS += ahci.o

1074 PCIIDE_OBJS += pci-ide.o

1076 PCEPP_OBJS += pcepp.o

1078 CPC_OBJS += cpc.o

1080 CPUID_OBJS += cpuid_drv.o

1082 SYSEVENT_OBJS += sysevent.o

1084 BL_OBJS += bl.o

1086 DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
1087             drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
1088             drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
1089             drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
1090             drm_cache.o drm_gem.o drm_mm.o ati_pcigart.o

1092 FM_OBJS += devfm.o devfm_machdep.o

1094 RTLS_OBJS +=    rtls.o

1096 #
1097 #                       exec modules
1098 #
1099 AOUTEXEC_OBJS +=aout.o

1101 ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o

1103 INTPEXEC_OBJS +=intp.o

1105 SHBINEXEC_OBJS +=shbin.o

1107 JAVAEXEC_OBJS +=java.o

1109 #
1110 #                       file system modules
1111 #
1112 AUTOFS_OBJS +=  auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o

1114 CACHEFS_OBJS += cachefs_cnode.o         cachefs_cod.o \
1115                 cachefs_dir.o           cachefs_dlog.o  cachefs_filegrp.o \
1116                 cachefs_fscache.o       cachefs_ioctl.o cachefs_log.o \
1117                 cachefs_module.o \
```



```
1118                 cachefs_noopc.o         cachefs_resource.o \
1119                 cachefs_strict.o \
1120                 cachefs_subr.o          cachefs_vfsops.o \
1121                 cachefs_vnops.o

1123 DCFS_OBJS +=    dc_vnops.o

1125 DEVFS_OBJS +=   devfs_subr.o    devfs_vfsops.o  devfs_vnops.o

1127 DEV_OBJS  +=    sdev_subr.o     sdev_vfsops.o   sdev_vnops.o    \
1128                 sdev_ptsops.o   sdev_zvolops.o  sdev_comm.o     \
1129                 sdev_profile.o  sdev_ncache.o   sdev_netops.o   \
1130                 sdev_ipnetops.o \
1131                 sdev_vtops.o

1133 CTFS_OBJS +=    ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
1134                 ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o

1136 OBJFS_OBJS +=   objfs_vfs.o     objfs_root.o    objfs_common.o \
1137                 objfs_odir.o    objfs_data.o

1139 FDFS_OBJS +=    fdops.o

1141 FIFO_OBJS +=    fifosubr.o      fifovnops.o

1143 PIPE_OBJS +=    pipe.o

1145 HSFS_OBJS +=    hsfs_node.o     hsfs_subr.o     hsfs_vfsops.o   hsfs_vnops.o \
1146                 hsfs_susp.o     hsfs_rrip.o     hsfs_susp_subr.o

1148 LOFS_OBJS +=    lofs_subr.o     lofs_vfsops.o   lofs_vnops.o

1150 NAMEFS_OBJS +=  namevfs.o       namevno.o

1152 NFS_OBJS +=     nfs_client.o    nfs_common.o    nfs_dump.o \
1153                 nfs_subr.o      nfs_vfsops.o    nfs_vnops.o \
1154                 nfs_xdr.o       nfs_sys.o       nfs_strerror.o \
1155                 nfs3_vfsops.o   nfs3_vnops.o    nfs3_xdr.o \
1156                 nfs_acl_vnops.o nfs_acl_xdr.o   nfs4_vfsops.o \
1157                 nfs4_vnops.o    nfs4_xdr.o      nfs4_idmap.o \
1158                 nfs4_shadow.o   nfs4_subr.o \
1159                 nfs4_attr.o     nfs4_rnode.o    nfs4_client.o \
1160                 nfs4_acache.o   nfs4_common.o   nfs4_client_state.o \
1161                 nfs4_callback.o nfs4_recovery.o nfs4_client_secinfo.o \
1162                 nfs4_client_debug.o     nfs_stats.o \
1163                 nfs4_acl.o      nfs4_stub_vnops.o       nfs_cmd.o

1165 NFSSRV_OBJS +=  nfs_server.o    nfs_srv.o       nfs3_srv.o \
1166                 nfs_acl_srv.o   nfs_auth.o      nfs_auth_xdr.o \
1167                 nfs_export.o    nfs_log.o       nfs_log_xdr.o \
1168                 nfs4_srv.o      nfs4_state.o    nfs4_srv_attr.o \
1169                 nfs4_srv_ns.o   nfs4_db.o       nfs4_srv_deleg.o \
1170                 nfs4_deleg_ops.o nfs4_srv_readdir.o nfs4_dispatch.o

1172 SMBSRV_SHARED_OBJS += \
1173                 smb_inet.o \
1174                 smb_match.o \
1175                 smb_msgbuf.o \
1176                 smb_oem.o \
1177                 smb_string.o \
1178                 smb_utf8.o \
1179                 smb_door_legacy.o \
1180                 smb_xdr.o \
1181                 smb_token.o \
1182                 smb_token_xdr.o \
1183                 smb_sid.o \
```

```
1184                smb_native.o \
1185                smb_netbios_util.o

1187 SMBSRV_OBJS +=  $(SMBSRV_SHARED_OBJS)                  \
1188                smb_acl.o                               \
1189                smb_alloc.o                             \
1190                smb_close.o                             \
1191                smb_common_open.o                       \
1192                smb_common_transact.o                   \
1193                smb_create.o                            \
1194                smb_delete.o                            \
1195                smb_directory.o                         \
1196                smb_dispatch.o                          \
1197                smb_echo.o                              \
1198                smb_fem.o                               \
1199                smb_find.o                              \
1200                smb_flush.o                             \
1201                smb_fsinfo.o                            \
1202                smb_fsops.o                             \
1203                smb_init.o                              \
1204                smb_kdoor.o                             \
1205                smb_kshare.o                            \
1206                smb_kutil.o                             \
1207                smb_lock.o                              \
1208                smb_lock_byte_range.o                   \
1209                smb_locking_andx.o                      \
1210                smb_logoff_andx.o                       \
1211                smb_mangle_name.o                       \
1212                smb_mbuf_marshaling.o                   \
1213                smb_mbuf_util.o                         \
1214                smb_negotiate.o                         \
1215                smb_net.o                               \
1216                smb_node.o                              \
1217                smb_nt_cancel.o                         \
1218                smb_nt_create_andx.o                    \
1219                smb_nt_transact_create.o                \
1220                smb_nt_transact_ioctl.o                 \
1221                smb_nt_transact_notify_change.o         \
1222                smb_nt_transact_quota.o                 \
1223                smb_nt_transact_security.o              \
1224                smb_odir.o                              \
1225                smb_ofile.o                             \
1226                smb_open_andx.o                         \
1227                smb_opipe.o                             \
1228                smb_oplock.o                            \
1229                smb_pathname.o                          \
1230                smb_print.o                             \
1231                smb_process_exit.o                      \
1232                smb_query_fileinfo.o                    \
1233                smb_read.o                              \
1234                smb_rename.o                            \
1235                smb_sd.o                                \
1236                smb_seek.o                              \
1237                smb_server.o                            \
1238                smb_session.o                           \
1239                smb_session_setup_andx.o                \
1240                smb_set_fileinfo.o                      \
1241                smb_signing.o                           \
1242                smb_tree.o                              \
1243                smb_trans2_create_directory.o           \
1244                smb_trans2_dfs.o                        \
1245                smb_trans2_find.o                       \
1246                smb_tree_connect.o                      \
1247                smb_unlock_byte_range.o                 \
1248                smb_user.o                              \
1249                smb_vfs.o                               \
```



```
1250                smb_vops.o                              \
1251                smb_vss.o                               \
1252                smb_write.o                             \
1253                smb_write_raw.o

1255 PCFS_OBJS +=    pc_alloc.o      pc_dir.o        pc_node.o       pc_subr.o \
1256                 pc_vfsops.o     pc_vnops.o

1258 PROC_OBJS +=    prcontrol.o     prioctl.o       prsubr.o        prusrio.o \
1259                 prvfsops.o      prvnops.o

1261 MNTFS_OBJS +=   mntvfsops.o     mntvnops.o

1263 SHAREFS_OBJS += sharetab.o      sharefs_vfsops.o        sharefs_vnops.o

1265 SPEC_OBJS +=    specsubr.o      specvfsops.o    specvnops.o

1267 SOCK_OBJS +=    socksubr.o      sockvfsops.o    sockparams.o    \
1268                 socksyscalls.o  socktpi.o       sockstr.o \
1269                 sockcommon_vnops.o      sockcommon_subr.o \
1270                 sockcommon_sops.o       sockcommon.o    \
1271                 sock_notsupp.o  socknotify.o \
1272                 nl7c.o          nl7curi.o       nl7chttp.o      nl7clogd.o \
1273                 nl7cnca.o       sodirect.o      sockfilter.o

1275 TMPFS_OBJS +=   tmp_dir.o       tmp_subr.o      tmp_tnode.o     tmp_vfsops.o \
1276                 tmp_vnops.o

1278 UDFS_OBJS +=    udf_alloc.o     udf_bmap.o      udf_dir.o       \
1279                 udf_inode.o     udf_subr.o      udf_vfsops.o    \
1280                 udf_vnops.o

1282 UFS_OBJS +=     ufs_alloc.o     ufs_bmap.o      ufs_dir.o       ufs_xattr.o \
1283                 ufs_inode.o     ufs_subr.o      ufs_tables.o    ufs_vfsops.o \
1284                 ufs_vnops.o     quota.o         quotacalls.o    quota_ufs.o \
1285                 ufs_filio.o     ufs_lockfs.o    ufs_thread.o    ufs_trans.o \
1286                 ufs_acl.o       ufs_panic.o     ufs_directio.o  ufs_log.o \
1287                 ufs_extvnops.o  ufs_snap.o      lufs.o          lufs_thread.o \
1288                 lufs_log.o      lufs_map.o      lufs_top.o      lufs_debug.o
1289 VSCAN_OBJS +=   vscan_drv.o     vscan_svc.o vscan_door.o

1291 NSMB_OBJS +=    smb_conn.o      smb_dev.o       smb_iod.o       smb_pass.o \
1292                 smb_rq.o        smb_sign.o      smb_smb.o       smb_subrs.o \
1293                 smb_time.o      smb_tran.o      smb_trantcp.o   smb_usr.o \
1294                 subr_mchain.o

1296 SMBFS_COMMON_OBJS += smbfs_ntacl.o
1297 SMBFS_OBJS +=    smbfs_vfsops.o  smbfs_vnops.o   smbfs_node.o    \
1298                 smbfs_acl.o     smbfs_client.o  smbfs_smb.o     \
1299                 smbfs_subr.o    smbfs_subr2.o   \
1300                 smbfs_rwlock.o  smbfs_xattr.o   \
1301                 $(SMBFS_COMMON_OBJS)


1304 #
1305 #                       LVM modules
1306 #
1307 MD_OBJS += md.o md_error.o md_ioctl.o md_mddb.o md_names.o \
1308         md_med.o md_rename.o md_subr.o

1310 MD_COMMON_OBJS = md_convert.o md_crc.o md_revchk.o

1312 MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

1314 SOFTPART_OBJS += sp.o sp_ioctl.o
```

```
1316 STRIPE_OBJS += stripe.o stripe_ioctl.o

1318 HOTSPARES_OBJS += hotspares.o

1320 RAID_OBJS += raid.o raid_ioctl.o raid_replay.o raid_resync.o raid_hotspare.o

1322 MIRROR_OBJS += mirror.o mirror_ioctl.o mirror_resync.o

1324 NOTIFY_OBJS += md_notify.o

1326 TRANS_OBJS += mdtrans.o trans_ioctl.o trans_log.o

1328 ZFS_COMMON_OBJS +=              \
1329         arc.o                   \
1330         bplist.o                \
1331         bpobj.o                 \
1332         bptree.o                \
1333         dbuf.o                  \
1334         ddt.o                   \
1335         ddt_zap.o               \
1336         dmu.o                   \
1337         dmu_diff.o              \
1338         dmu_send.o              \
1339         dmu_object.o            \
1340         dmu_objset.o            \
1341         dmu_traverse.o          \
1342         dmu_tx.o                \
1343         dnode.o                 \
1344         dnode_sync.o            \
1345         dsl_dir.o               \
1346         dsl_dataset.o           \
1347         dsl_deadlist.o          \
1348         dsl_pool.o              \
1349         dsl_synctask.o          \
1350         dmu_zfetch.o            \
1351         dsl_deleg.o             \
1352         dsl_prop.o              \
1353         dsl_scan.o              \
1354         zfeature.o              \
1355         gzip.o                  \
1356         lzjb.o                  \
1357         metaslab.o              \
1358         refcount.o              \
1359         sa.o                    \
1360         sha256.o                \
1361         spa.o                   \
1362         spa_config.o            \
1363         spa_errlog.o            \
1364         spa_history.o           \
1365         spa_misc.o              \
1366         space_map.o             \
1367         txg.o                   \
1368         uberblock.o             \
1369         unique.o                \
1370         vdev.o                  \
1371         vdev_cache.o            \
1372         vdev_file.o             \
1373         vdev_label.o            \
1374         vdev_mirror.o           \
1375         vdev_missing.o          \
1376         vdev_queue.o            \
1377         vdev_raidz.o            \
1378         vdev_root.o             \
1379         zap.o                   \
1380         zap_leaf.o              \
1381         zap_micro.o             \
```



```
1382         zfs_byteswap.o          \
1383         zfs_debug.o             \
1384         zfs_fm.o                \
1385         zfs_fuid.o              \
1386         zfs_sa.o                \
1387         zfs_znode.o             \
1388         zil.o                   \
1389         zio.o                   \
1390         zio_checksum.o          \
1391         zio_compress.o          \
1392         zio_inject.o            \
1393         zle.o                   \
1394         zrlock.o

1396 ZFS_SHARED_OBJS +=              \
1397         zfeature_common.o       \
1398         zfs_comutil.o           \
1399         zfs_deleg.o             \
1400         zfs_fletcher.o          \
1401         zfs_namecheck.o         \
1402         zfs_prop.o              \
1403         zpool_prop.o            \
1404         zprop_common.o

1406 ZFS_OBJS +=                     \
1407         $(ZFS_COMMON_OBJS)      \
1408         $(ZFS_SHARED_OBJS)      \
1409         vdev_disk.o             \
1410         zfs_acl.o               \
1411         zfs_ctldir.o            \
1412         zfs_dir.o               \
1413         zfs_ioctl.o             \
1414         zfs_log.o               \
1415         zfs_onexit.o            \
1416         zfs_replay.o            \
1417         zfs_rlock.o             \
1418         rrwlock.o               \
1419         zfs_vfsops.o            \
1420         zfs_vnops.o             \
1421         zvol.o

1423 ZUT_OBJS +=                     \
1424         zut.o

1426 #
1427 #                       streams modules
1428 #
1429 BUFMOD_OBJS     +=      bufmod.o

1431 CONNLD_OBJS +=  connld.o

1433 DEDUMP_OBJS +=  dedump.o

1435 DRCOMPAT_OBJS +=        drcompat.o

1437 LDLINUX_OBJS += ldlinux.o

1439 LDTERM_OBJS +=  ldterm.o uwidth.o

1441 PCKT_OBJS +=    pckt.o

1443 PFMOD_OBJS +=   pfmod.o

1445 PTEM_OBJS +=    ptem.o

1447 REDIRMOD_OBJS += strredirm.o
```

```
1449 TIMOD_OBJS +=   timod.o

1451 TIRDWR_OBJS +=  tirdwr.o

1453 TTCOMPAT_OBJS +=ttcompat.o

1455 LOG_OBJS +=     log.o

1457 PIPEMOD_OBJS += pipemod.o

1459 RPCMOD_OBJS +=  rpcmod.o        clnt_cots.o     clnt_clts.o \
1460                 clnt_gen.o      clnt_perr.o     mt_rpcinit.o    rpc_calmsg.o \
1461                 rpc_prot.o      rpc_sztypes.o   rpc_subr.o      rpcb_prot.o \
1462                 svc.o           svc_clts.o      svc_gen.o       svc_cots.o \
1463                 rpcsys.o        xdr_sizeof.o    clnt_rdma.o     svc_rdma.o \
1464                 xdr_rdma.o      rdma_subr.o     xdrrdma_sizeof.o

1466 TLIMOD_OBJS +=  tlimod.o        t_kalloc.o      t_kbind.o       t_kclose.o \
1467                 t_kconnect.o    t_kfree.o       t_kgtstate.o    t_kopen.o \
1468                 t_krcvudat.o    t_ksndudat.o    t_kspoll.o      t_kunbind.o \
1469                 t_kutil.o

1471 RLMOD_OBJS += rlmod.o

1473 TELMOD_OBJS += telmod.o

1475 CRYPTMOD_OBJS += cryptmod.o

1477 KB_OBJS +=      kbd.o           keytables.o

1479 #
1480 #                       ID mapping module
1481 #
1482 IDMAP_OBJS +=   idmap_mod.o     idmap_kapi.o    idmap_xdr.o     idmap_cache.o

1484 #
1485 #                       scheduling class modules
1486 #
1487 SDC_OBJS +=             sysdc.o

1489 RT_OBJS +=              rt.o
1490 RT_DPTBL_OBJS +=        rt_dptbl.o

1492 TS_OBJS +=              ts.o
1493 TS_DPTBL_OBJS +=        ts_dptbl.o

1495 IA_OBJS +=              ia.o

1497 FSS_OBJS +=             fss.o

1499 FX_OBJS +=              fx.o
1500 FX_DPTBL_OBJS +=        fx_dptbl.o

1502 #
1503 #                       Inter-Process Communication (IPC) modules
1504 #
1505 IPC_OBJS +=     ipc.o

1507 IPCMSG_OBJS +=  msg.o

1509 IPCSEM_OBJS +=  sem.o

1511 IPCSHM_OBJS +=  shm.o

1513 #
```



```
1514 #                       bignum module
1515 #
1516 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

1518 BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

1520 #
1521 #                       kernel cryptographic framework
1522 #
1523 KCF_OBJS +=     kcf.o kcf_callprov.o kcf_cbufcall.o kcf_cipher.o kcf_crypto.o \
1524                 kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1525                 kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1526                 kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1527                 kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1528                 kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1529                 fips_random.o

1531 CRYPTOADM_OBJS += cryptoadm.o

1533 CRYPTO_OBJS +=  crypto.o

1535 DPROV_OBJS +=   dprov.o

1537 DCA_OBJS +=     dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1538                 dca_rsa.o

1540 AESPROV_OBJS += aes.o aes_impl.o aes_modes.o

1542 ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o

1544 BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o

1546 ECCPROV_OBJS += ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1547                 ecp_384.o ecp_jac.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
1548                 ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1549                 ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1550                 mpi.o mplogic.o mpmontg.o mpprime.o oid.o \
1551                 secitem.o ec2_test.o ecp_test.o

1553 RSAPROV_OBJS += rsa.o rsa_impl.o pkcs1.o

1555 SWRANDPROV_OBJS += swrand.o

1557 #
1558 #                       kernel SSL
1559 #
1560 KSSL_OBJS +=    kssl.o kssl_ioctl.o

1562 KSSL_SOCKFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1564 #
1565 #                       misc. modules
1566 #

1568 C2AUDIT_OBJS += adr.o audit.o audit_event.o audit_io.o \
1569                 audit_path.o audit_start.o audit_syscalls.o audit_token.o \
1570                 audit_mem.o

1572 PCIC_OBJS +=    pcic.o

1574 RPCSEC_OBJS +=  secmod.o        sec_clnt.o      sec_svc.o       sec_gen.o \
1575                 auth_des.o      auth_kern.o     auth_none.o     auth_loopb.o\
1576                 authdesprt.o    authdesubr.o    authu_prot.o \
1577                 key_call.o      key_prot.o      svc_authu.o     svcauthdes.o

1579 RPCSEC_GSS_OBJS +=      rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
```

```
1580                 rpcsec_gss_utils.o svc_rpcsec_gss.o

1582 CONSCONFIG_OBJS += consconfig.o

1584 CONSCONFIG_DACF_OBJS  += consconfig_dacf.o consplat.o

1586 TEM_OBJS += tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1588 KBTRANS_OBJS +=                          \
1589                 kbtrans.o                \
1590                 kbtrans_keytables.o      \
1591                 kbtrans_polled.o         \
1592                 kbtrans_streams.o        \
1593                 usb_keytables.o

1595 KGSSD_OBJS +=   gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1596                 gss_display_name.o gss_release_name.o gss_import_name.o \
1597                 gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1599 KGSSD_DERIVED_OBJS = gssd_xdr.o

1601 KGSS_DUMMY_OBJS += dmech.o

1603 KSOCKET_OBJS += ksocket.o ksocket_mod.o

1605 CRYPTO= cksumtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1606         nfold.o verify_checksum.o prng.o block_size.o make_checksum.o\
1607         checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1609 # crypto/des
1610 CRYPTO_DES= f_cbc.o f_cksum.o f_parity.o weak_key.o d3_cbc.o ef_crypto.o

1612 CRYPTO_DK= checksum.o derive.o dk_decrypt.o dk_encrypt.o

1614 CRYPTO_ARCFOUR= k5_arcfour.o

1616 # crypto/enc_provider
1617 CRYPTO_ENC= des.o des3.o arcfour_provider.o aes_provider.o

1619 # crypto/hash_provider
1620 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_ksha1.o

1622 # crypto/keyhash_provider
1623 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

1625 # crypto/crc32
1626 CRYPTO_CRC32= crc32.o

1628 # crypto/old
1629 CRYPTO_OLD= old_decrypt.o old_encrypt.o

1631 # crypto/raw
1632 CRYPTO_RAW= raw_decrypt.o raw_encrypt.o

1634 K5_KRB= kfree.o copy_key.o \
1635         parse.o init_ctx.o \
1636         ser_adata.o ser_addr.o \
1637         ser_auth.o ser_cksum.o \
1638         ser_key.o ser_princ.o \
1639         serialize.o unparse.o \
1640         ser_actx.o

1642 K5_OS=  timeofday.o toffset.o \
1643         init_os_ctx.o c_ustime.o

1645 SEAL=
```



```
1646 # EXPORT DELETE START
1647 SEAL=   seal.o unseal.o
1648 # EXPORT DELETE END

1650 MECH=   delete_sec_context.o \
1651         import_sec_context.o \
1652         gssapi_krb5.o \
1653         k5seal.o k5unseal.o k5sealv3.o \
1654         ser_sctx.o \
1655         sign.o \
1656         util_crypt.o  \
1657         util_validate.o  util_ordering.o  \
1658         util_seqnum.o util_set.o util_seed.o \
1659         wrap_size_limit.o verify.o



1663 MECH_GEN= util_token.o


1666 KGSS_KRB5_OBJS += krb5mech.o \
1667         $(MECH) $(SEAL) $(MECH_GEN) \
1668         $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1669         $(CRYPTO_ENC) $(CRYPTO_HASH) \
1670         $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1671         $(CRYPTO_OLD) \
1672         $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1674 DES_OBJS +=     des_crypt.o des_impl.o des_ks.o des_soft.o

1676 DLBOOT_OBJS +=  bootparam_xdr.o nfs_dlinet.o scan.o

1678 KRTLD_OBJS +=   kobj_bootflags.o getoptstr.o \
1679                 kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1681 MOD_OBJS +=     modctl.o modsubr.o modsysfile.o modconf.o modhash.o

1683 STRPLUMB_OBJS += strplumb.o

1685 CPR_OBJS +=     cpr_driver.o cpr_dump.o \
1686                 cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1687                 cpr_uthread.o

1689 PROF_OBJS +=    prf.o

1691 SE_OBJS += se_driver.o

1693 SYSACCT_OBJS += acct.o

1695 ACCTCTL_OBJS += acctctl.o

1697 EXACCTSYS_OBJS += exacctsys.o

1699 KAIO_OBJS += aio.o

1701 PCMCIA_OBJS += pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o

1703 BUSRA_OBJS += busra.o

1705 PCS_OBJS += pcs.o

1707 PCAN_OBJS += pcan.o

1709 PCATA_OBJS += pcide.o pcdisk.o pclabel.o pcata.o

1711 PCSER_OBJS += pcser.o pcser_cis.o
```

```
1713 PCWL_OBJS += pcwl.o

1715 PSET_OBJS +=    pset.o

1717 OHCI_OBJS += ohci.o ohci_hub.o ohci_polled.o

1719 UHCI_OBJS += uhci.o uhciutil.o uhcitgt.o uhcihub.o uhcipolled.o

1721 EHCI_OBJS += ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o

1723 HUBD_OBJS += hubd.o

1725 USB_MID_OBJS += usb_mid.o

1727 USB_IA_OBJS += usb_ia.o

1729 UWBA_OBJS += uwba.o uwbai.o

1731 SCSA2USB_OBJS += scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o

1733 HWAHC_OBJS += hwahc.o hwahc_util.o

1735 WUSB_DF_OBJS += wusb_df.o
1736 WUSB_FWMOD_OBJS += wusb_fwmod.o

1738 IPF_OBJS += ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1739             ip_proxy.o ip_auth.o ip_pool.o ip_htable.o ip_lookup.o \
1740             ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o

1742 IBD_OBJS +=     ibd.o ibd_cm.o

1744 EIBNX_OBJS +=   enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1745                 enx_misc.o enx_q.o enx_ctl.o

1747 EOIB_OBJS +=    eib_adm.o eib_chan.o eib_cmn.o eib_ctl.o eib_data.o \
1748                 eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
1749                 eib_rsrc.o eib_svc.o eib_vnic.o

1751 DLPISTUB_OBJS += dlpistub.o

1753 SDP_OBJS +=     sdpddi.o

1755 TRILL_OBJS +=   trill.o

1757 CTF_OBJS += ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1758         ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o

1760 SMBIOS_OBJS += smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o

1762 RPCIB_OBJS += rpcib.o

1764 KMDB_OBJS += kdrv.o

1766 AFE_OBJS += afe.o

1768 BGE_OBJS += bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1769                 bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o

1771 DMFE_OBJS += dmfe_log.o dmfe_main.o dmfe_mii.o

1773 EFE_OBJS += efe.o

1775 ELXL_OBJS += elxl.o

1777 HME_OBJS += hme.o
```



```
1779 IXGB_OBJS += ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1780                 ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o

1782 NGE_OBJS += nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1783                 nge_log.o nge_rx.o nge_tx.o nge_xmii.o

1785 PCN_OBJS += pcn.o

1787 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o

1789 URTW_OBJS += urtw.o

1791 ARN_OBJS += arn_hw.o arn_eeprom.o arn_mac.o arn_calib.o arn_ani.o arn_phy.o arn_
1792                 arn_main.o arn_recv.o arn_xmit.o arn_rc.o

1794 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o

1796 ATU_OBJS += atu.o

1798 IPW_OBJS += ipw2100_hw.o ipw2100.o

1800 IWI_OBJS += ipw2200_hw.o ipw2200.o

1802 IWH_OBJS += iwh.o

1804 IWK_OBJS += iwk2.o

1806 IWP_OBJS += iwp.o

1808 MWL_OBJS += mwl.o

1810 MWLFW_OBJS += mwlfw_mode.o

1812 WPI_OBJS += wpi.o

1814 RAL_OBJS += rt2560.o ral_rate.o

1816 RUM_OBJS += rum.o

1818 RWD_OBJS += rt2661.o

1820 RWN_OBJS += rt2860.o

1822 UATH_OBJS += uath.o

1824 UATHFW_OBJS += uathfw_mod.o

1826 URAL_OBJS += ural.o

1828 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o

1830 ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o

1832 MXFE_OBJS += mxfe.o

1834 MPTSAS_OBJS += mptsas.o mptsas_impl.o mptsas_init.o mptsas_raid.o mptsas_smhba.o

1836 SFE_OBJS += sfe.o sfe_util.o

1838 BFE_OBJS += bfe.o

1840 BRIDGE_OBJS += bridge.o

1842 IDM_SHARED_OBJS += base64.o
```

```
1844 IDM_OBJS +=     $(IDM_SHARED_OBJS) \
1845                 idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o

1847 VR_OBJS += vr.o

1849 ATGE_OBJS += atge_main.o atge_l1e.o atge_mii.o atge_l1.o

1851 YGE_OBJS = yge.o

1853 #
1854 #       Build up defines and paths.
1855 #
1856 LINT_DEFS       += -Dunix

1858 #
1859 #       This duality can be removed when the native and target compilers
1860 #       are the same (or at least recognize the same command line syntax!)
1861 #       It is a bug in the current compilation system that the assember
1862 #       can't process the -Y I, flag.
1863 #
1864 NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1865 AS_INC_PATH     += $(INC_PATH) -I$(UTSBASE)/common
1866 INCLUDE_PATH    += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common

1868 PCIEB_OBJS += pcieb.o

1870 #       Chelsio N110 10G NIC driver module
1871 #
1872 CH_OBJS = ch.o glue.o pe.o sge.o

1874 CH_COM_OBJS =   ch_mac.o ch_subr.o cspi.o espi.o ixf1010.o mc3.o mc4.o mc5.o \
1875                 mv88e1xxx.o mv88x201x.o my3126.o pm3393.o tp.o ulp.o \
1876                 vsc7321.o vsc7326.o xpak.o

1878 #
1879 #       PCI strings file
1880 #
1881 PCI_STRING_OBJS = pci_strings.o

1883 NET_DACF_OBJS += net_dacf.o

1885 #
1886 #       Xframe 10G NIC driver module
1887 #
1888 XGE_OBJS = xge.o xgell.o

1890 XGE_HAL_OBJS =  xgehal-channel.o xgehal-fifo.o xgehal-ring.o  xgehal-config.o \
1891                 xgehal-driver.o  xgehal-mm.o xgehal-stats.o  xgehal-device.o \
1892                 xge-queue.o  xgehal-mgmt.o xgehal-mgmtaux.o

1894 #
1895 #       e1000g module
1896 #
1897 E1000G_OBJS +=  e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
1898                 e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
1899                 e1000_mac.o e1000_manage.o e1000_nvm.o e1000_osdep.o \
1900                 e1000_phy.o e1000g_debug.o e1000g_main.o e1000g_alloc.o \
1901                 e1000g_tx.o e1000g_rx.o e1000g_stat.o

1903 #
1904 #       Intel 82575 1G NIC driver module
1905 #
1906 IGB_OBJS =      igb_82575.o igb_api.o igb_mac.o igb_manage.o \
1907                 igb_nvm.o igb_osdep.o igb_phy.o igb_buf.o \
1908                 igb_debug.o igb_gld.o igb_log.o igb_main.o \
1909                 igb_rx.o igb_stat.o igb_tx.o
```



```
1911 #
1912 #       Intel Pro/100 NIC driver module
1913 #
1914 IPRB_OBJS =     iprb.o

1916 #
1917 #       Intel 10GbE PCIE NIC driver module
1918 #
1919 IXGBE_OBJS =    ixgbe_82598.o ixgbe_82599.o ixgbe_api.o         \
1920                 ixgbe_common.o ixgbe_phy.o                      \
1921                 ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o           \
1922                 ixgbe_log.o ixgbe_main.o                        \
1923                 ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o           \
1924                 ixgbe_tx.o  ixgbe_x540.o ixgbe_mbx.o
1924                 ixgbe_tx.o

1926 #
1927 #       NIU 10G/1G driver module
1928 #
1929 NXGE_OBJS =     nxge_mac.o nxge_ipp.o nxge_rxdma.o               \
1930                 nxge_txdma.o nxge_txc.o nxge_main.o             \
1931                 nxge_hw.o nxge_fzc.o nxge_virtual.o             \
1932                 nxge_send.o nxge_classify.o nxge_fflp.o         \
1933                 nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o       \
1934                 nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o      \
1935                 nxge_hio.o nxge_hio_guest.o nxge_intr.o

1937 NXGE_NPI_OBJS = \
1938                 npi.o npi_mac.o npi_ipp.o                       \
1939                 npi_txdma.o npi_rxdma.o npi_txc.o               \
1940                 npi_zcp.o npi_espc.o npi_fflp.o                 \
1941                 npi_vir.o

1943 NXGE_HCALL_OBJS =       \
1944                 nxge_hcall.o

1946 #
1947 #       kiconv modules
1948 #
1949 KICONV_EMEA_OBJS += kiconv_emea.o

1951 KICONV_JA_OBJS += kiconv_ja.o

1953 KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o

1955 KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o

1957 KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o

1959 #
1960 #       AAC module
1961 #
1962 AAC_OBJS = aac.o aac_ioctl.o

1964 #
1965 #       sdcard modules
1966 #
1967 SDA_OBJS =      sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
1968 SDHOST_OBJS =   sdhost.o

1970 #
1971 #       hxge 10G driver module
1972 #
1973 HXGE_OBJS =     hxge_main.o hxge_vmac.o hxge_send.o              \
1974                 hxge_txdma.o hxge_rxdma.o hxge_virtual.o        \
```

```
1975                 hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o    \
1976                 hxge_ndd.o hxge_pfc.o                           \
1977                 hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o        \
1978                 hpi_vir.o hpi_pfc.o

1980 #
1981 #       MEGARAID_SAS module
1982 #
1983 MEGA_SAS_OBJS = megaraid_sas.o

1985 #
1986 #       MR_SAS module
1987 #
1988 MR_SAS_OBJS = mr_sas.o

1990 #
1991 #       ISCSI_INITIATOR module
1992 #
1993 ISCSI_INITIATOR_OBJS =  chap.o iscsi_io.o iscsi_thread.o        \
1994                         iscsi_ioctl.o iscsid.o iscsi.o          \
1995                         iscsi_login.o isns_client.o iscsiAuthClient.o   \
1996                         iscsi_lun.o iscsiAuthClientGlue.o       \
1997                         iscsi_net.o nvfile.o iscsi_cmd.o        \
1998                         iscsi_queue.o persistent.o iscsi_conn.o \
1999                         iscsi_sess.o radius_auth.o iscsi_crc.o  \
2000                         iscsi_stats.o radius_packet.o iscsi_doorclt.o   \
2001                         iscsi_targetparam.o utils.o kifconf.o

2003 #
2004 #       ntxn 10Gb/1Gb NIC driver module
2005 #
2006 NTXN_OBJS =     unm_nic_init.o unm_gem.o unm_nic_hw.o unm_ndd.o \
2007                         unm_nic_main.o unm_nic_isr.o unm_nic_ctx.o niu.o

2009 #
2010 #       Myricom 10Gb NIC driver module
2011 #
2012 MYRI10GE_OBJS = myri10ge.o myri10ge_lro.o

2014 #       nulldriver module
2015 #
2016 NULLDRIVER_OBJS =       nulldriver.o

2018 TPM_OBJS =      tpm.o tpm_hcall.o
```

```
**********
   39876 Thu Jul 12 12:22:28 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_82598.c
XXXX Intel X540 support
**********
```

```
/******************************************************************************
  Copyright (c) 2001-2012, Intel Corporation
  Copyright (c) 2001-2010, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_82598.c,v 1.13 2012/07/05 20:51:44 jfv Exp $
/*$FreeBSD$*/

#include "ixgbe_type.h"
#include "ixgbe_82598.h"
#include "ixgbe_api.h"
#include "ixgbe_common.h"
#include "ixgbe_phy.h"

u32 ixgbe_get_pcie_msix_count_82598(struct ixgbe_hw *hw);
s32 ixgbe_init_ops_82598(struct ixgbe_hw *hw);
static s32 ixgbe_get_link_capabilities_82598(struct ixgbe_hw *hw,
                                             ixgbe_link_speed *speed,
                                             bool *autoneg);
static enum ixgbe_media_type ixgbe_get_media_type_82598(struct ixgbe_hw *hw);
s32 ixgbe_fc_enable_82598(struct ixgbe_hw *hw, s32 packetbuf_num);
static s32 ixgbe_start_mac_link_82598(struct ixgbe_hw *hw,
                                      bool autoneg_wait_to_complete);
static s32 ixgbe_check_mac_link_82598(struct ixgbe_hw *hw,
                                      ixgbe_link_speed *speed, bool *link_up,
                                      bool link_up_wait_to_complete);
static s32 ixgbe_setup_mac_link_82598(struct ixgbe_hw *hw,
                                      ixgbe_link_speed speed,
                                      bool autoneg,
                                      bool autoneg_wait_to_complete);
static s32 ixgbe_setup_copper_link_82598(struct ixgbe_hw *hw,
                                         ixgbe_link_speed speed,
                                         bool autoneg,
```



```
                                         bool autoneg_wait_to_complete);
static s32 ixgbe_reset_hw_82598(struct ixgbe_hw *hw);
s32 ixgbe_start_hw_82598(struct ixgbe_hw *hw);
void ixgbe_enable_relaxed_ordering_82598(struct ixgbe_hw *hw);
s32 ixgbe_set_vmdq_82598(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
static s32 ixgbe_clear_vmdq_82598(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
s32 ixgbe_set_vfta_82598(struct ixgbe_hw *hw, u32 vlan,
                         u32 vind, bool vlan_on);
static s32 ixgbe_clear_vfta_82598(struct ixgbe_hw *hw);
static void ixgbe_set_rxpba_82598(struct ixgbe_hw *hw, int num_pb,
                                  u32 headroom, int strategy);
s32 ixgbe_read_analog_reg8_82598(struct ixgbe_hw *hw, u32 reg, u8 *val);
s32 ixgbe_write_analog_reg8_82598(struct ixgbe_hw *hw, u32 reg, u8 val);
s32 ixgbe_read_i2c_eeprom_82598(struct ixgbe_hw *hw, u8 byte_offset,
                                u8 *eeprom_data);
u32 ixgbe_get_supported_physical_layer_82598(struct ixgbe_hw *hw);
s32 ixgbe_init_phy_ops_82598(struct ixgbe_hw *hw);
void ixgbe_set_lan_id_multi_port_pcie_82598(struct ixgbe_hw *hw);
void ixgbe_set_pcie_completion_timeout(struct ixgbe_hw *hw);

/**
 *  ixgbe_set_pcie_completion_timeout - set pci-e completion timeout
 *  @hw: pointer to the HW structure
 *
 *  The defaults for 82598 should be in the range of 50us to 50ms,
 *  however the hardware default for these parts is 500us to 1ms which is less
 *  than the 10ms recommended by the pci-e spec.  To address this we need to
 *  increase the value to either 10ms to 250ms for capability version 1 config,
 *  or 16ms to 55ms for version 2.
 **/
void ixgbe_set_pcie_completion_timeout(struct ixgbe_hw *hw)
{
        u32 gcr = IXGBE_READ_REG(hw, IXGBE_GCR);
        u16 pcie_devctl2;

        /* only take action if timeout value is defaulted to 0 */
        if (gcr & IXGBE_GCR_CMPL_TMOUT_MASK)
                goto out;

        /*
         * if capababilities version is type 1 we can write the
         * timeout of 10ms to 250ms through the GCR register
         */
        if (!(gcr & IXGBE_GCR_CAP_VER2)) {
                gcr |= IXGBE_GCR_CMPL_TMOUT_10ms;
                goto out;
        }

        /*
         * for version 2 capabilities we need to write the config space
         * directly in order to set the completion timeout value for
         * 16ms to 55ms
         */
        pcie_devctl2 = IXGBE_READ_PCIE_WORD(hw, IXGBE_PCI_DEVICE_CONTROL2);
        pcie_devctl2 |= IXGBE_PCI_DEVICE_CONTROL2_16ms;
        IXGBE_WRITE_PCIE_WORD(hw, IXGBE_PCI_DEVICE_CONTROL2, pcie_devctl2);
out:
        /* disable completion timeout resend */
        gcr &= ~IXGBE_GCR_CMPL_TMOUT_RESEND;
        IXGBE_WRITE_REG(hw, IXGBE_GCR, gcr);
}

/**
 *  ixgbe_get_pcie_msix_count_82598 - Gets MSI-X vector count
 *  @hw: pointer to hardware structure
 *
```

```
 *  Read PCIe configuration space, and get the MSI-X vector count from
 *  the capabilities table.
 **/
u32 ixgbe_get_pcie_msix_count_82598(struct ixgbe_hw *hw)
{
        u32 msix_count = 18;

        DEBUGFUNC("ixgbe_get_pcie_msix_count_82598");

        if (hw->mac.msix_vectors_from_pcie) {
                msix_count = IXGBE_READ_PCIE_WORD(hw,
                                                  IXGBE_PCIE_MSIX_82598_CAPS);
                msix_count &= IXGBE_PCIE_MSIX_TBL_SZ_MASK;

                /* MSI-X count is zero-based in HW, so increment to give
                 * proper value */
                msix_count++;
        }
        return msix_count;
}

/**
 *  ixgbe_init_ops_82598 - Inits func ptrs and MAC type
 *  @hw: pointer to hardware structure
 *
 *  Initialize the function pointers and assign the MAC type for 82598.
 *  Does not touch the hardware.
 **/
s32 ixgbe_init_ops_82598(struct ixgbe_hw *hw)
{
        struct ixgbe_mac_info *mac = &hw->mac;
        struct ixgbe_phy_info *phy = &hw->phy;
        s32 ret_val;

        DEBUGFUNC("ixgbe_init_ops_82598");

        ret_val = ixgbe_init_phy_ops_generic(hw);
        ret_val = ixgbe_init_ops_generic(hw);

        /* PHY */
        phy->ops.init = &ixgbe_init_phy_ops_82598;

        /* MAC */
        mac->ops.start_hw = &ixgbe_start_hw_82598;
        mac->ops.enable_relaxed_ordering = &ixgbe_enable_relaxed_ordering_82598;
        mac->ops.reset_hw = &ixgbe_reset_hw_82598;
        mac->ops.get_media_type = &ixgbe_get_media_type_82598;
        mac->ops.get_supported_physical_layer =
                                &ixgbe_get_supported_physical_layer_82598;
        mac->ops.read_analog_reg8 = &ixgbe_read_analog_reg8_82598;
        mac->ops.write_analog_reg8 = &ixgbe_write_analog_reg8_82598;
        mac->ops.set_lan_id = &ixgbe_set_lan_id_multi_port_pcie_82598;

        /* RAR, Multicast, VLAN */
        mac->ops.set_vmdq = &ixgbe_set_vmdq_82598;
        mac->ops.clear_vmdq = &ixgbe_clear_vmdq_82598;
        mac->ops.set_vfta = &ixgbe_set_vfta_82598;
        mac->ops.set_vlvf = NULL;
        mac->ops.clear_vfta = &ixgbe_clear_vfta_82598;

        /* Flow Control */
        mac->ops.fc_enable = &ixgbe_fc_enable_82598;

        mac->mcft_size          = 128;
        mac->vft_size           = 128;
        mac->num_rar_entries    = 16;
```



```
        mac->rx_pb_size         = 512;
        mac->max_tx_queues      = 32;
        mac->max_rx_queues      = 64;
        mac->max_msix_vectors   = ixgbe_get_pcie_msix_count_generic(hw);
        mac->max_msix_vectors = ixgbe_get_pcie_msix_count_82598(hw);

        /* SFP+ Module */
        phy->ops.read_i2c_eeprom = &ixgbe_read_i2c_eeprom_82598;

        /* Link */
        mac->ops.check_link = &ixgbe_check_mac_link_82598;
        mac->ops.setup_link = &ixgbe_setup_mac_link_82598;
        mac->ops.flap_tx_laser = NULL;
        mac->ops.get_link_capabilities = &ixgbe_get_link_capabilities_82598;
        mac->ops.setup_rxpba = &ixgbe_set_rxpba_82598;
        mac->ops.get_link_capabilities =
                                &ixgbe_get_link_capabilities_82598;

        /* Manageability interface */
        mac->ops.set_fw_drv_ver = NULL;

        return ret_val;
}

/**
 *  ixgbe_init_phy_ops_82598 - PHY/SFP specific init
 *  @hw: pointer to hardware structure
 *
 *  Initialize any function pointers that were not able to be
 *  set during init_shared_code because the PHY/SFP type was
 *  not known.  Perform the SFP init if necessary.
 *
 **/
s32 ixgbe_init_phy_ops_82598(struct ixgbe_hw *hw)
{
        struct ixgbe_mac_info *mac = &hw->mac;
        struct ixgbe_phy_info *phy = &hw->phy;
        s32 ret_val = IXGBE_SUCCESS;
        u16 list_offset, data_offset;

        DEBUGFUNC("ixgbe_init_phy_ops_82598");

        /* Identify the PHY */
        phy->ops.identify(hw);

        /* Overwrite the link function pointers if copper PHY */
        if (mac->ops.get_media_type(hw) == ixgbe_media_type_copper) {
                mac->ops.setup_link = &ixgbe_setup_copper_link_82598;
                mac->ops.get_link_capabilities =
                                  &ixgbe_get_copper_link_capabilities_generic;
        }

        switch (hw->phy.type) {
        case ixgbe_phy_tn:
                phy->ops.setup_link = &ixgbe_setup_phy_link_tnx;
                phy->ops.check_link = &ixgbe_check_phy_link_tnx;
                phy->ops.get_firmware_version =
                                        &ixgbe_get_phy_firmware_version_tnx;
                break;
        case ixgbe_phy_aq:
                phy->ops.get_firmware_version =
                                &ixgbe_get_phy_firmware_version_generic;
                break;
        case ixgbe_phy_nl:
                phy->ops.reset = &ixgbe_reset_phy_nl;
```

```
210                 /* Call SFP+ identify routine to get the SFP+ module type */
211                 ret_val = phy->ops.identify_sfp(hw);
212                 if (ret_val != IXGBE_SUCCESS)
213                         goto out;
214                 else if (hw->phy.sfp_type == ixgbe_sfp_type_unknown) {
215                         ret_val = IXGBE_ERR_SFP_NOT_SUPPORTED;
216                         goto out;
217                 }
219                 /* Check to see if SFP+ module is supported */
220                 ret_val = ixgbe_get_sfp_init_sequence_offsets(hw,
221                                                               &list_offset,
222                                                               &data_offset);
223                 if (ret_val != IXGBE_SUCCESS) {
224                         ret_val = IXGBE_ERR_SFP_NOT_SUPPORTED;
225                         goto out;
226                 }
227                 break;
228         default:
229                 break;
230         }
232 out:
233         return ret_val;
234 }
236 /**
237  *  ixgbe_start_hw_82598 - Prepare hardware for Tx/Rx
238  *  @hw: pointer to hardware structure
239  *
240  *  Starts the hardware using the generic start_hw function.
241  *  Disables relaxed ordering Then set pcie completion timeout
242  *
243  **/
244 s32 ixgbe_start_hw_82598(struct ixgbe_hw *hw)
245 {
246         u32 regval;
247         u32 i;
248         s32 ret_val = IXGBE_SUCCESS;
250         DEBUGFUNC("ixgbe_start_hw_82598");
252         ret_val = ixgbe_start_hw_generic(hw);
254         /* Disable relaxed ordering */
255         for (i = 0; ((i < hw->mac.max_tx_queues) &&
256              (i < IXGBE_DCA_MAX_QUEUES_82598)); i++) {
257                 regval = IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL(i));
258                 regval &= ~IXGBE_DCA_TXCTRL_DESC_WRO_EN;
296                 regval &= ~IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
259                 IXGBE_WRITE_REG(hw, IXGBE_DCA_TXCTRL(i), regval);
260         }
262         for (i = 0; ((i < hw->mac.max_rx_queues) &&
263              (i < IXGBE_DCA_MAX_QUEUES_82598)); i++) {
264                 regval = IXGBE_READ_REG(hw, IXGBE_DCA_RXCTRL(i));
265                 regval &= ~(IXGBE_DCA_RXCTRL_DATA_WRO_EN |
266                             IXGBE_DCA_RXCTRL_HEAD_WRO_EN);
304                             IXGBE_DCA_RXCTRL_DESC_HSRO_EN);
303                 regval &= ~(IXGBE_DCA_RXCTRL_DESC_WRO_EN |
267                 IXGBE_WRITE_REG(hw, IXGBE_DCA_RXCTRL(i), regval);
268         }
270         /* set the completion timeout for interface */
271         if (ret_val == IXGBE_SUCCESS)
272                 ixgbe_set_pcie_completion_timeout(hw);
```



```
274         return ret_val;
275 }
_____unchanged_portion_omitted_

338 /**
339  *  ixgbe_get_media_type_82598 - Determines media type
340  *  @hw: pointer to hardware structure
341  *
342  *  Returns the media type (fiber, copper, backplane)
343  **/
344 static enum ixgbe_media_type ixgbe_get_media_type_82598(struct ixgbe_hw *hw)
345 {
346         enum ixgbe_media_type media_type;
348         DEBUGFUNC("ixgbe_get_media_type_82598");
350         /* Detect if there is a copper PHY attached. */
351         switch (hw->phy.type) {
352         case ixgbe_phy_cu_unknown:
353         case ixgbe_phy_tn:
392         case ixgbe_phy_aq:
354                 media_type = ixgbe_media_type_copper;
355                 goto out;
356         default:
357                 break;
358         }
360         /* Media type for I82598 is based on device ID */
361         switch (hw->device_id) {
362         case IXGBE_DEV_ID_82598:
363         case IXGBE_DEV_ID_82598_BX:
364                 /* Default device ID is mezzanine card KX/KX4 */
365                 media_type = ixgbe_media_type_backplane;
366                 break;
367         case IXGBE_DEV_ID_82598AF_DUAL_PORT:
368         case IXGBE_DEV_ID_82598AF_SINGLE_PORT:
369         case IXGBE_DEV_ID_82598_DA_DUAL_PORT:
370         case IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM:
371         case IXGBE_DEV_ID_82598EB_XF_LR:
372         case IXGBE_DEV_ID_82598EB_SFP_LOM:
373                 media_type = ixgbe_media_type_fiber;
374                 break;
375         case IXGBE_DEV_ID_82598EB_CX4:
376         case IXGBE_DEV_ID_82598_CX4_DUAL_PORT:
377                 media_type = ixgbe_media_type_cx4;
378                 break;
379         case IXGBE_DEV_ID_82598AT:
380         case IXGBE_DEV_ID_82598AT2:
381                 media_type = ixgbe_media_type_copper;
382                 break;
383         default:
384                 media_type = ixgbe_media_type_unknown;
385                 break;
386         }
387 out:
388         return media_type;
389 }
391 /**
392  *  ixgbe_fc_enable_82598 - Enable flow control
393  *  @hw: pointer to hardware structure
433  *  @packetbuf_num: packet buffer number (0-7)
394  *
395  *  Enable flow control according to the current settings.
396  **/
```

```
s32 ixgbe_fc_enable_82598(struct ixgbe_hw *hw)
s32 ixgbe_fc_enable_82598(struct ixgbe_hw *hw, s32 packetbuf_num)
{
        s32 ret_val = IXGBE_SUCCESS;
        u32 fctrl_reg;
        u32 rmcs_reg;
        u32 reg;
        u32 fcrtl, fcrth;
        u32 rx_pba_size;
        u32 link_speed = 0;
        int i;
        bool link_up;

        DEBUGFUNC("ixgbe_fc_enable_82598");

        /* Validate the water mark configuration */
        if (!hw->fc.pause_time) {
                ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
                goto out;
        }

        /* Low water mark of zero causes XOFF floods */
        for (i = 0; i < IXGBE_DCB_MAX_TRAFFIC_CLASS; i++) {
                if ((hw->fc.current_mode & ixgbe_fc_tx_pause) &&
                    hw->fc.high_water[i]) {
                        if (!hw->fc.low_water[i] ||
                            hw->fc.low_water[i] >= hw->fc.high_water[i]) {
                                DEBUGOUT("Invalid water mark configuration\n");
                                ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
                                goto out;
                        }
                }
        }

        /*
         * On 82598 having Rx FC on causes resets while doing 1G
         * so if it's on turn it off once we know link_speed. For
         * more details see 82598 Specification update.
         */
        hw->mac.ops.check_link(hw, &link_speed, &link_up, FALSE);
        if (link_up && link_speed == IXGBE_LINK_SPEED_1GB_FULL) {
                switch (hw->fc.requested_mode) {
                case ixgbe_fc_full:
                        hw->fc.requested_mode = ixgbe_fc_tx_pause;
                        break;
                case ixgbe_fc_rx_pause:
                        hw->fc.requested_mode = ixgbe_fc_none;
                        break;
                default:
                        /* no change */
                        break;
                }
        }

        /* Negotiate the fc mode to use */
        ixgbe_fc_autoneg(hw);
        ret_val = ixgbe_fc_autoneg(hw);
        if (ret_val == IXGBE_ERR_FLOW_CONTROL)
                goto out;

        /* Disable any previous flow control settings */
        fctrl_reg = IXGBE_READ_REG(hw, IXGBE_FCTRL);
        fctrl_reg &= ~(IXGBE_FCTRL_RFCE | IXGBE_FCTRL_RPFCE);

        rmcs_reg = IXGBE_READ_REG(hw, IXGBE_RMCS);
        rmcs_reg &= ~(IXGBE_RMCS_TFCE_PRIORITY | IXGBE_RMCS_TFCE_802_3X);
```



```
        /*
         * The possible values of fc.current_mode are:
         * 0: Flow control is completely disabled
         * 1: Rx flow control is enabled (we can receive pause frames,
         *    but not send pause frames).
         * 2: Tx flow control is enabled (we can send pause frames but
         *     we do not support receiving pause frames).
         * 3: Both Rx and Tx flow control (symmetric) are enabled.
         * other: Invalid.
         */
        switch (hw->fc.current_mode) {
        case ixgbe_fc_none:
                /*
                 * Flow control is disabled by software override or autoneg.
                 * The code below will actually disable it in the HW.
                 */
                break;
        case ixgbe_fc_rx_pause:
                /*
                 * Rx Flow control is enabled and Tx Flow control is
                 * disabled by software override. Since there really
                 * isn't a way to advertise that we are capable of RX
                 * Pause ONLY, we will advertise that we support both
                 * symmetric and asymmetric Rx PAUSE.  Later, we will
                 * disable the adapter's ability to send PAUSE frames.
                 */
                fctrl_reg |= IXGBE_FCTRL_RFCE;
                break;
        case ixgbe_fc_tx_pause:
                /*
                 * Tx Flow control is enabled, and Rx Flow control is
                 * disabled by software override.
                 */
                rmcs_reg |= IXGBE_RMCS_TFCE_802_3X;
                break;
        case ixgbe_fc_full:
                /* Flow control (both Rx and Tx) is enabled by SW override. */
                fctrl_reg |= IXGBE_FCTRL_RFCE;
                rmcs_reg |= IXGBE_RMCS_TFCE_802_3X;
                break;
        default:
                DEBUGOUT("Flow control param set incorrectly\n");
                ret_val = IXGBE_ERR_CONFIG;
                goto out;
        }

        /* Set 802.3x based flow control settings. */
        fctrl_reg |= IXGBE_FCTRL_DPF;
        IXGBE_WRITE_REG(hw, IXGBE_FCTRL, fctrl_reg);
        IXGBE_WRITE_REG(hw, IXGBE_RMCS, rmcs_reg);

        /* Set up and enable Rx high/low water mark thresholds, enable XON. */
        for (i = 0; i < IXGBE_DCB_MAX_TRAFFIC_CLASS; i++) {
                if ((hw->fc.current_mode & ixgbe_fc_tx_pause) &&
                    hw->fc.high_water[i]) {
                        fcrtl = (hw->fc.low_water[i] << 10) | IXGBE_FCRTL_XONE;
                        fcrth = (hw->fc.high_water[i] << 10) | IXGBE_FCRTH_FCEN;
                        IXGBE_WRITE_REG(hw, IXGBE_FCRTL(i), fcrtl);
                        IXGBE_WRITE_REG(hw, IXGBE_FCRTH(i), fcrth);
                } else {
                        IXGBE_WRITE_REG(hw, IXGBE_FCRTL(i), 0);
                        IXGBE_WRITE_REG(hw, IXGBE_FCRTH(i), 0);
                }
        if (hw->fc.current_mode & ixgbe_fc_tx_pause) {
                rx_pba_size = IXGBE_READ_REG(hw, IXGBE_RXPBSIZE(packetbuf_num));
```

```
535                 rx_pba_size >>= IXGBE_RXPBSIZE_SHIFT;

537                 reg = (rx_pba_size - hw->fc.low_water) << 6;
538                 if (hw->fc.send_xon)
539                         reg |= IXGBE_FCRTL_XONE;

541                 IXGBE_WRITE_REG(hw, IXGBE_FCRTL(packetbuf_num), reg);

543                 reg = (rx_pba_size - hw->fc.high_water) << 6;
544                 reg |= IXGBE_FCRTH_FCEN;

546                 IXGBE_WRITE_REG(hw, IXGBE_FCRTH(packetbuf_num), reg);
523         }

525         /* Configure pause time (2 TCs per register) */
526         reg = hw->fc.pause_time * 0x00010001;
527         for (i = 0; i < (IXGBE_DCB_MAX_TRAFFIC_CLASS / 2); i++)
528                 IXGBE_WRITE_REG(hw, IXGBE_FCTTV(i), reg);
550         reg = IXGBE_READ_REG(hw, IXGBE_FCTTV(packetbuf_num / 2));
551         if ((packetbuf_num & 1) == 0)
552                 reg = (reg & 0xFFFF0000) | hw->fc.pause_time;
553         else
554                 reg = (reg & 0x0000FFFF) | (hw->fc.pause_time << 16);
555         IXGBE_WRITE_REG(hw, IXGBE_FCTTV(packetbuf_num / 2), reg);

530         /* Configure flow control refresh threshold value */
531         IXGBE_WRITE_REG(hw, IXGBE_FCRTV, hw->fc.pause_time / 2);
557         IXGBE_WRITE_REG(hw, IXGBE_FCRTV, (hw->fc.pause_time >> 1));

533 out:
534         return ret_val;
535 }
```

***unchanged_portion_omitted_***

```
620 /**
621  *  ixgbe_check_mac_link_82598 - Get link/speed status
622  *  @hw: pointer to hardware structure
623  *  @speed: pointer to link speed
624  *  @link_up: TRUE is link is up, FALSE otherwise
625  *  @link_up_wait_to_complete: bool used to wait for link up or not
626  *
627  *  Reads the links register to determine if link is up and the current speed
628  **/
629 static s32 ixgbe_check_mac_link_82598(struct ixgbe_hw *hw,
630                                       ixgbe_link_speed *speed, bool *link_up,
631                                       bool link_up_wait_to_complete)
632 {
633         u32 links_reg;
634         u32 i;
635         u16 link_reg, adapt_comp_reg;

637         DEBUGFUNC("ixgbe_check_mac_link_82598");

639         /*
640          * SERDES PHY requires us to read link status from undocumented
641          * register 0xC79F.  Bit 0 set indicates link is up/ready; clear
642          * indicates link down.  OxC00C is read to check that the XAUI lanes
643          * are active.  Bit 0 clear indicates active; set indicates inactive.
644          */
645         if (hw->phy.type == ixgbe_phy_nl) {
646                 hw->phy.ops.read_reg(hw, 0xC79F, IXGBE_TWINAX_DEV, &link_reg);
647                 hw->phy.ops.read_reg(hw, 0xC79F, IXGBE_TWINAX_DEV, &link_reg);
648                 hw->phy.ops.read_reg(hw, 0xC00C, IXGBE_TWINAX_DEV,
649                                      &adapt_comp_reg);
650                 if (link_up_wait_to_complete) {
651                         for (i = 0; i < IXGBE_LINK_UP_TIME; i++) {
```



```
652                                 if ((link_reg & 1) &&
653                                     ((adapt_comp_reg & 1) == 0)) {
654                                         *link_up = TRUE;
655                                         break;
656                                 } else {
657                                         *link_up = FALSE;
658                                 }
659                                 msec_delay(100);
660                                 hw->phy.ops.read_reg(hw, 0xC79F,
661                                                      IXGBE_TWINAX_DEV,
662                                                      &link_reg);
663                                 hw->phy.ops.read_reg(hw, 0xC00C,
664                                                      IXGBE_TWINAX_DEV,
665                                                      &adapt_comp_reg);
666                         }
667                 } else {
668                         if ((link_reg & 1) && ((adapt_comp_reg & 1) == 0))
669                                 *link_up = TRUE;
670                         else
671                                 *link_up = FALSE;
672                 }

674                 if (*link_up == FALSE)
675                         goto out;
676         }

678         links_reg = IXGBE_READ_REG(hw, IXGBE_LINKS);
679         if (link_up_wait_to_complete) {
680                 for (i = 0; i < IXGBE_LINK_UP_TIME; i++) {
681                         if (links_reg & IXGBE_LINKS_UP) {
682                                 *link_up = TRUE;
683                                 break;
684                         } else {
685                                 *link_up = FALSE;
686                         }
687                         msec_delay(100);
688                         links_reg = IXGBE_READ_REG(hw, IXGBE_LINKS);
689                 }
690         } else {
691                 if (links_reg & IXGBE_LINKS_UP)
692                         *link_up = TRUE;
693                 else
694                         *link_up = FALSE;
695         }

697         if (links_reg & IXGBE_LINKS_SPEED)
698                 *speed = IXGBE_LINK_SPEED_10GB_FULL;
699         else
700                 *speed = IXGBE_LINK_SPEED_1GB_FULL;

702         if ((hw->device_id == IXGBE_DEV_ID_82598AT2) && (*link_up == TRUE) &&
703             (ixgbe_validate_link_ready(hw) != IXGBE_SUCCESS))
704                 *link_up = FALSE;

732         /* if link is down, zero out the current_mode */
733         if (*link_up == FALSE) {
734                 hw->fc.current_mode = ixgbe_fc_none;
735                 hw->fc.fc_was_autonegged = FALSE;
736         }
706 out:
707         return IXGBE_SUCCESS;
708 }

710 /**
711  *  ixgbe_setup_mac_link_82598 - Set MAC link speed
712  *  @hw: pointer to hardware structure
```

```
 713  *  @speed: new link speed
 714  *  @autoneg: TRUE if autonegotiation enabled
 715  *  @autoneg_wait_to_complete: TRUE when waiting for completion is needed
 716  *
 717  *  Set the link speed in the AUTOC register and restarts link.
 718  **/
 719 static s32 ixgbe_setup_mac_link_82598(struct ixgbe_hw *hw,
 720                                       ixgbe_link_speed speed, bool autoneg,
 721                                       bool autoneg_wait_to_complete)
 722 {
 723         s32 status = IXGBE_SUCCESS;
 724         ixgbe_link_speed link_capabilities = IXGBE_LINK_SPEED_UNKNOWN;
 725         u32 curr_autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
 726         u32 autoc = curr_autoc;
 727         u32 link_mode = autoc & IXGBE_AUTOC_LMS_MASK;

 729         DEBUGFUNC("ixgbe_setup_mac_link_82598");

 731         /* Check to see if speed passed in is supported. */
 732         ixgbe_get_link_capabilities(hw, &link_capabilities, &autoneg);
 763         (void) ixgbe_get_link_capabilities(hw, &link_capabilities, &autoneg);
 733         speed &= link_capabilities;

 735         if (speed == IXGBE_LINK_SPEED_UNKNOWN)
 736                 status = IXGBE_ERR_LINK_SETUP;

 738         /* Set KX4/KX support according to speed requested */
 739         else if (link_mode == IXGBE_AUTOC_LMS_KX4_AN ||
 740                  link_mode == IXGBE_AUTOC_LMS_KX4_AN_1G_AN) {
 741                 autoc &= ~IXGBE_AUTOC_KX4_KX_SUPP_MASK;
 742                 if (speed & IXGBE_LINK_SPEED_10GB_FULL)
 743                         autoc |= IXGBE_AUTOC_KX4_SUPP;
 744                 if (speed & IXGBE_LINK_SPEED_1GB_FULL)
 745                         autoc |= IXGBE_AUTOC_KX_SUPP;
 746                 if (autoc != curr_autoc)
 747                         IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc);
 748         }

 750         if (status == IXGBE_SUCCESS) {
 751                 /*
 752                  * Setup and restart the link based on the new values in
 753                  * ixgbe_hw This will write the AUTOC register based on the new
 754                  * stored values
 755                  */
 756                 status = ixgbe_start_mac_link_82598(hw,
 757                                                     autoneg_wait_to_complete);
 758         }

 760         return status;
 761 }


 764 /**
 765  *  ixgbe_setup_copper_link_82598 - Set the PHY autoneg advertised field
 766  *  @hw: pointer to hardware structure
 767  *  @speed: new link speed
 768  *  @autoneg: TRUE if autonegotiation enabled
 769  *  @autoneg_wait_to_complete: TRUE if waiting is needed to complete
 770  *
 771  *  Sets the link speed in the AUTOC register in the MAC and restarts link.
 772  **/
 773 static s32 ixgbe_setup_copper_link_82598(struct ixgbe_hw *hw,
 774                                          ixgbe_link_speed speed,
 775                                          bool autoneg,
 776                                          bool autoneg_wait_to_complete)
 777 {
```



```
 778         s32 status;

 780         DEBUGFUNC("ixgbe_setup_copper_link_82598");

 782         /* Setup the PHY according to input speed */
 783         status = hw->phy.ops.setup_link_speed(hw, speed, autoneg,
 784                                               autoneg_wait_to_complete);
 785         /* Set up MAC */
 786         ixgbe_start_mac_link_82598(hw, autoneg_wait_to_complete);
 817         (void) ixgbe_start_mac_link_82598(hw, autoneg_wait_to_complete);

 788         return status;
 789 }

 791 /**
 792  *  ixgbe_reset_hw_82598 - Performs hardware reset
 793  *  @hw: pointer to hardware structure
 794  *
 795  *  Resets the hardware by resetting the transmit and receive units, masks and
 796  *  clears all interrupts, performing a PHY reset, and performing a link (MAC)
 797  *  reset.
 798  **/
 799 static s32 ixgbe_reset_hw_82598(struct ixgbe_hw *hw)
 800 {
 801         s32 status = IXGBE_SUCCESS;
 802         s32 phy_status = IXGBE_SUCCESS;
 803         u32 ctrl;
 804         u32 gheccr;
 805         u32 i;
 806         u32 autoc;
 807         u8  analog_val;

 809         DEBUGFUNC("ixgbe_reset_hw_82598");

 811         /* Call adapter stop to disable tx/rx and clear interrupts */
 812         status = hw->mac.ops.stop_adapter(hw);
 813         if (status != IXGBE_SUCCESS)
 814                 goto reset_hw_out;
 843         hw->mac.ops.stop_adapter(hw);

 816         /*
 817          * Power up the Atlas Tx lanes if they are currently powered down.
 818          * Atlas Tx lanes are powered down for MAC loopback tests, but
 819          * they are not automatically restored on reset.
 820          */
 821         hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_LPBK, &analog_val);
 822         if (analog_val & IXGBE_ATLAS_PDN_TX_REG_EN) {
 823                 /* Enable Tx Atlas so packets can be transmitted again */
 824                 hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_LPBK,
 825                                              &analog_val);
 826                 analog_val &= ~IXGBE_ATLAS_PDN_TX_REG_EN;
 827                 hw->mac.ops.write_analog_reg8(hw, IXGBE_ATLAS_PDN_LPBK,
 828                                               analog_val);

 830                 hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_10G,
 831                                              &analog_val);
 832                 analog_val &= ~IXGBE_ATLAS_PDN_TX_10G_QL_ALL;
 833                 hw->mac.ops.write_analog_reg8(hw, IXGBE_ATLAS_PDN_10G,
 834                                               analog_val);

 836                 hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_1G,
 837                                              &analog_val);
 838                 analog_val &= ~IXGBE_ATLAS_PDN_TX_1G_QL_ALL;
 839                 hw->mac.ops.write_analog_reg8(hw, IXGBE_ATLAS_PDN_1G,
 840                                               analog_val);
```

```
842                 hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_AN,
843                                              &analog_val);
844                 analog_val &= ~IXGBE_ATLAS_PDN_TX_AN_QL_ALL;
845                 hw->mac.ops.write_analog_reg8(hw, IXGBE_ATLAS_PDN_AN,
846                                               analog_val);
847         }

849         /* Reset PHY */
850         if (hw->phy.reset_disable == FALSE) {
851                 /* PHY ops must be identified and initialized prior to reset */

853                 /* Init PHY and function pointers, perform SFP setup */
854                 phy_status = hw->phy.ops.init(hw);
855                 if (phy_status == IXGBE_ERR_SFP_NOT_SUPPORTED)
856                         goto reset_hw_out;
857                 if (phy_status == IXGBE_ERR_SFP_NOT_PRESENT)
858                         goto mac_reset_top;
886                 else if (phy_status == IXGBE_ERR_SFP_NOT_PRESENT)
887                         goto no_phy_reset;

860                 hw->phy.ops.reset(hw);
861         }

892 no_phy_reset:
893         /*
894          * Prevent the PCI-E bus from from hanging by disabling PCI-E master
895          * access and verify no pending requests before reset
896          */
897         (void) ixgbe_disable_pcie_master(hw);

863 mac_reset_top:
864         /*
865          * Issue global reset to the MAC.  This needs to be a SW reset.
866          * If link reset is used, it might reset the MAC when mng is using it
867          */
868         ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL) | IXGBE_CTRL_RST;
869         IXGBE_WRITE_REG(hw, IXGBE_CTRL, ctrl);
904         ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
905         IXGBE_WRITE_REG(hw, IXGBE_CTRL, (ctrl | IXGBE_CTRL_RST));
870         IXGBE_WRITE_FLUSH(hw);

872         /* Poll for reset bit to self-clear indicating reset is complete */
873         for (i = 0; i < 10; i++) {
874                 usec_delay(1);
875                 ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
876                 if (!(ctrl & IXGBE_CTRL_RST))
877                         break;
878         }
879         if (ctrl & IXGBE_CTRL_RST) {
880                 status = IXGBE_ERR_RESET_FAILED;
881                 DEBUGOUT("Reset polling failed to complete.\n");
882         }

884         msec_delay(50);

886         /*
887          * Double resets are required for recovery from certain error
888          * conditions.  Between resets, it is necessary to stall to allow time
889          * for any pending HW events to complete.
923          * for any pending HW events to complete.  We use 1usec since that is
924          * what is needed for ixgbe_disable_pcie_master().  The second reset
925          * then clears out any effects of those events.
890          */
891         if (hw->mac.flags & IXGBE_FLAGS_DOUBLE_RESET_REQUIRED) {
892                 hw->mac.flags &= ~IXGBE_FLAGS_DOUBLE_RESET_REQUIRED;
929                 usec_delay(1);
```



```
893                 goto mac_reset_top;
894         }

933         msec_delay(50);

896         gheccr = IXGBE_READ_REG(hw, IXGBE_GHECCR);
897         gheccr &= ~((1 << 21) | (1 << 18) | (1 << 9) | (1 << 6));
898         IXGBE_WRITE_REG(hw, IXGBE_GHECCR, gheccr);

900         /*
901          * Store the original AUTOC value if it has not been
902          * stored off yet.  Otherwise restore the stored original
903          * AUTOC value since the reset operation sets back to deaults.
904          */
905         autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
906         if (hw->mac.orig_link_settings_stored == FALSE) {
907                 hw->mac.orig_autoc = autoc;
908                 hw->mac.orig_link_settings_stored = TRUE;
909         } else if (autoc != hw->mac.orig_autoc) {
910                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, hw->mac.orig_autoc);
911         }

913         /* Store the permanent mac address */
914         hw->mac.ops.get_mac_addr(hw, hw->mac.perm_addr);

916         /*
917          * Store MAC address from RAR0, clear receive address registers, and
918          * clear the multicast table
919          */
920         hw->mac.ops.init_rx_addrs(hw);

922 reset_hw_out:
923         if (phy_status != IXGBE_SUCCESS)
924                 status = phy_status;

926         return status;
927 }
_____unchanged_portion_omitted_

955 /**
956  *  ixgbe_clear_vmdq_82598 - Disassociate a VMDq set index from an rx address
957  *  @hw: pointer to hardware struct
958  *  @rar: receive address register index to associate with a VMDq index
959  *  @vmdq: VMDq clear index (not used in 82598, but elsewhere)
960  **/
961 static s32 ixgbe_clear_vmdq_82598(struct ixgbe_hw *hw, u32 rar, u32 vmdq)
962 {
963         u32 rar_high;
964         u32 rar_entries = hw->mac.num_rar_entries;

966         UNREFERENCED_1PARAMETER(vmdq);
1005        UNREFERENCED_PARAMETER(vmdq);

968         /* Make sure we are using a valid rar index range */
969         if (rar >= rar_entries) {
970                 DEBUGOUT1("RAR index %d is out of range.\n", rar);
971                 return IXGBE_ERR_INVALID_ARGUMENT;
972         }

974         rar_high = IXGBE_READ_REG(hw, IXGBE_RAH(rar));
975         if (rar_high & IXGBE_RAH_VIND_MASK) {
976                 rar_high &= ~IXGBE_RAH_VIND_MASK;
977                 IXGBE_WRITE_REG(hw, IXGBE_RAH(rar), rar_high);
978         }

980         return IXGBE_SUCCESS;
```

```
 981 }
______unchanged_portion_omitted_

1167 /**
1168  *  ixgbe_get_supported_physical_layer_82598 - Returns physical layer type
1169  *  @hw: pointer to hardware structure
1170  *
1171  *  Determines physical layer capabilities of the current configuration.
1172  **/
1173 u32 ixgbe_get_supported_physical_layer_82598(struct ixgbe_hw *hw)
1174 {
1175         u32 physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
1176         u32 autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
1177         u32 pma_pmd_10g = autoc & IXGBE_AUTOC_10G_PMA_PMD_MASK;
1178         u32 pma_pmd_1g = autoc & IXGBE_AUTOC_1G_PMA_PMD_MASK;
1179         u16 ext_ability = 0;

1181         DEBUGFUNC("ixgbe_get_supported_physical_layer_82598");

1183         hw->phy.ops.identify(hw);

1185         /* Copper PHY must be checked before AUTOC LMS to determine correct
1186          * physical layer because 10GBase-T PHYs use LMS = KX4/KX */
1187         switch (hw->phy.type) {
1188         case ixgbe_phy_tn:
1228         case ixgbe_phy_aq:
1189         case ixgbe_phy_cu_unknown:
1190                 hw->phy.ops.read_reg(hw, IXGBE_MDIO_PHY_EXT_ABILITY,
1191                 IXGBE_MDIO_PMA_PMD_DEV_TYPE, &ext_ability);
1192                 if (ext_ability & IXGBE_MDIO_PHY_10GBASET_ABILITY)
1193                         physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_T;
1194                 if (ext_ability & IXGBE_MDIO_PHY_1000BASET_ABILITY)
1195                         physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_T;
1196                 if (ext_ability & IXGBE_MDIO_PHY_100BASETX_ABILITY)
1197                         physical_layer |= IXGBE_PHYSICAL_LAYER_100BASE_TX;
1198                 goto out;
1199         default:
1200                 break;
1201         }

1203         switch (autoc & IXGBE_AUTOC_LMS_MASK) {
1204         case IXGBE_AUTOC_LMS_1G_AN:
1205         case IXGBE_AUTOC_LMS_1G_LINK_NO_AN:
1206                 if (pma_pmd_1g == IXGBE_AUTOC_1G_KX)
1207                         physical_layer = IXGBE_PHYSICAL_LAYER_1000BASE_KX;
1208                 else
1209                         physical_layer = IXGBE_PHYSICAL_LAYER_1000BASE_BX;
1210                 break;
1211         case IXGBE_AUTOC_LMS_10G_LINK_NO_AN:
1212                 if (pma_pmd_10g == IXGBE_AUTOC_10G_CX4)
1213                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_CX4;
1214                 else if (pma_pmd_10g == IXGBE_AUTOC_10G_KX4)
1215                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_KX4;
1216                 else /* XAUI */
1217                         physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
1218                 break;
1219         case IXGBE_AUTOC_LMS_KX4_AN:
1220         case IXGBE_AUTOC_LMS_KX4_AN_1G_AN:
1221                 if (autoc & IXGBE_AUTOC_KX_SUPP)
1222                         physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_KX;
1223                 if (autoc & IXGBE_AUTOC_KX4_SUPP)
1224                         physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_KX4;
1225                 break;
1226         default:
1227                 break;
1228         }
```



```
1230         if (hw->phy.type == ixgbe_phy_nl) {
1231                 hw->phy.ops.identify_sfp(hw);

1233                 switch (hw->phy.sfp_type) {
1234                 case ixgbe_sfp_type_da_cu:
1235                         physical_layer = IXGBE_PHYSICAL_LAYER_SFP_PLUS_CU;
1236                         break;
1237                 case ixgbe_sfp_type_sr:
1238                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_SR;
1239                         break;
1240                 case ixgbe_sfp_type_lr:
1241                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_LR;
1242                         break;
1243                 default:
1244                         physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
1245                         break;
1246                 }
1247         }

1249         switch (hw->device_id) {
1250         case IXGBE_DEV_ID_82598_DA_DUAL_PORT:
1251                 physical_layer = IXGBE_PHYSICAL_LAYER_SFP_PLUS_CU;
1252                 break;
1253         case IXGBE_DEV_ID_82598AF_DUAL_PORT:
1254         case IXGBE_DEV_ID_82598AF_SINGLE_PORT:
1255         case IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM:
1256                 physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_SR;
1257                 break;
1258         case IXGBE_DEV_ID_82598EB_XF_LR:
1259                 physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_LR;
1260                 break;
1261         default:
1262                 break;
1263         }

1265 out:
1266         return physical_layer;
1267 }
______unchanged_portion_omitted_

1303 /**
1304  *  ixgbe_enable_relaxed_ordering_82598 - enable relaxed ordering
1305  *  @hw: pointer to hardware structure
1306  *
1307  **/
1308 void ixgbe_enable_relaxed_ordering_82598(struct ixgbe_hw *hw)
1309 {
1310         u32 regval;
1311         u32 i;

1313         DEBUGFUNC("ixgbe_enable_relaxed_ordering_82598");

1315         /* Enable relaxed ordering */
1316         for (i = 0; ((i < hw->mac.max_tx_queues) &&
1317              (i < IXGBE_DCA_MAX_QUEUES_82598)); i++) {
1318                 regval = IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL(i));
1319                 regval |= IXGBE_DCA_TXCTRL_DESC_WRO_EN;
1359                 regval |= IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
1320                 IXGBE_WRITE_REG(hw, IXGBE_DCA_TXCTRL(i), regval);
1321         }

1323         for (i = 0; ((i < hw->mac.max_rx_queues) &&
1324              (i < IXGBE_DCA_MAX_QUEUES_82598)); i++) {
1325                 regval = IXGBE_READ_REG(hw, IXGBE_DCA_RXCTRL(i));
1326                 regval |= IXGBE_DCA_RXCTRL_DATA_WRO_EN |
```

```
1327                            IXGBE_DCA_RXCTRL_HEAD_WRO_EN;
1366                *regval |= (IXGBE_DCA_RXCTRL_DESC_WRO_EN |
1367                            IXGBE_DCA_RXCTRL_DESC_HSRO_EN);
1328                IXGBE_WRITE_REG(hw, IXGBE_DCA_RXCTRL(i), regval);
1329        }

1331 }

1333 /**
1334  * ixgbe_set_rxpba_82598 - Initialize RX packet buffer
1335  * @hw: pointer to hardware structure
1336  * @num_pb: number of packet buffers to allocate
1337  * @headroom: reserve n KB of headroom
1338  * @strategy: packet buffer allocation strategy
1339  **/
1340 static void ixgbe_set_rxpba_82598(struct ixgbe_hw *hw, int num_pb,
1341                                   u32 headroom, int strategy)
1342 {
1343         u32 rxpktsize = IXGBE_RXPBSIZE_64KB;
1344         u8 i = 0;
1345         UNREFERENCED_1PARAMETER(headroom);

1347         if (!num_pb)
1348                 return;

1350         /* Setup Rx packet buffer sizes */
1351         switch (strategy) {
1352         case PBA_STRATEGY_WEIGHTED:
1353                 /* Setup the first four at 80KB */
1354                 rxpktsize = IXGBE_RXPBSIZE_80KB;
1355                 for (; i < 4; i++)
1356                         IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), rxpktsize);
1357                 /* Setup the last four at 48KB...don't re-init i */
1358                 rxpktsize = IXGBE_RXPBSIZE_48KB;
1359                 /* Fall Through */
1360         case PBA_STRATEGY_EQUAL:
1361         default:
1362                 /* Divide the remaining Rx packet buffer evenly among the TCs */
1363                 for (; i < IXGBE_MAX_PACKET_BUFFERS; i++)
1364                         IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), rxpktsize);
1365                 break;
1366         }

1368         /* Setup Tx packet buffer sizes */
1369         for (i = 0; i < IXGBE_MAX_PACKET_BUFFERS; i++)
1370                 IXGBE_WRITE_REG(hw, IXGBE_TXPBSIZE(i), IXGBE_TXPBSIZE_40KB);

1372         return;
1373 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    2755 Thu Jul 12 12:22:29 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_82598.h
XXXX Intel X540 support
**********************************************************
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_82598.h,v 1.2 2012/07/05 20:51:44 jfv Exp $*

#ifndef _IXGBE_82598_H_
#define _IXGBE_82598_H_

u32 ixgbe_get_pcie_msix_count_82598(struct ixgbe_hw *hw);
s32 ixgbe_fc_enable_82598(struct ixgbe_hw *hw);
s32 ixgbe_start_hw_82598(struct ixgbe_hw *hw);
void ixgbe_enable_relaxed_ordering_82598(struct ixgbe_hw *hw);
s32 ixgbe_set_vmdq_82598(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
s32 ixgbe_set_vfta_82598(struct ixgbe_hw *hw, u32 vlan, u32 vind, bool vlan_on);
s32 ixgbe_read_analog_reg8_82598(struct ixgbe_hw *hw, u32 reg, u8 *val);
s32 ixgbe_write_analog_reg8_82598(struct ixgbe_hw *hw, u32 reg, u8 val);
s32 ixgbe_read_i2c_eeprom_82598(struct ixgbe_hw *hw, u8 byte_offset,
                                u8 *eeprom_data);
u32 ixgbe_get_supported_physical_layer_82598(struct ixgbe_hw *hw);
s32 ixgbe_init_phy_ops_82598(struct ixgbe_hw *hw);
void ixgbe_set_lan_id_multi_port_pcie_82598(struct ixgbe_hw *hw);
void ixgbe_set_pcie_completion_timeout(struct ixgbe_hw *hw);
#endif /* _IXGBE_82598_H_ */
```

```
**********************************************************
   69464 Thu Jul 12 12:22:29 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_82599.c
XXXX Intel X540 support
**********************************************************
```

```
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_82599.c,v 1.8 2012/07/05 20:51:44 jfv Exp $*
/*$FreeBSD$*/

#include "ixgbe_type.h"
#include "ixgbe_82599.h"
#include "ixgbe_api.h"
#include "ixgbe_common.h"
#include "ixgbe_phy.h"

s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw);
s32 ixgbe_get_link_capabilities_82599(struct ixgbe_hw *hw,
                                      ixgbe_link_speed *speed,
                                      bool *autoneg);
enum ixgbe_media_type ixgbe_get_media_type_82599(struct ixgbe_hw *hw);
void ixgbe_disable_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
void ixgbe_enable_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
void ixgbe_flap_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
s32 ixgbe_setup_mac_link_multispeed_fiber(struct ixgbe_hw *hw,
                                      ixgbe_link_speed speed, bool autoneg,
                                      bool autoneg_wait_to_complete);
s32 ixgbe_setup_mac_link_smartspeed(struct ixgbe_hw *hw,
                                      ixgbe_link_speed speed, bool autoneg,
                                      bool autoneg_wait_to_complete);
s32 ixgbe_start_mac_link_82599(struct ixgbe_hw *hw,
                               bool autoneg_wait_to_complete);
s32 ixgbe_setup_mac_link_82599(struct ixgbe_hw *hw,
                                      ixgbe_link_speed speed,
                                      bool autoneg,
                                      bool autoneg_wait_to_complete);
```



```
static s32 ixgbe_setup_copper_link_82599(struct ixgbe_hw *hw,
                                         ixgbe_link_speed speed,
                                         bool autoneg,
                                         bool autoneg_wait_to_complete);
s32 ixgbe_setup_sfp_modules_82599(struct ixgbe_hw *hw);
void ixgbe_init_mac_link_ops_82599(struct ixgbe_hw *hw);
s32 ixgbe_reset_hw_82599(struct ixgbe_hw *hw);
s32 ixgbe_read_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 *val);
s32 ixgbe_write_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 val);
s32 ixgbe_start_hw_rev_1_82599(struct ixgbe_hw *hw);
s32 ixgbe_identify_phy_82599(struct ixgbe_hw *hw);
s32 ixgbe_init_phy_ops_82599(struct ixgbe_hw *hw);
u32 ixgbe_get_supported_physical_layer_82599(struct ixgbe_hw *hw);
s32 ixgbe_enable_rx_dma_82599(struct ixgbe_hw *hw, u32 regval);
static s32 ixgbe_verify_fw_version_82599(struct ixgbe_hw *hw);
static s32 ixgbe_read_eeprom_82599(struct ixgbe_hw *hw,
                                   u16 offset, u16 *data);
static s32 ixgbe_read_eeprom_buffer_82599(struct ixgbe_hw *hw, u16 offset,
                                          u16 words, u16 *data);
bool ixgbe_verify_lesm_fw_enabled_82599(struct ixgbe_hw *hw);


void ixgbe_init_mac_link_ops_82599(struct ixgbe_hw *hw)
{
        struct ixgbe_mac_info *mac = &hw->mac;

        DEBUGFUNC("ixgbe_init_mac_link_ops_82599");

        /* enable the laser control functions for SFP+ fiber */
        if (mac->ops.get_media_type(hw) == ixgbe_media_type_fiber) {
                mac->ops.disable_tx_laser =
                                       &ixgbe_disable_tx_laser_multispeed_fiber;
                mac->ops.enable_tx_laser =
                                        &ixgbe_enable_tx_laser_multispeed_fiber;
                mac->ops.flap_tx_laser = &ixgbe_flap_tx_laser_multispeed_fiber;

        } else {
                mac->ops.disable_tx_laser = NULL;
                mac->ops.enable_tx_laser = NULL;
                mac->ops.flap_tx_laser = NULL;
        }

        if (hw->phy.multispeed_fiber) {
                /* Set up dual speed SFP+ support */
                mac->ops.setup_link = &ixgbe_setup_mac_link_multispeed_fiber;
        } else {
                if ((ixgbe_get_media_type(hw) == ixgbe_media_type_backplane) &&
                     (hw->phy.smart_speed == ixgbe_smart_speed_auto ||
                      hw->phy.smart_speed == ixgbe_smart_speed_on) &&
                      !ixgbe_verify_lesm_fw_enabled_82599(hw)) {
                        mac->ops.setup_link = &ixgbe_setup_mac_link_smartspeed;
                } else {
                        mac->ops.setup_link = &ixgbe_setup_mac_link_82599;
                }
        }
}

/**
 *  ixgbe_init_phy_ops_82599 - PHY/SFP specific init
 *  @hw: pointer to hardware structure
 *
 *  Initialize any function pointers that were not able to be
 *  set during init_shared_code because the PHY/SFP type was
 *  not known.  Perform the SFP init if necessary.
 *
 **/
```

```
 95 s32 ixgbe_init_phy_ops_82599(struct ixgbe_hw *hw)
 96 {
 97         struct ixgbe_mac_info *mac = &hw->mac;
 98         struct ixgbe_phy_info *phy = &hw->phy;
 99         s32 ret_val = IXGBE_SUCCESS;

101         DEBUGFUNC("ixgbe_init_phy_ops_82599");

103         /* Identify the PHY or SFP module */
104         ret_val = phy->ops.identify(hw);
105         if (ret_val == IXGBE_ERR_SFP_NOT_SUPPORTED)
106                 goto init_phy_ops_out;

108         /* Setup function pointers based on detected SFP module and speeds */
109         ixgbe_init_mac_link_ops_82599(hw);
110         if (hw->phy.sfp_type != ixgbe_sfp_type_unknown)
111                 hw->phy.ops.reset = NULL;

113         /* If copper media, overwrite with copper function pointers */
114         if (mac->ops.get_media_type(hw) == ixgbe_media_type_copper) {
115                 mac->ops.setup_link = &ixgbe_setup_copper_link_82599;
116                 mac->ops.get_link_capabilities =
117                                 &ixgbe_get_copper_link_capabilities_generic;
118         }

120         /* Set necessary function pointers based on phy type */
121         switch (hw->phy.type) {
122         case ixgbe_phy_tn:
123                 phy->ops.setup_link = &ixgbe_setup_phy_link_tnx;
124                 phy->ops.check_link = &ixgbe_check_phy_link_tnx;
125                 phy->ops.get_firmware_version =
126                                 &ixgbe_get_phy_firmware_version_tnx;
127                 break;
155         case ixgbe_phy_aq:
156                 phy->ops.get_firmware_version =
157                                 &ixgbe_get_phy_firmware_version_generic;
158                 break;
128         default:
129                 break;
130         }
131 init_phy_ops_out:
132         return ret_val;
133 }

135 s32 ixgbe_setup_sfp_modules_82599(struct ixgbe_hw *hw)
136 {
137         s32 ret_val = IXGBE_SUCCESS;
138         u32 reg_anlp1 = 0;
139         u32 i = 0;
140         u16 list_offset, data_offset, data_value;

142         DEBUGFUNC("ixgbe_setup_sfp_modules_82599");

144         if (hw->phy.sfp_type != ixgbe_sfp_type_unknown) {
145                 ixgbe_init_mac_link_ops_82599(hw);

147                 hw->phy.ops.reset = NULL;

149                 ret_val = ixgbe_get_sfp_init_sequence_offsets(hw, &list_offset,
150                                                               &data_offset);
151                 if (ret_val != IXGBE_SUCCESS)
152                         goto setup_sfp_out;

154                 /* PHY config will finish before releasing the semaphore */
155                 ret_val = hw->mac.ops.acquire_swfw_sync(hw,
156                                                         IXGBE_GSSR_MAC_CSR_SM);
```



```
186                 ret_val = ixgbe_acquire_swfw_sync(hw, IXGBE_GSSR_MAC_CSR_SM);
157                 if (ret_val != IXGBE_SUCCESS) {
158                         ret_val = IXGBE_ERR_SWFW_SYNC;
159                         goto setup_sfp_out;
160                 }

162                 hw->eeprom.ops.read(hw, ++data_offset, &data_value);
163                 while (data_value != 0xffff) {
164                         IXGBE_WRITE_REG(hw, IXGBE_CORECTL, data_value);
165                         IXGBE_WRITE_FLUSH(hw);
166                         hw->eeprom.ops.read(hw, ++data_offset, &data_value);
167                 }

169                 /* Release the semaphore */
170                 hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_MAC_CSR_SM);
200                 ixgbe_release_swfw_sync(hw, IXGBE_GSSR_MAC_CSR_SM);
171                 /* Delay obtaining semaphore again to allow FW access */
172                 msec_delay(hw->eeprom.semaphore_delay);

174                 /* Now restart DSP by setting Restart_AN and clearing LMS */
175                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, ((IXGBE_READ_REG(hw,
176                                 IXGBE_AUTOC) & ~IXGBE_AUTOC_LMS_MASK) |
177                                 IXGBE_AUTOC_AN_RESTART));

179                 /* Wait for AN to leave state 0 */
180                 for (i = 0; i < 10; i++) {
181                         msec_delay(4);
182                         reg_anlp1 = IXGBE_READ_REG(hw, IXGBE_ANLP1);
183                         if (reg_anlp1 & IXGBE_ANLP1_AN_STATE_MASK)
184                                 break;
185                 }
186                 if (!(reg_anlp1 & IXGBE_ANLP1_AN_STATE_MASK)) {
187                         DEBUGOUT("sfp module setup not complete\n");
188                         ret_val = IXGBE_ERR_SFP_SETUP_NOT_COMPLETE;
189                         goto setup_sfp_out;
190                 }

192                 /* Restart DSP by setting Restart_AN and return to SFI mode */
193                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, (IXGBE_READ_REG(hw,
194                                 IXGBE_AUTOC) | IXGBE_AUTOC_LMS_10G_SERIAL |
195                                 IXGBE_AUTOC_AN_RESTART));
196         }

198 setup_sfp_out:
199         return ret_val;
200 }

202 /**
203  *  ixgbe_init_ops_82599 - Inits func ptrs and MAC type
204  *  @hw: pointer to hardware structure
205  *
206  *  Initialize the function pointers and assign the MAC type for 82599.
207  *  Does not touch the hardware.
208  **/

210 s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw)
211 {
212         struct ixgbe_mac_info *mac = &hw->mac;
213         struct ixgbe_phy_info *phy = &hw->phy;
214         struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
215         s32 ret_val;

217         DEBUGFUNC("ixgbe_init_ops_82599");

219         ret_val = ixgbe_init_phy_ops_generic(hw);
220         ret_val = ixgbe_init_ops_generic(hw);
```

```
222         /* PHY */
223         phy->ops.identify = &ixgbe_identify_phy_82599;
224         phy->ops.init = &ixgbe_init_phy_ops_82599;
225
226         /* MAC */
227         mac->ops.reset_hw = &ixgbe_reset_hw_82599;
228         mac->ops.enable_relaxed_ordering = &ixgbe_enable_relaxed_ordering_gen2;
229         mac->ops.get_media_type = &ixgbe_get_media_type_82599;
230         mac->ops.get_supported_physical_layer =
231                                     &ixgbe_get_supported_physical_layer_82599;
232         mac->ops.disable_sec_rx_path = &ixgbe_disable_sec_rx_path_generic;
233         mac->ops.enable_sec_rx_path = &ixgbe_enable_sec_rx_path_generic;
234         mac->ops.enable_rx_dma = &ixgbe_enable_rx_dma_82599;
235         mac->ops.read_analog_reg8 = &ixgbe_read_analog_reg8_82599;
236         mac->ops.write_analog_reg8 = &ixgbe_write_analog_reg8_82599;
237         mac->ops.start_hw = &ixgbe_start_hw_82599;
264         mac->ops.start_hw = &ixgbe_start_hw_rev_1_82599;
238         mac->ops.get_san_mac_addr = &ixgbe_get_san_mac_addr_generic;
239         mac->ops.set_san_mac_addr = &ixgbe_set_san_mac_addr_generic;
240         mac->ops.get_device_caps = &ixgbe_get_device_caps_generic;
241         mac->ops.get_wwn_prefix = &ixgbe_get_wwn_prefix_generic;
242         mac->ops.get_fcoe_boot_status = &ixgbe_get_fcoe_boot_status_generic;
243
244         /* RAR, Multicast, VLAN */
245         mac->ops.set_vmdq = &ixgbe_set_vmdq_generic;
246         mac->ops.set_vmdq_san_mac = &ixgbe_set_vmdq_san_mac_generic;
247         mac->ops.clear_vmdq = &ixgbe_clear_vmdq_generic;
248         mac->ops.insert_mac_addr = &ixgbe_insert_mac_addr_generic;
249         mac->rar_highwater = 1;
250         mac->ops.set_vfta = &ixgbe_set_vfta_generic;
251         mac->ops.set_vlvf = &ixgbe_set_vlvf_generic;
252         mac->ops.clear_vfta = &ixgbe_clear_vfta_generic;
253         mac->ops.init_uta_tables = &ixgbe_init_uta_tables_generic;
254         mac->ops.setup_sfp = &ixgbe_setup_sfp_modules_82599;
255         mac->ops.set_mac_anti_spoofing = &ixgbe_set_mac_anti_spoofing;
256         mac->ops.set_vlan_anti_spoofing = &ixgbe_set_vlan_anti_spoofing;
257
258         /* Link */
259         mac->ops.get_link_capabilities = &ixgbe_get_link_capabilities_82599;
260         mac->ops.check_link = &ixgbe_check_mac_link_generic;
261         mac->ops.setup_rxpba = &ixgbe_set_rxpba_generic;
262         ixgbe_init_mac_link_ops_82599(hw);
263
264         mac->mcft_size          = 128;
265         mac->vft_size           = 128;
266         mac->num_rar_entries    = 128;
267         mac->rx_pb_size         = 512;
268         mac->max_tx_queues      = 128;
269         mac->max_rx_queues      = 128;
270         mac->max_msix_vectors   = ixgbe_get_pcie_msix_count_generic(hw);
271
272         mac->arc_subsystem_valid = (IXGBE_READ_REG(hw, IXGBE_FWSM) &
273                                     IXGBE_FWSM_MODE_MASK) ? TRUE : FALSE;
274
275         hw->mbx.ops.init_params = ixgbe_init_mbx_params_pf;
276
277         /* EEPROM */
278         eeprom->ops.read = &ixgbe_read_eeprom_82599;
279         eeprom->ops.read_buffer = &ixgbe_read_eeprom_buffer_82599;
280
281         /* Manageability interface */
282         mac->ops.set_fw_drv_ver = &ixgbe_set_fw_drv_ver_generic;
283
284
285         return ret_val;
```



```
286 }
287
288 /**
289  *  ixgbe_get_link_capabilities_82599 - Determines link capabilities
290  *  @hw: pointer to hardware structure
291  *  @speed: pointer to link speed
292  *  @negotiation: TRUE when autoneg or autotry is enabled
293  *
294  *  Determines the link capabilities by reading the AUTOC register.
295  **/
296 s32 ixgbe_get_link_capabilities_82599(struct ixgbe_hw *hw,
297                                       ixgbe_link_speed *speed,
298                                       bool *negotiation)
299 {
300         s32 status = IXGBE_SUCCESS;
301         u32 autoc = 0;
302
303         DEBUGFUNC("ixgbe_get_link_capabilities_82599");
304
305
306         /* Check if 1G SFP module. */
307         if (hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0 ||
308             hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1 ||
309             hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core0 ||
310             hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core1) {
319             hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1) {
311                 *speed = IXGBE_LINK_SPEED_1GB_FULL;
312                 *negotiation = TRUE;
313                 goto out;
314         }
315
316         /*
317          * Determine link capabilities based on the stored value of AUTOC,
318          * which represents EEPROM defaults.  If AUTOC value has not
319          * been stored, use the current register values.
320          */
321         if (hw->mac.orig_link_settings_stored)
322                 autoc = hw->mac.orig_autoc;
323         else
324                 autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
325
326         switch (autoc & IXGBE_AUTOC_LMS_MASK) {
327         case IXGBE_AUTOC_LMS_1G_LINK_NO_AN:
328                 *speed = IXGBE_LINK_SPEED_1GB_FULL;
329                 *negotiation = FALSE;
330                 break;
331
332         case IXGBE_AUTOC_LMS_10G_LINK_NO_AN:
333                 *speed = IXGBE_LINK_SPEED_10GB_FULL;
334                 *negotiation = FALSE;
335                 break;
336
337         case IXGBE_AUTOC_LMS_1G_AN:
338                 *speed = IXGBE_LINK_SPEED_1GB_FULL;
339                 *negotiation = TRUE;
340                 break;
341
342         case IXGBE_AUTOC_LMS_10G_SERIAL:
343                 *speed = IXGBE_LINK_SPEED_10GB_FULL;
344                 *negotiation = FALSE;
345                 break;
346
347         case IXGBE_AUTOC_LMS_KX4_KX_KR:
348         case IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN:
349                 *speed = IXGBE_LINK_SPEED_UNKNOWN;
350                 if (autoc & IXGBE_AUTOC_KR_SUPP)
```

```
351                         *speed |= IXGBE_LINK_SPEED_10GB_FULL;
352                 if (autoc & IXGBE_AUTOC_KX4_SUPP)
353                         *speed |= IXGBE_LINK_SPEED_10GB_FULL;
354                 if (autoc & IXGBE_AUTOC_KX_SUPP)
355                         *speed |= IXGBE_LINK_SPEED_1GB_FULL;
356                 *negotiation = TRUE;
357                 break;

359         case IXGBE_AUTOC_LMS_KX4_KX_KR_SGMII:
360                 *speed = IXGBE_LINK_SPEED_100_FULL;
361                 if (autoc & IXGBE_AUTOC_KR_SUPP)
362                         *speed |= IXGBE_LINK_SPEED_10GB_FULL;
363                 if (autoc & IXGBE_AUTOC_KX4_SUPP)
364                         *speed |= IXGBE_LINK_SPEED_10GB_FULL;
365                 if (autoc & IXGBE_AUTOC_KX_SUPP)
366                         *speed |= IXGBE_LINK_SPEED_1GB_FULL;
367                 *negotiation = TRUE;
368                 break;

370         case IXGBE_AUTOC_LMS_SGMII_1G_100M:
371                 *speed = IXGBE_LINK_SPEED_1GB_FULL | IXGBE_LINK_SPEED_100_FULL;
372                 *negotiation = FALSE;
373                 break;

375         default:
376                 status = IXGBE_ERR_LINK_SETUP;
377                 goto out;
378         }

380         if (hw->phy.multispeed_fiber) {
381                 *speed |= IXGBE_LINK_SPEED_10GB_FULL |
382                           IXGBE_LINK_SPEED_1GB_FULL;
383                 *negotiation = TRUE;
384         }

386 out:
387         return status;
388 }

390 /**
391  *  ixgbe_get_media_type_82599 - Get media type
392  *  @hw: pointer to hardware structure
393  *
394  *  Returns the media type (fiber, copper, backplane)
395  **/
396 enum ixgbe_media_type ixgbe_get_media_type_82599(struct ixgbe_hw *hw)
397 {
398         enum ixgbe_media_type media_type;

400         DEBUGFUNC("ixgbe_get_media_type_82599");

402         /* Detect if there is a copper PHY attached. */
403         switch (hw->phy.type) {
404         case ixgbe_phy_cu_unknown:
405         case ixgbe_phy_tn:
415         case ixgbe_phy_aq:
406                 media_type = ixgbe_media_type_copper;
407                 goto out;
408         default:
409                 break;
410         }

412         switch (hw->device_id) {
413         case IXGBE_DEV_ID_82599_KX4:
414         case IXGBE_DEV_ID_82599_KX4_MEZZ:
415         case IXGBE_DEV_ID_82599_COMBO_BACKPLANE:
```



```
416         case IXGBE_DEV_ID_82599_KR:
417         case IXGBE_DEV_ID_82599_BACKPLANE_FCOE:
418         case IXGBE_DEV_ID_82599_XAUI_LOM:
419                 /* Default device ID is mezzanine card KX/KX4 */
420                 media_type = ixgbe_media_type_backplane;
421                 break;
422         case IXGBE_DEV_ID_82599_SFP:
423         case IXGBE_DEV_ID_82599_SFP_FCOE:
424         case IXGBE_DEV_ID_82599_SFP_EM:
425         case IXGBE_DEV_ID_82599_SFP_SF2:
426         case IXGBE_DEV_ID_82599EN_SFP:
427                 media_type = ixgbe_media_type_fiber;
428                 break;
429         case IXGBE_DEV_ID_82599_CX4:
430                 media_type = ixgbe_media_type_cx4;
431                 break;
432         case IXGBE_DEV_ID_82599_T3_LOM:
433                 media_type = ixgbe_media_type_copper;
434                 break;
435         default:
436                 media_type = ixgbe_media_type_unknown;
437                 break;
438         }
439 out:
440         return media_type;
441 }
```

*unchanged_portion_omitted_*

```
934 /**
935  *  ixgbe_setup_copper_link_82599 - Set the PHY autoneg advertised field
936  *  @hw: pointer to hardware structure
937  *  @speed: new link speed
938  *  @autoneg: TRUE if autonegotiation enabled
939  *  @autoneg_wait_to_complete: TRUE if waiting is needed to complete
940  *
941  *  Restarts link on PHY and MAC based on settings passed in.
942  **/
943 static s32 ixgbe_setup_copper_link_82599(struct ixgbe_hw *hw,
944                                          ixgbe_link_speed speed,
945                                          bool autoneg,
946                                          bool autoneg_wait_to_complete)
947 {
948         s32 status;

950         DEBUGFUNC("ixgbe_setup_copper_link_82599");

952         /* Setup the PHY according to input speed */
953         status = hw->phy.ops.setup_link_speed(hw, speed, autoneg,
954                                               autoneg_wait_to_complete);
955         /* Set up MAC */
956         ixgbe_start_mac_link_82599(hw, autoneg_wait_to_complete);
966         (void) ixgbe_start_mac_link_82599(hw, autoneg_wait_to_complete);

958         return status;
959 }

961 /**
962  *  ixgbe_reset_hw_82599 - Perform hardware reset
963  *  @hw: pointer to hardware structure
964  *
965  *  Resets the hardware by resetting the transmit and receive units, masks
966  *  and clears all interrupts, perform a PHY reset, and perform a link (MAC)
967  *  reset.
968  **/
969 s32 ixgbe_reset_hw_82599(struct ixgbe_hw *hw)
970 {
```

```
        ixgbe_link_speed link_speed;
        s32 status;
        u32 ctrl, i, autoc, autoc2;
        bool link_up = FALSE;
        s32 status = IXGBE_SUCCESS;
        u32 ctrl;
        u32 i;
        u32 autoc;
        u32 autoc2;

        DEBUGFUNC("ixgbe_reset_hw_82599");

        /* Call adapter stop to disable tx/rx and clear interrupts */
        status = hw->mac.ops.stop_adapter(hw);
        if (status != IXGBE_SUCCESS)
                goto reset_hw_out;
        hw->mac.ops.stop_adapter(hw);

        /* flush pending Tx transactions */
        ixgbe_clear_tx_pending(hw);

        /* PHY ops must be identified and initialized prior to reset */

        /* Identify PHY and related function pointers */
        status = hw->phy.ops.init(hw);

        if (status == IXGBE_ERR_SFP_NOT_SUPPORTED)
                goto reset_hw_out;

        /* Setup SFP module if there is one present. */
        if (hw->phy.sfp_setup_needed) {
                status = hw->mac.ops.setup_sfp(hw);
                hw->phy.sfp_setup_needed = FALSE;
        }

        if (status == IXGBE_ERR_SFP_NOT_SUPPORTED)
                goto reset_hw_out;

        /* Reset PHY */
        if (hw->phy.reset_disable == FALSE && hw->phy.ops.reset != NULL)
                hw->phy.ops.reset(hw);

mac_reset_top:
        /*
         * Issue global reset to the MAC.  Needs to be SW reset if link is up.
         * If link reset is used when link is up, it might reset the PHY when
         * mng is using it.  If link is down or the flag to force full link
         * reset is set, then perform link reset.
         * Prevent the PCI-E bus from from hanging by disabling PCI-E master
         * access and verify no pending requests before reset
         */
        ctrl = IXGBE_CTRL_LNK_RST;
        if (!hw->force_full_reset) {
                hw->mac.ops.check_link(hw, &link_speed, &link_up, FALSE);
                if (link_up)
                        ctrl = IXGBE_CTRL_RST;
        }
        (void) ixgbe_disable_pcie_master(hw);

        ctrl |= IXGBE_READ_REG(hw, IXGBE_CTRL);
        IXGBE_WRITE_REG(hw, IXGBE_CTRL, ctrl);
mac_reset_top:
        /*
         * Issue global reset to the MAC.  This needs to be a SW reset.
         * If link reset is used, it might reset the MAC when mng is using it
         */
```



```
        ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
        IXGBE_WRITE_REG(hw, IXGBE_CTRL, (ctrl | IXGBE_CTRL_RST));
        IXGBE_WRITE_FLUSH(hw);

        /* Poll for reset bit to self-clear indicating reset is complete */
        for (i = 0; i < 10; i++) {
                usec_delay(1);
                ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
                if (!(ctrl & IXGBE_CTRL_RST_MASK))
                if (!(ctrl & IXGBE_CTRL_RST))
                        break;
        }

        if (ctrl & IXGBE_CTRL_RST_MASK) {
        if (ctrl & IXGBE_CTRL_RST) {
                status = IXGBE_ERR_RESET_FAILED;
                DEBUGOUT("Reset polling failed to complete.\n");
        }

        msec_delay(50);

        /*
         * Double resets are required for recovery from certain error
         * conditions.  Between resets, it is necessary to stall to allow time
         * for any pending HW events to complete.
         * for any pending HW events to complete.  We use 1usec since that is
         * what is needed for ixgbe_disable_pcie_master().  The second reset
         * then clears out any effects of those events.
         */
        if (hw->mac.flags & IXGBE_FLAGS_DOUBLE_RESET_REQUIRED) {
                hw->mac.flags &= ~IXGBE_FLAGS_DOUBLE_RESET_REQUIRED;
                usec_delay(1);
                goto mac_reset_top;
        }

        msec_delay(50);

        /*
         * Store the original AUTOC/AUTOC2 values if they have not been
         * stored off yet.  Otherwise restore the stored original
         * values since the reset operation sets back to defaults.
         */
        autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
        autoc2 = IXGBE_READ_REG(hw, IXGBE_AUTOC2);
        if (hw->mac.orig_link_settings_stored == FALSE) {
                hw->mac.orig_autoc = autoc;
                hw->mac.orig_autoc2 = autoc2;
                hw->mac.orig_link_settings_stored = TRUE;
        } else {
                if (autoc != hw->mac.orig_autoc)
                        IXGBE_WRITE_REG(hw, IXGBE_AUTOC, (hw->mac.orig_autoc |
                                        IXGBE_AUTOC_AN_RESTART));

                if ((autoc2 & IXGBE_AUTOC2_UPPER_MASK) !=
                    (hw->mac.orig_autoc2 & IXGBE_AUTOC2_UPPER_MASK)) {
                        autoc2 &= ~IXGBE_AUTOC2_UPPER_MASK;
                        autoc2 |= (hw->mac.orig_autoc2 &
                                   IXGBE_AUTOC2_UPPER_MASK);
                        IXGBE_WRITE_REG(hw, IXGBE_AUTOC2, autoc2);
                }
        }

        /* Store the permanent mac address */
        hw->mac.ops.get_mac_addr(hw, hw->mac.perm_addr);

        /*
```

```
1079          * Store MAC address from RAR0, clear receive address registers, and
1080          * clear the multicast table.  Also reset num_rar_entries to 128,
1081          * since we modify this value when programming the SAN MAC address.
1082          */
1083         hw->mac.num_rar_entries = 128;
1084         hw->mac.ops.init_rx_addrs(hw);

1086         /* Store the permanent SAN mac address */
1087         hw->mac.ops.get_san_mac_addr(hw, hw->mac.san_addr);

1089         /* Add the SAN MAC address to the RAR only if it's a valid address */
1090         if (ixgbe_validate_mac_addr(hw->mac.san_addr) == 0) {
1091                 hw->mac.ops.set_rar(hw, hw->mac.num_rar_entries - 1,
1092                                     hw->mac.san_addr, 0, IXGBE_RAH_AV);

1094                 /* Save the SAN MAC RAR index */
1095                 hw->mac.san_mac_rar_index = hw->mac.num_rar_entries - 1;

1097                 /* Reserve the last RAR for the SAN MAC address */
1098                 hw->mac.num_rar_entries--;
1099         }

1101         /* Store the alternative WWNN/WWPN prefix */
1102         hw->mac.ops.get_wwn_prefix(hw, &hw->mac.wwnn_prefix,
1103                                    &hw->mac.wwpn_prefix);

1105 reset_hw_out:
1106         return status;
1107 }

1109 /**
1110  *  ixgbe_reinit_fdir_tables_82599 - Reinitialize Flow Director tables.
1111  *  @hw: pointer to hardware structure
1112  **/
1113 s32 ixgbe_reinit_fdir_tables_82599(struct ixgbe_hw *hw)
1114 {
1115         int i;
1116         u32 fdirctrl = IXGBE_READ_REG(hw, IXGBE_FDIRCTRL);
1117         fdirctrl &= ~IXGBE_FDIRCTRL_INIT_DONE;

1119         DEBUGFUNC("ixgbe_reinit_fdir_tables_82599");

1121         /*
1122          * Before starting reinitialization process,
1123          * FDIRCMD.CMD must be zero.
1124          */
1125         for (i = 0; i < IXGBE_FDIRCMD_CMD_POLL; i++) {
1126                 if (!(IXGBE_READ_REG(hw, IXGBE_FDIRCMD) &
1127                       IXGBE_FDIRCMD_CMD_MASK))
1128                         break;
1129                 usec_delay(10);
1130         }
1131         if (i >= IXGBE_FDIRCMD_CMD_POLL) {
1132                 DEBUGOUT("Flow Director previous command isn't complete, "
1133                          "aborting table re-initialization.\n");
1135                          "aborting table re-initialization. \n");
1134                 return IXGBE_ERR_FDIR_REINIT_FAILED;
1135         }

1137         IXGBE_WRITE_REG(hw, IXGBE_FDIRFREE, 0);
1138         IXGBE_WRITE_FLUSH(hw);
1139         /*
1140          * 82599 adapters flow director init flow cannot be restarted,
1141          * Workaround 82599 silicon errata by performing the following steps
1142          * before re-writing the FDIRCTRL control register with the same value.
1143          * - write 1 to bit 8 of FDIRCMD register &
```



```
1144          * - write 0 to bit 8 of FDIRCMD register
1145          */
1146         IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD,
1147                         (IXGBE_READ_REG(hw, IXGBE_FDIRCMD) |
1148                          IXGBE_FDIRCMD_CLEARHT));
1149         IXGBE_WRITE_FLUSH(hw);
1150         IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD,
1151                         (IXGBE_READ_REG(hw, IXGBE_FDIRCMD) &
1152                          ~IXGBE_FDIRCMD_CLEARHT));
1153         IXGBE_WRITE_FLUSH(hw);
1154         /*
1155          * Clear FDIR Hash register to clear any leftover hashes
1156          * waiting to be programmed.
1157          */
1158         IXGBE_WRITE_REG(hw, IXGBE_FDIRHASH, 0x00);
1159         IXGBE_WRITE_FLUSH(hw);

1161         IXGBE_WRITE_REG(hw, IXGBE_FDIRCTRL, fdirctrl);
1162         IXGBE_WRITE_FLUSH(hw);

1164         /* Poll init-done after we write FDIRCTRL register */
1165         for (i = 0; i < IXGBE_FDIR_INIT_DONE_POLL; i++) {
1166                 if (IXGBE_READ_REG(hw, IXGBE_FDIRCTRL) &
1167                                    IXGBE_FDIRCTRL_INIT_DONE)
1168                         break;
1169                 usec_delay(10);
1170         }
1171         if (i >= IXGBE_FDIR_INIT_DONE_POLL) {
1172                 DEBUGOUT("Flow Director Signature poll time exceeded!\n");
1173                 return IXGBE_ERR_FDIR_REINIT_FAILED;
1174         }

1176         /* Clear FDIR statistics registers (read to clear) */
1177         IXGBE_READ_REG(hw, IXGBE_FDIRUSTAT);
1178         IXGBE_READ_REG(hw, IXGBE_FDIRFSTAT);
1179         IXGBE_READ_REG(hw, IXGBE_FDIRMATCH);
1180         IXGBE_READ_REG(hw, IXGBE_FDIRMISS);
1181         IXGBE_READ_REG(hw, IXGBE_FDIRLEN);
1179         (void) IXGBE_READ_REG(hw, IXGBE_FDIRUSTAT);
1180         (void) IXGBE_READ_REG(hw, IXGBE_FDIRFSTAT);
1181         (void) IXGBE_READ_REG(hw, IXGBE_FDIRMATCH);
1182         (void) IXGBE_READ_REG(hw, IXGBE_FDIRMISS);
1183         (void) IXGBE_READ_REG(hw, IXGBE_FDIRLEN);

1183         return IXGBE_SUCCESS;
1184 }

1186 /**
1187  *  ixgbe_fdir_enable_82599 - Initialize Flow Director control registers
1189  *  ixgbe_init_fdir_signature_82599 - Initialize Flow Director signature filters
1188  *  @hw: pointer to hardware structure
1189  *  @fdirctrl: value to write to flow director control register
1191  *  @pballoc: which mode to allocate filters with
1190  **/
1191 static void ixgbe_fdir_enable_82599(struct ixgbe_hw *hw, u32 fdirctrl)
1193 s32 ixgbe_init_fdir_signature_82599(struct ixgbe_hw *hw, u32 pballoc)
1192 {
1195         u32 fdirctrl = 0;
1196         u32 pbsize;
1193         int i;

1195         DEBUGFUNC("ixgbe_fdir_enable_82599");
1199         DEBUGFUNC("ixgbe_init_fdir_signature_82599");

1201         /*
1202          * Before enabling Flow Director, the Rx Packet Buffer size
```

```
1203          * must be reduced.  The new value is the current size minus
1204          * flow director memory usage size.
1205          */
1206         pbsize = (1 << (IXGBE_FDIR_PBALLOC_SIZE_SHIFT + pballoc));
1207         IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(0),
1208             (IXGBE_READ_REG(hw, IXGBE_RXPBSIZE(0)) - pbsize));

1210         /*
1211          * The defaults in the HW for RX PB 1-7 are not zero and so should be
1212          * intialized to zero for non DCB mode otherwise actual total RX PB
1213          * would be bigger than programmed and filter space would run into
1214          * the PB 0 region.
1215          */
1216         for (i = 1; i < 8; i++)
1217                 IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), 0);

1219         /* Send interrupt when 64 filters are left */
1220         fdirctrl |= 4 << IXGBE_FDIRCTRL_FULL_THRESH_SHIFT;

1222         /* Set the maximum length per hash bucket to 0xA filters */
1223         fdirctrl |= 0xA << IXGBE_FDIRCTRL_MAX_LENGTH_SHIFT;

1225         switch (pballoc) {
1226         case IXGBE_FDIR_PBALLOC_64K:
1227                 /* 8k - 1 signature filters */
1228                 fdirctrl |= IXGBE_FDIRCTRL_PBALLOC_64K;
1229                 break;
1230         case IXGBE_FDIR_PBALLOC_128K:
1231                 /* 16k - 1 signature filters */
1232                 fdirctrl |= IXGBE_FDIRCTRL_PBALLOC_128K;
1233                 break;
1234         case IXGBE_FDIR_PBALLOC_256K:
1235                 /* 32k - 1 signature filters */
1236                 fdirctrl |= IXGBE_FDIRCTRL_PBALLOC_256K;
1237                 break;
1238         default:
1239                 /* bad value */
1240                 return IXGBE_ERR_CONFIG;
1241         };

1243         /* Move the flexible bytes to use the ethertype - shift 6 words */
1244         fdirctrl |= (0x6 << IXGBE_FDIRCTRL_FLEX_SHIFT);


1197         /* Prime the keys for hashing */
1198         IXGBE_WRITE_REG(hw, IXGBE_FDIRHKEY, IXGBE_ATR_BUCKET_HASH_KEY);
1199         IXGBE_WRITE_REG(hw, IXGBE_FDIRSKEY, IXGBE_ATR_SIGNATURE_HASH_KEY);

1201         /*
1202          * Poll init-done after we write the register.  Estimated times:
1203          *      10G: PBALLOC = 11b, timing is 60us
1204          *       1G: PBALLOC = 11b, timing is 600us
1205          *     100M: PBALLOC = 11b, timing is 6ms
1206          *
1207          *     Multiple these timings by 4 if under full Rx load
1208          *
1209          * So we'll poll for IXGBE_FDIR_INIT_DONE_POLL times, sleeping for
1210          * 1 msec per poll time.  If we're at line rate and drop to 100M, then
1211          * this might not finish in our poll time, but we can live with that
1212          * for now.
1213          */
1214         IXGBE_WRITE_REG(hw, IXGBE_FDIRCTRL, fdirctrl);
1215         IXGBE_WRITE_FLUSH(hw);
1216         for (i = 0; i < IXGBE_FDIR_INIT_DONE_POLL; i++) {
1217                 if (IXGBE_READ_REG(hw, IXGBE_FDIRCTRL) &
1218                                    IXGBE_FDIRCTRL_INIT_DONE)
```



```
1219                         break;
1220                 msec_delay(1);
1221         }

1223         if (i >= IXGBE_FDIR_INIT_DONE_POLL)
1224                 DEBUGOUT("Flow Director poll time exceeded!\n");
1273                 DEBUGOUT("Flow Director Signature poll time exceeded!\n");

1275         return IXGBE_SUCCESS;
1225 }

1227 /**
1228  *  ixgbe_init_fdir_signature_82599 - Initialize Flow Director signature filters
1279  *  ixgbe_init_fdir_perfect_82599 - Initialize Flow Director perfect filters
1229  *  @hw: pointer to hardware structure
1230  *  @fdirctrl: value to write to flow director control register, initially
1231  *          contains just the value of the Rx packet buffer allocation
1281  *  @pballoc: which mode to allocate filters with
1232  **/
1233 s32 ixgbe_init_fdir_signature_82599(struct ixgbe_hw *hw, u32 fdirctrl)
1283 s32 ixgbe_init_fdir_perfect_82599(struct ixgbe_hw *hw, u32 pballoc)
1234 {
1235         DEBUGFUNC("ixgbe_init_fdir_signature_82599");
1285         u32 fdirctrl = 0;
1286         u32 pbsize;
1287         int i;

1289         DEBUGFUNC("ixgbe_init_fdir_perfect_82599");

1237         /*
1238          * Continue setup of fdirctrl register bits:
1239          *  Move the flexible bytes to use the ethertype - shift 6 words
1240          *  Set the maximum length per hash bucket to 0xA filters
1241          *  Send interrupt when 64 filters are left
1292          * Before enabling Flow Director, the Rx Packet Buffer size
1293          * must be reduced.  The new value is the current size minus
1294          * flow director memory usage size.
1242          */
1243         fdirctrl |= (0x6 << IXGBE_FDIRCTRL_FLEX_SHIFT) |
1244                     (0xA << IXGBE_FDIRCTRL_MAX_LENGTH_SHIFT) |
1245                     (4 << IXGBE_FDIRCTRL_FULL_THRESH_SHIFT);
1296         pbsize = (1 << (IXGBE_FDIR_PBALLOC_SIZE_SHIFT + pballoc));
1297         IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(0),
1298             (IXGBE_READ_REG(hw, IXGBE_RXPBSIZE(0)) - pbsize));

1247         /* write hashes and fdirctrl register, poll for completion */
1248         ixgbe_fdir_enable_82599(hw, fdirctrl);
1300         /*
1301          * The defaults in the HW for RX PB 1-7 are not zero and so should be
1302          * intialized to zero for non DCB mode otherwise actual total RX PB
1303          * would be bigger than programmed and filter space would run into
1304          * the PB 0 region.
1305          */
1306         for (i = 1; i < 8; i++)
1307                 IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), 0);

1309         /* Send interrupt when 64 filters are left */
1310         fdirctrl |= 4 << IXGBE_FDIRCTRL_FULL_THRESH_SHIFT;

1312         /* Initialize the drop queue to Rx queue 127 */
1313         fdirctrl |= (127 << IXGBE_FDIRCTRL_DROP_Q_SHIFT);

1315         switch (pballoc) {
1316         case IXGBE_FDIR_PBALLOC_64K:
1317                 /* 2k - 1 perfect filters */
1318                 fdirctrl |= IXGBE_FDIRCTRL_PBALLOC_64K;
```

```
1319                 break;
1320         case IXGBE_FDIR_PBALLOC_128K:
1321                 /* 4k - 1 perfect filters */
1322                 fdirctrl |= IXGBE_FDIRCTRL_PBALLOC_128K;
1323                 break;
1324         case IXGBE_FDIR_PBALLOC_256K:
1325                 /* 8k - 1 perfect filters */
1326                 fdirctrl |= IXGBE_FDIRCTRL_PBALLOC_256K;
1327                 break;
1328         default:
1329                 /* bad value */
1330                 return IXGBE_ERR_CONFIG;
1331         };

1333         /* Turn perfect match filtering on */
1334         fdirctrl |= IXGBE_FDIRCTRL_PERFECT_MATCH;
1335         fdirctrl |= IXGBE_FDIRCTRL_REPORT_STATUS;

1337         /* Move the flexible bytes to use the ethertype - shift 6 words */
1338         fdirctrl |= (0x6 << IXGBE_FDIRCTRL_FLEX_SHIFT);

1340         /* Prime the keys for hashing */
1341         IXGBE_WRITE_REG(hw, IXGBE_FDIRHKEY, IXGBE_ATR_BUCKET_HASH_KEY);
1342         IXGBE_WRITE_REG(hw, IXGBE_FDIRSKEY,IXGBE_ATR_SIGNATURE_HASH_KEY);

1344         /*
1345          * Poll init-done after we write the register.  Estimated times:
1346          *      10G: PBALLOC = 11b, timing is 60us
1347          *       1G: PBALLOC = 11b, timing is 600us
1348          *     100M: PBALLOC = 11b, timing is 6ms
1349          *
1350          *     Multiple these timings by 4 if under full Rx load
1351          *
1352          * So we'll poll for IXGBE_FDIR_INIT_DONE_POLL times, sleeping for
1353          * 1 msec per poll time.  If we're at line rate and drop to 100M, then
1354          * this might not finish in our poll time, but we can live with that
1355          * for now.
1356          */

1358         /* Set the maximum length per hash bucket to 0xA filters */
1359         fdirctrl |= (0xA << IXGBE_FDIRCTRL_MAX_LENGTH_SHIFT);

1361         IXGBE_WRITE_REG(hw, IXGBE_FDIRCTRL, fdirctrl);
1362         IXGBE_WRITE_FLUSH(hw);
1363         for (i = 0; i < IXGBE_FDIR_INIT_DONE_POLL; i++) {
1364                 if (IXGBE_READ_REG(hw, IXGBE_FDIRCTRL) &
1365                                    IXGBE_FDIRCTRL_INIT_DONE)
1366                         break;
1367                 msec_delay(1);
1368         }
1369         if (i >= IXGBE_FDIR_INIT_DONE_POLL)
1370                 DEBUGOUT("Flow Director Perfect poll time exceeded!\n");

1250         return IXGBE_SUCCESS;
1251 }

1253 /**
1254  *  ixgbe_init_fdir_perfect_82599 - Initialize Flow Director perfect filters
1255  *  @hw: pointer to hardware structure
1256  *  @fdirctrl: value to write to flow director control register, initially
1257  *          contains just the value of the Rx packet buffer allocation
1376  *  ixgbe_atr_compute_hash_82599 - Compute the hashes for SW ATR
1377  *  @stream: input bitstream to compute the hash on
1378  *  @key: 32-bit hash key
1258  **/
1259 s32 ixgbe_init_fdir_perfect_82599(struct ixgbe_hw *hw, u32 fdirctrl)
```



```
1380 u32 ixgbe_atr_compute_hash_82599(union ixgbe_atr_input *atr_input,
1381                                   u32 key)
1260 {
1261         DEBUGFUNC("ixgbe_init_fdir_perfect_82599");
1383         /*
1384          * The algorithm is as follows:
1385          *    Hash[15:0] = Sum { S[n] x K[n+16] }, n = 0...350
1386          *    where Sum {A[n]}, n = 0...n is bitwise XOR of A[0], A[1]...A[n]
1387          *    and A[n] x B[n] is bitwise AND between same length strings
1388          *
1389          *    K[n] is 16 bits, defined as:
1390          *       for n modulo 32 >= 15, K[n] = K[n % 32 : (n % 32) - 15]
1391          *       for n modulo 32 < 15, K[n] =
1392          *             K[(n % 32:0) | (31:31 - (14 - (n % 32)))]
1393          *
1394          *    S[n] is 16 bits, defined as:
1395          *       for n >= 15, S[n] = S[n:n - 15]
1396          *       for n < 15, S[n] = S[(n:0) | (350:350 - (14 - n))]
1397          *
1398          *    To simplify for programming, the algorithm is implemented
1399          *    in software this way:
1400          *
1401          *    key[31:0], hi_hash_dword[31:0], lo_hash_dword[31:0], hash[15:0]
1402          *
1403          *    for (i = 0; i < 352; i+=32)
1404          *        hi_hash_dword[31:0] ^= Stream[(i+31):i];
1405          *
1406          *    lo_hash_dword[15:0]  ^= Stream[15:0];
1407          *    lo_hash_dword[15:0]  ^= hi_hash_dword[31:16];
1408          *    lo_hash_dword[31:16] ^= hi_hash_dword[15:0];
1409          *
1410          *    hi_hash_dword[31:0]  ^= Stream[351:320];
1411          *
1412          *    if(key[0])
1413          *        hash[15:0] ^= Stream[15:0];
1414          *
1415          *    for (i = 0; i < 16; i++) {
1416          *        if (key[i])
1417          *            hash[15:0] ^= lo_hash_dword[(i+15):i];
1418          *        if (key[i + 16])
1419          *            hash[15:0] ^= hi_hash_dword[(i+15):i];
1420          *    }
1421          *
1422          */
1423         __be32 common_hash_dword = 0;
1424         u32 hi_hash_dword, lo_hash_dword, flow_vm_vlan;
1425         u32 hash_result = 0;
1426         u8 i;

1428         /* record the flow_vm_vlan bits as they are a key part to the hash */
1429         flow_vm_vlan = IXGBE_NTOHL(atr_input->dword_stream[0]);

1431         /* generate common hash dword */
1432         for (i = 10; i; i -= 2)
1433                 common_hash_dword ^= atr_input->dword_stream[i] ^
1434                                      atr_input->dword_stream[i - 1];

1436         hi_hash_dword = IXGBE_NTOHL(common_hash_dword);

1438         /* low dword is word swapped version of common */
1439         lo_hash_dword = (hi_hash_dword >> 16) | (hi_hash_dword << 16);

1441         /* apply flow ID/VM pool/VLAN ID bits to hash words */
1442         hi_hash_dword ^= flow_vm_vlan ^ (flow_vm_vlan >> 16);

1444         /* Process bits 0 and 16 */
```

```
1445         if (key & 0x0001) hash_result ^= lo_hash_dword;
1446         if (key & 0x00010000) hash_result ^= hi_hash_dword;

1263         /*
1264          * Continue setup of fdirctrl register bits:
1265          *  Turn perfect match filtering on
1266          *  Report hash in RSS field of Rx wb descriptor
1267          *  Initialize the drop queue
1268          *  Move the flexible bytes to use the ethertype - shift 6 words
1269          *  Set the maximum length per hash bucket to 0xA filters
1270          *  Send interrupt when 64 (0x4 * 16) filters are left
1449          * apply flow ID/VM pool/VLAN ID bits to lo hash dword, we had to
1450          * delay this because bit 0 of the stream should not be processed
1451          * so we do not add the vlan until after bit 0 was processed
1271          */
1272         fdirctrl |= IXGBE_FDIRCTRL_PERFECT_MATCH |
1273                     IXGBE_FDIRCTRL_REPORT_STATUS |
1274                     (IXGBE_FDIR_DROP_QUEUE << IXGBE_FDIRCTRL_DROP_Q_SHIFT) |
1275                     (0x6 << IXGBE_FDIRCTRL_FLEX_SHIFT) |
1276                     (0xA << IXGBE_FDIRCTRL_MAX_LENGTH_SHIFT) |
1277                     (4 << IXGBE_FDIRCTRL_FULL_THRESH_SHIFT);
1453         lo_hash_dword ^= flow_vm_vlan ^ (flow_vm_vlan << 16);

1279         /* write hashes and fdirctrl register, poll for completion */
1280         ixgbe_fdir_enable_82599(hw, fdirctrl);

1282         return IXGBE_SUCCESS;
1456         /* process the remaining 30 bits in the key 2 bits at a time */
1457         for (i = 15; i; i-- ) {
1458                 if (key & (0x0001 << i)) hash_result ^= lo_hash_dword >> i;
1459                 if (key & (0x00010000 << i)) hash_result ^= hi_hash_dword >> i;
1460         }

1462         return hash_result & IXGBE_ATR_HASH_MASK;
1283 }

1285 /*
1286  * These defines allow us to quickly generate all of the necessary instructions
1287  * in the function below by simply calling out IXGBE_COMPUTE_SIG_HASH_ITERATION
1288  * for values 0 through 15
1289  */
1290 #define IXGBE_ATR_COMMON_HASH_KEY \
1291                 (IXGBE_ATR_BUCKET_HASH_KEY & IXGBE_ATR_SIGNATURE_HASH_KEY)
1292 #define IXGBE_COMPUTE_SIG_HASH_ITERATION(_n) \
1293 do { \
1473 { \
1294         u32 n = (_n); \
1295         if (IXGBE_ATR_COMMON_HASH_KEY & (0x01 << n)) \
1296                 common_hash ^= lo_hash_dword >> n; \
1297         else if (IXGBE_ATR_BUCKET_HASH_KEY & (0x01 << n)) \
1298                 bucket_hash ^= lo_hash_dword >> n; \
1299         else if (IXGBE_ATR_SIGNATURE_HASH_KEY & (0x01 << n)) \
1300                 sig_hash ^= lo_hash_dword << (16 - n); \
1301         if (IXGBE_ATR_COMMON_HASH_KEY & (0x01 << (n + 16))) \
1302                 common_hash ^= hi_hash_dword >> n; \
1303         else if (IXGBE_ATR_BUCKET_HASH_KEY & (0x01 << (n + 16))) \
1304                 bucket_hash ^= hi_hash_dword >> n; \
1305         else if (IXGBE_ATR_SIGNATURE_HASH_KEY & (0x01 << (n + 16))) \
1306                 sig_hash ^= hi_hash_dword << (16 - n); \
1307 } while (0);
1487 }

1309 /**
1310  *  ixgbe_atr_compute_sig_hash_82599 - Compute the signature hash
1311  *  @stream: input bitstream to compute the hash on
1312  *
```



```
1313  *  This function is almost identical to the function above but contains
1314  *  several optomizations such as unwinding all of the loops, letting the
1315  *  compiler work out all of the conditional ifs since the keys are static
1316  *  defines, and computing two keys at once since the hashed dword stream
1317  *  will be the same for both keys.
1318  **/
1319 u32 ixgbe_atr_compute_sig_hash_82599(union ixgbe_atr_hash_dword input,
1499 static u32 ixgbe_atr_compute_sig_hash_82599(union ixgbe_atr_hash_dword input,
1320                                             union ixgbe_atr_hash_dword common)
1321 {
1322         u32 hi_hash_dword, lo_hash_dword, flow_vm_vlan;
1323         u32 sig_hash = 0, bucket_hash = 0, common_hash = 0;

1325         /* record the flow_vm_vlan bits as they are a key part to the hash */
1326         flow_vm_vlan = IXGBE_NTOHL(input.dword);

1328         /* generate common hash dword */
1329         hi_hash_dword = IXGBE_NTOHL(common.dword);

1331         /* low dword is word swapped version of common */
1332         lo_hash_dword = (hi_hash_dword >> 16) | (hi_hash_dword << 16);

1334         /* apply flow ID/VM pool/VLAN ID bits to hash words */
1335         hi_hash_dword ^= flow_vm_vlan ^ (flow_vm_vlan >> 16);

1337         /* Process bits 0 and 16 */
1338         IXGBE_COMPUTE_SIG_HASH_ITERATION(0);

1340         /*
1341          * apply flow ID/VM pool/VLAN ID bits to lo hash dword, we had to
1342          * delay this because bit 0 of the stream should not be processed
1343          * so we do not add the vlan until after bit 0 was processed
1344          */
1345         lo_hash_dword ^= flow_vm_vlan ^ (flow_vm_vlan << 16);

1347         /* Process remaining 30 bit of the key */
1348         IXGBE_COMPUTE_SIG_HASH_ITERATION(1);
1349         IXGBE_COMPUTE_SIG_HASH_ITERATION(2);
1350         IXGBE_COMPUTE_SIG_HASH_ITERATION(3);
1351         IXGBE_COMPUTE_SIG_HASH_ITERATION(4);
1352         IXGBE_COMPUTE_SIG_HASH_ITERATION(5);
1353         IXGBE_COMPUTE_SIG_HASH_ITERATION(6);
1354         IXGBE_COMPUTE_SIG_HASH_ITERATION(7);
1355         IXGBE_COMPUTE_SIG_HASH_ITERATION(8);
1356         IXGBE_COMPUTE_SIG_HASH_ITERATION(9);
1357         IXGBE_COMPUTE_SIG_HASH_ITERATION(10);
1358         IXGBE_COMPUTE_SIG_HASH_ITERATION(11);
1359         IXGBE_COMPUTE_SIG_HASH_ITERATION(12);
1360         IXGBE_COMPUTE_SIG_HASH_ITERATION(13);
1361         IXGBE_COMPUTE_SIG_HASH_ITERATION(14);
1362         IXGBE_COMPUTE_SIG_HASH_ITERATION(15);

1364         /* combine common_hash result with signature and bucket hashes */
1365         bucket_hash ^= common_hash;
1366         bucket_hash &= IXGBE_ATR_HASH_MASK;

1368         sig_hash ^= common_hash << 16;
1369         sig_hash &= IXGBE_ATR_HASH_MASK << 16;

1371         /* return completed signature hash */
1372         return sig_hash ^ bucket_hash;
1373 }

1375 /**
1376  *  ixgbe_atr_add_signature_filter_82599 - Adds a signature hash filter
1377  *  @hw: pointer to hardware structure
```

```
 *  @input: unique input dword
 *  @common: compressed common input dword
 *  @stream: input bitstream
 *  @queue: queue index to direct traffic to
 **/
s32 ixgbe_fdir_add_signature_filter_82599(struct ixgbe_hw *hw,
                                          union ixgbe_atr_hash_dword input,
                                          union ixgbe_atr_hash_dword common,
                                          u8 queue)
{
        u64  fdirhashcmd;
        u32  fdircmd;

        DEBUGFUNC("ixgbe_fdir_add_signature_filter_82599");

        /*
         * Get the flow_type in order to program FDIRCMD properly
         * lowest 2 bits are FDIRCMD.L4TYPE, third lowest bit is FDIRCMD.IPV6
         */
        switch (input.formatted.flow_type) {
        case IXGBE_ATR_FLOW_TYPE_TCPV4:
        case IXGBE_ATR_FLOW_TYPE_UDPV4:
        case IXGBE_ATR_FLOW_TYPE_SCTPV4:
        case IXGBE_ATR_FLOW_TYPE_TCPV6:
        case IXGBE_ATR_FLOW_TYPE_UDPV6:
        case IXGBE_ATR_FLOW_TYPE_SCTPV6:
                break;
        default:
                DEBUGOUT(" Error on flow type input\n");
                return IXGBE_ERR_CONFIG;
        }

        /* configure FDIRCMD register */
        fdircmd = IXGBE_FDIRCMD_CMD_ADD_FLOW | IXGBE_FDIRCMD_FILTER_UPDATE |
                  IXGBE_FDIRCMD_LAST | IXGBE_FDIRCMD_QUEUE_EN;
        fdircmd |= input.formatted.flow_type << IXGBE_FDIRCMD_FLOW_TYPE_SHIFT;
        fdircmd |= (u32)queue << IXGBE_FDIRCMD_RX_QUEUE_SHIFT;

        /*
         * The lower 32-bits of fdirhashcmd is for FDIRHASH, the upper 32-bits
         * is for FDIRCMD.  Then do a 64-bit register write from FDIRHASH.
         */
        fdirhashcmd = (u64)fdircmd << 32;
        fdirhashcmd |= ixgbe_atr_compute_sig_hash_82599(input, common);
        IXGBE_WRITE_REG64(hw, IXGBE_FDIRHASH, fdirhashcmd);

        DEBUGOUT2("Tx Queue=%x hash=%x\n", queue, (u32)fdirhashcmd);

        return IXGBE_SUCCESS;
}

#define IXGBE_COMPUTE_BKT_HASH_ITERATION(_n) \
do { \
        u32 n = (_n); \
        if (IXGBE_ATR_BUCKET_HASH_KEY & (0x01 << n)) \
                bucket_hash ^= lo_hash_dword >> n; \
        if (IXGBE_ATR_BUCKET_HASH_KEY & (0x01 << (n + 16))) \
                bucket_hash ^= hi_hash_dword >> n; \
} while (0);

/**
 *  ixgbe_atr_compute_perfect_hash_82599 - Compute the perfect filter hash
 *  @atr_input: input bitstream to compute the hash on
 *  @input_mask: mask for the input bitstream
 *
 *  This function serves two main purposes.  First it applys the input_mask
```



```
 *  to the atr_input resulting in a cleaned up atr_input data stream.
 *  Secondly it computes the hash and stores it in the bkt_hash field at
 *  the end of the input byte stream.  This way it will be available for
 *  future use without needing to recompute the hash.
 **/
void ixgbe_atr_compute_perfect_hash_82599(union ixgbe_atr_input *input,
                                          union ixgbe_atr_input *input_mask)
{

        u32 hi_hash_dword, lo_hash_dword, flow_vm_vlan;
        u32 bucket_hash = 0;

        /* Apply masks to input data */
        input->dword_stream[0]  &= input_mask->dword_stream[0];
        input->dword_stream[1]  &= input_mask->dword_stream[1];
        input->dword_stream[2]  &= input_mask->dword_stream[2];
        input->dword_stream[3]  &= input_mask->dword_stream[3];
        input->dword_stream[4]  &= input_mask->dword_stream[4];
        input->dword_stream[5]  &= input_mask->dword_stream[5];
        input->dword_stream[6]  &= input_mask->dword_stream[6];
        input->dword_stream[7]  &= input_mask->dword_stream[7];
        input->dword_stream[8]  &= input_mask->dword_stream[8];
        input->dword_stream[9]  &= input_mask->dword_stream[9];
        input->dword_stream[10] &= input_mask->dword_stream[10];

        /* record the flow_vm_vlan bits as they are a key part to the hash */
        flow_vm_vlan = IXGBE_NTOHL(input->dword_stream[0]);

        /* generate common hash dword */
        hi_hash_dword = IXGBE_NTOHL(input->dword_stream[1] ^
                                    input->dword_stream[2] ^
                                    input->dword_stream[3] ^
                                    input->dword_stream[4] ^
                                    input->dword_stream[5] ^
                                    input->dword_stream[6] ^
                                    input->dword_stream[7] ^
                                    input->dword_stream[8] ^
                                    input->dword_stream[9] ^
                                    input->dword_stream[10]);

        /* low dword is word swapped version of common */
        lo_hash_dword = (hi_hash_dword >> 16) | (hi_hash_dword << 16);

        /* apply flow ID/VM pool/VLAN ID bits to hash words */
        hi_hash_dword ^= flow_vm_vlan ^ (flow_vm_vlan >> 16);

        /* Process bits 0 and 16 */
        IXGBE_COMPUTE_BKT_HASH_ITERATION(0);

        /*
         * apply flow ID/VM pool/VLAN ID bits to lo hash dword, we had to
         * delay this because bit 0 of the stream should not be processed
         * so we do not add the vlan until after bit 0 was processed
         */
        lo_hash_dword ^= flow_vm_vlan ^ (flow_vm_vlan << 16);

        /* Process remaining 30 bit of the key */
        IXGBE_COMPUTE_BKT_HASH_ITERATION(1);
        IXGBE_COMPUTE_BKT_HASH_ITERATION(2);
        IXGBE_COMPUTE_BKT_HASH_ITERATION(3);
        IXGBE_COMPUTE_BKT_HASH_ITERATION(4);
        IXGBE_COMPUTE_BKT_HASH_ITERATION(5);
        IXGBE_COMPUTE_BKT_HASH_ITERATION(6);
        IXGBE_COMPUTE_BKT_HASH_ITERATION(7);
        IXGBE_COMPUTE_BKT_HASH_ITERATION(8);
        IXGBE_COMPUTE_BKT_HASH_ITERATION(9);
```

```
1509         IXGBE_COMPUTE_BKT_HASH_ITERATION(10);
1510         IXGBE_COMPUTE_BKT_HASH_ITERATION(11);
1511         IXGBE_COMPUTE_BKT_HASH_ITERATION(12);
1512         IXGBE_COMPUTE_BKT_HASH_ITERATION(13);
1513         IXGBE_COMPUTE_BKT_HASH_ITERATION(14);
1514         IXGBE_COMPUTE_BKT_HASH_ITERATION(15);

1516         /*
1517          * Limit hash to 13 bits since max bucket count is 8K.
1518          * Store result at the end of the input stream.
1519          */
1520         input->formatted.bkt_hash = bucket_hash & 0x1FFF;
1521 }

1523 /**
1524  *  ixgbe_get_fdirtcpm_82599 - generate a tcp port from atr_input_masks
1525  *  @input_mask: mask to be bit swapped
1526  *
1527  *  The source and destination port masks for flow director are bit swapped
1528  *  in that bit 15 effects bit 0, 14 effects 1, 13, 2 etc.  In order to
1529  *  generate a correctly swapped value we need to bit swap the mask and that
1530  *  is what is accomplished by this function.
1531  **/
1532 static u32 ixgbe_get_fdirtcpm_82599(union ixgbe_atr_input *input_mask)
1616 static u32 ixgbe_get_fdirtcpm_82599(struct ixgbe_atr_input_masks *input_masks)
1533 {
1534         u32 mask = IXGBE_NTOHS(input_mask->formatted.dst_port);
1618         u32 mask = IXGBE_NTOHS(input_masks->dst_port_mask);
1535         mask <<= IXGBE_FDIRTCPM_DPORTM_SHIFT;
1536         mask |= IXGBE_NTOHS(input_mask->formatted.src_port);
1620         mask |= IXGBE_NTOHS(input_masks->src_port_mask);
1537         mask = ((mask & 0x55555555) << 1) | ((mask & 0xAAAAAAAA) >> 1);
1538         mask = ((mask & 0x33333333) << 2) | ((mask & 0xCCCCCCCC) >> 2);
1539         mask = ((mask & 0x0F0F0F0F) << 4) | ((mask & 0xF0F0F0F0) >> 4);
1540         return ((mask & 0x00FF00FF) << 8) | ((mask & 0xFF00FF00) >> 8);
1541 }

1543 /*
1544  * These two macros are meant to address the fact that we have registers
1545  * that are either all or in part big-endian.  As a result on big-endian
1546  * systems we will end up byte swapping the value to little-endian before
1547  * it is byte swapped again and written to the hardware in the original
1548  * big-endian format.
1549  */
1550 #define IXGBE_STORE_AS_BE32(_value) \
1551         (((u32)(_value) >> 24) | (((u32)(_value) & 0x00FF0000) >> 8) | \
1552          (((u32)(_value) & 0x0000FF00) << 8) | ((u32)(_value) << 24))

1554 #define IXGBE_WRITE_REG_BE32(a, reg, value) \
1555         IXGBE_WRITE_REG((a), (reg), IXGBE_STORE_AS_BE32(IXGBE_NTOHL(value)))

1557 #define IXGBE_STORE_AS_BE16(_value) \
1558         IXGBE_NTOHS(((u16)(_value) >> 8) | ((u16)(_value) << 8))
1642         (((u16)(_value) >> 8) | ((u16)(_value) << 8))

1560 s32 ixgbe_fdir_set_input_mask_82599(struct ixgbe_hw *hw,
1561                                     union ixgbe_atr_input *input_mask)

1645 /**
1646  *  ixgbe_fdir_add_perfect_filter_82599 - Adds a perfect filter
1647  *  @hw: pointer to hardware structure
1648  *  @input: input bitstream
1649  *  @input_masks: masks for the input bitstream
1650  *  @soft_id: software index for the filters
1651  *  @queue: queue index to direct traffic to
1652  *
```



```
1653  *  Note that the caller to this function must lock before calling, since the
1654  *  hardware writes must be protected from one another.
1655  **/
1656 s32 ixgbe_fdir_add_perfect_filter_82599(struct ixgbe_hw *hw,
1657                                         union ixgbe_atr_input *input,
1658                                         struct ixgbe_atr_input_masks *input_masks,
1659                                         u16 soft_id, u8 queue)
1562 {
1563         /* mask IPv6 since it is currently not supported */
1564         u32 fdirm = IXGBE_FDIRM_DIPv6;
1565         u32 fdirtcpm;
1661         u32 fdirhash;
1662         u32 fdircmd;
1663         u32 fdirport, fdirtcpm;
1664         u32 fdirvlan;
1665         /* start with VLAN, flex bytes, VM pool, and IPv6 destination masked */
1666         u32 fdirm = IXGBE_FDIRM_VLANID | IXGBE_FDIRM_VLANP | IXGBE_FDIRM_FLEX |
1667                     IXGBE_FDIRM_POOL | IXGBE_FDIRM_DIPv6;

1567         DEBUGFUNC("ixgbe_fdir_set_atr_input_mask_82599");
1669         DEBUGFUNC("ixgbe_fdir_add_perfect_filter_82599");

1569         /*
1672          * Check flow_type formatting, and bail out before we touch the hardware
1673          * if there's a configuration issue
1674          */
1675         switch (input->formatted.flow_type) {
1676         case IXGBE_ATR_FLOW_TYPE_IPV4:
1677                 /* use the L4 protocol mask for raw IPv4/IPv6 traffic */
1678                 fdirm |= IXGBE_FDIRM_L4P;
1679                 /* FALLTHRU */
1680         case IXGBE_ATR_FLOW_TYPE_SCTPV4:
1681                 if (input_masks->dst_port_mask || input_masks->src_port_mask) {
1682                         DEBUGOUT(" Error on src/dst port mask\n");
1683                         return IXGBE_ERR_CONFIG;
1684                 }
1685                 break;
1686         case IXGBE_ATR_FLOW_TYPE_TCPV4:
1687                 break;
1688         case IXGBE_ATR_FLOW_TYPE_UDPV4:
1689                 break;
1690         default:
1691                 DEBUGOUT(" Error on flow type input\n");
1692                 return IXGBE_ERR_CONFIG;
1693         }

1695         /*
1570          * Program the relevant mask registers.  If src/dst_port or src/dst_addr
1571          * are zero, then assume a full mask for that field.  Also assume that
1572          * a VLAN of 0 is unspecified, so mask that out as well.  L4type
1573          * cannot be masked out in this implementation.
1574          *
1575          * This also assumes IPv4 only.  IPv6 masking isn't supported at this
1576          * point in time.
1577          */

1579         /* verify bucket hash is cleared on hash generation */
1580         if (input_mask->formatted.bkt_hash)
1581                 DEBUGOUT(" bucket hash should always be 0 in mask\n");

1583         /* Program FDIRM and verify partial masks */
1584         switch (input_mask->formatted.vm_pool & 0x7F) {
1585         case 0x0:
1586                 fdirm |= IXGBE_FDIRM_POOL;
1587         case 0x7F:
1705         /* Program FDIRM */
```

```
1706         switch (IXGBE_NTOHS(input_masks->vlan_id_mask) & 0xEFFF) {
1707         case 0xEFFF:
1708                 /* Unmask VLAN ID - bit 0 and fall through to unmask prio */
1709                 fdirm &= ~IXGBE_FDIRM_VLANID;
1710                 /* FALLTHRU */
1711         case 0xE000:
1712                 /* Unmask VLAN prio - bit 1 */
1713                 fdirm &= ~IXGBE_FDIRM_VLANP;
1588                 break;
1589         default:
1590                 DEBUGOUT(" Error on vm pool mask\n");
1591                 return IXGBE_ERR_CONFIG;
1592         }

1594         switch (input_mask->formatted.flow_type & IXGBE_ATR_L4TYPE_MASK) {
1595         case 0x0:
1596                 fdirm |= IXGBE_FDIRM_L4P;
1597                 if (input_mask->formatted.dst_port ||
1598                     input_mask->formatted.src_port) {
1599                         DEBUGOUT(" Error on src/dst port mask\n");
1600                         return IXGBE_ERR_CONFIG;
1601                 }
1602         case IXGBE_ATR_L4TYPE_MASK:
1715         case 0x0FFF:
1716                 /* Unmask VLAN ID - bit 0 */
1717                 fdirm &= ~IXGBE_FDIRM_VLANID;
1603                 break;
1604         default:
1605                 DEBUGOUT(" Error on flow type mask\n");
1606                 return IXGBE_ERR_CONFIG;
1607         }

1609         switch (IXGBE_NTOHS(input_mask->formatted.vlan_id) & 0xEFFF) {
1610         case 0x0000:
1611                 /* mask VLAN ID, fall through to mask VLAN priority */
1612                 fdirm |= IXGBE_FDIRM_VLANID;
1613         case 0x0FFF:
1614                 /* mask VLAN priority */
1615                 fdirm |= IXGBE_FDIRM_VLANP;
1720                 /* do nothing, vlans already masked */
1616                 break;
1617         case 0xE000:
1618                 /* mask VLAN ID only, fall through */
1619                 fdirm |= IXGBE_FDIRM_VLANID;
1620         case 0xEFFF:
1621                 /* no VLAN fields masked */
1622                 break;
1623         default:
1624                 DEBUGOUT(" Error on VLAN mask\n");
1625                 return IXGBE_ERR_CONFIG;
1626         }

1628         switch (input_mask->formatted.flex_bytes & 0xFFFF) {
1629         case 0x0000:
1630                 /* Mask Flex Bytes, fall through */
1631                 fdirm |= IXGBE_FDIRM_FLEX;
1632         case 0xFFFF:
1633                 break;
1634         default:
1727         if (input_masks->flex_mask & 0xFFFF) {
1728                 if ((input_masks->flex_mask & 0xFFFF) != 0xFFFF) {
1635                 DEBUGOUT(" Error on flexible byte mask\n");
1636                 return IXGBE_ERR_CONFIG;
1637         }
1732                 /* Unmask Flex Bytes - bit 4 */
1733                 fdirm &= ~IXGBE_FDIRM_FLEX;
```



```
1734         }

1639         /* Now mask VM pool and destination IPv6 - bits 5 and 2 */
1640         IXGBE_WRITE_REG(hw, IXGBE_FDIRM, fdirm);

1642         /* store the TCP/UDP port masks, bit reversed from port layout */
1643         fdirtcpm = ixgbe_get_fdirtcpm_82599(input_mask);
1740         fdirtcpm = ixgbe_get_fdirtcpm_82599(input_masks);

1645         /* write both the same so that UDP and TCP use the same mask */
1646         IXGBE_WRITE_REG(hw, IXGBE_FDIRTCPM, ~fdirtcpm);
1647         IXGBE_WRITE_REG(hw, IXGBE_FDIRUDPM, ~fdirtcpm);

1649         /* store source and destination IP masks (big-enian) */
1650         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIP4M,
1651                              ~input_mask->formatted.src_ip[0]);
1748                              ~input_masks->src_ip_mask[0]);
1652         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRDIP4M,
1653                              ~input_mask->formatted.dst_ip[0]);
1750                              ~input_masks->dst_ip_mask[0]);

1655         return IXGBE_SUCCESS;
1656 }
1752         /* Apply masks to input data */
1753         input->formatted.vlan_id &= input_masks->vlan_id_mask;
1754         input->formatted.flex_bytes &= input_masks->flex_mask;
1755         input->formatted.src_port &= input_masks->src_port_mask;
1756         input->formatted.dst_port &= input_masks->dst_port_mask;
1757         input->formatted.src_ip[0] &= input_masks->src_ip_mask[0];
1758         input->formatted.dst_ip[0] &= input_masks->dst_ip_mask[0];

1658 s32 ixgbe_fdir_write_perfect_filter_82599(struct ixgbe_hw *hw,
1659                                           union ixgbe_atr_input *input,
1660                                           u16 soft_id, u8 queue)
1661 {
1662         u32 fdirport, fdirvlan, fdirhash, fdircmd;
1760         /* record vlan (little-endian) and flex_bytes(big-endian) */
1761         fdirvlan =
1762                 IXGBE_STORE_AS_BE16(IXGBE_NTOHS(input->formatted.flex_bytes));
1763         fdirvlan <<= IXGBE_FDIRVLAN_FLEX_SHIFT;
1764         fdirvlan |= IXGBE_NTOHS(input->formatted.vlan_id);
1765         IXGBE_WRITE_REG(hw, IXGBE_FDIRVLAN, fdirvlan);

1664         DEBUGFUNC("ixgbe_fdir_write_perfect_filter_82599");

1666         /* currently IPv6 is not supported, must be programmed with 0 */
1667         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIPv6(0),
1668                              input->formatted.src_ip[0]);
1669         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIPv6(1),
1670                              input->formatted.src_ip[1]);
1671         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIPv6(2),
1672                              input->formatted.src_ip[2]);

1674         /* record the source address (big-endian) */
1675         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRIPSA, input->formatted.src_ip[0]);

1677         /* record the first 32 bits of the destination address (big-endian) */
1678         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRIPDA, input->formatted.dst_ip[0]);

1680         /* record source and destination port (little-endian)*/
1681         fdirport = IXGBE_NTOHS(input->formatted.dst_port);
1682         fdirport <<= IXGBE_FDIRPORT_DESTINATION_SHIFT;
1683         fdirport |= IXGBE_NTOHS(input->formatted.src_port);
1684         IXGBE_WRITE_REG(hw, IXGBE_FDIRPORT, fdirport);

1686         /* record vlan (little-endian) and flex_bytes(big-endian) */
```

```
1687         fdirvlan = IXGBE_STORE_AS_BE16(input->formatted.flex_bytes);
1688         fdirvlan <<= IXGBE_FDIRVLAN_FLEX_SHIFT;
1689         fdirvlan |= IXGBE_NTOHS(input->formatted.vlan_id);
1690         IXGBE_WRITE_REG(hw, IXGBE_FDIRVLAN, fdirvlan);
1773         /* record the first 32 bits of the destination address (big-endian) */
1774         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRIPDA, input->formatted.dst_ip[0]);

1692         /* configure FDIRHASH register */
1693         fdirhash = input->formatted.bkt_hash;
1694         fdirhash |= soft_id << IXGBE_FDIRHASH_SIG_SW_INDEX_SHIFT;
1695         IXGBE_WRITE_REG(hw, IXGBE_FDIRHASH, fdirhash);
1776         /* record the source address (big-endian) */
1777         IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRIPSA, input->formatted.src_ip[0]);

1697         /*
1698          * flush all previous writes to make certain registers are
1699          * programmed prior to issuing the command
1700          */
1701         IXGBE_WRITE_FLUSH(hw);

1703         /* configure FDIRCMD register */
1704         fdircmd = IXGBE_FDIRCMD_CMD_ADD_FLOW | IXGBE_FDIRCMD_FILTER_UPDATE |
1705                   IXGBE_FDIRCMD_LAST | IXGBE_FDIRCMD_QUEUE_EN;
1706         if (queue == IXGBE_FDIR_DROP_QUEUE)
1707                 fdircmd |= IXGBE_FDIRCMD_DROP;
1708         fdircmd |= input->formatted.flow_type << IXGBE_FDIRCMD_FLOW_TYPE_SHIFT;
1709         fdircmd |= (u32)queue << IXGBE_FDIRCMD_RX_QUEUE_SHIFT;
1710         fdircmd |= (u32)input->formatted.vm_pool << IXGBE_FDIRCMD_VT_POOL_SHIFT;

1712         IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD, fdircmd);

1714         return IXGBE_SUCCESS;
1715 }

1717 s32 ixgbe_fdir_erase_perfect_filter_82599(struct ixgbe_hw *hw,
1718                                           union ixgbe_atr_input *input,
1719                                           u16 soft_id)
1720 {
1721         u32 fdirhash;
1722         u32 fdircmd = 0;
1723         u32 retry_count;
1724         s32 err = IXGBE_SUCCESS;

1726         /* configure FDIRHASH register */
1727         fdirhash = input->formatted.bkt_hash;
1785         /* we only want the bucket hash so drop the upper 16 bits */
1786         fdirhash = ixgbe_atr_compute_hash_82599(input,
1787                                                 IXGBE_ATR_BUCKET_HASH_KEY);
1728         fdirhash |= soft_id << IXGBE_FDIRHASH_SIG_SW_INDEX_SHIFT;
1729         IXGBE_WRITE_REG(hw, IXGBE_FDIRHASH, fdirhash);

1731         /* flush hash to HW */
1732         IXGBE_WRITE_FLUSH(hw);

1734         /* Query if filter is present */
1735         IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD, IXGBE_FDIRCMD_CMD_QUERY_REM_FILT);

1737         for (retry_count = 10; retry_count; retry_count--) {
1738                 /* allow 10us for query to process */
1739                 usec_delay(10);
1740                 /* verify query completed successfully */
1741                 fdircmd = IXGBE_READ_REG(hw, IXGBE_FDIRCMD);
1742                 if (!(fdircmd & IXGBE_FDIRCMD_CMD_MASK))
1743                         break;
1744         }
```



```
1746         if (!retry_count)
1747                 err = IXGBE_ERR_FDIR_REINIT_FAILED;

1749         /* if filter exists in hardware then remove it */
1750         if (fdircmd & IXGBE_FDIRCMD_FILTER_VALID) {
1751                 IXGBE_WRITE_REG(hw, IXGBE_FDIRHASH, fdirhash);
1752                 IXGBE_WRITE_FLUSH(hw);
1753                 IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD,
1754                                 IXGBE_FDIRCMD_CMD_REMOVE_FLOW);
1755         }
1791         IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD, fdircmd);

1757         return err;
1793         return IXGBE_SUCCESS;
1758 }

1760 /**
1761  *  ixgbe_fdir_add_perfect_filter_82599 - Adds a perfect filter
1762  *  @hw: pointer to hardware structure
1763  *  @input: input bitstream
1764  *  @input_mask: mask for the input bitstream
1765  *  @soft_id: software index for the filters
1766  *  @queue: queue index to direct traffic to
1767  *
1768  *  Note that the caller to this function must lock before calling, since the
1769  *  hardware writes must be protected from one another.
1770  **/
1771 s32 ixgbe_fdir_add_perfect_filter_82599(struct ixgbe_hw *hw,
1772                                         union ixgbe_atr_input *input,
1773                                         union ixgbe_atr_input *input_mask,
1774                                         u16 soft_id, u8 queue)
1775 {
1776         s32 err = IXGBE_ERR_CONFIG;

1778         DEBUGFUNC("ixgbe_fdir_add_perfect_filter_82599");

1780         /*
1781          * Check flow_type formatting, and bail out before we touch the hardware
1782          * if there's a configuration issue
1783          */
1784         switch (input->formatted.flow_type) {
1785         case IXGBE_ATR_FLOW_TYPE_IPV4:
1786                 input_mask->formatted.flow_type = IXGBE_ATR_L4TYPE_IPV6_MASK;
1787                 if (input->formatted.dst_port || input->formatted.src_port) {
1788                         DEBUGOUT(" Error on src/dst port\n");
1789                         return IXGBE_ERR_CONFIG;
1790                 }
1791                 break;
1792         case IXGBE_ATR_FLOW_TYPE_SCTPV4:
1793                 if (input->formatted.dst_port || input->formatted.src_port) {
1794                         DEBUGOUT(" Error on src/dst port\n");
1795                         return IXGBE_ERR_CONFIG;
1796                 }
1797         case IXGBE_ATR_FLOW_TYPE_TCPV4:
1798         case IXGBE_ATR_FLOW_TYPE_UDPV4:
1799                 input_mask->formatted.flow_type = IXGBE_ATR_L4TYPE_IPV6_MASK |
1800                                                   IXGBE_ATR_L4TYPE_MASK;
1801                 break;
1802         default:
1803                 DEBUGOUT(" Error on flow type input\n");
1804                 return err;
1805         }

1807         /* program input mask into the HW */
1808         err = ixgbe_fdir_set_input_mask_82599(hw, input_mask);
1809         if (err)
```

```
1810                 return err;

1812         /* apply mask and compute/store hash */
1813         ixgbe_atr_compute_perfect_hash_82599(input, input_mask);

1815         /* program filters to filter memory */
1816         return ixgbe_fdir_write_perfect_filter_82599(hw, input,
1817                                                      soft_id, queue);
1818 }

1820 /**
1821  *  ixgbe_read_analog_reg8_82599 - Reads 8 bit Omer analog register
1822  *  @hw: pointer to hardware structure
1823  *  @reg: analog register to read
1824  *  @val: read value
1825  *
1826  *  Performs read operation to Omer analog register specified.
1827  **/
1828 s32 ixgbe_read_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 *val)
1829 {
1830         u32  core_ctl;

1832         DEBUGFUNC("ixgbe_read_analog_reg8_82599");

1834         IXGBE_WRITE_REG(hw, IXGBE_CORECTL, IXGBE_CORECTL_WRITE_CMD |
1835                         (reg << 8));
1836         IXGBE_WRITE_FLUSH(hw);
1837         usec_delay(10);
1838         core_ctl = IXGBE_READ_REG(hw, IXGBE_CORECTL);
1839         *val = (u8)core_ctl;

1841         return IXGBE_SUCCESS;
1842 }
______unchanged_portion_omitted_

1866 /**
1867  *  ixgbe_start_hw_82599 - Prepare hardware for Tx/Rx
1843  *  ixgbe_start_hw_rev_1_82599 - Prepare hardware for Tx/Rx
1868  *  @hw: pointer to hardware structure
1869  *
1870  *  Starts the hardware using the generic start_hw function
1871  *  and the generation start_hw function.
1872  *  Then performs revision-specific operations, if any.
1873  **/
1874 s32 ixgbe_start_hw_82599(struct ixgbe_hw *hw)
1850 s32 ixgbe_start_hw_rev_1_82599(struct ixgbe_hw *hw)
1875 {
1876         s32 ret_val = IXGBE_SUCCESS;

1878         DEBUGFUNC("ixgbe_start_hw_82599");
1854         DEBUGFUNC("ixgbe_start_hw_rev_1__82599");

1880         ret_val = ixgbe_start_hw_generic(hw);
1881         if (ret_val != IXGBE_SUCCESS)
1882                 goto out;

1884         ret_val = ixgbe_start_hw_gen2(hw);
1885         if (ret_val != IXGBE_SUCCESS)
1886                 goto out;

1888         /* We need to run link autotry after the driver loads */
1889         hw->mac.autotry_restart = TRUE;

1891         if (ret_val == IXGBE_SUCCESS)
1892                 ret_val = ixgbe_verify_fw_version_82599(hw);
1893 out:
```



```
1894         return ret_val;
1895 }

1897 /**
1898  *  ixgbe_identify_phy_82599 - Get physical layer module
1899  *  @hw: pointer to hardware structure
1900  *
1901  *  Determines the physical layer module found on the current adapter.
1902  *  If PHY already detected, maintains current PHY type in hw struct,
1903  *  otherwise executes the PHY detection routine.
1904  **/
1905 s32 ixgbe_identify_phy_82599(struct ixgbe_hw *hw)
1906 {
1907         s32 status = IXGBE_ERR_PHY_ADDR_INVALID;

1909         DEBUGFUNC("ixgbe_identify_phy_82599");

1911         /* Detect PHY if not unknown - returns success if already detected. */
1912         status = ixgbe_identify_phy_generic(hw);
1913         if (status != IXGBE_SUCCESS) {
1914                 /* 82599 10GBASE-T requires an external PHY */
1915                 if (hw->mac.ops.get_media_type(hw) == ixgbe_media_type_copper)
1916                         goto out;
1917                 else
1918                         status = ixgbe_identify_module_generic(hw);
1894                         status = ixgbe_identify_sfp_module_generic(hw);
1919         }

1921         /* Set PHY type none if no PHY detected */
1922         if (hw->phy.type == ixgbe_phy_unknown) {
1923                 hw->phy.type = ixgbe_phy_none;
1924                 status = IXGBE_SUCCESS;
1925         }

1927         /* Return error if SFP module has been detected but is not supported */
1928         if (hw->phy.type == ixgbe_phy_sfp_unsupported)
1929                 status = IXGBE_ERR_SFP_NOT_SUPPORTED;

1931 out:
1932         return status;
1933 }

1935 /**
1936  *  ixgbe_get_supported_physical_layer_82599 - Returns physical layer type
1937  *  @hw: pointer to hardware structure
1938  *
1939  *  Determines physical layer capabilities of the current configuration.
1940  **/
1941 u32 ixgbe_get_supported_physical_layer_82599(struct ixgbe_hw *hw)
1942 {
1943         u32 physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
1944         u32 autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
1945         u32 autoc2 = IXGBE_READ_REG(hw, IXGBE_AUTOC2);
1946         u32 pma_pmd_10g_serial = autoc2 & IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_MASK;
1947         u32 pma_pmd_10g_parallel = autoc & IXGBE_AUTOC_10G_PMA_PMD_MASK;
1948         u32 pma_pmd_1g = autoc & IXGBE_AUTOC_1G_PMA_PMD_MASK;
1949         u16 ext_ability = 0;
1950         u8 comp_codes_10g = 0;
1951         u8 comp_codes_1g = 0;

1953         DEBUGFUNC("ixgbe_get_support_physical_layer_82599");

1955         hw->phy.ops.identify(hw);

1957         switch (hw->phy.type) {
1958         case ixgbe_phy_tn:
```

```
1935         case ixgbe_phy_aq:
1959         case ixgbe_phy_cu_unknown:
1960                 hw->phy.ops.read_reg(hw, IXGBE_MDIO_PHY_EXT_ABILITY,
1961                 IXGBE_MDIO_PMA_PMD_DEV_TYPE, &ext_ability);
1962                 if (ext_ability & IXGBE_MDIO_PHY_10GBASET_ABILITY)
1963                         physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_T;
1964                 if (ext_ability & IXGBE_MDIO_PHY_1000BASET_ABILITY)
1965                         physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_T;
1966                 if (ext_ability & IXGBE_MDIO_PHY_100BASETX_ABILITY)
1967                         physical_layer |= IXGBE_PHYSICAL_LAYER_100BASE_TX;
1968                 goto out;
1969         default:
1970                 break;
1971         }

1973         switch (autoc & IXGBE_AUTOC_LMS_MASK) {
1974         case IXGBE_AUTOC_LMS_1G_AN:
1975         case IXGBE_AUTOC_LMS_1G_LINK_NO_AN:
1976                 if (pma_pmd_1g == IXGBE_AUTOC_1G_KX_BX) {
1977                         physical_layer = IXGBE_PHYSICAL_LAYER_1000BASE_KX |
1978                             IXGBE_PHYSICAL_LAYER_1000BASE_BX;
1979                         goto out;
1980                 }
1981                 /* SFI mode so read SFP module */
1982                 goto sfp_check;
1983         case IXGBE_AUTOC_LMS_10G_LINK_NO_AN:
1984                 if (pma_pmd_10g_parallel == IXGBE_AUTOC_10G_CX4)
1985                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_CX4;
1986                 else if (pma_pmd_10g_parallel == IXGBE_AUTOC_10G_KX4)
1987                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_KX4;
1988                 else if (pma_pmd_10g_parallel == IXGBE_AUTOC_10G_XAUI)
1989                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_XAUI;
1990                 goto out;
1991         case IXGBE_AUTOC_LMS_10G_SERIAL:
1992                 if (pma_pmd_10g_serial == IXGBE_AUTOC2_10G_KR) {
1993                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_KR;
1994                         goto out;
1995                 } else if (pma_pmd_10g_serial == IXGBE_AUTOC2_10G_SFI)
1996                         goto sfp_check;
1997                 break;
1998         case IXGBE_AUTOC_LMS_KX4_KX_KR:
1999         case IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN:
2000                 if (autoc & IXGBE_AUTOC_KX_SUPP)
2001                         physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_KX;
2002                 if (autoc & IXGBE_AUTOC_KX4_SUPP)
2003                         physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_KX4;
2004                 if (autoc & IXGBE_AUTOC_KR_SUPP)
2005                         physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_KR;
2006                 goto out;
2007         default:
2008                 goto out;
2009         }

2011 sfp_check:
2012         /* SFP check must be done last since DA modules are sometimes used to
2013          * test KR mode -  we need to id KR mode correctly before SFP module.
2014          * Call identify_sfp because the pluggable module may have changed */
2015         hw->phy.ops.identify_sfp(hw);
2016         if (hw->phy.sfp_type == ixgbe_sfp_type_not_present)
2017                 goto out;

2019         switch (hw->phy.type) {
2020         case ixgbe_phy_sfp_passive_tyco:
2021         case ixgbe_phy_sfp_passive_unknown:
2022                 physical_layer = IXGBE_PHYSICAL_LAYER_SFP_PLUS_CU;
2023                 break;
```



```
2024         case ixgbe_phy_sfp_ftl_active:
2025         case ixgbe_phy_sfp_active_unknown:
2026                 physical_layer = IXGBE_PHYSICAL_LAYER_SFP_ACTIVE_DA;
2027                 break;
2028         case ixgbe_phy_sfp_avago:
2029         case ixgbe_phy_sfp_ftl:
2030         case ixgbe_phy_sfp_intel:
2031         case ixgbe_phy_sfp_unknown:
2032                 hw->phy.ops.read_i2c_eeprom(hw,
2033                       IXGBE_SFF_1GBE_COMP_CODES, &comp_codes_1g);
2034                 hw->phy.ops.read_i2c_eeprom(hw,
2035                       IXGBE_SFF_10GBE_COMP_CODES, &comp_codes_10g);
2036                 if (comp_codes_10g & IXGBE_SFF_10GBASESR_CAPABLE)
2037                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_SR;
2038                 else if (comp_codes_10g & IXGBE_SFF_10GBASELR_CAPABLE)
2039                         physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_LR;
2040                 else if (comp_codes_1g & IXGBE_SFF_1GBASET_CAPABLE)
2041                         physical_layer = IXGBE_PHYSICAL_LAYER_1000BASE_T;
2042                 else if (comp_codes_1g & IXGBE_SFF_1GBASESX_CAPABLE)
2043                         physical_layer = IXGBE_PHYSICAL_LAYER_1000BASE_SX;
2044                 break;
2045         default:
2046                 break;
2047         }

2049 out:
2050         return physical_layer;
2051 }

2053 /**
2054  *  ixgbe_enable_rx_dma_82599 - Enable the Rx DMA unit on 82599
2055  *  @hw: pointer to hardware structure
2056  *  @regval: register value to write to RXCTRL
2057  *
2058  *  Enables the Rx DMA unit for 82599
2059  **/
2060 s32 ixgbe_enable_rx_dma_82599(struct ixgbe_hw *hw, u32 regval)
2061 {
2037 #define IXGBE_MAX_SECRX_POLL 30
2038         int i;
2039         int secrxreg;

2063         DEBUGFUNC("ixgbe_enable_rx_dma_82599");

2065         /*
2066          * Workaround for 82599 silicon errata when enabling the Rx datapath.
2067          * If traffic is incoming before we enable the Rx unit, it could hang
2068          * the Rx DMA unit.  Therefore, make sure the security engine is
2069          * completely disabled prior to enabling the Rx unit.
2070          */
2049         secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXCTRL);
2050         secrxreg |= IXGBE_SECRXCTRL_RX_DIS;
2051         IXGBE_WRITE_REG(hw, IXGBE_SECRXCTRL, secrxreg);
2052         for (i = 0; i < IXGBE_MAX_SECRX_POLL; i++) {
2053                 secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXSTAT);
2054                 if (secrxreg & IXGBE_SECRXSTAT_SECRX_RDY)
2055                         break;
2056                 else
2057                         /* Use interrupt-safe sleep just in case */
2058                         usec_delay(10);
2059         }

2072         hw->mac.ops.disable_sec_rx_path(hw);
2061         /* For informational purposes only */
2062         if (i >= IXGBE_MAX_SECRX_POLL)
2063                 DEBUGOUT("Rx unit being enabled before security "
```

```
                           "path fully disabled.  Continuing with init.\n");

        IXGBE_WRITE_REG(hw, IXGBE_RXCTRL, regval);
        secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXCTRL);
        secrxreg &= ~IXGBE_SECRXCTRL_RX_DIS;
        IXGBE_WRITE_REG(hw, IXGBE_SECRXCTRL, secrxreg);
        IXGBE_WRITE_FLUSH(hw);

        hw->mac.ops.enable_sec_rx_path(hw);

        return IXGBE_SUCCESS;
}

/**
 *  ixgbe_verify_fw_version_82599 - verify fw version for 82599
 *  @hw: pointer to hardware structure
 *
 *  Verifies that installed the firmware version is 0.6 or higher
 *  for SFI devices. All 82599 SFI devices should have version 0.6 or higher.
 *
 *  Returns IXGBE_ERR_EEPROM_VERSION if the FW is not present or
 *  if the FW version is not supported.
 **/
static s32 ixgbe_verify_fw_version_82599(struct ixgbe_hw *hw)
{
        s32 status = IXGBE_ERR_EEPROM_VERSION;
        u16 fw_offset, fw_ptp_cfg_offset;
        u16 fw_version = 0;

        DEBUGFUNC("ixgbe_verify_fw_version_82599");

        /* firmware check is only necessary for SFI devices */
        if (hw->phy.media_type != ixgbe_media_type_fiber) {
                status = IXGBE_SUCCESS;
                goto fw_version_out;
        }

        /* get the offset to the Firmware Module block */
        hw->eeprom.ops.read(hw, IXGBE_FW_PTR, &fw_offset);

        if ((fw_offset == 0) || (fw_offset == 0xFFFF))
                goto fw_version_out;

        /* get the offset to the Pass Through Patch Configuration block */
        hw->eeprom.ops.read(hw, (fw_offset +
                                 IXGBE_FW_PASSTHROUGH_PATCH_CONFIG_PTR),
                                 &fw_ptp_cfg_offset);

        if ((fw_ptp_cfg_offset == 0) || (fw_ptp_cfg_offset == 0xFFFF))
                goto fw_version_out;

        /* get the firmware version */
        hw->eeprom.ops.read(hw, (fw_ptp_cfg_offset +
                            IXGBE_FW_PATCH_VERSION_4), &fw_version);
                                 IXGBE_FW_PATCH_VERSION_4),
                                 &fw_version);

        if (fw_version > 0x5)
                status = IXGBE_SUCCESS;

fw_version_out:
        return status;
}
```

*unchanged_portion_omitted_*

```
/**
```



```
 *  ixgbe_read_eeprom_buffer_82599 - Read EEPROM word(s) using
 *  fastest available method
 *
 *  @hw: pointer to hardware structure
 *  @offset: offset of  word in EEPROM to read
 *  @words: number of words
 *  @data: word(s) read from the EEPROM
 *
 *  Retrieves 16 bit word(s) read from EEPROM
 **/
static s32 ixgbe_read_eeprom_buffer_82599(struct ixgbe_hw *hw, u16 offset,
                                          u16 words, u16 *data)
{
        struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
        s32 ret_val = IXGBE_ERR_CONFIG;

        DEBUGFUNC("ixgbe_read_eeprom_buffer_82599");

        /*
         * If EEPROM is detected and can be addressed using 14 bits,
         * use EERD otherwise use bit bang
         */
        if ((eeprom->type == ixgbe_eeprom_spi) &&
            (offset + (words - 1) <= IXGBE_EERD_MAX_ADDR))
                ret_val = ixgbe_read_eerd_buffer_generic(hw, offset, words,
                                                         data);
        else
                ret_val = ixgbe_read_eeprom_buffer_bit_bang_generic(hw, offset,
                                                                    words,
                                                                    data);

        return ret_val;
}

/**
 *  ixgbe_read_eeprom_82599 - Read EEPROM word using
 *  fastest available method
 *
 *  @hw: pointer to hardware structure
 *  @offset: offset of  word in the EEPROM to read
 *  @data: word read from the EEPROM
 *
 *  Reads a 16 bit word from the EEPROM
 **/
static s32 ixgbe_read_eeprom_82599(struct ixgbe_hw *hw,
                                   u16 offset, u16 *data)
{
        struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
        s32 ret_val = IXGBE_ERR_CONFIG;

        DEBUGFUNC("ixgbe_read_eeprom_82599");

        /*
         * If EEPROM is detected and can be addressed using 14 bits,
         * use EERD otherwise use bit bang
         */
        if ((eeprom->type == ixgbe_eeprom_spi) &&
            (offset <= IXGBE_EERD_MAX_ADDR))
                ret_val = ixgbe_read_eerd_generic(hw, offset, data);
        else
                ret_val = ixgbe_read_eeprom_bit_bang_generic(hw, offset, data);

        return ret_val;
}
```

**********
*3452 Thu Jul 12 12:22:30 2012*
*new/usr/src/uts/common/io/ixgbe/ixgbe_82599.h*
*XXXX Intel X540 support*
**********

```
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_82599.h,v 1.1 2012/01/30 16:42:02 jfv Exp $*

#ifndef _IXGBE_82599_H_
#define _IXGBE_82599_H_

s32 ixgbe_get_link_capabilities_82599(struct ixgbe_hw *hw,
                                      ixgbe_link_speed *speed, bool *autoneg);
enum ixgbe_media_type ixgbe_get_media_type_82599(struct ixgbe_hw *hw);
void ixgbe_disable_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
void ixgbe_enable_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
void ixgbe_flap_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
s32 ixgbe_setup_mac_link_multispeed_fiber(struct ixgbe_hw *hw,
                                          ixgbe_link_speed speed, bool autoneg,
                                          bool autoneg_wait_to_complete);
s32 ixgbe_setup_mac_link_smartspeed(struct ixgbe_hw *hw,
                                    ixgbe_link_speed speed, bool autoneg,
                                    bool autoneg_wait_to_complete);
s32 ixgbe_start_mac_link_82599(struct ixgbe_hw *hw,
                               bool autoneg_wait_to_complete);
s32 ixgbe_setup_mac_link_82599(struct ixgbe_hw *hw, ixgbe_link_speed speed,
                               bool autoneg, bool autoneg_wait_to_complete);
s32 ixgbe_setup_sfp_modules_82599(struct ixgbe_hw *hw);
void ixgbe_init_mac_link_ops_82599(struct ixgbe_hw *hw);
s32 ixgbe_reset_hw_82599(struct ixgbe_hw *hw);
s32 ixgbe_read_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 *val);
s32 ixgbe_write_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 val);
s32 ixgbe_start_hw_82599(struct ixgbe_hw *hw);
s32 ixgbe_identify_phy_82599(struct ixgbe_hw *hw);
s32 ixgbe_init_phy_ops_82599(struct ixgbe_hw *hw);
```



```
u32 ixgbe_get_supported_physical_layer_82599(struct ixgbe_hw *hw);
s32 ixgbe_enable_rx_dma_82599(struct ixgbe_hw *hw, u32 regval);
bool ixgbe_verify_lesm_fw_enabled_82599(struct ixgbe_hw *hw);
#endif /* _IXGBE_82599_H_ */
```

```
**********
   35393 Thu Jul 12 12:22:31 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_api.c
XXXX Intel X540 support
**********
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_api.c,v 1.13 2012/07/05 20:51:44 jfv Exp $*/
/*$FreeBSD$*/

#include "ixgbe_api.h"
#include "ixgbe_common.h"

extern s32 ixgbe_init_ops_82598(struct ixgbe_hw *hw);
extern s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw);

/**
 *  ixgbe_init_shared_code - Initialize the shared code
 *  @hw: pointer to hardware structure
 *
 *  This will assign function pointers and assign the MAC type and PHY code.
 *  Does not touch the hardware. This function must be called prior to any
 *  other function in the shared code. The ixgbe_hw structure should be
 *  memset to 0 prior to calling this function.  The following fields in
 *  hw structure should be filled in prior to calling this function:
 *  hw_addr, back, device_id, vendor_id, subsystem_device_id,
 *  subsystem_vendor_id, and revision_id
 **/
s32 ixgbe_init_shared_code(struct ixgbe_hw *hw)
{
        s32 status;

        DEBUGFUNC("ixgbe_init_shared_code");

        /*
         * Set the mac type
```



```
         */
        ixgbe_set_mac_type(hw);
        (void) ixgbe_set_mac_type(hw);

        switch (hw->mac.type) {
        case ixgbe_mac_82598EB:
                status = ixgbe_init_ops_82598(hw);
                break;
        case ixgbe_mac_82599EB:
                status = ixgbe_init_ops_82599(hw);
                break;
#if 0
/* XXX KEBE ASKS - nuke this? */
        case ixgbe_mac_82599_vf:
        case ixgbe_mac_X540_vf:
                status = ixgbe_init_ops_vf(hw);
                break;
#endif
        case ixgbe_mac_X540:
                status = ixgbe_init_ops_X540(hw);
                break;
        default:
                status = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
                break;
        }

        return status;
}

/**
 *  ixgbe_set_mac_type - Sets MAC type
 *  @hw: pointer to the HW structure
 *
 *  This function sets the mac type of the adapter based on the
 *  vendor ID and device ID stored in the hw structure.
 **/
s32 ixgbe_set_mac_type(struct ixgbe_hw *hw)
{
        s32 ret_val = IXGBE_SUCCESS;

        DEBUGFUNC("ixgbe_set_mac_type\n");

        if (hw->vendor_id == IXGBE_INTEL_VENDOR_ID) {
                switch (hw->device_id) {
                case IXGBE_DEV_ID_82598:
                case IXGBE_DEV_ID_82598_BX:
                case IXGBE_DEV_ID_82598AF_SINGLE_PORT:
                case IXGBE_DEV_ID_82598AF_DUAL_PORT:
                case IXGBE_DEV_ID_82598AT:
                case IXGBE_DEV_ID_82598AT2:
                case IXGBE_DEV_ID_82598EB_CX4:
                case IXGBE_DEV_ID_82598_CX4_DUAL_PORT:
                case IXGBE_DEV_ID_82598_DA_DUAL_PORT:
                case IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM:
                case IXGBE_DEV_ID_82598EB_XF_LR:
                case IXGBE_DEV_ID_82598EB_SFP_LOM:
                        hw->mac.type = ixgbe_mac_82598EB;
                        break;
                case IXGBE_DEV_ID_82599_KX4:
                case IXGBE_DEV_ID_82599_KX4_MEZZ:
                case IXGBE_DEV_ID_82599_XAUI_LOM:
                case IXGBE_DEV_ID_82599_COMBO_BACKPLANE:
                case IXGBE_DEV_ID_82599_KR:
                case IXGBE_DEV_ID_82599_SFP:
                case IXGBE_DEV_ID_82599_BACKPLANE_FCOE:
                case IXGBE_DEV_ID_82599_SFP_FCOE:
```

```
123                 case IXGBE_DEV_ID_82599_SFP_EM:
124                 case IXGBE_DEV_ID_82599_SFP_SF2:
125                 case IXGBE_DEV_ID_82599EN_SFP:
126                 case IXGBE_DEV_ID_82599_CX4:
127                 case IXGBE_DEV_ID_82599_T3_LOM:
128                         hw->mac.type = ixgbe_mac_82599EB;
129                         break;
130                 case IXGBE_DEV_ID_82599_VF:
131                         hw->mac.type = ixgbe_mac_82599_vf;
132                         break;
133                 case IXGBE_DEV_ID_X540_VF:
134                         hw->mac.type = ixgbe_mac_X540_vf;
135                         break;
136                 case IXGBE_DEV_ID_X540T:
137                 case IXGBE_DEV_ID_X540T1:
138                         hw->mac.type = ixgbe_mac_X540;
139                         break;
140                 default:
141                         ret_val = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
142                         break;
143                 }
144         } else {
145                 ret_val = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
146         }

148         DEBUGOUT2("ixgbe_set_mac_type found mac: %d, returns: %d\n",
149                   hw->mac.type, ret_val);
150         return ret_val;
151 }
```

*unchanged_portion_omitted_*

```
382 /**
366  *  ixgbe_read_pba_length - Reads part number string length from EEPROM
367  *  @hw: pointer to hardware structure
368  *  @pba_num_size: part number string buffer length
369  *
370  *  Reads the part number length from the EEPROM.
371  *  Returns expected buffer size in pba_num_size.
372  **/
373 s32 ixgbe_read_pba_length(struct ixgbe_hw *hw, u32 *pba_num_size)
374 {
375         return ixgbe_read_pba_length_generic(hw, pba_num_size);
376 }

378 /**
383  *  ixgbe_read_pba_num - Reads part number from EEPROM
384  *  @hw: pointer to hardware structure
385  *  @pba_num: stores the part number from the EEPROM
386  *
387  *  Reads the part number from the EEPROM.
388  **/
389 s32 ixgbe_read_pba_num(struct ixgbe_hw *hw, u32 *pba_num)
390 {
391         return ixgbe_read_pba_num_generic(hw, pba_num);
392 }
```

*unchanged_portion_omitted_*

```
447 /**
448  *  ixgbe_read_phy_reg - Read PHY register
449  *  @hw: pointer to hardware structure
450  *  @reg_addr: 32 bit address of PHY register to read
451  *  @phy_data: Pointer to read data from PHY register
452  *
453  *  Reads a value from a specified PHY register
454  **/
455 s32 ixgbe_read_phy_reg(struct ixgbe_hw *hw, u32 reg_addr, u32 device_type,
```



```
456                        u16 *phy_data)
457 {
458         if (hw->phy.id == 0)
459                 ixgbe_identify_phy(hw);
455                 (void) ixgbe_identify_phy(hw);

461         return ixgbe_call_func(hw, hw->phy.ops.read_reg, (hw, reg_addr,
462                                device_type, phy_data), IXGBE_NOT_IMPLEMENTED);
463 }

465 /**
466  *  ixgbe_write_phy_reg - Write PHY register
467  *  @hw: pointer to hardware structure
468  *  @reg_addr: 32 bit PHY register to write
469  *  @phy_data: Data to write to the PHY register
470  *
471  *  Writes a value to specified PHY register
472  **/
473 s32 ixgbe_write_phy_reg(struct ixgbe_hw *hw, u32 reg_addr, u32 device_type,
474                         u16 phy_data)
475 {
476         if (hw->phy.id == 0)
477                 ixgbe_identify_phy(hw);
473                 (void) ixgbe_identify_phy(hw);

479         return ixgbe_call_func(hw, hw->phy.ops.write_reg, (hw, reg_addr,
480                                device_type, phy_data), IXGBE_NOT_IMPLEMENTED);
481 }
```

*unchanged_portion_omitted_*

```
690 /**
691  *  ixgbe_write_eeprom_buffer - Write word(s) to EEPROM
692  *  @hw: pointer to hardware structure
693  *  @offset: offset within the EEPROM to be written to
694  *  @data: 16 bit word(s) to be written to the EEPROM
695  *  @words: number of words
696  *
697  *  Writes 16 bit word(s) to EEPROM. If ixgbe_eeprom_update_checksum is not
698  *  called after this function, the EEPROM will most likely contain an
699  *  invalid checksum.
700  **/
701 s32 ixgbe_write_eeprom_buffer(struct ixgbe_hw *hw, u16 offset, u16 words,
702                               u16 *data)
703 {
704         return ixgbe_call_func(hw, hw->eeprom.ops.write_buffer,
705                                (hw, offset, words, data),
706                                IXGBE_NOT_IMPLEMENTED);
707 }

709 /**
710  *  ixgbe_read_eeprom - Read word from EEPROM
711  *  @hw: pointer to hardware structure
712  *  @offset: offset within the EEPROM to be read
713  *  @data: read 16 bit value from EEPROM
714  *
715  *  Reads 16 bit value from EEPROM
716  **/
717 s32 ixgbe_read_eeprom(struct ixgbe_hw *hw, u16 offset, u16 *data)
718 {
719         return ixgbe_call_func(hw, hw->eeprom.ops.read, (hw, offset, data),
720                                IXGBE_NOT_IMPLEMENTED);
721 }

723 /**
724  *  ixgbe_read_eeprom_buffer - Read word(s) from EEPROM
725  *  @hw: pointer to hardware structure
```

```
 *  @offset: offset within the EEPROM to be read
 *  @data: read 16 bit word(s) from EEPROM
 *  @words: number of words
 *
 *  Reads 16 bit word(s) from EEPROM
 **/
s32 ixgbe_read_eeprom_buffer(struct ixgbe_hw *hw, u16 offset,
                             u16 words, u16 *data)
{
        return ixgbe_call_func(hw, hw->eeprom.ops.read_buffer,
                               (hw, offset, words, data),
                               IXGBE_NOT_IMPLEMENTED);
}

/**
 *  ixgbe_validate_eeprom_checksum - Validate EEPROM checksum
 *  @hw: pointer to hardware structure
 *  @checksum_val: calculated checksum
 *
 *  Performs checksum calculation and validates the EEPROM checksum
 **/
s32 ixgbe_validate_eeprom_checksum(struct ixgbe_hw *hw, u16 *checksum_val)
{
        return ixgbe_call_func(hw, hw->eeprom.ops.validate_checksum,
                               (hw, checksum_val), IXGBE_NOT_IMPLEMENTED);
}
```

*unchanged_portion_omitted_*

```
/**
 *  ixgbe_set_vmdq - Associate a VMDq index with a receive address
 *  @hw: pointer to hardware structure
 *  @rar: receive address register index to associate with VMDq index
 *  @vmdq: VMDq set or pool index
 **/
s32 ixgbe_set_vmdq(struct ixgbe_hw *hw, u32 rar, u32 vmdq)
{
        return ixgbe_call_func(hw, hw->mac.ops.set_vmdq, (hw, rar, vmdq),
                               IXGBE_NOT_IMPLEMENTED);

}

/**
 *  ixgbe_set_vmdq_san_mac - Associate VMDq index 127 with a receive address
 *  @hw: pointer to hardware structure
 *  @vmdq: VMDq default pool index
 **/
s32 ixgbe_set_vmdq_san_mac(struct ixgbe_hw *hw, u32 vmdq)
{
        return ixgbe_call_func(hw, hw->mac.ops.set_vmdq_san_mac,
                               (hw, vmdq), IXGBE_NOT_IMPLEMENTED);
}

/**
 *  ixgbe_clear_vmdq - Disassociate a VMDq index from a receive address
 *  @hw: pointer to hardware structure
 *  @rar: receive address register index to disassociate with VMDq index
 *  @vmdq: VMDq set or pool index
 **/
s32 ixgbe_clear_vmdq(struct ixgbe_hw *hw, u32 rar, u32 vmdq)
{
        return ixgbe_call_func(hw, hw->mac.ops.clear_vmdq, (hw, rar, vmdq),
                               IXGBE_NOT_IMPLEMENTED);
}
```

*unchanged_portion_omitted_*

```
/**
```



```
 *  ixgbe_update_mc_addr_list - Updates the MAC's list of multicast addresses
 *  @hw: pointer to hardware structure
 *  @mc_addr_list: the list of new multicast addresses
 *  @mc_addr_count: number of addresses
 *  @func: iterator function to walk the multicast address list
 *
 *  The given list replaces any existing list. Clears the MC addrs from receive
 *  address registers and the multicast table. Uses unused receive address
 *  registers for the first multicast addresses, and hashes the rest into the
 *  multicast table.
 **/
s32 ixgbe_update_mc_addr_list(struct ixgbe_hw *hw, u8 *mc_addr_list,
                              u32 mc_addr_count, ixgbe_mc_addr_itr func,
                              bool clear)
                              u32 mc_addr_count, ixgbe_mc_addr_itr func)
{
        return ixgbe_call_func(hw, hw->mac.ops.update_mc_addr_list, (hw,
                               mc_addr_list, mc_addr_count, func, clear),
                               mc_addr_list, mc_addr_count, func),
                               IXGBE_NOT_IMPLEMENTED);
}
```

*unchanged_portion_omitted_*

```
/**
 *  ixgbe_set_vlvf - Set VLAN Pool Filter
 *  @hw: pointer to hardware structure
 *  @vlan: VLAN id to write to VLAN filter
 *  @vind: VMDq output index that maps queue to VLAN id in VFVFB
 *  @vlan_on: boolean flag to turn on/off VLAN in VFVF
 *  @vfta_changed: pointer to boolean flag which indicates whether VFTA
 *                 should be changed
 *
 *  Turn on/off specified bit in VLVF table.
 **/
s32 ixgbe_set_vlvf(struct ixgbe_hw *hw, u32 vlan, u32 vind, bool vlan_on,
                   bool *vfta_changed)
{
        return ixgbe_call_func(hw, hw->mac.ops.set_vlvf, (hw, vlan, vind,
                               vlan_on, vfta_changed), IXGBE_NOT_IMPLEMENTED);
}

/**
 *  ixgbe_fc_enable - Enable flow control
 *  @hw: pointer to hardware structure
 *  @packetbuf_num: packet buffer number (0-7)
 *
 *  Configures the flow control settings based on SW configuration.
 **/
s32 ixgbe_fc_enable(struct ixgbe_hw *hw)
s32 ixgbe_fc_enable(struct ixgbe_hw *hw, s32 packetbuf_num)
{
        return ixgbe_call_func(hw, hw->mac.ops.fc_enable, (hw),
        return ixgbe_call_func(hw, hw->mac.ops.fc_enable, (hw, packetbuf_num),
                               IXGBE_NOT_IMPLEMENTED);
}

/**
 * ixgbe_set_fw_drv_ver - Try to send the driver version number FW
 * @hw: pointer to hardware structure
 * @maj: driver major number to be sent to firmware
 * @min: driver minor number to be sent to firmware
 * @build: driver build number to be sent to firmware
 * @ver: driver version number to be sent to firmware
 **/
s32 ixgbe_set_fw_drv_ver(struct ixgbe_hw *hw, u8 maj, u8 min, u8 build,
                         u8 ver)
```

```
 999 {
1000         return ixgbe_call_func(hw, hw->mac.ops.set_fw_drv_ver, (hw, maj, min,
1001                                build, ver), IXGBE_NOT_IMPLEMENTED);
1002 }


1005 /**
1006  *  ixgbe_read_analog_reg8 - Reads 8 bit analog register
1007  *  @hw: pointer to hardware structure
1008  *  @reg: analog register to read
1009  *  @val: read value
1010  *
1011  *  Performs write operation to analog register specified.
1012  **/
1013 s32 ixgbe_read_analog_reg8(struct ixgbe_hw *hw, u32 reg, u8 *val)
1014 {
1015         return ixgbe_call_func(hw, hw->mac.ops.read_analog_reg8, (hw, reg,
1016                                val), IXGBE_NOT_IMPLEMENTED);
1017 }
```

*unchanged_portion_omitted_*

```
1120 /**
1121  *  ixgbe_enable_rx_dma - Enables Rx DMA unit, dependent on device specifics
1035  *  ixgbe_enable_rx_dma - Enables Rx DMA unit, dependant on device specifics
1122  *  @hw: pointer to hardware structure
1123  *  @regval: bitfield to write to the Rx DMA register
1124  *
1125  *  Enables the Rx DMA unit of the device.
1126  **/
1127 s32 ixgbe_enable_rx_dma(struct ixgbe_hw *hw, u32 regval)
1128 {
1129         return ixgbe_call_func(hw, hw->mac.ops.enable_rx_dma,
1130                                (hw, regval), IXGBE_NOT_IMPLEMENTED);
1131 }

1133 /**
1134  *  ixgbe_disable_sec_rx_path - Stops the receive data path
1135  *  @hw: pointer to hardware structure
1136  *
1137  *  Stops the receive data path.
1138  **/
1139 s32 ixgbe_disable_sec_rx_path(struct ixgbe_hw *hw)
1140 {
1141         return ixgbe_call_func(hw, hw->mac.ops.disable_sec_rx_path,
1142                                (hw), IXGBE_NOT_IMPLEMENTED);
1143 }

1145 /**
1146  *  ixgbe_enable_sec_rx_path - Enables the receive data path
1147  *  @hw: pointer to hardware structure
1148  *
1149  *  Enables the receive data path.
1150  **/
1151 s32 ixgbe_enable_sec_rx_path(struct ixgbe_hw *hw)
1152 {
1153         return ixgbe_call_func(hw, hw->mac.ops.enable_sec_rx_path,
1154                                (hw), IXGBE_NOT_IMPLEMENTED);
1155 }

1157 /**
1158  *  ixgbe_acquire_swfw_semaphore - Acquire SWFW semaphore
1159  *  @hw: pointer to hardware structure
1160  *  @mask: Mask to specify which semaphore to acquire
1161  *
1162  *  Acquires the SWFW semaphore through SW_FW_SYNC register for the specified
1163  *  function (CSR, PHY0, PHY1, EEPROM, Flash)
```



```
1164  **/
1165 s32 ixgbe_acquire_swfw_semaphore(struct ixgbe_hw *hw, u16 mask)
1166 {
1167         return ixgbe_call_func(hw, hw->mac.ops.acquire_swfw_sync,
1168                                (hw, mask), IXGBE_NOT_IMPLEMENTED);
1169 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    8429 Thu Jul 12 12:22:31 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_api.h
XXXX Intel X540 support
**********************************************************
/******************************************************************************
  Copyright (c) 2001-2012, Intel Corporation
  Copyright (c) 2001-2010, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_api.h,v 1.14 2012/07/05 20:51:44 jfv Exp $*/

#ifndef _IXGBE_API_H_
#define _IXGBE_API_H_

#include "ixgbe_type.h"

s32 ixgbe_init_shared_code(struct ixgbe_hw *hw);

extern s32 ixgbe_init_ops_82598(struct ixgbe_hw *hw);
extern s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw);
extern s32 ixgbe_init_ops_X540(struct ixgbe_hw *hw);
extern s32 ixgbe_init_ops_vf(struct ixgbe_hw *hw);

s32 ixgbe_set_mac_type(struct ixgbe_hw *hw);
s32 ixgbe_init_hw(struct ixgbe_hw *hw);
s32 ixgbe_reset_hw(struct ixgbe_hw *hw);
s32 ixgbe_start_hw(struct ixgbe_hw *hw);
void ixgbe_enable_relaxed_ordering(struct ixgbe_hw *hw);
s32 ixgbe_clear_hw_cntrs(struct ixgbe_hw *hw);
enum ixgbe_media_type ixgbe_get_media_type(struct ixgbe_hw *hw);
s32 ixgbe_get_mac_addr(struct ixgbe_hw *hw, u8 *mac_addr);
s32 ixgbe_get_bus_info(struct ixgbe_hw *hw);
u32 ixgbe_get_num_of_tx_queues(struct ixgbe_hw *hw);
u32 ixgbe_get_num_of_rx_queues(struct ixgbe_hw *hw);
s32 ixgbe_stop_adapter(struct ixgbe_hw *hw);
s32 ixgbe_read_pba_num(struct ixgbe_hw *hw, u32 *pba_num);
s32 ixgbe_read_pba_string(struct ixgbe_hw *hw, u8 *pba_num, u32 pba_num_size);
```



```
s32 ixgbe_read_pba_length(struct ixgbe_hw *hw, u32 *pba_num_size);

s32 ixgbe_identify_phy(struct ixgbe_hw *hw);
s32 ixgbe_reset_phy(struct ixgbe_hw *hw);
s32 ixgbe_read_phy_reg(struct ixgbe_hw *hw, u32 reg_addr, u32 device_type,
                       u16 *phy_data);
s32 ixgbe_write_phy_reg(struct ixgbe_hw *hw, u32 reg_addr, u32 device_type,
                        u16 phy_data);

s32 ixgbe_setup_phy_link(struct ixgbe_hw *hw);
s32 ixgbe_check_phy_link(struct ixgbe_hw *hw,
                         ixgbe_link_speed *speed,
                         bool *link_up);
s32 ixgbe_setup_phy_link_speed(struct ixgbe_hw *hw,
                               ixgbe_link_speed speed,
                               bool autoneg,
                               bool autoneg_wait_to_complete);
void ixgbe_disable_tx_laser(struct ixgbe_hw *hw);
void ixgbe_enable_tx_laser(struct ixgbe_hw *hw);
void ixgbe_flap_tx_laser(struct ixgbe_hw *hw);
s32 ixgbe_setup_link(struct ixgbe_hw *hw, ixgbe_link_speed speed,
                     bool autoneg, bool autoneg_wait_to_complete);
s32 ixgbe_check_link(struct ixgbe_hw *hw, ixgbe_link_speed *speed,
                     bool *link_up, bool link_up_wait_to_complete);
s32 ixgbe_get_link_capabilities(struct ixgbe_hw *hw, ixgbe_link_speed *speed,
                                bool *autoneg);
s32 ixgbe_led_on(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_led_off(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_blink_led_start(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_blink_led_stop(struct ixgbe_hw *hw, u32 index);

s32 ixgbe_init_eeprom_params(struct ixgbe_hw *hw);
s32 ixgbe_write_eeprom(struct ixgbe_hw *hw, u16 offset, u16 data);
s32 ixgbe_write_eeprom_buffer(struct ixgbe_hw *hw, u16 offset,
                              u16 words, u16 *data);
s32 ixgbe_read_eeprom(struct ixgbe_hw *hw, u16 offset, u16 *data);
s32 ixgbe_read_eeprom_buffer(struct ixgbe_hw *hw, u16 offset,
                             u16 words, u16 *data);

s32 ixgbe_validate_eeprom_checksum(struct ixgbe_hw *hw, u16 *checksum_val);
s32 ixgbe_update_eeprom_checksum(struct ixgbe_hw *hw);

s32 ixgbe_insert_mac_addr(struct ixgbe_hw *hw, u8 *addr, u32 vmdq);
s32 ixgbe_set_rar(struct ixgbe_hw *hw, u32 index, u8 *addr, u32 vmdq,
                  u32 enable_addr);
s32 ixgbe_clear_rar(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_set_vmdq(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
s32 ixgbe_set_vmdq_san_mac(struct ixgbe_hw *hw, u32 vmdq);
s32 ixgbe_clear_vmdq(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
s32 ixgbe_init_rx_addrs(struct ixgbe_hw *hw);
u32 ixgbe_get_num_rx_addrs(struct ixgbe_hw *hw);
s32 ixgbe_update_uc_addr_list(struct ixgbe_hw *hw, u8 *addr_list,
                              u32 addr_count, ixgbe_mc_addr_itr func);
s32 ixgbe_update_mc_addr_list(struct ixgbe_hw *hw, u8 *mc_addr_list,
                              u32 mc_addr_count, ixgbe_mc_addr_itr func,
                              bool clear);
                              u32 mc_addr_count, ixgbe_mc_addr_itr func);
void ixgbe_add_uc_addr(struct ixgbe_hw *hw, u8 *addr_list, u32 vmdq);
s32 ixgbe_enable_mc(struct ixgbe_hw *hw);
s32 ixgbe_disable_mc(struct ixgbe_hw *hw);
s32 ixgbe_clear_vfta(struct ixgbe_hw *hw);
s32 ixgbe_set_vfta(struct ixgbe_hw *hw, u32 vlan,
                   u32 vind, bool vlan_on);
s32 ixgbe_set_vlvf(struct ixgbe_hw *hw, u32 vlan, u32 vind,
                   bool vlan_on, bool *vfta_changed);
s32 ixgbe_fc_enable(struct ixgbe_hw *hw);
```

```
125 s32 ixgbe_set_fw_drv_ver(struct ixgbe_hw *hw, u8 maj, u8 min, u8 build,
126                          u8 ver);

111 s32 ixgbe_fc_enable(struct ixgbe_hw *hw, s32 packetbuf_num);

127 void ixgbe_set_mta(struct ixgbe_hw *hw, u8 *mc_addr);
128 s32 ixgbe_get_phy_firmware_version(struct ixgbe_hw *hw,
129                                    u16 *firmware_version);
130 s32 ixgbe_read_analog_reg8(struct ixgbe_hw *hw, u32 reg, u8 *val);
131 s32 ixgbe_write_analog_reg8(struct ixgbe_hw *hw, u32 reg, u8 val);
132 s32 ixgbe_init_uta_tables(struct ixgbe_hw *hw);
133 s32 ixgbe_read_i2c_eeprom(struct ixgbe_hw *hw, u8 byte_offset, u8 *eeprom_data);
134 u32 ixgbe_get_supported_physical_layer(struct ixgbe_hw *hw);
135 s32 ixgbe_enable_rx_dma(struct ixgbe_hw *hw, u32 regval);
136 s32 ixgbe_disable_sec_rx_path(struct ixgbe_hw *hw);
137 s32 ixgbe_enable_sec_rx_path(struct ixgbe_hw *hw);
138 s32 ixgbe_reinit_fdir_tables_82599(struct ixgbe_hw *hw);
139 s32 ixgbe_init_fdir_signature_82599(struct ixgbe_hw *hw, u32 fdirctrl);
140 s32 ixgbe_init_fdir_perfect_82599(struct ixgbe_hw *hw, u32 fdirctrl);
123 s32 ixgbe_init_fdir_signature_82599(struct ixgbe_hw *hw, u32 pballoc);
124 s32 ixgbe_init_fdir_perfect_82599(struct ixgbe_hw *hw, u32 pballoc);
141 s32 ixgbe_fdir_add_signature_filter_82599(struct ixgbe_hw *hw,
142                                           union ixgbe_atr_hash_dword input,
143                                           union ixgbe_atr_hash_dword common,
144                                           u8 queue);
145 s32 ixgbe_fdir_set_input_mask_82599(struct ixgbe_hw *hw,
146                                     union ixgbe_atr_input *input_mask);
147 s32 ixgbe_fdir_write_perfect_filter_82599(struct ixgbe_hw *hw,
148                                           union ixgbe_atr_input *input,
149                                           u16 soft_id, u8 queue);
150 s32 ixgbe_fdir_erase_perfect_filter_82599(struct ixgbe_hw *hw,
151                                           union ixgbe_atr_input *input,
152                                           u16 soft_id);
153 s32 ixgbe_fdir_add_perfect_filter_82599(struct ixgbe_hw *hw,
154                                         union ixgbe_atr_input *input,
155                                         union ixgbe_atr_input *mask,
131                                         struct ixgbe_atr_input_masks *masks,
156                                         u16 soft_id,
157                                         u8 queue);
158 void ixgbe_atr_compute_perfect_hash_82599(union ixgbe_atr_input *input,
159                                           union ixgbe_atr_input *mask);
160 u32 ixgbe_atr_compute_sig_hash_82599(union ixgbe_atr_hash_dword input,
161                                      union ixgbe_atr_hash_dword common);
134 u32 ixgbe_atr_compute_hash_82599(union ixgbe_atr_input *input, u32 key);
162 s32 ixgbe_read_i2c_byte(struct ixgbe_hw *hw, u8 byte_offset, u8 dev_addr,
163                         u8 *data);
164 s32 ixgbe_write_i2c_byte(struct ixgbe_hw *hw, u8 byte_offset, u8 dev_addr,
165                          u8 data);
166 s32 ixgbe_write_i2c_eeprom(struct ixgbe_hw *hw, u8 byte_offset, u8 eeprom_data);
167 s32 ixgbe_get_san_mac_addr(struct ixgbe_hw *hw, u8 *san_mac_addr);
168 s32 ixgbe_set_san_mac_addr(struct ixgbe_hw *hw, u8 *san_mac_addr);
169 s32 ixgbe_get_device_caps(struct ixgbe_hw *hw, u16 *device_caps);
170 s32 ixgbe_acquire_swfw_semaphore(struct ixgbe_hw *hw, u16 mask);
171 void ixgbe_release_swfw_semaphore(struct ixgbe_hw *hw, u16 mask);
172 s32 ixgbe_get_wwn_prefix(struct ixgbe_hw *hw, u16 *wwnn_prefix,
173                          u16 *wwpn_prefix);
174 s32 ixgbe_get_fcoe_boot_status(struct ixgbe_hw *hw, u16 *bs);

176 #endif /* _IXGBE_API_H_ */
```

```
**********************************************************
  116733 Thu Jul 12 12:22:32 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_common.c
XXXX Intel X540 support
**********************************************************
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  Copyright (c) 2001-2010, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_common.c,v 1.14 2012/07/05 20:51:44 jfv Exp
/*$FreeBSD$*/

#include "ixgbe_common.h"
#include "ixgbe_phy.h"
#include "ixgbe_api.h"

static s32 ixgbe_acquire_eeprom(struct ixgbe_hw *hw);
static s32 ixgbe_get_eeprom_semaphore(struct ixgbe_hw *hw);
static void ixgbe_release_eeprom_semaphore(struct ixgbe_hw *hw);
static s32 ixgbe_ready_eeprom(struct ixgbe_hw *hw);
static void ixgbe_standby_eeprom(struct ixgbe_hw *hw);
static void ixgbe_shift_out_eeprom_bits(struct ixgbe_hw *hw, u16 data,
                                        u16 count);
static u16 ixgbe_shift_in_eeprom_bits(struct ixgbe_hw *hw, u16 count);
static void ixgbe_raise_eeprom_clk(struct ixgbe_hw *hw, u32 *eec);
static void ixgbe_lower_eeprom_clk(struct ixgbe_hw *hw, u32 *eec);
static void ixgbe_release_eeprom(struct ixgbe_hw *hw);

static s32 ixgbe_mta_vector(struct ixgbe_hw *hw, u8 *mc_addr);
static s32 ixgbe_get_san_mac_addr_offset(struct ixgbe_hw *hw,
                                         u16 *san_mac_offset);
static s32 ixgbe_read_eeprom_buffer_bit_bang(struct ixgbe_hw *hw, u16 offset,
                                             u16 words, u16 *data);
static s32 ixgbe_write_eeprom_buffer_bit_bang(struct ixgbe_hw *hw, u16 offset,
                                              u16 words, u16 *data);
static s32 ixgbe_detect_eeprom_page_size_generic(struct ixgbe_hw *hw,
                                                 u16 offset);
```



```
static s32 ixgbe_fc_autoneg_fiber(struct ixgbe_hw *hw);
static s32 ixgbe_fc_autoneg_backplane(struct ixgbe_hw *hw);
static s32 ixgbe_fc_autoneg_copper(struct ixgbe_hw *hw);
static s32 ixgbe_device_supports_autoneg_fc(struct ixgbe_hw *hw);
static s32 ixgbe_negotiate_fc(struct ixgbe_hw *hw, u32 adv_reg, u32 lp_reg,
                              u32 adv_sym, u32 adv_asm, u32 lp_sym, u32 lp_asm);

s32 ixgbe_find_vlvf_slot(struct ixgbe_hw *hw, u32 vlan);

/**
 *  ixgbe_init_ops_generic - Inits function ptrs
 *  @hw: pointer to the hardware structure
 *
 *  Initialize the function pointers.
 **/
s32 ixgbe_init_ops_generic(struct ixgbe_hw *hw)
{
        struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
        struct ixgbe_mac_info *mac = &hw->mac;
        u32 eec = IXGBE_READ_REG(hw, IXGBE_EEC);

        DEBUGFUNC("ixgbe_init_ops_generic");

        /* EEPROM */
        eeprom->ops.init_params = &ixgbe_init_eeprom_params_generic;
        /* If EEPROM is valid (bit 8 = 1), use EERD otherwise use bit bang */
        if (eec & IXGBE_EEC_PRES) {
        if (eec & (1 << 8))
                eeprom->ops.read = &ixgbe_read_eerd_generic;
                eeprom->ops.read_buffer = &ixgbe_read_eerd_buffer_generic;
        } else {
        else
                eeprom->ops.read = &ixgbe_read_eeprom_bit_bang_generic;
                eeprom->ops.read_buffer =
                                &ixgbe_read_eeprom_buffer_bit_bang_generic;
        }
        eeprom->ops.write = &ixgbe_write_eeprom_generic;
        eeprom->ops.write_buffer = &ixgbe_write_eeprom_buffer_bit_bang_generic;
        eeprom->ops.validate_checksum =
                                        &ixgbe_validate_eeprom_checksum_generic;
        eeprom->ops.update_checksum = &ixgbe_update_eeprom_checksum_generic;
        eeprom->ops.calc_checksum = &ixgbe_calc_eeprom_checksum_generic;

        /* MAC */
        mac->ops.init_hw = &ixgbe_init_hw_generic;
        mac->ops.reset_hw = NULL;
        mac->ops.start_hw = &ixgbe_start_hw_generic;
        mac->ops.clear_hw_cntrs = &ixgbe_clear_hw_cntrs_generic;
        mac->ops.get_media_type = NULL;
        mac->ops.get_supported_physical_layer = NULL;
        mac->ops.enable_rx_dma = &ixgbe_enable_rx_dma_generic;
        mac->ops.get_mac_addr = &ixgbe_get_mac_addr_generic;
        mac->ops.stop_adapter = &ixgbe_stop_adapter_generic;
        mac->ops.get_bus_info = &ixgbe_get_bus_info_generic;
        mac->ops.set_lan_id = &ixgbe_set_lan_id_multi_port_pcie;
        mac->ops.acquire_swfw_sync = &ixgbe_acquire_swfw_sync;
        mac->ops.release_swfw_sync = &ixgbe_release_swfw_sync;

        /* LEDs */
        mac->ops.led_on = &ixgbe_led_on_generic;
        mac->ops.led_off = &ixgbe_led_off_generic;
        mac->ops.blink_led_start = &ixgbe_blink_led_start_generic;
        mac->ops.blink_led_stop = &ixgbe_blink_led_stop_generic;

        /* RAR, Multicast, VLAN */
        mac->ops.set_rar = &ixgbe_set_rar_generic;
```

```
116         mac->ops.clear_rar = &ixgbe_clear_rar_generic;
117         mac->ops.insert_mac_addr = NULL;
118         mac->ops.set_vmdq = NULL;
119         mac->ops.clear_vmdq = NULL;
120         mac->ops.init_rx_addrs = &ixgbe_init_rx_addrs_generic;
121         mac->ops.update_uc_addr_list = &ixgbe_update_uc_addr_list_generic;
122         mac->ops.update_mc_addr_list = &ixgbe_update_mc_addr_list_generic;
123         mac->ops.enable_mc = &ixgbe_enable_mc_generic;
124         mac->ops.disable_mc = &ixgbe_disable_mc_generic;
125         mac->ops.clear_vfta = NULL;
126         mac->ops.set_vfta = NULL;
127         mac->ops.set_vlvf = NULL;
128         mac->ops.init_uta_tables = NULL;

130         /* Flow Control */
131         mac->ops.fc_enable = &ixgbe_fc_enable_generic;

133         /* Link */
134         mac->ops.get_link_capabilities = NULL;
135         mac->ops.setup_link = NULL;
136         mac->ops.check_link = NULL;

138         return IXGBE_SUCCESS;
139 }

141 /**
142  *  ixgbe_device_supports_autoneg_fc - Check if phy supports autoneg flow
143  *  control
144  *  @hw: pointer to hardware structure
145  *
146  *  There are several phys that do not support autoneg flow control. This
147  *  function check the device id to see if the associated phy supports
148  *  autoneg flow control.
149  **/
150 static s32 ixgbe_device_supports_autoneg_fc(struct ixgbe_hw *hw)
151 {

153         DEBUGFUNC("ixgbe_device_supports_autoneg_fc");

155         switch (hw->device_id) {
156         case IXGBE_DEV_ID_X540T:
157         case IXGBE_DEV_ID_X540T1:
158                 return IXGBE_SUCCESS;
159         case IXGBE_DEV_ID_82599_T3_LOM:
160                 return IXGBE_SUCCESS;
161         default:
162                 return IXGBE_ERR_FC_NOT_SUPPORTED;
163         }
164 }

166 /**
167  *  ixgbe_setup_fc - Set up flow control
168  *  @hw: pointer to hardware structure
169  *
170  *  Called at init time to set up flow control.
171  **/
172 static s32 ixgbe_setup_fc(struct ixgbe_hw *hw)
173 {
174         s32 ret_val = IXGBE_SUCCESS;
175         u32 reg = 0, reg_bp = 0;
176         u16 reg_cu = 0;

178         DEBUGFUNC("ixgbe_setup_fc");

180         /*
181          * Validate the requested mode.  Strict IEEE mode does not allow
```



```
182          * ixgbe_fc_rx_pause because it will cause us to fail at UNH.
183          */
184         if (hw->fc.strict_ieee && hw->fc.requested_mode == ixgbe_fc_rx_pause) {
185                 DEBUGOUT("ixgbe_fc_rx_pause not valid in strict IEEE mode\n");
186                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
187                 goto out;
188         }

190         /*
191          * 10gig parts do not have a word in the EEPROM to determine the
192          * default flow control setting, so we explicitly set it to full.
193          */
194         if (hw->fc.requested_mode == ixgbe_fc_default)
195                 hw->fc.requested_mode = ixgbe_fc_full;

197         /*
198          * Set up the 1G and 10G flow control advertisement registers so the
199          * HW will be able to do fc autoneg once the cable is plugged in.  If
200          * we link at 10G, the 1G advertisement is harmless and vice versa.
201          */
202         switch (hw->phy.media_type) {
203         case ixgbe_media_type_fiber:
204         case ixgbe_media_type_backplane:
205                 reg = IXGBE_READ_REG(hw, IXGBE_PCS1GANA);
206                 reg_bp = IXGBE_READ_REG(hw, IXGBE_AUTOC);
207                 break;
208         case ixgbe_media_type_copper:
209                 hw->phy.ops.read_reg(hw, IXGBE_MDIO_AUTO_NEG_ADVT,
210                                      IXGBE_MDIO_AUTO_NEG_DEV_TYPE, &reg_cu);
211                 break;
212         default:
213                 break;
214         }

216         /*
217          * The possible values of fc.requested_mode are:
218          * 0: Flow control is completely disabled
219          * 1: Rx flow control is enabled (we can receive pause frames,
220          *    but not send pause frames).
221          * 2: Tx flow control is enabled (we can send pause frames but
222          *    we do not support receiving pause frames).
223          * 3: Both Rx and Tx flow control (symmetric) are enabled.
224          * other: Invalid.
225          */
226         switch (hw->fc.requested_mode) {
227         case ixgbe_fc_none:
228                 /* Flow control completely disabled by software override. */
229                 reg &= ~(IXGBE_PCS1GANA_SYM_PAUSE | IXGBE_PCS1GANA_ASM_PAUSE);
230                 if (hw->phy.media_type == ixgbe_media_type_backplane)
231                         reg_bp &= ~(IXGBE_AUTOC_SYM_PAUSE |
232                                     IXGBE_AUTOC_ASM_PAUSE);
233                 else if (hw->phy.media_type == ixgbe_media_type_copper)
234                         reg_cu &= ~(IXGBE_TAF_SYM_PAUSE | IXGBE_TAF_ASM_PAUSE);
235                 break;
236         case ixgbe_fc_tx_pause:
237                 /*
238                  * Tx Flow control is enabled, and Rx Flow control is
239                  * disabled by software override.
240                  */
241                 reg |= IXGBE_PCS1GANA_ASM_PAUSE;
242                 reg &= ~IXGBE_PCS1GANA_SYM_PAUSE;
243                 if (hw->phy.media_type == ixgbe_media_type_backplane) {
244                         reg_bp |= IXGBE_AUTOC_ASM_PAUSE;
245                         reg_bp &= ~IXGBE_AUTOC_SYM_PAUSE;
246                 } else if (hw->phy.media_type == ixgbe_media_type_copper) {
247                         reg_cu |= IXGBE_TAF_ASM_PAUSE;
```

```
248                         reg_cu &= ~IXGBE_TAF_SYM_PAUSE;
249                 }
250                 break;
251         case ixgbe_fc_rx_pause:
252                 /*
253                  * Rx Flow control is enabled and Tx Flow control is
254                  * disabled by software override. Since there really
255                  * isn't a way to advertise that we are capable of RX
256                  * Pause ONLY, we will advertise that we support both
257                  * symmetric and asymmetric Rx PAUSE, as such we fall
258                  * through to the fc_full statement.  Later, we will
259                  * disable the adapter's ability to send PAUSE frames.
260                  */
261         case ixgbe_fc_full:
262                 /* Flow control (both Rx and Tx) is enabled by SW override. */
263                 reg |= IXGBE_PCS1GANA_SYM_PAUSE | IXGBE_PCS1GANA_ASM_PAUSE;
264                 if (hw->phy.media_type == ixgbe_media_type_backplane)
265                         reg_bp |= IXGBE_AUTOC_SYM_PAUSE |
266                                   IXGBE_AUTOC_ASM_PAUSE;
267                 else if (hw->phy.media_type == ixgbe_media_type_copper)
268                         reg_cu |= IXGBE_TAF_SYM_PAUSE | IXGBE_TAF_ASM_PAUSE;
269                 break;
270         default:
271                 DEBUGOUT("Flow control param set incorrectly\n");
272                 ret_val = IXGBE_ERR_CONFIG;
273                 goto out;
274         }

276         if (hw->mac.type != ixgbe_mac_X540) {
277                 /*
278                  * Enable auto-negotiation between the MAC & PHY;
279                  * the MAC will advertise clause 37 flow control.
280                  */
281                 IXGBE_WRITE_REG(hw, IXGBE_PCS1GANA, reg);
282                 reg = IXGBE_READ_REG(hw, IXGBE_PCS1GLCTL);

284                 /* Disable AN timeout */
285                 if (hw->fc.strict_ieee)
286                         reg &= ~IXGBE_PCS1GLCTL_AN_1G_TIMEOUT_EN;

288                 IXGBE_WRITE_REG(hw, IXGBE_PCS1GLCTL, reg);
289                 DEBUGOUT1("Set up FC; PCS1GLCTL = 0x%08X\n", reg);
290         }

292         /*
293          * AUTOC restart handles negotiation of 1G and 10G on backplane
294          * and copper. There is no need to set the PCS1GCTL register.
295          *
296          */
297         if (hw->phy.media_type == ixgbe_media_type_backplane) {
298                 reg_bp |= IXGBE_AUTOC_AN_RESTART;
299                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, reg_bp);
300         } else if ((hw->phy.media_type == ixgbe_media_type_copper) &&
301                     (ixgbe_device_supports_autoneg_fc(hw) == IXGBE_SUCCESS)) {
302                 hw->phy.ops.write_reg(hw, IXGBE_MDIO_AUTO_NEG_ADVT,
303                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE, reg_cu);
304         }

306         DEBUGOUT1("Set up FC; IXGBE_AUTOC = 0x%08X\n", reg);
307 out:
308         return ret_val;
309 }

311 /**
312  *  ixgbe_start_hw_generic - Prepare hardware for Tx/Rx
313  *  @hw: pointer to hardware structure
```



```
314  *
315  *  Starts the hardware by filling the bus info structure and media type, clears
316  *  all on chip counters, initializes receive address registers, multicast
317  *  table, VLAN filter table, calls routine to set up link and flow control
318  *  settings, and leaves transmit and receive units disabled and uninitialized
319  **/
320 s32 ixgbe_start_hw_generic(struct ixgbe_hw *hw)
321 {
322         s32 ret_val;
323         u32 ctrl_ext;

325         DEBUGFUNC("ixgbe_start_hw_generic");

327         /* Set the media type */
328         hw->phy.media_type = hw->mac.ops.get_media_type(hw);

330         /* PHY ops initialization must be done in reset_hw() */

332         /* Clear the VLAN filter table */
333         hw->mac.ops.clear_vfta(hw);

335         /* Clear statistics registers */
336         hw->mac.ops.clear_hw_cntrs(hw);

338         /* Set No Snoop Disable */
339         ctrl_ext = IXGBE_READ_REG(hw, IXGBE_CTRL_EXT);
340         ctrl_ext |= IXGBE_CTRL_EXT_NS_DIS;
341         IXGBE_WRITE_REG(hw, IXGBE_CTRL_EXT, ctrl_ext);
342         IXGBE_WRITE_FLUSH(hw);

344         /* Setup flow control */
345         ret_val = ixgbe_setup_fc(hw);
346         if (ret_val != IXGBE_SUCCESS)
347                 goto out;
170         (void) ixgbe_setup_fc(hw, 0);

349         /* Clear adapter stopped flag */
350         hw->adapter_stopped = FALSE;

352 out:
353         return ret_val;
175         return IXGBE_SUCCESS;
354 }

356 /**
357  *  ixgbe_start_hw_gen2 - Init sequence for common device family
358  *  @hw: pointer to hw structure
359  *
360  * Performs the init sequence common to the second generation
361  * of 10 GbE devices.
362  * Devices in the second generation:
363  *     82599
364  *     X540
365  **/
366 s32 ixgbe_start_hw_gen2(struct ixgbe_hw *hw)
367 {
368         u32 i;
369         u32 regval;

371         /* Clear the rate limiters */
372         for (i = 0; i < hw->mac.max_tx_queues; i++) {
373                 IXGBE_WRITE_REG(hw, IXGBE_RTTDQSEL, i);
374                 IXGBE_WRITE_REG(hw, IXGBE_RTTBCNRC, 0);
375         }
376         IXGBE_WRITE_FLUSH(hw);
```

```
        /* Disable relaxed ordering */
        for (i = 0; i < hw->mac.max_tx_queues; i++) {
                regval = IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL_82599(i));
                regval &= ~IXGBE_DCA_TXCTRL_DESC_WRO_EN;
                regval &= ~IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
                IXGBE_WRITE_REG(hw, IXGBE_DCA_TXCTRL_82599(i), regval);
        }

        for (i = 0; i < hw->mac.max_rx_queues; i++) {
                regval = IXGBE_READ_REG(hw, IXGBE_DCA_RXCTRL(i));
                regval &= ~(IXGBE_DCA_RXCTRL_DATA_WRO_EN |
                            IXGBE_DCA_RXCTRL_HEAD_WRO_EN);
                regval &= ~(IXGBE_DCA_RXCTRL_DESC_WRO_EN |
                                    IXGBE_DCA_RXCTRL_DESC_HSRO_EN);
                IXGBE_WRITE_REG(hw, IXGBE_DCA_RXCTRL(i), regval);
        }

        return IXGBE_SUCCESS;
}
```

*unchanged_portion_omitted_*

```
/**
 *  ixgbe_clear_hw_cntrs_generic - Generic clear hardware counters
 *  @hw: pointer to hardware structure
 *
 *  Clears all hardware statistics counters by reading them from the hardware
 *  Statistics counters are clear on read.
 **/
s32 ixgbe_clear_hw_cntrs_generic(struct ixgbe_hw *hw)
{
        u16 i = 0;

        DEBUGFUNC("ixgbe_clear_hw_cntrs_generic");

        IXGBE_READ_REG(hw, IXGBE_CRCERRS);
        IXGBE_READ_REG(hw, IXGBE_ILLERRC);
        IXGBE_READ_REG(hw, IXGBE_ERRBC);
        IXGBE_READ_REG(hw, IXGBE_MSPDC);
        (void) IXGBE_READ_REG(hw, IXGBE_CRCERRS);
        (void) IXGBE_READ_REG(hw, IXGBE_ILLERRC);
        (void) IXGBE_READ_REG(hw, IXGBE_ERRBC);
        (void) IXGBE_READ_REG(hw, IXGBE_MSPDC);
        for (i = 0; i < 8; i++)
                IXGBE_READ_REG(hw, IXGBE_MPC(i));
                (void) IXGBE_READ_REG(hw, IXGBE_MPC(i));

        IXGBE_READ_REG(hw, IXGBE_MLFC);
        IXGBE_READ_REG(hw, IXGBE_MRFC);
        IXGBE_READ_REG(hw, IXGBE_RLEC);
        IXGBE_READ_REG(hw, IXGBE_LXONTXC);
        IXGBE_READ_REG(hw, IXGBE_LXOFFTXC);
        (void) IXGBE_READ_REG(hw, IXGBE_MLFC);
        (void) IXGBE_READ_REG(hw, IXGBE_MRFC);
        (void) IXGBE_READ_REG(hw, IXGBE_RLEC);
        (void) IXGBE_READ_REG(hw, IXGBE_LXONTXC);
        (void) IXGBE_READ_REG(hw, IXGBE_LXOFFTXC);
        if (hw->mac.type >= ixgbe_mac_82599EB) {
                IXGBE_READ_REG(hw, IXGBE_LXONRXCNT);
                IXGBE_READ_REG(hw, IXGBE_LXOFFRXCNT);
                (void) IXGBE_READ_REG(hw, IXGBE_LXONRXCNT);
                (void) IXGBE_READ_REG(hw, IXGBE_LXOFFRXCNT);
        } else {
                IXGBE_READ_REG(hw, IXGBE_LXONRXC);
                IXGBE_READ_REG(hw, IXGBE_LXOFFRXC);
                (void) IXGBE_READ_REG(hw, IXGBE_LXONRXC);
                (void) IXGBE_READ_REG(hw, IXGBE_LXOFFRXC);
```



```
        }

        for (i = 0; i < 8; i++) {
                IXGBE_READ_REG(hw, IXGBE_PXONTXC(i));
                IXGBE_READ_REG(hw, IXGBE_PXOFFTXC(i));
                (void) IXGBE_READ_REG(hw, IXGBE_PXONTXC(i));
                (void) IXGBE_READ_REG(hw, IXGBE_PXOFFTXC(i));
                if (hw->mac.type >= ixgbe_mac_82599EB) {
                        IXGBE_READ_REG(hw, IXGBE_PXONRXCNT(i));
                        IXGBE_READ_REG(hw, IXGBE_PXOFFRXCNT(i));
                        (void) IXGBE_READ_REG(hw, IXGBE_PXONRXCNT(i));
                        (void) IXGBE_READ_REG(hw, IXGBE_PXOFFRXCNT(i));
                } else {
                        IXGBE_READ_REG(hw, IXGBE_PXONRXC(i));
                        IXGBE_READ_REG(hw, IXGBE_PXOFFRXC(i));
                        (void) IXGBE_READ_REG(hw, IXGBE_PXONRXC(i));
                        (void) IXGBE_READ_REG(hw, IXGBE_PXOFFRXC(i));
                }
        }
        if (hw->mac.type >= ixgbe_mac_82599EB)
                for (i = 0; i < 8; i++)
                        IXGBE_READ_REG(hw, IXGBE_PXON2OFFCNT(i));
        IXGBE_READ_REG(hw, IXGBE_PRC64);
        IXGBE_READ_REG(hw, IXGBE_PRC127);
        IXGBE_READ_REG(hw, IXGBE_PRC255);
        IXGBE_READ_REG(hw, IXGBE_PRC511);
        IXGBE_READ_REG(hw, IXGBE_PRC1023);
        IXGBE_READ_REG(hw, IXGBE_PRC1522);
        IXGBE_READ_REG(hw, IXGBE_GPRC);
        IXGBE_READ_REG(hw, IXGBE_BPRC);
        IXGBE_READ_REG(hw, IXGBE_MPRC);
        IXGBE_READ_REG(hw, IXGBE_GPTC);
        IXGBE_READ_REG(hw, IXGBE_GORCL);
        IXGBE_READ_REG(hw, IXGBE_GORCH);
        IXGBE_READ_REG(hw, IXGBE_GOTCL);
        IXGBE_READ_REG(hw, IXGBE_GOTCH);
        if (hw->mac.type == ixgbe_mac_82598EB)
                        (void) IXGBE_READ_REG(hw, IXGBE_PXON2OFFCNT(i));
        (void) IXGBE_READ_REG(hw, IXGBE_PRC64);
        (void) IXGBE_READ_REG(hw, IXGBE_PRC127);
        (void) IXGBE_READ_REG(hw, IXGBE_PRC255);
        (void) IXGBE_READ_REG(hw, IXGBE_PRC511);
        (void) IXGBE_READ_REG(hw, IXGBE_PRC1023);
        (void) IXGBE_READ_REG(hw, IXGBE_PRC1522);
        (void) IXGBE_READ_REG(hw, IXGBE_GPRC);
        (void) IXGBE_READ_REG(hw, IXGBE_BPRC);
        (void) IXGBE_READ_REG(hw, IXGBE_MPRC);
        (void) IXGBE_READ_REG(hw, IXGBE_GPTC);
        (void) IXGBE_READ_REG(hw, IXGBE_GORCL);
        (void) IXGBE_READ_REG(hw, IXGBE_GORCH);
        (void) IXGBE_READ_REG(hw, IXGBE_GOTCL);
        (void) IXGBE_READ_REG(hw, IXGBE_GOTCH);
                for (i = 0; i < 8; i++)
                        IXGBE_READ_REG(hw, IXGBE_RNBC(i));
        IXGBE_READ_REG(hw, IXGBE_RUC);
        IXGBE_READ_REG(hw, IXGBE_RFC);
        IXGBE_READ_REG(hw, IXGBE_ROC);
        IXGBE_READ_REG(hw, IXGBE_RJC);
        IXGBE_READ_REG(hw, IXGBE_MNGPRC);
        IXGBE_READ_REG(hw, IXGBE_MNGPDC);
        IXGBE_READ_REG(hw, IXGBE_MNGPTC);
        IXGBE_READ_REG(hw, IXGBE_TORL);
        IXGBE_READ_REG(hw, IXGBE_TORH);
        IXGBE_READ_REG(hw, IXGBE_TPR);
        IXGBE_READ_REG(hw, IXGBE_TPT);
        IXGBE_READ_REG(hw, IXGBE_PTC64);
```

```
498         IXGBE_READ_REG(hw, IXGBE_PTC127);
499         IXGBE_READ_REG(hw, IXGBE_PTC255);
500         IXGBE_READ_REG(hw, IXGBE_PTC511);
501         IXGBE_READ_REG(hw, IXGBE_PTC1023);
502         IXGBE_READ_REG(hw, IXGBE_PTC1522);
503         IXGBE_READ_REG(hw, IXGBE_MPTC);
504         IXGBE_READ_REG(hw, IXGBE_BPTC);
306                 (void) IXGBE_READ_REG(hw, IXGBE_RNBC(i));
307         (void) IXGBE_READ_REG(hw, IXGBE_RUC);
308         (void) IXGBE_READ_REG(hw, IXGBE_RFC);
309         (void) IXGBE_READ_REG(hw, IXGBE_ROC);
310         (void) IXGBE_READ_REG(hw, IXGBE_RJC);
311         (void) IXGBE_READ_REG(hw, IXGBE_MNGPRC);
312         (void) IXGBE_READ_REG(hw, IXGBE_MNGPDC);
313         (void) IXGBE_READ_REG(hw, IXGBE_MNGPTC);
314         (void) IXGBE_READ_REG(hw, IXGBE_TORL);
315         (void) IXGBE_READ_REG(hw, IXGBE_TORH);
316         (void) IXGBE_READ_REG(hw, IXGBE_TPR);
317         (void) IXGBE_READ_REG(hw, IXGBE_TPT);
318         (void) IXGBE_READ_REG(hw, IXGBE_PTC64);
319         (void) IXGBE_READ_REG(hw, IXGBE_PTC127);
320         (void) IXGBE_READ_REG(hw, IXGBE_PTC255);
321         (void) IXGBE_READ_REG(hw, IXGBE_PTC511);
322         (void) IXGBE_READ_REG(hw, IXGBE_PTC1023);
323         (void) IXGBE_READ_REG(hw, IXGBE_PTC1522);
324         (void) IXGBE_READ_REG(hw, IXGBE_MPTC);
325         (void) IXGBE_READ_REG(hw, IXGBE_BPTC);
505         for (i = 0; i < 16; i++) {
506                 IXGBE_READ_REG(hw, IXGBE_QPRC(i));
507                 IXGBE_READ_REG(hw, IXGBE_QPTC(i));
327                 (void) IXGBE_READ_REG(hw, IXGBE_QPRC(i));
328                 (void) IXGBE_READ_REG(hw, IXGBE_QPTC(i));
508                 if (hw->mac.type >= ixgbe_mac_82599EB) {
509                         IXGBE_READ_REG(hw, IXGBE_QBRC_L(i));
510                         IXGBE_READ_REG(hw, IXGBE_QBRC_H(i));
511                         IXGBE_READ_REG(hw, IXGBE_QBTC_L(i));
512                         IXGBE_READ_REG(hw, IXGBE_QBTC_H(i));
513                         IXGBE_READ_REG(hw, IXGBE_QPRDC(i));
330                         (void) IXGBE_READ_REG(hw, IXGBE_QBRC_L(i));
331                         (void) IXGBE_READ_REG(hw, IXGBE_QBRC_H(i));
332                         (void) IXGBE_READ_REG(hw, IXGBE_QBTC_L(i));
333                         (void) IXGBE_READ_REG(hw, IXGBE_QBTC_H(i));
334                         (void) IXGBE_READ_REG(hw, IXGBE_QPRDC(i));
514                 } else {
515                         IXGBE_READ_REG(hw, IXGBE_QBRC(i));
516                         IXGBE_READ_REG(hw, IXGBE_QBTC(i));
336                         (void) IXGBE_READ_REG(hw, IXGBE_QBRC(i));
337                         (void) IXGBE_READ_REG(hw, IXGBE_QBTC(i));
517                 }
518         }

520         if (hw->mac.type == ixgbe_mac_X540) {
521                 if (hw->phy.id == 0)
522                         ixgbe_identify_phy(hw);
523                 hw->phy.ops.read_reg(hw, IXGBE_PCRC8ECL,
524                                      IXGBE_MDIO_PCS_DEV_TYPE, &i);
525                 hw->phy.ops.read_reg(hw, IXGBE_PCRC8ECH,
526                                      IXGBE_MDIO_PCS_DEV_TYPE, &i);
527                 hw->phy.ops.read_reg(hw, IXGBE_LDPCECL,
528                                      IXGBE_MDIO_PCS_DEV_TYPE, &i);
529                 hw->phy.ops.read_reg(hw, IXGBE_LDPCECH,
530                                      IXGBE_MDIO_PCS_DEV_TYPE, &i);
531         }

533         return IXGBE_SUCCESS;
534 }
```

*unchanged_portion_omitted_*



```
647 /**
456  *  ixgbe_read_pba_length_generic - Reads part number length from EEPROM
457  *  @hw: pointer to hardware structure
458  *  @pba_num_size: part number string buffer length
459  *
460  *  Reads the part number length from the EEPROM.
461  *  Returns expected buffer size in pba_num_size
462  **/
463 s32 ixgbe_read_pba_length_generic(struct ixgbe_hw *hw, u32 *pba_num_size)
464 {
465         s32 ret_val;
466         u16 data;
467         u16 pba_ptr;
468         u16 length;

470         DEBUGFUNC("ixgbe_read_pba_length_generic");

472         if (pba_num_size == NULL) {
473                 DEBUGOUT("PBA buffer size was null\n");
474                 return IXGBE_ERR_INVALID_ARGUMENT;
475         }

477         ret_val = hw->eeprom.ops.read(hw, IXGBE_PBANUM0_PTR, &data);
478         if (ret_val) {
479                 DEBUGOUT("NVM Read Error\n");
480                 return ret_val;
481         }

483         ret_val = hw->eeprom.ops.read(hw, IXGBE_PBANUM1_PTR, &pba_ptr);
484         if (ret_val) {
485                 DEBUGOUT("NVM Read Error\n");
486                 return ret_val;
487         }

489          /* if data is not ptr guard the PBA must be in legacy format */
490         if (data != IXGBE_PBANUM_PTR_GUARD) {
491                 *pba_num_size = 11;
492                 return IXGBE_SUCCESS;
493         }

495         ret_val = hw->eeprom.ops.read(hw, pba_ptr, &length);
496         if (ret_val) {
497                 DEBUGOUT("NVM Read Error\n");
498                 return ret_val;
499         }

501         if (length == 0xFFFF || length == 0) {
502                 DEBUGOUT("NVM PBA number section invalid length\n");
503                 return IXGBE_ERR_PBA_SECTION;
504         }

506         /*
507          * Convert from length in u16 values to u8 chars, add 1 for NULL,
508          * and subtract 2 because length field is included in length.
509          */
510         *pba_num_size = ((u32)length * 2) - 1;

512         return IXGBE_SUCCESS;
513 }

515 /**
648  *  ixgbe_read_pba_num_generic - Reads part number from EEPROM
649  *  @hw: pointer to hardware structure
650  *  @pba_num: stores the part number from the EEPROM
651  *
```

```
 652  *  Reads the part number from the EEPROM.
 653  **/
 654 s32 ixgbe_read_pba_num_generic(struct ixgbe_hw *hw, u32 *pba_num)
 655 {
 656         s32 ret_val;
 657         u16 data;

 659         DEBUGFUNC("ixgbe_read_pba_num_generic");

 661         ret_val = hw->eeprom.ops.read(hw, IXGBE_PBANUM0_PTR, &data);
 662         if (ret_val) {
 663                 DEBUGOUT("NVM Read Error\n");
 664                 return ret_val;
 665         } else if (data == IXGBE_PBANUM_PTR_GUARD) {
 666                 DEBUGOUT("NVM Not supported\n");
 667                 return IXGBE_NOT_IMPLEMENTED;
 668         }
 669         *pba_num = (u32)(data << 16);

 671         ret_val = hw->eeprom.ops.read(hw, IXGBE_PBANUM1_PTR, &data);
 672         if (ret_val) {
 673                 DEBUGOUT("NVM Read Error\n");
 674                 return ret_val;
 675         }
 676         *pba_num |= data;

 678         return IXGBE_SUCCESS;
 679 }
_____unchanged_portion_omitted_

 710 /**
 711  *  ixgbe_get_bus_info_generic - Generic set PCI bus info
 712  *  @hw: pointer to hardware structure
 713  *
 714  *  Sets the PCI bus info (speed, width, type) within the ixgbe_hw structure
 715  **/
 716 s32 ixgbe_get_bus_info_generic(struct ixgbe_hw *hw)
 717 {
 718         struct ixgbe_mac_info *mac = &hw->mac;
 719         u16 link_status;

 721         DEBUGFUNC("ixgbe_get_bus_info_generic");

 723         hw->bus.type = ixgbe_bus_type_pci_express;

 725         /* Get the negotiated link width and speed from PCI config space */
 726         link_status = IXGBE_READ_PCIE_WORD(hw, IXGBE_PCI_LINK_STATUS);

 728         switch (link_status & IXGBE_PCI_LINK_WIDTH) {
 729         case IXGBE_PCI_LINK_WIDTH_1:
 730                 hw->bus.width = ixgbe_bus_width_pcie_x1;
 731                 break;
 732         case IXGBE_PCI_LINK_WIDTH_2:
 733                 hw->bus.width = ixgbe_bus_width_pcie_x2;
 734                 break;
 735         case IXGBE_PCI_LINK_WIDTH_4:
 736                 hw->bus.width = ixgbe_bus_width_pcie_x4;
 737                 break;
 738         case IXGBE_PCI_LINK_WIDTH_8:
 739                 hw->bus.width = ixgbe_bus_width_pcie_x8;
 740                 break;
 741         default:
 742                 hw->bus.width = ixgbe_bus_width_unknown;
 743                 break;
 744         }
```



```
 746         switch (link_status & IXGBE_PCI_LINK_SPEED) {
 747         case IXGBE_PCI_LINK_SPEED_2500:
 748                 hw->bus.speed = ixgbe_bus_speed_2500;
 749                 break;
 750         case IXGBE_PCI_LINK_SPEED_5000:
 751                 hw->bus.speed = ixgbe_bus_speed_5000;
 752                 break;
 753         case IXGBE_PCI_LINK_SPEED_8000:
 754                 hw->bus.speed = ixgbe_bus_speed_8000;
 755                 break;
 756         default:
 757                 hw->bus.speed = ixgbe_bus_speed_unknown;
 758                 break;
 759         }

 761         mac->ops.set_lan_id(hw);

 763         return IXGBE_SUCCESS;
 764 }
_____unchanged_portion_omitted_

 790 /**
 791  *  ixgbe_stop_adapter_generic - Generic stop Tx/Rx units
 792  *  @hw: pointer to hardware structure
 793  *
 794  *  Sets the adapter_stopped flag within ixgbe_hw struct. Clears interrupts,
 795  *  disables transmit and receive units. The adapter_stopped flag is used by
 796  *  the shared code and drivers to determine if the adapter is in a stopped
 797  *  state and should not touch the hardware.
 798  **/
 799 s32 ixgbe_stop_adapter_generic(struct ixgbe_hw *hw)
 800 {
 666         u32 number_of_queues;
 801         u32 reg_val;
 802         u16 i;

 804         DEBUGFUNC("ixgbe_stop_adapter_generic");

 806         /*
 807          * Set the adapter_stopped flag so other driver functions stop touching
 808          * the hardware
 809          */
 810         hw->adapter_stopped = TRUE;

 812         /* Disable the receive unit */
 813         IXGBE_WRITE_REG(hw, IXGBE_RXCTRL, 0);
 679         reg_val = IXGBE_READ_REG(hw, IXGBE_RXCTRL);
 680         reg_val &= ~(IXGBE_RXCTRL_RXEN);
 681         IXGBE_WRITE_REG(hw, IXGBE_RXCTRL, reg_val);
 682         IXGBE_WRITE_FLUSH(hw);
 683         msec_delay(2);

 815         /* Clear interrupt mask to stop interrupts from being generated */
 685         /* Clear interrupt mask to stop from interrupts being generated */
 816         IXGBE_WRITE_REG(hw, IXGBE_EIMC, IXGBE_IRQ_CLEAR_MASK);

 818         /* Clear any pending interrupts, flush previous writes */
 819         IXGBE_READ_REG(hw, IXGBE_EICR);
 688         /* Clear any pending interrupts */
 689         (void) IXGBE_READ_REG(hw, IXGBE_EICR);

 821         /* Disable the transmit unit.  Each queue must be disabled. */
 822         for (i = 0; i < hw->mac.max_tx_queues; i++)
 823                 IXGBE_WRITE_REG(hw, IXGBE_TXDCTL(i), IXGBE_TXDCTL_SWFLSH);

 825         /* Disable the receive unit by stopping each queue */
```

```
826         for (i = 0; i < hw->mac.max_rx_queues; i++) {
827                 reg_val = IXGBE_READ_REG(hw, IXGBE_RXDCTL(i));
828                 reg_val &= ~IXGBE_RXDCTL_ENABLE;
829                 reg_val |= IXGBE_RXDCTL_SWFLSH;
830                 IXGBE_WRITE_REG(hw, IXGBE_RXDCTL(i), reg_val);
692         number_of_queues = hw->mac.max_tx_queues;
693         for (i = 0; i < number_of_queues; i++) {
694                 reg_val = IXGBE_READ_REG(hw, IXGBE_TXDCTL(i));
695                 if (reg_val & IXGBE_TXDCTL_ENABLE) {
696                         reg_val &= ~IXGBE_TXDCTL_ENABLE;
697                         IXGBE_WRITE_REG(hw, IXGBE_TXDCTL(i), reg_val);
831         }
699         }

833         /* flush all queues disables */
834         IXGBE_WRITE_FLUSH(hw);
835         msec_delay(2);

837         /*
838          * Prevent the PCI-E bus from from hanging by disabling PCI-E master
839          * access and verify no pending requests
840          */
841         return ixgbe_disable_pcie_master(hw);
705         (void) ixgbe_disable_pcie_master(hw);

707         return IXGBE_SUCCESS;
842 }
_____unchanged_portion_omitted_

884 /**
885  *  ixgbe_init_eeprom_params_generic - Initialize EEPROM params
886  *  @hw: pointer to hardware structure
887  *
888  *  Initializes the EEPROM parameters ixgbe_eeprom_info within the
889  *  ixgbe_hw struct in order to set up EEPROM access.
890  **/
891 s32 ixgbe_init_eeprom_params_generic(struct ixgbe_hw *hw)
892 {
893         struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
894         u32 eec;
895         u16 eeprom_size;

897         DEBUGFUNC("ixgbe_init_eeprom_params_generic");

899         if (eeprom->type == ixgbe_eeprom_uninitialized) {
900                 eeprom->type = ixgbe_eeprom_none;
901                 /* Set default semaphore delay to 10ms which is a well
902                  * tested value */
903                 eeprom->semaphore_delay = 10;
904                 /* Clear EEPROM page size, it will be initialized as needed */
905                 eeprom->word_page_size = 0;

907                 /*
908                  * Check for EEPROM present first.
909                  * If not present leave as none
910                  */
911                 eec = IXGBE_READ_REG(hw, IXGBE_EEC);
912                 if (eec & IXGBE_EEC_PRES) {
913                         eeprom->type = ixgbe_eeprom_spi;

915                         /*
916                          * SPI EEPROM is assumed here.  This code would need to
917                          * change if a future EEPROM is not SPI.
918                          */
919                         eeprom_size = (u16)((eec & IXGBE_EEC_SIZE) >>
920                                             IXGBE_EEC_SIZE_SHIFT);
```



```
921                         eeprom->word_size = 1 << (eeprom_size +
922                                                   IXGBE_EEPROM_WORD_SIZE_SHIFT);
786                                                   IXGBE_EEPROM_WORD_SIZE_BASE_SHIFT);
923                 }

925                 if (eec & IXGBE_EEC_ADDR_SIZE)
926                         eeprom->address_bits = 16;
927                 else
928                         eeprom->address_bits = 8;
929                 DEBUGOUT3("Eeprom params: type = %d, size = %d, address bits: "
930                           "%d\n", eeprom->type, eeprom->word_size,
931                           eeprom->address_bits);
932         }

934         return IXGBE_SUCCESS;
935 }

937 /**
938  *  ixgbe_write_eeprom_buffer_bit_bang_generic - Write EEPROM using bit-bang
802  *  ixgbe_write_eeprom_generic - Writes 16 bit value to EEPROM
939  *  @hw: pointer to hardware structure
940  *  @offset: offset within the EEPROM to write
941  *  @words: number of word(s)
942  *  @data: 16 bit word(s) to write to EEPROM
804  *  @offset: offset within the EEPROM to be written to
805  *  @data: 16 bit word to be written to the EEPROM
943  *
944  *  Reads 16 bit word(s) from EEPROM through bit-bang method
807  *  If ixgbe_eeprom_update_checksum is not called after this function, the
808  *  EEPROM will most likely contain an invalid checksum.
945  **/
946 s32 ixgbe_write_eeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
947                                                u16 words, u16 *data)
810 s32 ixgbe_write_eeprom_generic(struct ixgbe_hw *hw, u16 offset, u16 data)
948 {
949         s32 status = IXGBE_SUCCESS;
950         u16 i, count;
812         s32 status;
813         u8 write_opcode = IXGBE_EEPROM_WRITE_OPCODE_SPI;

952         DEBUGFUNC("ixgbe_write_eeprom_buffer_bit_bang_generic");
815         DEBUGFUNC("ixgbe_write_eeprom_generic");

954         hw->eeprom.ops.init_params(hw);

956         if (words == 0) {
957                 status = IXGBE_ERR_INVALID_ARGUMENT;
958                 goto out;
959         }

961         if (offset + words > hw->eeprom.word_size) {
819         if (offset >= hw->eeprom.word_size) {
962                 status = IXGBE_ERR_EEPROM;
963                 goto out;
964         }

966         /*
967          * The EEPROM page size cannot be queried from the chip. We do lazy
968          * initialization. It is worth to do that when we write large buffer.
969          */
970         if ((hw->eeprom.word_page_size == 0) &&
971             (words > IXGBE_EEPROM_PAGE_SIZE_MAX))
972                 ixgbe_detect_eeprom_page_size_generic(hw, offset);

974         /*
975          * We cannot hold synchronization semaphores for too long
```

```
 976          * to avoid other entity starvation. However it is more efficient
 977          * to read in bursts than synchronizing access for each word.
 978          */
 979         for (i = 0; i < words; i += IXGBE_EEPROM_RD_BUFFER_MAX_COUNT) {
 980                 count = (words - i) / IXGBE_EEPROM_RD_BUFFER_MAX_COUNT > 0 ?
 981                         IXGBE_EEPROM_RD_BUFFER_MAX_COUNT : (words - i);
 982                 status = ixgbe_write_eeprom_buffer_bit_bang(hw, offset + i,
 983                                                             count, &data[i]);

 985                 if (status != IXGBE_SUCCESS)
 986                         break;
 987         }

 989 out:
 990         return status;
 991 }

 993 /**
 994  *  ixgbe_write_eeprom_buffer_bit_bang - Writes 16 bit word(s) to EEPROM
 995  *  @hw: pointer to hardware structure
 996  *  @offset: offset within the EEPROM to be written to
 997  *  @words: number of word(s)
 998  *  @data: 16 bit word(s) to be written to the EEPROM
 999  *
1000  *  If ixgbe_eeprom_update_checksum is not called after this function, the
1001  *  EEPROM will most likely contain an invalid checksum.
1002  **/
1003 static s32 ixgbe_write_eeprom_buffer_bit_bang(struct ixgbe_hw *hw, u16 offset,
1004                                               u16 words, u16 *data)
1005 {
1006         s32 status;
1007         u16 word;
1008         u16 page_size;
1009         u16 i;
1010         u8 write_opcode = IXGBE_EEPROM_WRITE_OPCODE_SPI;

1012         DEBUGFUNC("ixgbe_write_eeprom_buffer_bit_bang");

1014         /* Prepare the EEPROM for writing  */
1015         status = ixgbe_acquire_eeprom(hw);

1017         if (status == IXGBE_SUCCESS) {
1018                 if (ixgbe_ready_eeprom(hw) != IXGBE_SUCCESS) {
1019                         ixgbe_release_eeprom(hw);
1020                         status = IXGBE_ERR_EEPROM;
1021                 }
1022         }

1024         if (status == IXGBE_SUCCESS) {
1025                 for (i = 0; i < words; i++) {
1026                         ixgbe_standby_eeprom(hw);

1028                         /*  Send the WRITE ENABLE command (8 bit opcode )  */
1029                         ixgbe_shift_out_eeprom_bits(hw,
1030                                                     IXGBE_EEPROM_WREN_OPCODE_SPI,
 838                 ixgbe_shift_out_eeprom_bits(hw, IXGBE_EEPROM_WREN_OPCODE_SPI,
1031                                                     IXGBE_EEPROM_OPCODE_BITS);

1033                         ixgbe_standby_eeprom(hw);

1035                         /*
1036                          * Some SPI eeproms use the 8th address bit embedded
1037                          * in the opcode
 844                  * Some SPI eeproms use the 8th address bit embedded in the
 845                  * opcode
1038                          */
```



```
1039                         if ((hw->eeprom.address_bits == 8) &&
1040                             ((offset + i) >= 128))
 847                 if ((hw->eeprom.address_bits == 8) && (offset >= 128))
1041                         write_opcode |= IXGBE_EEPROM_A8_OPCODE_SPI;

1043                         /* Send the Write command (8-bit opcode + addr) */
1044                         ixgbe_shift_out_eeprom_bits(hw, write_opcode,
1045                                                     IXGBE_EEPROM_OPCODE_BITS);
1046                         ixgbe_shift_out_eeprom_bits(hw, (u16)((offset + i) * 2),
 853                 ixgbe_shift_out_eeprom_bits(hw, (u16)(offset*2),
1047                                                     hw->eeprom.address_bits);

1049                         page_size = hw->eeprom.word_page_size;

1051                         /* Send the data in burst via SPI*/
1052                         do {
1053                                 word = data[i];
1054                                 word = (word >> 8) | (word << 8);
1055                                 ixgbe_shift_out_eeprom_bits(hw, word, 16);

1057                                 if (page_size == 0)
1058                                         break;

1060                                 /* do not wrap around page */
1061                                 if (((offset + i) & (page_size - 1)) ==
1062                                     (page_size - 1))
1063                                         break;
1064                         } while (++i < words);

 856                 /* Send the data */
 857                 data = (data >> 8) | (data << 8);
 858                 ixgbe_shift_out_eeprom_bits(hw, data, 16);
1066                         ixgbe_standby_eeprom(hw);
1067                         msec_delay(10);
1068                 }

1069                 /* Done with writing - release the EEPROM */
1070                 ixgbe_release_eeprom(hw);
1071         }

 865 out:
1073         return status;
1074 }

1076 /**
1077  *  ixgbe_write_eeprom_generic - Writes 16 bit value to EEPROM
 870  *  ixgbe_read_eeprom_bit_bang_generic - Read EEPROM word using bit-bang
1078  *  @hw: pointer to hardware structure
1079  *  @offset: offset within the EEPROM to be written to
1080  *  @data: 16 bit word to be written to the EEPROM
 872  *  @offset: offset within the EEPROM to be read
 873  *  @data: read 16 bit value from EEPROM
1081  *
1082  *  If ixgbe_eeprom_update_checksum is not called after this function, the
1083  *  EEPROM will most likely contain an invalid checksum.
 875  *  Reads 16 bit value from EEPROM through bit-bang method
1084  **/
1085 s32 ixgbe_write_eeprom_generic(struct ixgbe_hw *hw, u16 offset, u16 data)
 877 s32 ixgbe_read_eeprom_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
 878                                        u16 *data)
1086 {
1087         s32 status;
 881         u16 word_in;
 882         u8 read_opcode = IXGBE_EEPROM_READ_OPCODE_SPI;

1089         DEBUGFUNC("ixgbe_write_eeprom_generic");
```

```
 884         DEBUGFUNC("ixgbe_read_eeprom_bit_bang_generic");

1091         hw->eeprom.ops.init_params(hw);

1093         if (offset >= hw->eeprom.word_size) {
1094                 status = IXGBE_ERR_EEPROM;
1095                 goto out;
1096         }

1098         status = ixgbe_write_eeprom_buffer_bit_bang(hw, offset, 1, &data);

1100 out:
1101         return status;
1102 }

1104 /**
1105  *  ixgbe_read_eeprom_buffer_bit_bang_generic - Read EEPROM using bit-bang
1106  *  @hw: pointer to hardware structure
1107  *  @offset: offset within the EEPROM to be read
1108  *  @data: read 16 bit words(s) from EEPROM
1109  *  @words: number of word(s)
1110  *
1111  *  Reads 16 bit word(s) from EEPROM through bit-bang method
1112  **/
1113 s32 ixgbe_read_eeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
1114                                               u16 words, u16 *data)
1115 {
1116         s32 status = IXGBE_SUCCESS;
1117         u16 i, count;

1119         DEBUGFUNC("ixgbe_read_eeprom_buffer_bit_bang_generic");

1121         hw->eeprom.ops.init_params(hw);

1123         if (words == 0) {
1124                 status = IXGBE_ERR_INVALID_ARGUMENT;
1125                 goto out;
1126         }

1128         if (offset + words > hw->eeprom.word_size) {
1129                 status = IXGBE_ERR_EEPROM;
1130                 goto out;
1131         }

1133         /*
1134          * We cannot hold synchronization semaphores for too long
1135          * to avoid other entity starvation. However it is more efficient
1136          * to read in bursts than synchronizing access for each word.
1137          */
1138         for (i = 0; i < words; i += IXGBE_EEPROM_RD_BUFFER_MAX_COUNT) {
1139                 count = (words - i) / IXGBE_EEPROM_RD_BUFFER_MAX_COUNT > 0 ?
1140                         IXGBE_EEPROM_RD_BUFFER_MAX_COUNT : (words - i);

1142                 status = ixgbe_read_eeprom_buffer_bit_bang(hw, offset + i,
1143                                                            count, &data[i]);

1145                 if (status != IXGBE_SUCCESS)
1146                         break;
1147         }

1149 out:
1150         return status;
1151 }

1153 /**
1154  *  ixgbe_read_eeprom_buffer_bit_bang - Read EEPROM using bit-bang
```



```
1155  *  @hw: pointer to hardware structure
1156  *  @offset: offset within the EEPROM to be read
1157  *  @words: number of word(s)
1158  *  @data: read 16 bit word(s) from EEPROM
1159  *
1160  *  Reads 16 bit word(s) from EEPROM through bit-bang method
1161  **/
1162 static s32 ixgbe_read_eeprom_buffer_bit_bang(struct ixgbe_hw *hw, u16 offset,
1163                                              u16 words, u16 *data)
1164 {
1165         s32 status;
1166         u16 word_in;
1167         u8 read_opcode = IXGBE_EEPROM_READ_OPCODE_SPI;
1168         u16 i;

1170         DEBUGFUNC("ixgbe_read_eeprom_buffer_bit_bang");

1172         /* Prepare the EEPROM for reading  */
1173         status = ixgbe_acquire_eeprom(hw);

1175         if (status == IXGBE_SUCCESS) {
1176                 if (ixgbe_ready_eeprom(hw) != IXGBE_SUCCESS) {
1177                         ixgbe_release_eeprom(hw);
1178                         status = IXGBE_ERR_EEPROM;
1179                 }
1180         }

1182         if (status == IXGBE_SUCCESS) {
1183                 for (i = 0; i < words; i++) {
1184                         ixgbe_standby_eeprom(hw);

1185                         /*
1186                          * Some SPI eeproms use the 8th address bit embedded
1187                          * in the opcode
 907                  * Some SPI eeproms use the 8th address bit embedded in the
 908                  * opcode
1188                          */
1189                         if ((hw->eeprom.address_bits == 8) &&
1190                             ((offset + i) >= 128))
 910                 if ((hw->eeprom.address_bits == 8) && (offset >= 128))
1191                                 read_opcode |= IXGBE_EEPROM_A8_OPCODE_SPI;

1193                         /* Send the READ command (opcode + addr) */
1194                         ixgbe_shift_out_eeprom_bits(hw, read_opcode,
1195                                                     IXGBE_EEPROM_OPCODE_BITS);
1196                         ixgbe_shift_out_eeprom_bits(hw, (u16)((offset + i) * 2),
 916                 ixgbe_shift_out_eeprom_bits(hw, (u16)(offset*2),
1197                                                     hw->eeprom.address_bits);

1199                         /* Read the data. */
1200                         word_in = ixgbe_shift_in_eeprom_bits(hw, 16);
1201                         data[i] = (word_in >> 8) | (word_in << 8);
1202                 }
 921                 *data = (word_in >> 8) | (word_in << 8);

1204                 /* End this read operation */
1205                 ixgbe_release_eeprom(hw);
1206         }

1208         return status;
1209 }

1211 /**
1212  *  ixgbe_read_eeprom_bit_bang_generic - Read EEPROM word using bit-bang
1213  *  @hw: pointer to hardware structure
1214  *  @offset: offset within the EEPROM to be read
```

```
 *  @data: read 16 bit value from EEPROM
 *
 *  Reads 16 bit value from EEPROM through bit-bang method
 **/
s32 ixgbe_read_eeprom_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
                                       u16 *data)
{
        s32 status;

        DEBUGFUNC("ixgbe_read_eeprom_bit_bang_generic");

        hw->eeprom.ops.init_params(hw);

        if (offset >= hw->eeprom.word_size) {
                status = IXGBE_ERR_EEPROM;
                goto out;
        }

        status = ixgbe_read_eeprom_buffer_bit_bang(hw, offset, 1, data);

out:
        return status;
}

/**
 *  ixgbe_read_eerd_buffer_generic - Read EEPROM word(s) using EERD
 *  ixgbe_read_eerd_generic - Read EEPROM word using EERD
 *  @hw: pointer to hardware structure
 *  @offset: offset of word in the EEPROM to read
 *  @words: number of word(s)
 *  @data: 16 bit word(s) from the EEPROM
 *  @data: word read from the EEPROM
 *
 *  Reads a 16 bit word(s) from the EEPROM using the EERD register.
 *  Reads a 16 bit word from the EEPROM using the EERD register.
 **/
s32 ixgbe_read_eerd_buffer_generic(struct ixgbe_hw *hw, u16 offset,
                                   u16 words, u16 *data)
s32 ixgbe_read_eerd_generic(struct ixgbe_hw *hw, u16 offset, u16 *data)
{
        u32 eerd;
        s32 status = IXGBE_SUCCESS;
        u32 i;
        s32 status;

        DEBUGFUNC("ixgbe_read_eerd_buffer_generic");
        DEBUGFUNC("ixgbe_read_eerd_generic");

        hw->eeprom.ops.init_params(hw);

        if (words == 0) {
                status = IXGBE_ERR_INVALID_ARGUMENT;
                goto out;
        }

        if (offset >= hw->eeprom.word_size) {
                status = IXGBE_ERR_EEPROM;
                goto out;
        }

        for (i = 0; i < words; i++) {
                eerd = ((offset + i) << IXGBE_EEPROM_RW_ADDR_SHIFT) +
        eerd = (offset << IXGBE_EEPROM_RW_ADDR_SHIFT) +
                       IXGBE_EEPROM_RW_REG_START;

                IXGBE_WRITE_REG(hw, IXGBE_EERD, eerd);
```



```
                status = ixgbe_poll_eerd_eewr_done(hw, IXGBE_NVM_POLL_READ);

                if (status == IXGBE_SUCCESS) {
                        data[i] = (IXGBE_READ_REG(hw, IXGBE_EERD) >>
        if (status == IXGBE_SUCCESS)
                        *data = (IXGBE_READ_REG(hw, IXGBE_EERD) >>
                                 IXGBE_EEPROM_RW_REG_DATA);
                } else {
        else
                        DEBUGOUT("Eeprom read timed out\n");
                        goto out;
                }
        }
out:
        return status;
}

/**
 *  ixgbe_detect_eeprom_page_size_generic - Detect EEPROM page size
 *  @hw: pointer to hardware structure
 *  @offset: offset within the EEPROM to be used as a scratch pad
 *
 *  Discover EEPROM page size by writing marching data at given offset.
 *  This function is called only when we are writing a new large buffer
 *  at given offset so the data would be overwritten anyway.
 **/
static s32 ixgbe_detect_eeprom_page_size_generic(struct ixgbe_hw *hw,
                                                 u16 offset)
{
        u16 data[IXGBE_EEPROM_PAGE_SIZE_MAX];
        s32 status = IXGBE_SUCCESS;
        u16 i;

        DEBUGFUNC("ixgbe_detect_eeprom_page_size_generic");

        for (i = 0; i < IXGBE_EEPROM_PAGE_SIZE_MAX; i++)
                data[i] = i;

        hw->eeprom.word_page_size = IXGBE_EEPROM_PAGE_SIZE_MAX;
        status = ixgbe_write_eeprom_buffer_bit_bang(hw, offset,
                                                    IXGBE_EEPROM_PAGE_SIZE_MAX, data);
        hw->eeprom.word_page_size = 0;
        if (status != IXGBE_SUCCESS)
                goto out;

        status = ixgbe_read_eeprom_buffer_bit_bang(hw, offset, 1, data);
        if (status != IXGBE_SUCCESS)
                goto out;

        /*
         * When writing in burst more than the actual page size
         * EEPROM address wraps around current page.
         */
        hw->eeprom.word_page_size = IXGBE_EEPROM_PAGE_SIZE_MAX - data[0];

        DEBUGOUT1("Detected EEPROM page size = %d words.",
                  hw->eeprom.word_page_size);
out:
        return status;
}

/**
 *  ixgbe_read_eerd_generic - Read EEPROM word using EERD
 *  ixgbe_write_eewr_generic - Write EEPROM word using EEWR
 *  @hw: pointer to hardware structure
 *  @offset: offset of  word in the EEPROM to read
```

```
 *  @data: word read from the EEPROM
 *
 *  Reads a 16 bit word from the EEPROM using the EERD register.
 **/
s32 ixgbe_read_eerd_generic(struct ixgbe_hw *hw, u16 offset, u16 *data)
{
        return ixgbe_read_eerd_buffer_generic(hw, offset, 1, data);
}

/**
 *  ixgbe_write_eewr_buffer_generic - Write EEPROM word(s) using EEWR
 *  @hw: pointer to hardware structure
 *  @offset: offset of  word in the EEPROM to write
 *  @words: number of word(s)
 *  @data: word(s) write to the EEPROM
 *  @data: word write to the EEPROM
 *
 *  Write a 16 bit word(s) to the EEPROM using the EEWR register.
 *  Write a 16 bit word to the EEPROM using the EEWR register.
 **/
s32 ixgbe_write_eewr_buffer_generic(struct ixgbe_hw *hw, u16 offset,
                                    u16 words, u16 *data)
s32 ixgbe_write_eewr_generic(struct ixgbe_hw *hw, u16 offset, u16 data)
{
        u32 eewr;
        s32 status = IXGBE_SUCCESS;
        u16 i;
        s32 status;

        DEBUGFUNC("ixgbe_write_eewr_generic");

        hw->eeprom.ops.init_params(hw);

        if (words == 0) {
                status = IXGBE_ERR_INVALID_ARGUMENT;
                goto out;
        }

        if (offset >= hw->eeprom.word_size) {
                status = IXGBE_ERR_EEPROM;
                goto out;
        }

        for (i = 0; i < words; i++) {
                eewr = ((offset + i) << IXGBE_EEPROM_RW_ADDR_SHIFT) |
                       (data[i] << IXGBE_EEPROM_RW_REG_DATA) |
                       IXGBE_EEPROM_RW_REG_START;
        eewr = (offset << IXGBE_EEPROM_RW_ADDR_SHIFT) |
               (data << IXGBE_EEPROM_RW_REG_DATA) | IXGBE_EEPROM_RW_REG_START;

                status = ixgbe_poll_eerd_eewr_done(hw, IXGBE_NVM_POLL_WRITE);
                if (status != IXGBE_SUCCESS) {
                        DEBUGOUT("Eeprom write EEWR timed out\n");
                        goto out;
                }

                IXGBE_WRITE_REG(hw, IXGBE_EEWR, eewr);

                status = ixgbe_poll_eerd_eewr_done(hw, IXGBE_NVM_POLL_WRITE);
                if (status != IXGBE_SUCCESS) {
                        DEBUGOUT("Eeprom write EEWR timed out\n");
                        goto out;
                }
        }

out:
```



```
        return status;
}

/**
 *  ixgbe_write_eewr_generic - Write EEPROM word using EEWR
 *  @hw: pointer to hardware structure
 *  @offset: offset of  word in the EEPROM to write
 *  @data: word write to the EEPROM
 *
 *  Write a 16 bit word to the EEPROM using the EEWR register.
 **/
s32 ixgbe_write_eewr_generic(struct ixgbe_hw *hw, u16 offset, u16 data)
{
        return ixgbe_write_eewr_buffer_generic(hw, offset, 1, &data);
}

/**
 *  ixgbe_poll_eerd_eewr_done - Poll EERD read or EEWR write status
 *  @hw: pointer to hardware structure
 *  @ee_reg: EEPROM flag for polling
 *
 *  Polls the status bit (bit 1) of the EERD or EEWR to determine when the
 *  read or write is done respectively.
 **/
s32 ixgbe_poll_eerd_eewr_done(struct ixgbe_hw *hw, u32 ee_reg)
{
        u32 i;
        u32 reg;
        s32 status = IXGBE_ERR_EEPROM;

        DEBUGFUNC("ixgbe_poll_eerd_eewr_done");

        for (i = 0; i < IXGBE_EERD_EEWR_ATTEMPTS; i++) {
                if (ee_reg == IXGBE_NVM_POLL_READ)
                        reg = IXGBE_READ_REG(hw, IXGBE_EERD);
                else
                        reg = IXGBE_READ_REG(hw, IXGBE_EEWR);

                if (reg & IXGBE_EEPROM_RW_REG_DONE) {
                        status = IXGBE_SUCCESS;
                        break;
                }
                usec_delay(5);
        }
        return status;
}

/**
 *  ixgbe_acquire_eeprom - Acquire EEPROM using bit-bang
 *  @hw: pointer to hardware structure
 *
 *  Prepares EEPROM for access using bit-bang method. This function should
 *  be called before issuing a command to the EEPROM.
 **/
static s32 ixgbe_acquire_eeprom(struct ixgbe_hw *hw)
{
        s32 status = IXGBE_SUCCESS;
        u32 eec;
        u32 i;

        DEBUGFUNC("ixgbe_acquire_eeprom");

        if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM)
            != IXGBE_SUCCESS)
        if (ixgbe_acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) != IXGBE_SUCCESS)
                status = IXGBE_ERR_SWFW_SYNC;
```

```
1462         if (status == IXGBE_SUCCESS) {
1463                 eec = IXGBE_READ_REG(hw, IXGBE_EEC);

1465                 /* Request EEPROM Access */
1466                 eec |= IXGBE_EEC_REQ;
1467                 IXGBE_WRITE_REG(hw, IXGBE_EEC, eec);

1469                 for (i = 0; i < IXGBE_EEPROM_GRANT_ATTEMPTS; i++) {
1470                         eec = IXGBE_READ_REG(hw, IXGBE_EEC);
1471                         if (eec & IXGBE_EEC_GNT)
1472                                 break;
1473                         usec_delay(5);
1474                 }

1476                 /* Release if grant not acquired */
1477                 if (!(eec & IXGBE_EEC_GNT)) {
1478                         eec &= ~IXGBE_EEC_REQ;
1479                         IXGBE_WRITE_REG(hw, IXGBE_EEC, eec);
1480                         DEBUGOUT("Could not acquire EEPROM grant\n");

1482                         hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
1081                         ixgbe_release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
1483                         status = IXGBE_ERR_EEPROM;
1484                 }

1486                 /* Setup EEPROM for Read/Write */
1487                 if (status == IXGBE_SUCCESS) {
1488                         /* Clear CS and SK */
1489                         eec &= ~(IXGBE_EEC_CS | IXGBE_EEC_SK);
1490                         IXGBE_WRITE_REG(hw, IXGBE_EEC, eec);
1491                         IXGBE_WRITE_FLUSH(hw);
1492                         usec_delay(1);
1493                 }
1494         }
1495         return status;
1496 }

1498 /**
1499  *  ixgbe_get_eeprom_semaphore - Get hardware semaphore
1500  *  @hw: pointer to hardware structure
1501  *
1502  *  Sets the hardware semaphores so EEPROM access can occur for bit-bang method
1503  **/
1504 static s32 ixgbe_get_eeprom_semaphore(struct ixgbe_hw *hw)
1505 {
1506         s32 status = IXGBE_ERR_EEPROM;
1507         u32 timeout = 2000;
1508         u32 i;
1509         u32 swsm;

1511         DEBUGFUNC("ixgbe_get_eeprom_semaphore");


1514         /* Get SMBI software semaphore between device drivers first */
1515         for (i = 0; i < timeout; i++) {
1516                 /*
1517                  * If the SMBI bit is 0 when we read it, then the bit will be
1518                  * set and we have the semaphore
1519                  */
1520                 swsm = IXGBE_READ_REG(hw, IXGBE_SWSM);
1521                 if (!(swsm & IXGBE_SWSM_SMBI)) {
1522                         status = IXGBE_SUCCESS;
1523                         break;
1524                 }
1525                 usec_delay(50);
```



```
1526         }

1528         if (i == timeout) {
1529                 DEBUGOUT("Driver can't access the Eeprom - SMBI Semaphore "
1530                          "not granted.\n");
1531                 /*
1532                  * this release is particularly important because our attempts
1533                  * above to get the semaphore may have succeeded, and if there
1534                  * was a timeout, we should unconditionally clear the semaphore
1535                  * bits to free the driver to make progress
1536                  */
1537                 ixgbe_release_eeprom_semaphore(hw);

1539                 usec_delay(50);
1540                 /*
1541                  * one last try
1542                  * If the SMBI bit is 0 when we read it, then the bit will be
1543                  * set and we have the semaphore
1544                  */
1545                 swsm = IXGBE_READ_REG(hw, IXGBE_SWSM);
1546                 if (!(swsm & IXGBE_SWSM_SMBI))
1547                         status = IXGBE_SUCCESS;
1548         }

1550         /* Now get the semaphore between SW/FW through the SWESMBI bit */
1551         if (status == IXGBE_SUCCESS) {
1552                 for (i = 0; i < timeout; i++) {
1553                         swsm = IXGBE_READ_REG(hw, IXGBE_SWSM);

1555                         /* Set the SW EEPROM semaphore bit to request access */
1556                         swsm |= IXGBE_SWSM_SWESMBI;
1557                         IXGBE_WRITE_REG(hw, IXGBE_SWSM, swsm);

1559                         /*
1560                          * If we set the bit successfully then we got the
1561                          * semaphore.
1562                          */
1563                         swsm = IXGBE_READ_REG(hw, IXGBE_SWSM);
1564                         if (swsm & IXGBE_SWSM_SWESMBI)
1565                                 break;

1567                         usec_delay(50);
1568                 }

1570                 /*
1571                  * Release semaphores and return error if SW EEPROM semaphore
1572                  * was not granted because we don't have access to the EEPROM
1573                  */
1574                 if (i >= timeout) {
1575                         DEBUGOUT("SWESMBI Software EEPROM semaphore "
1576                                  "not granted.\n");
1577                         ixgbe_release_eeprom_semaphore(hw);
1578                         status = IXGBE_ERR_EEPROM;
1579                 }
1580         } else {
1581                 DEBUGOUT("Software semaphore SMBI between device drivers "
1582                          "not granted.\n");
1583         }

1585         return status;
1586 }
______unchanged_portion_omitted_

1806 /**
1807  *  ixgbe_release_eeprom - Release EEPROM, release semaphores
1808  *  @hw: pointer to hardware structure
```

```
 **/
static void ixgbe_release_eeprom(struct ixgbe_hw *hw)
{
        u32 eec;

        DEBUGFUNC("ixgbe_release_eeprom");

        eec = IXGBE_READ_REG(hw, IXGBE_EEC);

        eec |= IXGBE_EEC_CS;  /* Pull CS high */
        eec &= ~IXGBE_EEC_SK; /* Lower SCK */

        IXGBE_WRITE_REG(hw, IXGBE_EEC, eec);
        IXGBE_WRITE_FLUSH(hw);

        usec_delay(1);

        /* Stop requesting EEPROM access */
        eec &= ~IXGBE_EEC_REQ;
        IXGBE_WRITE_REG(hw, IXGBE_EEC, eec);

        hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
        ixgbe_release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);

        /* Delay before attempt to obtain semaphore again to allow FW access */
        msec_delay(hw->eeprom.semaphore_delay);
}
```

**_unchanged_portion_omitted_**

```
/**
 *  ixgbe_init_rx_addrs_generic - Initializes receive address filters.
 *  @hw: pointer to hardware structure
 *
 *  Places the MAC address in receive address register 0 and clears the rest
 *  of the receive address registers. Clears the multicast table. Assumes
 *  the receiver is in reset when the routine is called.
 **/
s32 ixgbe_init_rx_addrs_generic(struct ixgbe_hw *hw)
{
        u32 i;
        u32 rar_entries = hw->mac.num_rar_entries;

        DEBUGFUNC("ixgbe_init_rx_addrs_generic");

        /*
         * If the current mac address is valid, assume it is a software override
         * to the permanent address.
         * Otherwise, use the permanent address from the eeprom.
         */
        if (ixgbe_validate_mac_addr(hw->mac.addr) ==
            IXGBE_ERR_INVALID_MAC_ADDR) {
                /* Get the MAC address from the RAR0 for later reference */
                hw->mac.ops.get_mac_addr(hw, hw->mac.addr);

                DEBUGOUT3(" Keeping Current RAR0 Addr =%.2X %.2X %.2X ",
                          hw->mac.addr[0], hw->mac.addr[1],
                          hw->mac.addr[2]);
                DEBUGOUT3("%.2X %.2X %.2X\n", hw->mac.addr[3],
                          hw->mac.addr[4], hw->mac.addr[5]);
        } else {
                /* Setup the receive address. */
                DEBUGOUT("Overriding MAC Address in RAR[0]\n");
                DEBUGOUT3(" New MAC Addr =%.2X %.2X %.2X ",
                          hw->mac.addr[0], hw->mac.addr[1],
                          hw->mac.addr[2]);
                DEBUGOUT3("%.2X %.2X %.2X\n", hw->mac.addr[3],
```



```
                          hw->mac.addr[4], hw->mac.addr[5]);

                hw->mac.ops.set_rar(hw, 0, hw->mac.addr, 0, IXGBE_RAH_AV);

                /* clear VMDq pool/queue selection for RAR 0 */
                hw->mac.ops.clear_vmdq(hw, 0, IXGBE_CLEAR_VMDQ_ALL);
        }
        hw->addr_ctrl.overflow_promisc = 0;

        hw->addr_ctrl.rar_used_count = 1;

        /* Zero out the other receive addresses. */
        DEBUGOUT1("Clearing RAR[1-%d]\n", rar_entries - 1);
        for (i = 1; i < rar_entries; i++) {
                IXGBE_WRITE_REG(hw, IXGBE_RAL(i), 0);
                IXGBE_WRITE_REG(hw, IXGBE_RAH(i), 0);
        }

        /* Clear the MTA */
        hw->addr_ctrl.mta_in_use = 0;
        IXGBE_WRITE_REG(hw, IXGBE_MCSTCTRL, hw->mac.mc_filter_type);

        DEBUGOUT(" Clearing MTA\n");
        for (i = 0; i < hw->mac.mcft_size; i++)
                IXGBE_WRITE_REG(hw, IXGBE_MTA(i), 0);

        ixgbe_init_uta_tables(hw);
        (void) ixgbe_init_uta_tables(hw);

        return IXGBE_SUCCESS;
}
```

**_unchanged_portion_omitted_**

```
/**
 *  ixgbe_update_mc_addr_list_generic - Updates MAC list of multicast addresses
 *  @hw: pointer to hardware structure
 *  @mc_addr_list: the list of new multicast addresses
 *  @mc_addr_count: number of addresses
 *  @next: iterator function to walk the multicast address list
 *  @clear: flag, when set clears the table beforehand
 *
 *  When the clear flag is set, the given list replaces any existing list.
 *  Hashes the given addresses into the multicast table.
 *  The given list replaces any existing list. Clears the MC addrs from receive
 *  address registers and the multicast table. Uses unused receive address
 *  registers for the first multicast addresses, and hashes the rest into the
 *  multicast table.
 **/
s32 ixgbe_update_mc_addr_list_generic(struct ixgbe_hw *hw, u8 *mc_addr_list,
                                      u32 mc_addr_count, ixgbe_mc_addr_itr next,
                                      bool clear)
                                      u32 mc_addr_count, ixgbe_mc_addr_itr next)
{
        u32 i;
        u32 vmdq;

        DEBUGFUNC("ixgbe_update_mc_addr_list_generic");

        /*
         * Set the new number of MC addresses that we are being requested to
         * use.
         */
        hw->addr_ctrl.num_mc_addrs = mc_addr_count;
        hw->addr_ctrl.mta_in_use = 0;

        /* Clear mta_shadow */
```

```
2351         if (clear) {
2352                 DEBUGOUT(" Clearing MTA\n");
2353                 memset(&hw->mac.mta_shadow, 0, sizeof(hw->mac.mta_shadow));
2354         }
1929         (void) memset(&hw->mac.mta_shadow, 0, sizeof(hw->mac.mta_shadow));

2356         /* Update mta_shadow */
2357         for (i = 0; i < mc_addr_count; i++) {
2358                 DEBUGOUT(" Adding the multicast addresses:\n");
2359                 ixgbe_set_mta(hw, next(hw, &mc_addr_list, &vmdq));
2360         }

2362         /* Enable mta */
2363         for (i = 0; i < hw->mac.mcft_size; i++)
2364                 IXGBE_WRITE_REG_ARRAY(hw, IXGBE_MTA(0), i,
2365                                       hw->mac.mta_shadow[i]);

2367         if (hw->addr_ctrl.mta_in_use > 0)
2368                 IXGBE_WRITE_REG(hw, IXGBE_MCSTCTRL,
2369                                 IXGBE_MCSTCTRL_MFE | hw->mac.mc_filter_type);

2371         DEBUGOUT("ixgbe_update_mc_addr_list_generic Complete\n");
2372         return IXGBE_SUCCESS;
2373 }
______unchanged_portion_omitted_

2412 /**
2413  *  ixgbe_fc_enable_generic - Enable flow control
2414  *  @hw: pointer to hardware structure
1990  *  @packetbuf_num: packet buffer number (0-7)
2415  *
2416  *  Enable flow control according to the current settings.
2417  **/
2418 s32 ixgbe_fc_enable_generic(struct ixgbe_hw *hw)
1994 s32 ixgbe_fc_enable_generic(struct ixgbe_hw *hw, s32 packetbuf_num)
2419 {
2420         s32 ret_val = IXGBE_SUCCESS;
2421         u32 mflcn_reg, fccfg_reg;
2422         u32 reg;
1999         u32 rx_pba_size;
2423         u32 fcrtl, fcrth;
2424         int i;

2426         DEBUGFUNC("ixgbe_fc_enable_generic");

2428         /* Validate the water mark configuration */
2429         if (!hw->fc.pause_time) {
2430                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
2004         /* Negotiate the fc mode to use */
2005         ret_val = ixgbe_fc_autoneg(hw);
2006         if (ret_val == IXGBE_ERR_FLOW_CONTROL)
2431                 goto out;
2432         }

2434         /* Low water mark of zero causes XOFF floods */
2435         for (i = 0; i < IXGBE_DCB_MAX_TRAFFIC_CLASS; i++) {
2436                 if ((hw->fc.current_mode & ixgbe_fc_tx_pause) &&
2437                     hw->fc.high_water[i]) {
2438                         if (!hw->fc.low_water[i] ||
2439                             hw->fc.low_water[i] >= hw->fc.high_water[i]) {
2440                                 DEBUGOUT("Invalid water mark configuration\n");
2441                                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
2442                                 goto out;
2443                         }
2444                 }
2445         }
```



```
2447         /* Negotiate the fc mode to use */
2448         ixgbe_fc_autoneg(hw);

2450         /* Disable any previous flow control settings */
2451         mflcn_reg = IXGBE_READ_REG(hw, IXGBE_MFLCN);
2452         mflcn_reg &= ~(IXGBE_MFLCN_RPFCE_MASK | IXGBE_MFLCN_RFCE);
2011         mflcn_reg &= ~(IXGBE_MFLCN_RFCE | IXGBE_MFLCN_RPFCE);

2454         fccfg_reg = IXGBE_READ_REG(hw, IXGBE_FCCFG);
2455         fccfg_reg &= ~(IXGBE_FCCFG_TFCE_802_3X | IXGBE_FCCFG_TFCE_PRIORITY);

2457         /*
2458          * The possible values of fc.current_mode are:
2459          * 0: Flow control is completely disabled
2460          * 1: Rx flow control is enabled (we can receive pause frames,
2461          *    but not send pause frames).
2462          * 2: Tx flow control is enabled (we can send pause frames but
2463          *    we do not support receiving pause frames).
2464          * 3: Both Rx and Tx flow control (symmetric) are enabled.
2465          * other: Invalid.
2466          */
2467         switch (hw->fc.current_mode) {
2468         case ixgbe_fc_none:
2469                 /*
2470                  * Flow control is disabled by software override or autoneg.
2471                  * The code below will actually disable it in the HW.
2472                  */
2473                 break;
2474         case ixgbe_fc_rx_pause:
2475                 /*
2476                  * Rx Flow control is enabled and Tx Flow control is
2477                  * disabled by software override. Since there really
2478                  * isn't a way to advertise that we are capable of RX
2479                  * Pause ONLY, we will advertise that we support both
2480                  * symmetric and asymmetric Rx PAUSE.  Later, we will
2481                  * disable the adapter's ability to send PAUSE frames.
2482                  */
2483                 mflcn_reg |= IXGBE_MFLCN_RFCE;
2484                 break;
2485         case ixgbe_fc_tx_pause:
2486                 /*
2487                  * Tx Flow control is enabled, and Rx Flow control is
2488                  * disabled by software override.
2489                  */
2490                 fccfg_reg |= IXGBE_FCCFG_TFCE_802_3X;
2491                 break;
2492         case ixgbe_fc_full:
2493                 /* Flow control (both Rx and Tx) is enabled by SW override. */
2494                 mflcn_reg |= IXGBE_MFLCN_RFCE;
2495                 fccfg_reg |= IXGBE_FCCFG_TFCE_802_3X;
2496                 break;
2497         default:
2498                 DEBUGOUT("Flow control param set incorrectly\n");
2499                 ret_val = IXGBE_ERR_CONFIG;
2500                 goto out;
2501         }

2503         /* Set 802.3x based flow control settings. */
2504         mflcn_reg |= IXGBE_MFLCN_DPF;
2505         IXGBE_WRITE_REG(hw, IXGBE_MFLCN, mflcn_reg);
2506         IXGBE_WRITE_REG(hw, IXGBE_FCCFG, fccfg_reg);

2067         rx_pba_size = IXGBE_READ_REG(hw, IXGBE_RXPBSIZE(packetbuf_num));
2068         rx_pba_size >>= IXGBE_RXPBSIZE_SHIFT;
```

```
2509         /* Set up and enable Rx high/low water mark thresholds, enable XON. */
2510         for (i = 0; i < IXGBE_DCB_MAX_TRAFFIC_CLASS; i++) {
2511                 if ((hw->fc.current_mode & ixgbe_fc_tx_pause) &&
2512                     hw->fc.high_water[i]) {
2513                         fcrtl = (hw->fc.low_water[i] << 10) | IXGBE_FCRTL_XONE;
2514                         IXGBE_WRITE_REG(hw, IXGBE_FCRTL_82599(i), fcrtl);
2515                         fcrth = (hw->fc.high_water[i] << 10) | IXGBE_FCRTH_FCEN;
2516                 } else {
2517                         IXGBE_WRITE_REG(hw, IXGBE_FCRTL_82599(i), 0);
2518                         /*
2519                          * In order to prevent Tx hangs when the internal Tx
2520                          * switch is enabled we must set the high water mark
2521                          * to the maximum FCRTH value.  This allows the Tx
2522                          * switch to function even under heavy Rx workloads.
2523                          */
2524                         fcrth = IXGBE_READ_REG(hw, IXGBE_RXPBSIZE(i)) - 32;
2525                 }
2070         fcrth = (rx_pba_size - hw->fc.high_water) << 10;
2071         fcrtl = (rx_pba_size - hw->fc.low_water) << 10;

2527                 IXGBE_WRITE_REG(hw, IXGBE_FCRTH_82599(i), fcrth);
2073         if (hw->fc.current_mode & ixgbe_fc_tx_pause) {
2074                 fcrth |= IXGBE_FCRTH_FCEN;
2075                 if (hw->fc.send_xon)
2076                         fcrtl |= IXGBE_FCRTL_XONE;
2528         }

2079         IXGBE_WRITE_REG(hw, IXGBE_FCRTH_82599(packetbuf_num), fcrth);
2080         IXGBE_WRITE_REG(hw, IXGBE_FCRTL_82599(packetbuf_num), fcrtl);

2530         /* Configure pause time (2 TCs per register) */
2531         reg = hw->fc.pause_time * 0x00010001;
2532         for (i = 0; i < (IXGBE_DCB_MAX_TRAFFIC_CLASS / 2); i++)
2533                 IXGBE_WRITE_REG(hw, IXGBE_FCTTV(i), reg);
2083         reg = IXGBE_READ_REG(hw, IXGBE_FCTTV(packetbuf_num / 2));
2084         if ((packetbuf_num & 1) == 0)
2085                 reg = (reg & 0xFFFF0000) | hw->fc.pause_time;
2086         else
2087                 reg = (reg & 0x0000FFFF) | (hw->fc.pause_time << 16);
2088         IXGBE_WRITE_REG(hw, IXGBE_FCTTV(packetbuf_num / 2), reg);

2535         /* Configure flow control refresh threshold value */
2536         IXGBE_WRITE_REG(hw, IXGBE_FCRTV, hw->fc.pause_time / 2);
2090         IXGBE_WRITE_REG(hw, IXGBE_FCRTV, (hw->fc.pause_time >> 1));

2538 out:
2539         return ret_val;
2540 }

2542 /**
2543  *  ixgbe_negotiate_fc - Negotiate flow control
2097  *  ixgbe_fc_autoneg - Configure flow control
2544  *  @hw: pointer to hardware structure
2545  *  @adv_reg: flow control advertised settings
2546  *  @lp_reg: link partner's flow control settings
2547  *  @adv_sym: symmetric pause bit in advertisement
2548  *  @adv_asm: asymmetric pause bit in advertisement
2549  *  @lp_sym: symmetric pause bit in link partner advertisement
2550  *  @lp_asm: asymmetric pause bit in link partner advertisement
2551  *
2552  *  Find the intersection between advertised settings and link partner's
2553  *  advertised settings
2100  *  Compares our advertised flow control capabilities to those advertised by
2101  *  our link partner, and determines the proper flow control mode to use.
2554  **/
2555 static s32 ixgbe_negotiate_fc(struct ixgbe_hw *hw, u32 adv_reg, u32 lp_reg,
```



```
2556                               u32 adv_sym, u32 adv_asm, u32 lp_sym, u32 lp_asm)
2103 s32 ixgbe_fc_autoneg(struct ixgbe_hw *hw)
2557 {
2558         if ((!(adv_reg)) ||  (!(lp_reg)))
2559                 return IXGBE_ERR_FC_NOT_NEGOTIATED;
2105         s32 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2106         ixgbe_link_speed speed;
2107         bool link_up;

2561         if ((adv_reg & adv_sym) && (lp_reg & lp_sym)) {
2109         DEBUGFUNC("ixgbe_fc_autoneg");

2111         if (hw->fc.disable_fc_autoneg)
2112                 goto out;

2562                 /*
2563                  * Now we need to check if the user selected Rx ONLY
2564                  * of pause frames.  In this case, we had to advertise
2565                  * FULL flow control because we could not advertise RX
2566                  * ONLY. Hence, we must now check to see if we need to
2567                  * turn OFF the TRANSMISSION of PAUSE frames.
2115          * AN should have completed when the cable was plugged in.
2116          * Look for reasons to bail out.  Bail out if:
2117          * - FC autoneg is disabled, or if
2118          * - link is not up.
2119          *
2120          * Since we're being called from an LSC, link is already known to be up.
2121          * So use link_up_wait_to_complete=FALSE.
2568                  */
2569                 if (hw->fc.requested_mode == ixgbe_fc_full) {
2570                         hw->fc.current_mode = ixgbe_fc_full;
2571                         DEBUGOUT("Flow Control = FULL.\n");
2572                 } else {
2573                         hw->fc.current_mode = ixgbe_fc_rx_pause;
2574                         DEBUGOUT("Flow Control=RX PAUSE frames only\n");
2123         hw->mac.ops.check_link(hw, &speed, &link_up, FALSE);
2124         if (!link_up) {
2125                 ret_val = IXGBE_ERR_FLOW_CONTROL;
2126                 goto out;
2575                 }
2576         } else if (!(adv_reg & adv_sym) && (adv_reg & adv_asm) &&
2577                    (lp_reg & lp_sym) && (lp_reg & lp_asm)) {
2578                 hw->fc.current_mode = ixgbe_fc_tx_pause;
2579                 DEBUGOUT("Flow Control = TX PAUSE frames only.\n");
2580         } else if ((adv_reg & adv_sym) && (adv_reg & adv_asm) &&
2581                    !(lp_reg & lp_sym) && (lp_reg & lp_asm)) {
2582                 hw->fc.current_mode = ixgbe_fc_rx_pause;
2583                 DEBUGOUT("Flow Control = RX PAUSE frames only.\n");

2129         switch (hw->phy.media_type) {
2130         /* Autoneg flow control on fiber adapters */
2131         case ixgbe_media_type_fiber:
2132                 if (speed == IXGBE_LINK_SPEED_1GB_FULL)
2133                         ret_val = ixgbe_fc_autoneg_fiber(hw);
2134                 break;

2136         /* Autoneg flow control on backplane adapters */
2137         case ixgbe_media_type_backplane:
2138                 ret_val = ixgbe_fc_autoneg_backplane(hw);
2139                 break;

2141         /* Autoneg flow control on copper adapters */
2142         case ixgbe_media_type_copper:
2143                 if (ixgbe_device_supports_autoneg_fc(hw) == IXGBE_SUCCESS)
2144                         ret_val = ixgbe_fc_autoneg_copper(hw);
2145                 break;
```

```
2147         default:
2148                 break;
2149         }

2151 out:
2152         if (ret_val == IXGBE_SUCCESS) {
2153                 hw->fc.fc_was_autonegged = TRUE;
2584         } else {
2585                 hw->fc.current_mode = ixgbe_fc_none;
2586                 DEBUGOUT("Flow Control = NONE.\n");
2155                 hw->fc.fc_was_autonegged = FALSE;
2156                 hw->fc.current_mode = hw->fc.requested_mode;
2587         }
2588         return IXGBE_SUCCESS;
2158         return ret_val;
2589 }

2591 /**
2592  *  ixgbe_fc_autoneg_fiber - Enable flow control on 1 gig fiber
2593  *  @hw: pointer to hardware structure
2164  *  @speed:
2165  *  @link_up
2594  *
2595  *  Enable flow control according on 1 gig fiber.
2596  **/
2597 static s32 ixgbe_fc_autoneg_fiber(struct ixgbe_hw *hw)
2598 {
2599         u32 pcs_anadv_reg, pcs_lpab_reg, linkstat;
2600         s32 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2172         s32 ret_val;

2602         /*
2603          * On multispeed fiber at 1g, bail out if
2604          * - link is up but AN did not complete, or if
2605          * - link is up and AN completed but timed out
2606          */

2608         linkstat = IXGBE_READ_REG(hw, IXGBE_PCS1GLSTA);
2609         if ((!!(linkstat & IXGBE_PCS1GLSTA_AN_COMPLETE) == 0) ||
2610             (!!(linkstat & IXGBE_PCS1GLSTA_AN_TIMED_OUT) == 1))
2181         if (((linkstat & IXGBE_PCS1GLSTA_AN_COMPLETE) == 0) ||
2182             ((linkstat & IXGBE_PCS1GLSTA_AN_TIMED_OUT) == 1)) {
2183                 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2611                 goto out;
2185         }

2613         pcs_anadv_reg = IXGBE_READ_REG(hw, IXGBE_PCS1GANA);
2614         pcs_lpab_reg = IXGBE_READ_REG(hw, IXGBE_PCS1GANLP);

2616         ret_val =  ixgbe_negotiate_fc(hw, pcs_anadv_reg,
2617                                       pcs_lpab_reg, IXGBE_PCS1GANA_SYM_PAUSE,
2618                                       IXGBE_PCS1GANA_ASM_PAUSE,
2619                                       IXGBE_PCS1GANA_SYM_PAUSE,
2620                                       IXGBE_PCS1GANA_ASM_PAUSE);

2622 out:
2623         return ret_val;
2624 }

2626 /**
2627  *  ixgbe_fc_autoneg_backplane - Enable flow control IEEE clause 37
2628  *  @hw: pointer to hardware structure
2629  *
2630  *  Enable flow control according to IEEE clause 37.
2631  **/
```



```
2632 static s32 ixgbe_fc_autoneg_backplane(struct ixgbe_hw *hw)
2633 {
2634         u32 links2, anlp1_reg, autoc_reg, links;
2635         s32 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2209         s32 ret_val;

2637         /*
2638          * On backplane, bail out if
2639          * - backplane autoneg was not completed, or if
2640          * - we are 82599 and link partner is not AN enabled
2641          */
2642         links = IXGBE_READ_REG(hw, IXGBE_LINKS);
2643         if ((links & IXGBE_LINKS_KX_AN_COMP) == 0)
2217         if ((links & IXGBE_LINKS_KX_AN_COMP) == 0) {
2218                 hw->fc.fc_was_autonegged = FALSE;
2219                 hw->fc.current_mode = hw->fc.requested_mode;
2220                 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2644                 goto out;
2222         }

2646         if (hw->mac.type == ixgbe_mac_82599EB) {
2647                 links2 = IXGBE_READ_REG(hw, IXGBE_LINKS2);
2648                 if ((links2 & IXGBE_LINKS2_AN_SUPPORTED) == 0)
2226                 if ((links2 & IXGBE_LINKS2_AN_SUPPORTED) == 0) {
2227                         hw->fc.fc_was_autonegged = FALSE;
2228                         hw->fc.current_mode = hw->fc.requested_mode;
2229                         ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2649                         goto out;
2650         }
2232         }
2651         /*
2652          * Read the 10g AN autoc and LP ability registers and resolve
2653          * local flow control settings accordingly
2654          */
2655         autoc_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC);
2656         anlp1_reg = IXGBE_READ_REG(hw, IXGBE_ANLP1);

2658         ret_val = ixgbe_negotiate_fc(hw, autoc_reg,
2659                 anlp1_reg, IXGBE_AUTOC_SYM_PAUSE, IXGBE_AUTOC_ASM_PAUSE,
2660                 IXGBE_ANLP1_SYM_PAUSE, IXGBE_ANLP1_ASM_PAUSE);

2662 out:
2663         return ret_val;
2664 }
_____unchanged_portion_omitted_

2690 /**
2691  *  ixgbe_fc_autoneg - Configure flow control
2273  *  ixgbe_negotiate_fc - Negotiate flow control
2692  *  @hw: pointer to hardware structure
2275  *  @adv_reg: flow control advertised settings
2276  *  @lp_reg: link partner's flow control settings
2277  *  @adv_sym: symmetric pause bit in advertisement
2278  *  @adv_asm: asymmetric pause bit in advertisement
2279  *  @lp_sym: symmetric pause bit in link partner advertisement
2280  *  @lp_asm: asymmetric pause bit in link partner advertisement
2693  *
2694  *  Compares our advertised flow control capabilities to those advertised by
2695  *  our link partner, and determines the proper flow control mode to use.
2282  *  Find the intersection between advertised settings and link partner's
2283  *  advertised settings
2696  **/
2697 void ixgbe_fc_autoneg(struct ixgbe_hw *hw)
2285 static s32 ixgbe_negotiate_fc(struct ixgbe_hw *hw, u32 adv_reg, u32 lp_reg,
2286                               u32 adv_sym, u32 adv_asm, u32 lp_sym, u32 lp_asm)
2698 {
```

```
2699         s32 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2700         ixgbe_link_speed speed;
2701         bool link_up;
2288         if ((!(adv_reg)) ||  (!(lp_reg)))
2289                 return IXGBE_ERR_FC_NOT_NEGOTIATED;

2703         DEBUGFUNC("ixgbe_fc_autoneg");
2291         if ((adv_reg & adv_sym) && (lp_reg & lp_sym)) {
2292                 /*
2293                  * Now we need to check if the user selected Rx ONLY
2294                  * of pause frames.  In this case, we had to advertise
2295                  * FULL flow control because we could not advertise RX
2296                  * ONLY. Hence, we must now check to see if we need to
2297                  * turn OFF the TRANSMISSION of PAUSE frames.
2298                  */
2299                 if (hw->fc.requested_mode == ixgbe_fc_full) {
2300                         hw->fc.current_mode = ixgbe_fc_full;
2301                         DEBUGOUT("Flow Control = FULL.\n");
2302                 } else {
2303                         hw->fc.current_mode = ixgbe_fc_rx_pause;
2304                         DEBUGOUT("Flow Control=RX PAUSE frames only\n");
2305                 }
2306         } else if (!(adv_reg & adv_sym) && (adv_reg & adv_asm) &&
2307                    (lp_reg & lp_sym) && (lp_reg & lp_asm)) {
2308                 hw->fc.current_mode = ixgbe_fc_tx_pause;
2309                 DEBUGOUT("Flow Control = TX PAUSE frames only.\n");
2310         } else if ((adv_reg & adv_sym) && (adv_reg & adv_asm) &&
2311                    !(lp_reg & lp_sym) && (lp_reg & lp_asm)) {
2312                 hw->fc.current_mode = ixgbe_fc_rx_pause;
2313                 DEBUGOUT("Flow Control = RX PAUSE frames only.\n");
2314         } else {
2315                 hw->fc.current_mode = ixgbe_fc_none;
2316                 DEBUGOUT("Flow Control = NONE.\n");
2317         }
2318         return IXGBE_SUCCESS;
2319 }

2321 /**
2322  *  ixgbe_setup_fc - Set up flow control
2323  *  @hw: pointer to hardware structure
2324  *
2325  *  Called at init time to set up flow control.
2326  **/
2327 s32 ixgbe_setup_fc(struct ixgbe_hw *hw, s32 packetbuf_num)
2328 {
2329         s32 ret_val = IXGBE_SUCCESS;
2330         u32 reg = 0, reg_bp = 0;
2331         u16 reg_cu = 0;

2333         DEBUGFUNC("ixgbe_setup_fc");

2335         /* Validate the packetbuf configuration */
2336         if (packetbuf_num < 0 || packetbuf_num > 7) {
2337                 DEBUGOUT1("Invalid packet buffer number [%d], expected range is"
2338                           " 0-7\n", packetbuf_num);
2339                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
2340                 goto out;
2341         }

2705         /*
2706          * AN should have completed when the cable was plugged in.
2707          * Look for reasons to bail out.  Bail out if:
2708          * - FC autoneg is disabled, or if
2709          * - link is not up.
2344          * Validate the water mark configuration.  Zero water marks are invalid
2345          * because it causes the controller to just blast out fc packets.
```



```
2710          */
2711         if (hw->fc.disable_fc_autoneg)
2347         if (!hw->fc.low_water || !hw->fc.high_water || !hw->fc.pause_time) {
2348                 DEBUGOUT("Invalid water mark configuration\n");
2349                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
2712                 goto out;
2351         }

2714         hw->mac.ops.check_link(hw, &speed, &link_up, FALSE);
2715         if (!link_up)
2353         /*
2354          * Validate the requested mode.  Strict IEEE mode does not allow
2355          * ixgbe_fc_rx_pause because it will cause us to fail at UNH.
2356          */
2357         if (hw->fc.strict_ieee && hw->fc.requested_mode == ixgbe_fc_rx_pause) {
2358                 DEBUGOUT("ixgbe_fc_rx_pause not valid in strict IEEE mode\n");
2359                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
2716                 goto out;
2361         }

2363         /*
2364          * 10gig parts do not have a word in the EEPROM to determine the
2365          * default flow control setting, so we explicitly set it to full.
2366          */
2367         if (hw->fc.requested_mode == ixgbe_fc_default)
2368                 hw->fc.requested_mode = ixgbe_fc_full;

2370         /*
2371          * Set up the 1G and 10G flow control advertisement registers so the
2372          * HW will be able to do fc autoneg once the cable is plugged in.  If
2373          * we link at 10G, the 1G advertisement is harmless and vice versa.
2374          */

2718         switch (hw->phy.media_type) {
2719         /* Autoneg flow control on fiber adapters */
2720         case ixgbe_media_type_fiber:
2721                 if (speed == IXGBE_LINK_SPEED_1GB_FULL)
2722                         ret_val = ixgbe_fc_autoneg_fiber(hw);
2723                 break;

2725         /* Autoneg flow control on backplane adapters */
2726         case ixgbe_media_type_backplane:
2727                 ret_val = ixgbe_fc_autoneg_backplane(hw);
2379                 reg = IXGBE_READ_REG(hw, IXGBE_PCS1GANA);
2380                 reg_bp = IXGBE_READ_REG(hw, IXGBE_AUTOC);
2728                 break;

2730         /* Autoneg flow control on copper adapters */
2731         case ixgbe_media_type_copper:
2732                 if (ixgbe_device_supports_autoneg_fc(hw) == IXGBE_SUCCESS)
2733                         ret_val = ixgbe_fc_autoneg_copper(hw);
2384                 hw->phy.ops.read_reg(hw, IXGBE_MDIO_AUTO_NEG_ADVT,
2385                                      IXGBE_MDIO_AUTO_NEG_DEV_TYPE, &reg_cu);
2734                 break;

2736         default:
2389                 ;
2390         }

2392         /*
2393          * The possible values of fc.requested_mode are:
2394          * 0: Flow control is completely disabled
2395          * 1: Rx flow control is enabled (we can receive pause frames,
2396          *    but not send pause frames).
2397          * 2: Tx flow control is enabled (we can send pause frames but
2398          *    we do not support receiving pause frames).
```

```
2399          * 3: Both Rx and Tx flow control (symmetric) are enabled.
2400          * other: Invalid.
2401          */
2402         switch (hw->fc.requested_mode) {
2403         case ixgbe_fc_none:
2404                 /* Flow control completely disabled by software override. */
2405                 reg &= ~(IXGBE_PCS1GANA_SYM_PAUSE | IXGBE_PCS1GANA_ASM_PAUSE);
2406                 if (hw->phy.media_type == ixgbe_media_type_backplane)
2407                         reg_bp &= ~(IXGBE_AUTOC_SYM_PAUSE |
2408                                     IXGBE_AUTOC_ASM_PAUSE);
2409                 else if (hw->phy.media_type == ixgbe_media_type_copper)
2410                         reg_cu &= ~(IXGBE_TAF_SYM_PAUSE | IXGBE_TAF_ASM_PAUSE);
2737                 break;
2412         case ixgbe_fc_rx_pause:
2413                 /*
2414                  * Rx Flow control is enabled and Tx Flow control is
2415                  * disabled by software override. Since there really
2416                  * isn't a way to advertise that we are capable of RX
2417                  * Pause ONLY, we will advertise that we support both
2418                  * symmetric and asymmetric Rx PAUSE.  Later, we will
2419                  * disable the adapter's ability to send PAUSE frames.
2420                  */
2421                 reg |= (IXGBE_PCS1GANA_SYM_PAUSE | IXGBE_PCS1GANA_ASM_PAUSE);
2422                 if (hw->phy.media_type == ixgbe_media_type_backplane)
2423                         reg_bp |= (IXGBE_AUTOC_SYM_PAUSE |
2424                                    IXGBE_AUTOC_ASM_PAUSE);
2425                 else if (hw->phy.media_type == ixgbe_media_type_copper)
2426                         reg_cu |= (IXGBE_TAF_SYM_PAUSE | IXGBE_TAF_ASM_PAUSE);
2427                 break;
2428         case ixgbe_fc_tx_pause:
2429                 /*
2430                  * Tx Flow control is enabled, and Rx Flow control is
2431                  * disabled by software override.
2432                  */
2433                 reg |= (IXGBE_PCS1GANA_ASM_PAUSE);
2434                 reg &= ~(IXGBE_PCS1GANA_SYM_PAUSE);
2435                 if (hw->phy.media_type == ixgbe_media_type_backplane) {
2436                         reg_bp |= (IXGBE_AUTOC_ASM_PAUSE);
2437                         reg_bp &= ~(IXGBE_AUTOC_SYM_PAUSE);
2438                 } else if (hw->phy.media_type == ixgbe_media_type_copper) {
2439                         reg_cu |= (IXGBE_TAF_ASM_PAUSE);
2440                         reg_cu &= ~(IXGBE_TAF_SYM_PAUSE);
2738         }
2442                 break;
2443         case ixgbe_fc_full:
2444                 /* Flow control (both Rx and Tx) is enabled by SW override. */
2445                 reg |= (IXGBE_PCS1GANA_SYM_PAUSE | IXGBE_PCS1GANA_ASM_PAUSE);
2446                 if (hw->phy.media_type == ixgbe_media_type_backplane)
2447                         reg_bp |= (IXGBE_AUTOC_SYM_PAUSE |
2448                                    IXGBE_AUTOC_ASM_PAUSE);
2449                 else if (hw->phy.media_type == ixgbe_media_type_copper)
2450                         reg_cu |= (IXGBE_TAF_SYM_PAUSE | IXGBE_TAF_ASM_PAUSE);
2451                 break;
2452         default:
2453                 DEBUGOUT("Flow control param set incorrectly\n");
2454                 ret_val = IXGBE_ERR_CONFIG;
2455                 goto out;
2456         }

2740 out:
2741         if (ret_val == IXGBE_SUCCESS) {
2742                 hw->fc.fc_was_autonegged = TRUE;
2743         } else {
2744                 hw->fc.fc_was_autonegged = FALSE;
2745                 hw->fc.current_mode = hw->fc.requested_mode;
2458         /*
```



```
2459          * Enable auto-negotiation between the MAC & PHY;
2460          * the MAC will advertise clause 37 flow control.
2461          */
2462         IXGBE_WRITE_REG(hw, IXGBE_PCS1GANA, reg);
2463         reg = IXGBE_READ_REG(hw, IXGBE_PCS1GLCTL);

2465         /* Disable AN timeout */
2466         if (hw->fc.strict_ieee)
2467                 reg &= ~IXGBE_PCS1GLCTL_AN_1G_TIMEOUT_EN;

2469         IXGBE_WRITE_REG(hw, IXGBE_PCS1GLCTL, reg);
2470         DEBUGOUT1("Set up FC; PCS1GLCTL = 0x%08X\n", reg);

2472         /*
2473          * AUTOC restart handles negotiation of 1G and 10G on backplane
2474          * and copper. There is no need to set the PCS1GCTL register.
2475          *
2476          */
2477         if (hw->phy.media_type == ixgbe_media_type_backplane) {
2478                 reg_bp |= IXGBE_AUTOC_AN_RESTART;
2479                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, reg_bp);
2480         } else if ((hw->phy.media_type == ixgbe_media_type_copper) &&
2481                     (ixgbe_device_supports_autoneg_fc(hw) == IXGBE_SUCCESS)) {
2482                 hw->phy.ops.write_reg(hw, IXGBE_MDIO_AUTO_NEG_ADVT,
2483                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE, reg_cu);
2746         }

2486         DEBUGOUT1("Set up FC; IXGBE_AUTOC = 0x%08X\n", reg);
2487 out:
2488         return ret_val;
2747 }

2749 /**
2750  *  ixgbe_disable_pcie_master - Disable PCI-express master access
2751  *  @hw: pointer to hardware structure
2752  *
2753  *  Disables PCI-Express master access and verifies there are no pending
2754  *  requests. IXGBE_ERR_MASTER_REQUESTS_PENDING is returned if master disable
2755  *  bit hasn't caused the master requests to be disabled, else IXGBE_SUCCESS
2756  *  is returned signifying master requests disabled.
2757  **/
2758 s32 ixgbe_disable_pcie_master(struct ixgbe_hw *hw)
2759 {
2760         s32 status = IXGBE_SUCCESS;
2761         u32 i;
2503         u32 reg_val;
2504         u32 number_of_queues;
2505         s32 status = IXGBE_SUCCESS;

2763         DEBUGFUNC("ixgbe_disable_pcie_master");

2765         /* Always set this bit to ensure any future transactions are blocked */
2766         IXGBE_WRITE_REG(hw, IXGBE_CTRL, IXGBE_CTRL_GIO_DIS);

2768         /* Exit if master requets are blocked */
2509         /* Just jump out if bus mastering is already disabled */
2769         if (!(IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_GIO))
2770                 goto out;

2772         /* Poll for master request bit to clear */
2513         /* Disable the receive unit by stopping each queue */
2514         number_of_queues = hw->mac.max_rx_queues;
2515         for (i = 0; i < number_of_queues; i++) {
2516                 reg_val = IXGBE_READ_REG(hw, IXGBE_RXDCTL(i));
2517                 if (reg_val & IXGBE_RXDCTL_ENABLE) {
2518                         reg_val &= ~IXGBE_RXDCTL_ENABLE;
```

```
2519                        IXGBE_WRITE_REG(hw, IXGBE_RXDCTL(i), reg_val);
2520                }
2521        }

2523        reg_val = IXGBE_READ_REG(hw, IXGBE_CTRL);
2524        reg_val |= IXGBE_CTRL_GIO_DIS;
2525        IXGBE_WRITE_REG(hw, IXGBE_CTRL, reg_val);

2773        for (i = 0; i < IXGBE_PCI_MASTER_DISABLE_TIMEOUT; i++) {
2774                usec_delay(100);
2775                if (!(IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_GIO))
2776                        goto out;
2529                        goto check_device_status;
2530                usec_delay(100);
2777        }

2779        /*
2780         * Two consecutive resets are required via CTRL.RST per datasheet
2781         * 5.2.5.3.2 Master Disable.  We set a flag to inform the reset routine
2782         * of this need.  The first reset prevents new master requests from
2783         * being issued by our device.  We then must wait 1usec or more for any
2784         * remaining completions from the PCIe bus to trickle in, and then reset
2785         * again to clear out any effects they may have had on our device.
2786         */
2787        DEBUGOUT("GIO Master Disable bit didn't clear - requesting resets\n");
2788        hw->mac.flags |= IXGBE_FLAGS_DOUBLE_RESET_REQUIRED;
2534        status = IXGBE_ERR_MASTER_REQUESTS_PENDING;

2790        /*
2791         * Before proceeding, make sure that the PCIe block does not have
2792         * transactions pending.
2793         */
2540 check_device_status:
2794        for (i = 0; i < IXGBE_PCI_MASTER_DISABLE_TIMEOUT; i++) {
2795                usec_delay(100);
2796                if (!(IXGBE_READ_PCIE_WORD(hw, IXGBE_PCI_DEVICE_STATUS) &
2797                    IXGBE_PCI_DEVICE_STATUS_TRANSACTION_PENDING))
2798                        goto out;
2544                        break;
2545                usec_delay(100);
2799        }

2548        if (i == IXGBE_PCI_MASTER_DISABLE_TIMEOUT)
2801        DEBUGOUT("PCIe transaction pending bit also did not clear.\n");
2802        status = IXGBE_ERR_MASTER_REQUESTS_PENDING;
2550        else
2551                goto out;

2553        /*
2554         * Two consecutive resets are required via CTRL.RST per datasheet
2555         * 5.2.5.3.2 Master Disable.  We set a flag to inform the reset routine
2556         * of this need.  The first reset prevents new master requests from
2557         * being issued by our device.  We then must wait 1usec for any
2558         * remaining completions from the PCIe bus to trickle in, and then reset
2559         * again to clear out any effects they may have had on our device.
2560         */
2561         hw->mac.flags |= IXGBE_FLAGS_DOUBLE_RESET_REQUIRED;

2804 out:
2805        return status;
2806 }


2808 /**
2809  *  ixgbe_acquire_swfw_sync - Acquire SWFW semaphore
2810  *  @hw: pointer to hardware structure
```



```
2811  *  @mask: Mask to specify which semaphore to acquire
2812  *
2813  *  Acquires the SWFW semaphore through the GSSR register for the specified
2573  *  Acquires the SWFW semaphore thought the GSSR register for the specified
2814  *  function (CSR, PHY0, PHY1, EEPROM, Flash)
2815  **/
2816 s32 ixgbe_acquire_swfw_sync(struct ixgbe_hw *hw, u16 mask)
2817 {
2818         u32 gssr;
2819         u32 swmask = mask;
2820         u32 fwmask = mask << 5;
2821         s32 timeout = 200;

2823         DEBUGFUNC("ixgbe_acquire_swfw_sync");

2825         while (timeout) {
2826                 /*
2827                  * SW EEPROM semaphore bit is used for access to all
2828                  * SW_FW_SYNC/GSSR bits (not just EEPROM)
2829                  */
2830                 if (ixgbe_get_eeprom_semaphore(hw))
2831                         return IXGBE_ERR_SWFW_SYNC;

2833                 gssr = IXGBE_READ_REG(hw, IXGBE_GSSR);
2834                 if (!(gssr & (fwmask | swmask)))
2835                         break;

2837                 /*
2838                  * Firmware currently using resource (fwmask) or other software
2839                  * thread currently using resource (swmask)
2840                  */
2841                 ixgbe_release_eeprom_semaphore(hw);
2842                 msec_delay(5);
2843                 timeout--;
2844         }

2846         if (!timeout) {
2847                 DEBUGOUT("Driver can't access resource, SW_FW_SYNC timeout.\n");
2848                 return IXGBE_ERR_SWFW_SYNC;
2849         }

2851         gssr |= swmask;
2852         IXGBE_WRITE_REG(hw, IXGBE_GSSR, gssr);

2854         ixgbe_release_eeprom_semaphore(hw);
2855         return IXGBE_SUCCESS;
2856 }

2858 /**
2859  *  ixgbe_release_swfw_sync - Release SWFW semaphore
2860  *  @hw: pointer to hardware structure
2861  *  @mask: Mask to specify which semaphore to release
2862  *
2863  *  Releases the SWFW semaphore through the GSSR register for the specified
2623  *  Releases the SWFW semaphore thought the GSSR register for the specified
2864  *  function (CSR, PHY0, PHY1, EEPROM, Flash)
2865  **/
2866 void ixgbe_release_swfw_sync(struct ixgbe_hw *hw, u16 mask)
2867 {
2868         u32 gssr;
2869         u32 swmask = mask;

2871         DEBUGFUNC("ixgbe_release_swfw_sync");

2873         ixgbe_get_eeprom_semaphore(hw);
2633         (void) ixgbe_get_eeprom_semaphore(hw);
```

```
2875         gssr = IXGBE_READ_REG(hw, IXGBE_GSSR);
2876         gssr &= ~swmask;
2877         IXGBE_WRITE_REG(hw, IXGBE_GSSR, gssr);

2879         ixgbe_release_eeprom_semaphore(hw);
2880 }

2882 /**
2883  *  ixgbe_disable_sec_rx_path_generic - Stops the receive data path
2884  *  @hw: pointer to hardware structure
2885  *
2886  *  Stops the receive data path and waits for the HW to internally empty
2887  *  the Rx security block
2888  **/
2889 s32 ixgbe_disable_sec_rx_path_generic(struct ixgbe_hw *hw)
2890 {
2891 #define IXGBE_MAX_SECRX_POLL 40

2893         int i;
2894         int secrxreg;

2896         DEBUGFUNC("ixgbe_disable_sec_rx_path_generic");


2899         secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXCTRL);
2900         secrxreg |= IXGBE_SECRXCTRL_RX_DIS;
2901         IXGBE_WRITE_REG(hw, IXGBE_SECRXCTRL, secrxreg);
2902         for (i = 0; i < IXGBE_MAX_SECRX_POLL; i++) {
2903                 secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXSTAT);
2904                 if (secrxreg & IXGBE_SECRXSTAT_SECRX_RDY)
2905                         break;
2906                 else
2907                         /* Use interrupt-safe sleep just in case */
2908                         usec_delay(1000);
2909         }

2911         /* For informational purposes only */
2912         if (i >= IXGBE_MAX_SECRX_POLL)
2913                 DEBUGOUT("Rx unit being enabled before security "
2914                          "path fully disabled.  Continuing with init.\n");

2916         return IXGBE_SUCCESS;
2917 }

2919 /**
2920  *  ixgbe_enable_sec_rx_path_generic - Enables the receive data path
2921  *  @hw: pointer to hardware structure
2922  *
2923  *  Enables the receive data path.
2924  **/
2925 s32 ixgbe_enable_sec_rx_path_generic(struct ixgbe_hw *hw)
2926 {
2927         int secrxreg;

2929         DEBUGFUNC("ixgbe_enable_sec_rx_path_generic");

2931         secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXCTRL);
2932         secrxreg &= ~IXGBE_SECRXCTRL_RX_DIS;
2933         IXGBE_WRITE_REG(hw, IXGBE_SECRXCTRL, secrxreg);
2934         IXGBE_WRITE_FLUSH(hw);

2936         return IXGBE_SUCCESS;
2937 }

2939 /**
```



```
2940  *  ixgbe_enable_rx_dma_generic - Enable the Rx DMA unit
2941  *  @hw: pointer to hardware structure
2942  *  @regval: register value to write to RXCTRL
2943  *
2944  *  Enables the Rx DMA unit
2945  **/
2946 s32 ixgbe_enable_rx_dma_generic(struct ixgbe_hw *hw, u32 regval)
2947 {
2948         DEBUGFUNC("ixgbe_enable_rx_dma_generic");

2950         IXGBE_WRITE_REG(hw, IXGBE_RXCTRL, regval);

2952         return IXGBE_SUCCESS;
2953 }

2955 /**
2956  *  ixgbe_blink_led_start_generic - Blink LED based on index.
2957  *  @hw: pointer to hardware structure
2958  *  @index: led number to blink
2959  **/
2960 s32 ixgbe_blink_led_start_generic(struct ixgbe_hw *hw, u32 index)
2961 {
2962         ixgbe_link_speed speed = 0;
2963         bool link_up = 0;
2964         u32 autoc_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC);
2965         u32 led_reg = IXGBE_READ_REG(hw, IXGBE_LEDCTL);

2967         DEBUGFUNC("ixgbe_blink_led_start_generic");

2969         /*
2970          * Link must be up to auto-blink the LEDs;
2971          * Force it if link is down.
2972          */
2973         hw->mac.ops.check_link(hw, &speed, &link_up, FALSE);

2975         if (!link_up) {
2976                 autoc_reg |= IXGBE_AUTOC_AN_RESTART;
2977                 autoc_reg |= IXGBE_AUTOC_FLU;
2978                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc_reg);
2979                 IXGBE_WRITE_FLUSH(hw);
2980                 msec_delay(10);
2981         }

2983         led_reg &= ~IXGBE_LED_MODE_MASK(index);
2984         led_reg |= IXGBE_LED_BLINK(index);
2985         IXGBE_WRITE_REG(hw, IXGBE_LEDCTL, led_reg);
2986         IXGBE_WRITE_FLUSH(hw);

2988         return IXGBE_SUCCESS;
2989 }
______unchanged_portion_omitted_

3040 /**
3041  *  ixgbe_get_san_mac_addr_generic - SAN MAC address retrieval from the EEPROM
3042  *  @hw: pointer to hardware structure
3043  *  @san_mac_addr: SAN MAC address
3044  *
3045  *  Reads the SAN MAC address from the EEPROM, if it's available.  This is
3046  *  per-port, so set_lan_id() must be called before reading the addresses.
3047  *  set_lan_id() is called by identify_sfp(), but this cannot be relied
3048  *  upon for non-SFP connections, so we must call it here.
3049  **/
3050 s32 ixgbe_get_san_mac_addr_generic(struct ixgbe_hw *hw, u8 *san_mac_addr)
3051 {
3052         u16 san_mac_data, san_mac_offset;
3053         u8 i;
```

```
3055         DEBUGFUNC("ixgbe_get_san_mac_addr_generic");

3057         /*
3058          * First read the EEPROM pointer to see if the MAC addresses are
3059          * available.  If they're not, no point in calling set_lan_id() here.
3060          */
3061         ixgbe_get_san_mac_addr_offset(hw, &san_mac_offset);
2763         (void) ixgbe_get_san_mac_addr_offset(hw, &san_mac_offset);

3063         if ((san_mac_offset == 0) || (san_mac_offset == 0xFFFF)) {
3064                 /*
3065                  * No addresses available in this EEPROM.  It's not an
3066                  * error though, so just wipe the local address and return.
3067                  */
3068                 for (i = 0; i < 6; i++)
3069                         san_mac_addr[i] = 0xFF;

3071                 goto san_mac_addr_out;
3072         }

3074         /* make sure we know which port we need to program */
3075         hw->mac.ops.set_lan_id(hw);
3076         /* apply the port offset to the address offset */
3077         (hw->bus.func) ? (san_mac_offset += IXGBE_SAN_MAC_ADDR_PORT1_OFFSET) :
3078                          (san_mac_offset += IXGBE_SAN_MAC_ADDR_PORT0_OFFSET);
3079         for (i = 0; i < 3; i++) {
3080                 hw->eeprom.ops.read(hw, san_mac_offset, &san_mac_data);
3081                 san_mac_addr[i * 2] = (u8)(san_mac_data);
3082                 san_mac_addr[i * 2 + 1] = (u8)(san_mac_data >> 8);
3083                 san_mac_offset++;
3084         }

3086 san_mac_addr_out:
3087         return IXGBE_SUCCESS;
3088 }

3090 /**
3091  *  ixgbe_set_san_mac_addr_generic - Write the SAN MAC address to the EEPROM
3092  *  @hw: pointer to hardware structure
3093  *  @san_mac_addr: SAN MAC address
3094  *
3095  *  Write a SAN MAC address to the EEPROM.
3096  **/
3097 s32 ixgbe_set_san_mac_addr_generic(struct ixgbe_hw *hw, u8 *san_mac_addr)
3098 {
3099         s32 status = IXGBE_SUCCESS;
3100         u16 san_mac_data, san_mac_offset;
3101         u8 i;

3103         DEBUGFUNC("ixgbe_set_san_mac_addr_generic");

3105         /* Look for SAN mac address pointer.  If not defined, return */
3106         ixgbe_get_san_mac_addr_offset(hw, &san_mac_offset);
2808         (void) ixgbe_get_san_mac_addr_offset(hw, &san_mac_offset);

3108         if ((san_mac_offset == 0) || (san_mac_offset == 0xFFFF)) {
3109                 status = IXGBE_ERR_NO_SAN_ADDR_PTR;
3110                 goto san_mac_addr_out;
3111         }

3113         /* Make sure we know which port we need to write */
3114         hw->mac.ops.set_lan_id(hw);
3115         /* Apply the port offset to the address offset */
3116         (hw->bus.func) ? (san_mac_offset += IXGBE_SAN_MAC_ADDR_PORT1_OFFSET) :
3117                          (san_mac_offset += IXGBE_SAN_MAC_ADDR_PORT0_OFFSET);
```



```
3119         for (i = 0; i < 3; i++) {
3120                 san_mac_data = (u16)((u16)(san_mac_addr[i * 2 + 1]) << 8);
3121                 san_mac_data |= (u16)(san_mac_addr[i * 2]);
3122                 hw->eeprom.ops.write(hw, san_mac_offset, san_mac_data);
3123                 san_mac_offset++;
3124         }

3126 san_mac_addr_out:
3127         return status;
3128 }

3130 /**
3131  *  ixgbe_get_pcie_msix_count_generic - Gets MSI-X vector count
3132  *  @hw: pointer to hardware structure
3133  *
3134  *  Read PCIe configuration space, and get the MSI-X vector count from
3135  *  the capabilities table.
3136  **/
3137 u16 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw)
2839 u32 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw)
3138 {
3139         u16 msix_count = 1;
3140         u16 max_msix_count;
3141         u16 pcie_offset;
2841         u32 msix_count = 64;

3143         switch (hw->mac.type) {
3144         case ixgbe_mac_82598EB:
3145                 pcie_offset = IXGBE_PCIE_MSIX_82598_CAPS;
3146                 max_msix_count = IXGBE_MAX_MSIX_VECTORS_82598;
3147                 break;
3148         case ixgbe_mac_82599EB:
3149         case ixgbe_mac_X540:
3150                 pcie_offset = IXGBE_PCIE_MSIX_82599_CAPS;
3151                 max_msix_count = IXGBE_MAX_MSIX_VECTORS_82599;
3152                 break;
3153         default:
3154                 return msix_count;
3155         }

3157         DEBUGFUNC("ixgbe_get_pcie_msix_count_generic");
3158         msix_count = IXGBE_READ_PCIE_WORD(hw, pcie_offset);
2844         if (hw->mac.msix_vectors_from_pcie) {
2845                 msix_count = IXGBE_READ_PCIE_WORD(hw,
2846                                                   IXGBE_PCIE_MSIX_82599_CAPS);
3159         msix_count &= IXGBE_PCIE_MSIX_TBL_SZ_MASK;

3161         /* MSI-X count is zero-based in HW */
2849                 /* MSI-X count is zero-based in HW, so increment to give
2850                  * proper value */
3162         msix_count++;
2852         }

3164         if (msix_count > max_msix_count)
3165                 msix_count = max_msix_count;

3167         return msix_count;
3168 }

3170 /**
3171  *  ixgbe_insert_mac_addr_generic - Find a RAR for this mac address
3172  *  @hw: pointer to hardware structure
3173  *  @addr: Address to put into receive address register
3174  *  @vmdq: VMDq pool to assign
3175  *
```

```
3176  *  Puts an ethernet address into a receive address register, or
3177  *  finds the rar that it is aleady in; adds to the pool list
3178  **/
3179 s32 ixgbe_insert_mac_addr_generic(struct ixgbe_hw *hw, u8 *addr, u32 vmdq)
3180 {
3181         static const u32 NO_EMPTY_RAR_FOUND = 0xFFFFFFFF;
3182         u32 first_empty_rar = NO_EMPTY_RAR_FOUND;
3183         u32 rar;
3184         u32 rar_low, rar_high;
3185         u32 addr_low, addr_high;

3187         DEBUGFUNC("ixgbe_insert_mac_addr_generic");

3189         /* swap bytes for HW little endian */
3190         addr_low  = addr[0] | (addr[1] << 8)
3191                             | (addr[2] << 16)
3192                             | (addr[3] << 24);
3193         addr_high = addr[4] | (addr[5] << 8);

3195         /*
3196          * Either find the mac_id in rar or find the first empty space.
3197          * rar_highwater points to just after the highest currently used
3198          * rar in order to shorten the search.  It grows when we add a new
3199          * rar to the top.
3200          */
3201         for (rar = 0; rar < hw->mac.rar_highwater; rar++) {
3202                 rar_high = IXGBE_READ_REG(hw, IXGBE_RAH(rar));

3204                 if (((IXGBE_RAH_AV & rar_high) == 0)
3205                     && first_empty_rar == NO_EMPTY_RAR_FOUND) {
3206                         first_empty_rar = rar;
3207                 } else if ((rar_high & 0xFFFF) == addr_high) {
3208                         rar_low = IXGBE_READ_REG(hw, IXGBE_RAL(rar));
3209                         if (rar_low == addr_low)
3210                                 break;    /* found it already in the rars */
3211                 }
3212         }

3214         if (rar < hw->mac.rar_highwater) {
3215                 /* already there so just add to the pool bits */
3216                 ixgbe_set_vmdq(hw, rar, vmdq);
2903                 (void) ixgbe_set_vmdq(hw, rar, vmdq);
3217         } else if (first_empty_rar != NO_EMPTY_RAR_FOUND) {
3218                 /* stick it into first empty RAR slot we found */
3219                 rar = first_empty_rar;
3220                 ixgbe_set_rar(hw, rar, addr, vmdq, IXGBE_RAH_AV);
2907                 (void) ixgbe_set_rar(hw, rar, addr, vmdq, IXGBE_RAH_AV);
3221         } else if (rar == hw->mac.rar_highwater) {
3222                 /* add it to the top of the list and inc the highwater mark */
3223                 ixgbe_set_rar(hw, rar, addr, vmdq, IXGBE_RAH_AV);
2910                 (void) ixgbe_set_rar(hw, rar, addr, vmdq, IXGBE_RAH_AV);
3224                 hw->mac.rar_highwater++;
3225         } else if (rar >= hw->mac.num_rar_entries) {
3226                 return IXGBE_ERR_INVALID_MAC_ADDR;
3227         }

3229         /*
3230          * If we found rar[0], make sure the default pool bit (we use pool 0)
3231          * remains cleared to be sure default pool packets will get delivered
3232          */
3233         if (rar == 0)
3234                 ixgbe_clear_vmdq(hw, rar, 0);
2921                 (void) ixgbe_clear_vmdq(hw, rar, 0);

3236         return rar;
3237 }
```

_____***unchanged_portion_omitted_***



```
3319 /**
3320  *  This function should only be involved in the IOV mode.
3321  *  In IOV mode, Default pool is next pool after the number of
3322  *  VFs advertized and not 0.
3323  *  MPSAR table needs to be updated for SAN_MAC RAR [hw->mac.san_mac_rar_index]
3324  *
3325  *  ixgbe_set_vmdq_san_mac - Associate default VMDq pool index with a rx address
3326  *  @hw: pointer to hardware struct
3327  *  @vmdq: VMDq pool index
3328  **/
3329 s32 ixgbe_set_vmdq_san_mac_generic(struct ixgbe_hw *hw, u32 vmdq)
3330 {
3331         u32 rar = hw->mac.san_mac_rar_index;

3333         DEBUGFUNC("ixgbe_set_vmdq_san_mac");

3335         if (vmdq < 32) {
3336                 IXGBE_WRITE_REG(hw, IXGBE_MPSAR_LO(rar), 1 << vmdq);
3337                 IXGBE_WRITE_REG(hw, IXGBE_MPSAR_HI(rar), 0);
3338         } else {
3339                 IXGBE_WRITE_REG(hw, IXGBE_MPSAR_LO(rar), 0);
3340                 IXGBE_WRITE_REG(hw, IXGBE_MPSAR_HI(rar), 1 << (vmdq - 32));
3341         }

3343         return IXGBE_SUCCESS;
3344 }

3346 /**
3347  *  ixgbe_init_uta_tables_generic - Initialize the Unicast Table Array
3348  *  @hw: pointer to hardware structure
3349  **/
3350 s32 ixgbe_init_uta_tables_generic(struct ixgbe_hw *hw)
3351 {
3352         int i;

3354         DEBUGFUNC("ixgbe_init_uta_tables_generic");
3355         DEBUGOUT(" Clearing UTA\n");

3357         for (i = 0; i < 128; i++)
3358                 IXGBE_WRITE_REG(hw, IXGBE_UTA(i), 0);

3360         return IXGBE_SUCCESS;
3361 }
```

_____***unchanged_portion_omitted_***

```
3410 /**
3411  *  ixgbe_set_vfta_generic - Set VLAN filter table
3412  *  @hw: pointer to hardware structure
3413  *  @vlan: VLAN id to write to VLAN filter
3414  *  @vind: VMDq output index that maps queue to VLAN id in VFVFB
3415  *  @vlan_on: boolean flag to turn on/off VLAN in VFVF
3416  *
3417  *  Turn on/off specified VLAN in the VLAN filter table.
3418  **/
3419 s32 ixgbe_set_vfta_generic(struct ixgbe_hw *hw, u32 vlan, u32 vind,
3420                            bool vlan_on)
3421 {
3422         s32 regindex;
3423         u32 bitindex;
3424         u32 vfta;
3085         u32 bits;
3086         u32 vt;
3425         u32 targetbit;
3426         s32 ret_val = IXGBE_SUCCESS;
3427         bool vfta_changed = FALSE;
```

```
3429         DEBUGFUNC("ixgbe_set_vfta_generic");

3431         if (vlan > 4095)
3432                 return IXGBE_ERR_PARAM;

3434         /*
3435          * this is a 2 part operation - first the VFTA, then the
3436          * VLVF and VLVFB if VT Mode is set
3437          * We don't write the VFTA until we know the VLVF part succeeded.
3438          */

3440         /* Part 1
3441          * The VFTA is a bitstring made up of 128 32-bit registers
3442          * that enable the particular VLAN id, much like the MTA:
3443          *    bits[11-5]: which register
3444          *    bits[4-0]:  which bit in the register
3445          */
3446         regindex = (vlan >> 5) & 0x7F;
3447         bitindex = vlan & 0x1F;
3448         targetbit = (1 << bitindex);
3449         vfta = IXGBE_READ_REG(hw, IXGBE_VFTA(regindex));

3451         if (vlan_on) {
3452                 if (!(vfta & targetbit)) {
3453                         vfta |= targetbit;
3454                         vfta_changed = TRUE;
3455                 }
3456         } else {
3457                 if ((vfta & targetbit)) {
3458                         vfta &= ~targetbit;
3459                         vfta_changed = TRUE;
3460                 }
3461         }

3463         /* Part 2
3464          * Call ixgbe_set_vlvf_generic to set VLVFB and VLVF
3465          */
3466         ret_val = ixgbe_set_vlvf_generic(hw, vlan, vind, vlan_on,
3467                                          &vfta_changed);
3468         if (ret_val != IXGBE_SUCCESS)
3469                 return ret_val;

3471         if (vfta_changed)
3472                 IXGBE_WRITE_REG(hw, IXGBE_VFTA(regindex), vfta);

3474         return IXGBE_SUCCESS;
3475 }

3477 /**
3478  *  ixgbe_set_vlvf_generic - Set VLAN Pool Filter
3479  *  @hw: pointer to hardware structure
3480  *  @vlan: VLAN id to write to VLAN filter
3481  *  @vind: VMDq output index that maps queue to VLAN id in VFVFB
3482  *  @vlan_on: boolean flag to turn on/off VLAN in VFVF
3483  *  @vfta_changed: pointer to boolean flag which indicates whether VFTA
3484  *                 should be changed
3485  *
3486  *  Turn on/off specified bit in VLVF table.
3487  **/
3488 s32 ixgbe_set_vlvf_generic(struct ixgbe_hw *hw, u32 vlan, u32 vind,
3489                            bool vlan_on, bool *vfta_changed)
3490 {
3491         u32 vt;

3493         DEBUGFUNC("ixgbe_set_vlvf_generic");
```



```
3495         if (vlan > 4095)
3496                 return IXGBE_ERR_PARAM;

3498         /* If VT Mode is set
3125          * If VT Mode is set
3499          *   Either vlan_on
3500          *     make sure the vlan is in VLVF
3501          *     set the vind bit in the matching VLVFB
3502          *   Or !vlan_on
3503          *     clear the pool bit and possibly the vind
3504          */
3505         vt = IXGBE_READ_REG(hw, IXGBE_VT_CTL);
3506         if (vt & IXGBE_VT_CTL_VT_ENABLE) {
3507                 s32 vlvf_index;
3508                 u32 bits;

3510                 vlvf_index = ixgbe_find_vlvf_slot(hw, vlan);
3511                 if (vlvf_index < 0)
3512                         return vlvf_index;

3514                 if (vlan_on) {
3515                         /* set the pool bit */
3516                         if (vind < 32) {
3517                                 bits = IXGBE_READ_REG(hw,
3518                                         IXGBE_VLVFB(vlvf_index * 2));
3144                                         IXGBE_VLVFB(vlvf_index*2));
3519                                 bits |= (1 << vind);
3520                                 IXGBE_WRITE_REG(hw,
3521                                         IXGBE_VLVFB(vlvf_index * 2),
3147                                         IXGBE_VLVFB(vlvf_index*2),
3522                                         bits);
3523                         } else {
3524                                 bits = IXGBE_READ_REG(hw,
3525                                         IXGBE_VLVFB((vlvf_index * 2) + 1));
3526                                 bits |= (1 << (vind - 32));
3151                                         IXGBE_VLVFB((vlvf_index*2)+1));
3152                                 bits |= (1 << (vind-32));
3527                                 IXGBE_WRITE_REG(hw,
3528                                         IXGBE_VLVFB((vlvf_index * 2) + 1),
3154                                         IXGBE_VLVFB((vlvf_index*2)+1),
3529                                         bits);
3530                         }
3531                 } else {
3532                         /* clear the pool bit */
3533                         if (vind < 32) {
3534                                 bits = IXGBE_READ_REG(hw,
3535                                         IXGBE_VLVFB(vlvf_index * 2));
3161                                         IXGBE_VLVFB(vlvf_index*2));
3536                                 bits &= ~(1 << vind);
3537                                 IXGBE_WRITE_REG(hw,
3538                                         IXGBE_VLVFB(vlvf_index * 2),
3164                                         IXGBE_VLVFB(vlvf_index*2),
3539                                         bits);
3540                                 bits |= IXGBE_READ_REG(hw,
3541                                         IXGBE_VLVFB((vlvf_index * 2) + 1));
3167                                         IXGBE_VLVFB((vlvf_index*2)+1));
3542                         } else {
3543                                 bits = IXGBE_READ_REG(hw,
3544                                         IXGBE_VLVFB((vlvf_index * 2) + 1));
3545                                 bits &= ~(1 << (vind - 32));
3170                                         IXGBE_VLVFB((vlvf_index*2)+1));
3171                                 bits &= ~(1 << (vind-32));
3546                                 IXGBE_WRITE_REG(hw,
3547                                         IXGBE_VLVFB((vlvf_index * 2) + 1),
3173                                         IXGBE_VLVFB((vlvf_index*2)+1),
```

```
3548                                    bits);
3549                            bits |= IXGBE_READ_REG(hw,
3550                                                   IXGBE_VLVFB(vlvf_index * 2));
3176                                                   IXGBE_VLVFB(vlvf_index*2));
3551                    }
3552            }

3554            /*
3555             * If there are still bits set in the VLVFB registers
3556             * for the VLAN ID indicated we need to see if the
3557             * caller is requesting that we clear the VFTA entry bit.
3558             * If the caller has requested that we clear the VFTA
3559             * entry bit but there are still pools/VFs using this VLAN
3560             * ID entry then ignore the request.  We're not worried
3561             * about the case where we're turning the VFTA VLAN ID
3562             * entry bit on, only when requested to turn it off as
3563             * there may be multiple pools and/or VFs using the
3564             * VLAN ID entry.  In that case we cannot clear the
3565             * VFTA bit until all pools/VFs using that VLAN ID have also
3566             * been cleared.  This will be indicated by "bits" being
3567             * zero.
3568             */
3569            if (bits) {
3570                    IXGBE_WRITE_REG(hw, IXGBE_VLVF(vlvf_index),
3571                                    (IXGBE_VLVF_VIEN | vlan));
3572                    if ((!vlan_on) && (vfta_changed != NULL)) {
3198                    if (!vlan_on) {
3573                            /* someone wants to clear the vfta entry
3574                             * but some pools/VFs are still using it.
3575                             * Ignore it. */
3576                            *vfta_changed = FALSE;
3202                            vfta_changed = FALSE;
3577                    }
3578            } else
3204             }
3205             else
3579                    IXGBE_WRITE_REG(hw, IXGBE_VLVF(vlvf_index), 0);
3580    }

3209    if (vfta_changed)
3210            IXGBE_WRITE_REG(hw, IXGBE_VFTA(regindex), vfta);

3582    return IXGBE_SUCCESS;
3583 }

3585 /**
3586  *  ixgbe_clear_vfta_generic - Clear VLAN filter table
3587  *  @hw: pointer to hardware structure
3588  *
3589  *  Clears the VLAN filer table, and the VMDq index associated with the filter
3590  **/
3591 s32 ixgbe_clear_vfta_generic(struct ixgbe_hw *hw)
3592 {
3593        u32 offset;

3595        DEBUGFUNC("ixgbe_clear_vfta_generic");

3597        for (offset = 0; offset < hw->mac.vft_size; offset++)
3598                IXGBE_WRITE_REG(hw, IXGBE_VFTA(offset), 0);

3600        for (offset = 0; offset < IXGBE_VLVF_ENTRIES; offset++) {
3601                IXGBE_WRITE_REG(hw, IXGBE_VLVF(offset), 0);
3602                IXGBE_WRITE_REG(hw, IXGBE_VLVFB(offset * 2), 0);
3603                IXGBE_WRITE_REG(hw, IXGBE_VLVFB((offset * 2) + 1), 0);
3232                IXGBE_WRITE_REG(hw, IXGBE_VLVFB(offset*2), 0);
3233                IXGBE_WRITE_REG(hw, IXGBE_VLVFB((offset*2)+1), 0);
```



```
3604        }

3606        return IXGBE_SUCCESS;
3607 }

3609 /**
3610  *  ixgbe_check_mac_link_generic - Determine link and speed status
3611  *  @hw: pointer to hardware structure
3612  *  @speed: pointer to link speed
3613  *  @link_up: TRUE when link is up
3614  *  @link_up_wait_to_complete: bool used to wait for link up or not
3615  *
3616  *  Reads the links register to determine if link is up and the current speed
3617  **/
3618 s32 ixgbe_check_mac_link_generic(struct ixgbe_hw *hw, ixgbe_link_speed *speed,
3619                                  bool *link_up, bool link_up_wait_to_complete)
3620 {
3621        u32 links_reg, links_orig;
3622        u32 i;

3624        DEBUGFUNC("ixgbe_check_mac_link_generic");

3626        /* clear the old state */
3627        links_orig = IXGBE_READ_REG(hw, IXGBE_LINKS);

3629        links_reg = IXGBE_READ_REG(hw, IXGBE_LINKS);

3631        if (links_orig != links_reg) {
3632                DEBUGOUT2("LINKS changed from %08X to %08X\n",
3633                          links_orig, links_reg);
3634        }

3636        if (link_up_wait_to_complete) {
3637                for (i = 0; i < IXGBE_LINK_UP_TIME; i++) {
3638                        if (links_reg & IXGBE_LINKS_UP) {
3639                                *link_up = TRUE;
3640                                break;
3641                        } else {
3642                                *link_up = FALSE;
3643                        }
3644                        msec_delay(100);
3645                        links_reg = IXGBE_READ_REG(hw, IXGBE_LINKS);
3646                }
3647        } else {
3648                if (links_reg & IXGBE_LINKS_UP)
3649                        *link_up = TRUE;
3650                else
3651                        *link_up = FALSE;
3652        }

3654        if ((links_reg & IXGBE_LINKS_SPEED_82599) ==
3655            IXGBE_LINKS_SPEED_10G_82599)
3656                *speed = IXGBE_LINK_SPEED_10GB_FULL;
3657        else if ((links_reg & IXGBE_LINKS_SPEED_82599) ==
3658                 IXGBE_LINKS_SPEED_1G_82599)
3659                *speed = IXGBE_LINK_SPEED_1GB_FULL;
3660        else if ((links_reg & IXGBE_LINKS_SPEED_82599) ==
3661                 IXGBE_LINKS_SPEED_100_82599)
3662                *speed = IXGBE_LINK_SPEED_100_FULL;
3663        else
3664                *speed = IXGBE_LINK_SPEED_UNKNOWN;

3296        /* if link is down, zero out the current_mode */
3297        if (*link_up == FALSE) {
3298                hw->fc.current_mode = ixgbe_fc_none;
3299                hw->fc.fc_was_autonegged = FALSE;
```

```
3300         }

3666         return IXGBE_SUCCESS;
3667 }
______unchanged_portion_omitted_

3765 /**
3402  *  ixgbe_device_supports_autoneg_fc - Check if phy supports autoneg flow
3403  *  control
3404  *  @hw: pointer to hardware structure
3405  *
3406  *  There are several phys that do not support autoneg flow control. This
3407  *  function check the device id to see if the associated phy supports
3408  *  autoneg flow control.
3409  **/
3410 static s32 ixgbe_device_supports_autoneg_fc(struct ixgbe_hw *hw)
3411 {

3413         DEBUGFUNC("ixgbe_device_supports_autoneg_fc");

3415         switch (hw->device_id) {
3416         case IXGBE_DEV_ID_82599_T3_LOM:
3417                 return IXGBE_SUCCESS;
3418         default:
3419                 return IXGBE_ERR_FC_NOT_SUPPORTED;
3420         }
3421 }

3423 /**
3766  *  ixgbe_set_mac_anti_spoofing - Enable/Disable MAC anti-spoofing
3767  *  @hw: pointer to hardware structure
3768  *  @enable: enable or disable switch for anti-spoofing
3769  *  @pf: Physical Function pool - do not enable anti-spoofing for the PF
3770  *
3771  **/
3772 void ixgbe_set_mac_anti_spoofing(struct ixgbe_hw *hw, bool enable, int pf)
3773 {
3774         int j;
3775         int pf_target_reg = pf >> 3;
3776         int pf_target_shift = pf % 8;
3777         u32 pfvfspoof = 0;

3779         if (hw->mac.type == ixgbe_mac_82598EB)
3780                 return;

3782         if (enable)
3783                 pfvfspoof = IXGBE_SPOOF_MACAS_MASK;

3785         /*
3786          * PFVFSPOOF register array is size 8 with 8 bits assigned to
3787          * MAC anti-spoof enables in each register array element.
3788          */
3789         for (j = 0; j < pf_target_reg; j++)
3447         for (j = 0; j < IXGBE_PFVFSPOOF_REG_COUNT; j++)
3790                 IXGBE_WRITE_REG(hw, IXGBE_PFVFSPOOF(j), pfvfspoof);

3450         /* If not enabling anti-spoofing then done */
3451         if (!enable)
3452                 return;

3792         /*
3793          * The PF should be allowed to spoof so that it can support
3794          * emulation mode NICs.  Do not set the bits assigned to the PF
3456          * emulation mode NICs.  Reset the bit assigned to the PF
3795          */
3796         pfvfspoof &= (1 << pf_target_shift) - 1;
```



```
3797         IXGBE_WRITE_REG(hw, IXGBE_PFVFSPOOF(j), pfvfspoof);

3799         /*
3800          * Remaining pools belong to the PF so they do not need to have
3801          * anti-spoofing enabled.
3802          */
3803         for (j++; j < IXGBE_PFVFSPOOF_REG_COUNT; j++)
3804                 IXGBE_WRITE_REG(hw, IXGBE_PFVFSPOOF(j), 0);
3458         pfvfspoof = IXGBE_READ_REG(hw, IXGBE_PFVFSPOOF(pf_target_reg));
3459         pfvfspoof ^= (1 << pf_target_shift);
3460         IXGBE_WRITE_REG(hw, IXGBE_PFVFSPOOF(pf_target_reg), pfvfspoof);
3805 }
______unchanged_portion_omitted_

3848 /**
3849  *  ixgbe_enable_relaxed_ordering_gen2 - Enable relaxed ordering
3850  *  @hw: pointer to hardware structure
3851  *
3852  **/
3853 void ixgbe_enable_relaxed_ordering_gen2(struct ixgbe_hw *hw)
3854 {
3855         u32 regval;
3856         u32 i;

3858         DEBUGFUNC("ixgbe_enable_relaxed_ordering_gen2");

3860         /* Enable relaxed ordering */
3861         for (i = 0; i < hw->mac.max_tx_queues; i++) {
3862                 regval = IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL_82599(i));
3863                 regval |= IXGBE_DCA_TXCTRL_DESC_WRO_EN;
3519                 regval |= IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
3864                 IXGBE_WRITE_REG(hw, IXGBE_DCA_TXCTRL_82599(i), regval);
3865         }

3867         for (i = 0; i < hw->mac.max_rx_queues; i++) {
3868                 regval = IXGBE_READ_REG(hw, IXGBE_DCA_RXCTRL(i));
3869                 regval |= IXGBE_DCA_RXCTRL_DATA_WRO_EN |
3870                           IXGBE_DCA_RXCTRL_HEAD_WRO_EN;
3525                 regval |= (IXGBE_DCA_RXCTRL_DESC_WRO_EN |
3526                            IXGBE_DCA_RXCTRL_DESC_HSRO_EN);
3871                 IXGBE_WRITE_REG(hw, IXGBE_DCA_RXCTRL(i), regval);
3872         }

3874 }

3876 /**
3877  *  ixgbe_calculate_checksum - Calculate checksum for buffer
3878  *  @buffer: pointer to EEPROM
3879  *  @length: size of EEPROM to calculate a checksum for
3880  *  Calculates the checksum for some buffer on a specified length.  The
3881  *  checksum calculated is returned.
3882  **/
3883 static u8 ixgbe_calculate_checksum(u8 *buffer, u32 length)
3884 {
3885         u32 i;
3886         u8 sum = 0;

3888         DEBUGFUNC("ixgbe_calculate_checksum");

3890         if (!buffer)
3891                 return 0;

3893         for (i = 0; i < length; i++)
3894                 sum += buffer[i];

3896         return (u8) (0 - sum);
```

```
3897 }
3899 /**
3900  *  ixgbe_host_interface_command - Issue command to manageability block
3901  *  @hw: pointer to the HW structure
3902  *  @buffer: contains the command to write and where the return status will
3903  *   be placed
3904  *  @length: length of buffer, must be multiple of 4 bytes
3905  *
3906  *  Communicates with the manageability block.  On success return IXGBE_SUCCESS
3907  *  else return IXGBE_ERR_HOST_INTERFACE_COMMAND.
3908  **/
3909 static s32 ixgbe_host_interface_command(struct ixgbe_hw *hw, u32 *buffer,
3910                                         u32 length)
3911 {
3912         u32 hicr, i, bi;
3913         u32 hdr_size = sizeof(struct ixgbe_hic_hdr);
3914         u8 buf_len, dword_len;
3916         s32 ret_val = IXGBE_SUCCESS;
3918         DEBUGFUNC("ixgbe_host_interface_command");
3920         if (length == 0 || length & 0x3 ||
3921             length > IXGBE_HI_MAX_BLOCK_BYTE_LENGTH) {
3922                 DEBUGOUT("Buffer length failure.\n");
3923                 ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;
3924                 goto out;
3925         }
3927         /* Check that the host interface is enabled. */
3928         hicr = IXGBE_READ_REG(hw, IXGBE_HICR);
3929         if ((hicr & IXGBE_HICR_EN) == 0) {
3930                 DEBUGOUT("IXGBE_HOST_EN bit disabled.\n");
3931                 ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;
3932                 goto out;
3933         }
3935         /* Calculate length in DWORDs */
3936         dword_len = length >> 2;
3938         /*
3939          * The device driver writes the relevant command block
3940          * into the ram area.
3941          */
3942         for (i = 0; i < dword_len; i++)
3943                 IXGBE_WRITE_REG_ARRAY(hw, IXGBE_FLEX_MNG,
3944                                       i, IXGBE_CPU_TO_LE32(buffer[i]));
3946         /* Setting this bit tells the ARC that a new command is pending. */
3947         IXGBE_WRITE_REG(hw, IXGBE_HICR, hicr | IXGBE_HICR_C);
3949         for (i = 0; i < IXGBE_HI_COMMAND_TIMEOUT; i++) {
3950                 hicr = IXGBE_READ_REG(hw, IXGBE_HICR);
3951                 if (!(hicr & IXGBE_HICR_C))
3952                         break;
3953                 msec_delay(1);
3954         }
3956         /* Check command successful completion. */
3957         if (i == IXGBE_HI_COMMAND_TIMEOUT ||
3958             (!(IXGBE_READ_REG(hw, IXGBE_HICR) & IXGBE_HICR_SV))) {
3959                 DEBUGOUT("Command has failed with no status valid.\n");
3960                 ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;
3961                 goto out;
3962         }
```



```
3964         /* Calculate length in DWORDs */
3965         dword_len = hdr_size >> 2;
3967         /* first pull in the header so we know the buffer length */
3968         for (bi = 0; bi < dword_len; bi++) {
3969                 buffer[bi] = IXGBE_READ_REG_ARRAY(hw, IXGBE_FLEX_MNG, bi);
3970                 IXGBE_LE32_TO_CPUS(&buffer[bi]);
3971         }
3973         /* If there is any thing in data position pull it in */
3974         buf_len = ((struct ixgbe_hic_hdr *)buffer)->buf_len;
3975         if (buf_len == 0)
3976                 goto out;
3978         if (length < (buf_len + hdr_size)) {
3979                 DEBUGOUT("Buffer not large enough for reply message.\n");
3980                 ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;
3981                 goto out;
3982         }
3984         /* Calculate length in DWORDs, add 3 for odd lengths */
3985         dword_len = (buf_len + 3) >> 2;
3987         /* Pull in the rest of the buffer (bi is where we left off)*/
3988         for (; bi <= dword_len; bi++) {
3989                 buffer[bi] = IXGBE_READ_REG_ARRAY(hw, IXGBE_FLEX_MNG, bi);
3990                 IXGBE_LE32_TO_CPUS(&buffer[bi]);
3991         }
3993 out:
3994         return ret_val;
3995 }
3997 /**
3998  *  ixgbe_set_fw_drv_ver_generic - Sends driver version to firmware
3999  *  @hw: pointer to the HW structure
4000  *  @maj: driver version major number
4001  *  @min: driver version minor number
4002  *  @build: driver version build number
4003  *  @sub: driver version sub build number
4004  *
4005  *  Sends driver version number to firmware through the manageability
4006  *  block.  On success return IXGBE_SUCCESS
4007  *  else returns IXGBE_ERR_SWFW_SYNC when encountering an error acquiring
4008  *  semaphore or IXGBE_ERR_HOST_INTERFACE_COMMAND when command fails.
4009  **/
4010 s32 ixgbe_set_fw_drv_ver_generic(struct ixgbe_hw *hw, u8 maj, u8 min,
4011                                  u8 build, u8 sub)
4012 {
4013         struct ixgbe_hic_drv_info fw_cmd;
4014         int i;
4015         s32 ret_val = IXGBE_SUCCESS;
4017         DEBUGFUNC("ixgbe_set_fw_drv_ver_generic");
4019         if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_SW_MNG_SM)
4020             != IXGBE_SUCCESS) {
4021                 ret_val = IXGBE_ERR_SWFW_SYNC;
4022                 goto out;
4023         }
4025         fw_cmd.hdr.cmd = FW_CEM_CMD_DRIVER_INFO;
4026         fw_cmd.hdr.buf_len = FW_CEM_CMD_DRIVER_INFO_LEN;
4027         fw_cmd.hdr.cmd_or_resp.cmd_resv = FW_CEM_CMD_RESERVED;
4028         fw_cmd.port_num = (u8)hw->bus.func;
```

```
4029         fw_cmd.ver_maj = maj;
4030         fw_cmd.ver_min = min;
4031         fw_cmd.ver_build = build;
4032         fw_cmd.ver_sub = sub;
4033         fw_cmd.hdr.checksum = 0;
4034         fw_cmd.hdr.checksum = ixgbe_calculate_checksum((u8 *)&fw_cmd,
4035                                 (FW_CEM_HDR_LEN + fw_cmd.hdr.buf_len));
4036         fw_cmd.pad = 0;
4037         fw_cmd.pad2 = 0;

4039         for (i = 0; i <= FW_CEM_MAX_RETRIES; i++) {
4040                 ret_val = ixgbe_host_interface_command(hw, (u32 *)&fw_cmd,
4041                                                        sizeof(fw_cmd));
4042                 if (ret_val != IXGBE_SUCCESS)
4043                         continue;

4045                 if (fw_cmd.hdr.cmd_or_resp.ret_status ==
4046                     FW_CEM_RESP_STATUS_SUCCESS)
4047                         ret_val = IXGBE_SUCCESS;
4048                 else
4049                         ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;

4051                 break;
4052         }

4054         hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_SW_MNG_SM);
4055 out:
4056         return ret_val;
4057 }

4059 /**
4060  * ixgbe_set_rxpba_generic - Initialize Rx packet buffer
4061  * @hw: pointer to hardware structure
4062  * @num_pb: number of packet buffers to allocate
4063  * @headroom: reserve n KB of headroom
4064  * @strategy: packet buffer allocation strategy
4065  **/
4066 void ixgbe_set_rxpba_generic(struct ixgbe_hw *hw, int num_pb, u32 headroom,
4067                              int strategy)
4068 {
4069         u32 pbsize = hw->mac.rx_pb_size;
4070         int i = 0;
4071         u32 rxpktsize, txpktsize, txpbthresh;

4073         /* Reserve headroom */
4074         pbsize -= headroom;

4076         if (!num_pb)
4077                 num_pb = 1;

4079         /* Divide remaining packet buffer space amongst the number of packet
4080          * buffers requested using supplied strategy.
4081          */
4082         switch (strategy) {
4083         case PBA_STRATEGY_WEIGHTED:
4084                 /* ixgbe_dcb_pba_80_48 strategy weight first half of packet
4085                  * buffer with 5/8 of the packet buffer space.
4086                  */
4087                 rxpktsize = (pbsize * 5) / (num_pb * 4);
4088                 pbsize -= rxpktsize * (num_pb / 2);
4089                 rxpktsize <<= IXGBE_RXPBSIZE_SHIFT;
4090                 for (; i < (num_pb / 2); i++)
4091                         IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), rxpktsize);
4092                 /* Fall through to configure remaining packet buffers */
4093         case PBA_STRATEGY_EQUAL:
4094                 rxpktsize = (pbsize / (num_pb - i)) << IXGBE_RXPBSIZE_SHIFT;
```



```
4095                 for (; i < num_pb; i++)
4096                         IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), rxpktsize);
4097                 break;
4098         default:
4099                 break;
4100         }

4102         /* Only support an equally distributed Tx packet buffer strategy. */
4103         txpktsize = IXGBE_TXPBSIZE_MAX / num_pb;
4104         txpbthresh = (txpktsize / 1024) - IXGBE_TXPKT_SIZE_MAX;
4105         for (i = 0; i < num_pb; i++) {
4106                 IXGBE_WRITE_REG(hw, IXGBE_TXPBSIZE(i), txpktsize);
4107                 IXGBE_WRITE_REG(hw, IXGBE_TXPBTHRESH(i), txpbthresh);
4108         }

4110         /* Clear unused TCs, if any, to zero buffer size*/
4111         for (; i < IXGBE_MAX_PB; i++) {
4112                 IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), 0);
4113                 IXGBE_WRITE_REG(hw, IXGBE_TXPBSIZE(i), 0);
4114                 IXGBE_WRITE_REG(hw, IXGBE_TXPBTHRESH(i), 0);
4115         }
4116 }

4118 /**
4119  * ixgbe_clear_tx_pending - Clear pending TX work from the PCIe fifo
4120  * @hw: pointer to the hardware structure
4121  *
4122  * The 82599 and x540 MACs can experience issues if TX work is still pending
4123  * when a reset occurs.  This function prevents this by flushing the PCIe
4124  * buffers on the system.
4125  **/
4126 void ixgbe_clear_tx_pending(struct ixgbe_hw *hw)
4127 {
4128         u32 gcr_ext, hlreg0;

4130         /*
4131          * If double reset is not requested then all transactions should
4132          * already be clear and as such there is no work to do
4133          */
4134         if (!(hw->mac.flags & IXGBE_FLAGS_DOUBLE_RESET_REQUIRED))
4135                 return;

4137         /*
4138          * Set loopback enable to prevent any transmits from being sent
4139          * should the link come up.  This assumes that the RXCTRL.RXEN bit
4140          * has already been cleared.
4141          */
4142         hlreg0 = IXGBE_READ_REG(hw, IXGBE_HLREG0);
4143         IXGBE_WRITE_REG(hw, IXGBE_HLREG0, hlreg0 | IXGBE_HLREG0_LPBK);

4145         /* initiate cleaning flow for buffers in the PCIe transaction layer */
4146         gcr_ext = IXGBE_READ_REG(hw, IXGBE_GCR_EXT);
4147         IXGBE_WRITE_REG(hw, IXGBE_GCR_EXT,
4148                         gcr_ext | IXGBE_GCR_EXT_BUFFERS_CLEAR);

4150         /* Flush all writes and allow 20usec for all transactions to clear */
4151         IXGBE_WRITE_FLUSH(hw);
4152         usec_delay(20);

4154         /* restore previous register values */
4155         IXGBE_WRITE_REG(hw, IXGBE_GCR_EXT, gcr_ext);
4156         IXGBE_WRITE_REG(hw, IXGBE_HLREG0, hlreg0);
4157 }
```

```
**********************************************************
    6867 Thu Jul 12 12:22:33 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_common.h
XXXX Intel X540 support
**********************************************************
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  Copyright (c) 2001-2010, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_common.h,v 1.12 2012/07/05 20:51:44 jfv Exp
/*$FreeBSD$*/

#ifndef _IXGBE_COMMON_H_
#define _IXGBE_COMMON_H_

#include "ixgbe_type.h"
#define IXGBE_WRITE_REG64(hw, reg, value) \
        do { \
        { \
                IXGBE_WRITE_REG(hw, reg, (u32) value); \
                IXGBE_WRITE_REG(hw, reg + 4, (u32) (value >> 32)); \
        } while (0)
        }

u16 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw);
u32 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw);

s32 ixgbe_init_ops_generic(struct ixgbe_hw *hw);
s32 ixgbe_init_hw_generic(struct ixgbe_hw *hw);
s32 ixgbe_start_hw_generic(struct ixgbe_hw *hw);
s32 ixgbe_start_hw_gen2(struct ixgbe_hw *hw);
s32 ixgbe_clear_hw_cntrs_generic(struct ixgbe_hw *hw);
s32 ixgbe_read_pba_num_generic(struct ixgbe_hw *hw, u32 *pba_num);
s32 ixgbe_read_pba_string_generic(struct ixgbe_hw *hw, u8 *pba_num,
                                  u32 pba_num_size);
s32 ixgbe_read_pba_length_generic(struct ixgbe_hw *hw, u32 *pba_num_size);
s32 ixgbe_get_mac_addr_generic(struct ixgbe_hw *hw, u8 *mac_addr);
```



```
s32 ixgbe_get_bus_info_generic(struct ixgbe_hw *hw);
void ixgbe_set_lan_id_multi_port_pcie(struct ixgbe_hw *hw);
s32 ixgbe_stop_adapter_generic(struct ixgbe_hw *hw);

s32 ixgbe_led_on_generic(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_led_off_generic(struct ixgbe_hw *hw, u32 index);

s32 ixgbe_init_eeprom_params_generic(struct ixgbe_hw *hw);
s32 ixgbe_write_eeprom_generic(struct ixgbe_hw *hw, u16 offset, u16 data);
s32 ixgbe_write_eeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
                                               u16 words, u16 *data);
s32 ixgbe_read_eerd_generic(struct ixgbe_hw *hw, u16 offset, u16 *data);
s32 ixgbe_read_eerd_buffer_generic(struct ixgbe_hw *hw, u16 offset,
                                   u16 words, u16 *data);
s32 ixgbe_write_eewr_generic(struct ixgbe_hw *hw, u16 offset, u16 data);
s32 ixgbe_write_eewr_buffer_generic(struct ixgbe_hw *hw, u16 offset,
                                    u16 words, u16 *data);
s32 ixgbe_read_eeprom_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
                                       u16 *data);
s32 ixgbe_read_eeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
                                              u16 words, u16 *data);
u16 ixgbe_calc_eeprom_checksum_generic(struct ixgbe_hw *hw);
s32 ixgbe_validate_eeprom_checksum_generic(struct ixgbe_hw *hw,
                                           u16 *checksum_val);
s32 ixgbe_update_eeprom_checksum_generic(struct ixgbe_hw *hw);
s32 ixgbe_poll_eerd_eewr_done(struct ixgbe_hw *hw, u32 ee_reg);

s32 ixgbe_set_rar_generic(struct ixgbe_hw *hw, u32 index, u8 *addr, u32 vmdq,
                          u32 enable_addr);
s32 ixgbe_clear_rar_generic(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_init_rx_addrs_generic(struct ixgbe_hw *hw);
s32 ixgbe_update_mc_addr_list_generic(struct ixgbe_hw *hw, u8 *mc_addr_list,
                                      u32 mc_addr_count,
                                      ixgbe_mc_addr_itr func, bool clear);
                                      ixgbe_mc_addr_itr func);
s32 ixgbe_update_uc_addr_list_generic(struct ixgbe_hw *hw, u8 *addr_list,
                                      u32 addr_count, ixgbe_mc_addr_itr func);
s32 ixgbe_enable_mc_generic(struct ixgbe_hw *hw);
s32 ixgbe_disable_mc_generic(struct ixgbe_hw *hw);
s32 ixgbe_enable_rx_dma_generic(struct ixgbe_hw *hw, u32 regval);
s32 ixgbe_disable_sec_rx_path_generic(struct ixgbe_hw *hw);
s32 ixgbe_enable_sec_rx_path_generic(struct ixgbe_hw *hw);

s32 ixgbe_fc_enable_generic(struct ixgbe_hw *hw);
void ixgbe_fc_autoneg(struct ixgbe_hw *hw);
s32 ixgbe_setup_fc(struct ixgbe_hw *hw, s32 packetbuf_num);
s32 ixgbe_fc_enable_generic(struct ixgbe_hw *hw, s32 packtetbuf_num);
s32 ixgbe_fc_autoneg(struct ixgbe_hw *hw);

s32 ixgbe_validate_mac_addr(u8 *mac_addr);
s32 ixgbe_acquire_swfw_sync(struct ixgbe_hw *hw, u16 mask);
void ixgbe_release_swfw_sync(struct ixgbe_hw *hw, u16 mask);
s32 ixgbe_disable_pcie_master(struct ixgbe_hw *hw);

s32 ixgbe_blink_led_start_generic(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_blink_led_stop_generic(struct ixgbe_hw *hw, u32 index);

s32 ixgbe_get_san_mac_addr_generic(struct ixgbe_hw *hw, u8 *san_mac_addr);
s32 ixgbe_set_san_mac_addr_generic(struct ixgbe_hw *hw, u8 *san_mac_addr);

s32 ixgbe_set_vmdq_generic(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
s32 ixgbe_set_vmdq_san_mac_generic(struct ixgbe_hw *hw, u32 vmdq);
s32 ixgbe_clear_vmdq_generic(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
s32 ixgbe_insert_mac_addr_generic(struct ixgbe_hw *hw, u8 *addr, u32 vmdq);
s32 ixgbe_init_uta_tables_generic(struct ixgbe_hw *hw);
s32 ixgbe_set_vfta_generic(struct ixgbe_hw *hw, u32 vlan,
```

```
118                          u32 vind, bool vlan_on);
119 s32 ixgbe_set_vlvf_generic(struct ixgbe_hw *hw, u32 vlan, u32 vind,
120                            bool vlan_on, bool *vfta_changed);
121 s32 ixgbe_clear_vfta_generic(struct ixgbe_hw *hw);
122 s32 ixgbe_find_vlvf_slot(struct ixgbe_hw *hw, u32 vlan);

124 s32 ixgbe_check_mac_link_generic(struct ixgbe_hw *hw,
125                                  ixgbe_link_speed *speed,
126                                  bool *link_up, bool link_up_wait_to_complete);

128 s32 ixgbe_get_wwn_prefix_generic(struct ixgbe_hw *hw, u16 *wwnn_prefix,
129                                  u16 *wwpn_prefix);

131 s32 ixgbe_get_fcoe_boot_status_generic(struct ixgbe_hw *hw, u16 *bs);
132 void ixgbe_set_mac_anti_spoofing(struct ixgbe_hw *hw, bool enable, int pf);
133 void ixgbe_set_vlan_anti_spoofing(struct ixgbe_hw *hw, bool enable, int vf);
134 s32 ixgbe_get_device_caps_generic(struct ixgbe_hw *hw, u16 *device_caps);
135 void ixgbe_set_rxpba_generic(struct ixgbe_hw *hw, int num_pb, u32 headroom,
136                              int strategy);
137 void ixgbe_enable_relaxed_ordering_gen2(struct ixgbe_hw *hw);
138 s32 ixgbe_set_fw_drv_ver_generic(struct ixgbe_hw *hw, u8 maj, u8 min,
139                                  u8 build, u8 ver);
140 void ixgbe_clear_tx_pending(struct ixgbe_hw *hw);
141 #endif /* IXGBE_COMMON */
```

```
*********************************************************
   18624 Thu Jul 12 12:22:34 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_gld.c
XXXX Intel X540 support
*********************************************************
______unchanged_portion_omitted_

 605 /* ARGSUSED */
 606 int
 607 ixgbe_set_priv_prop(ixgbe_t *ixgbe, const char *pr_name,
 608     uint_t pr_valsize, const void *pr_val)
 609 {
 610         int err = 0;
 611         long result;
 612         struct ixgbe_hw *hw = &ixgbe->hw;
 613         int i;

 615         if (strcmp(pr_name, "_tx_copy_thresh") == 0) {
 616                 if (pr_val == NULL) {
 617                         err = EINVAL;
 618                         return (err);
 619                 }
 620                 (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
 621                 if (result < MIN_TX_COPY_THRESHOLD ||
 622                     result > MAX_TX_COPY_THRESHOLD)
 623                         err = EINVAL;
 624                 else {
 625                         ixgbe->tx_copy_thresh = (uint32_t)result;
 626                 }
 627                 return (err);
 628         }
 629         if (strcmp(pr_name, "_tx_recycle_thresh") == 0) {
 630                 if (pr_val == NULL) {
 631                         err = EINVAL;
 632                         return (err);
 633                 }
 634                 (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
 635                 if (result < MIN_TX_RECYCLE_THRESHOLD ||
 636                     result > MAX_TX_RECYCLE_THRESHOLD)
 637                         err = EINVAL;
 638                 else {
 639                         ixgbe->tx_recycle_thresh = (uint32_t)result;
 640                 }
 641                 return (err);
 642         }
 643         if (strcmp(pr_name, "_tx_overload_thresh") == 0) {
 644                 if (pr_val == NULL) {
 645                         err = EINVAL;
 646                         return (err);
 647                 }
 648                 (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
 649                 if (result < MIN_TX_OVERLOAD_THRESHOLD ||
 650                     result > MAX_TX_OVERLOAD_THRESHOLD)
 651                         err = EINVAL;
 652                 else {
 653                         ixgbe->tx_overload_thresh = (uint32_t)result;
 654                 }
 655                 return (err);
 656         }
 657         if (strcmp(pr_name, "_tx_resched_thresh") == 0) {
 658                 if (pr_val == NULL) {
 659                         err = EINVAL;
 660                         return (err);
 661                 }
 662                 (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
 663                 if (result < MIN_TX_RESCHED_THRESHOLD ||
```



```
 664                     result > MAX_TX_RESCHED_THRESHOLD)
 665                         err = EINVAL;
 666                 else {
 667                         ixgbe->tx_resched_thresh = (uint32_t)result;
 668                 }
 669                 return (err);
 670         }
 671         if (strcmp(pr_name, "_rx_copy_thresh") == 0) {
 672                 if (pr_val == NULL) {
 673                         err = EINVAL;
 674                         return (err);
 675                 }
 676                 (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
 677                 if (result < MIN_RX_COPY_THRESHOLD ||
 678                     result > MAX_RX_COPY_THRESHOLD)
 679                         err = EINVAL;
 680                 else {
 681                         ixgbe->rx_copy_thresh = (uint32_t)result;
 682                 }
 683                 return (err);
 684         }
 685         if (strcmp(pr_name, "_rx_limit_per_intr") == 0) {
 686                 if (pr_val == NULL) {
 687                         err = EINVAL;
 688                         return (err);
 689                 }
 690                 (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
 691                 if (result < MIN_RX_LIMIT_PER_INTR ||
 692                     result > MAX_RX_LIMIT_PER_INTR)
 693                         err = EINVAL;
 694                 else {
 695                         ixgbe->rx_limit_per_intr = (uint32_t)result;
 696                 }
 697                 return (err);
 698         }
 699         if (strcmp(pr_name, "_intr_throttling") == 0) {
 700                 if (pr_val == NULL) {
 701                         err = EINVAL;
 702                         return (err);
 703                 }
 704                 (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);

 706                 if (result < ixgbe->capab->min_intr_throttle ||
 707                     result > ixgbe->capab->max_intr_throttle)
 708                         err = EINVAL;
 709                 else {
 710                         ixgbe->intr_throttling[0] = (uint32_t)result;

 712                         /*
 713                          * 82599 and X540 require the interupt throttling
 714                          * rate is a multiple of 8. This is enforced by the
 715                          * register definiton.
 713                          * 82599 requires the interupt throttling rate is
 714                          * a multiple of 8. This is enforced by the register
 715                          * definiton.
 716                          */
 717                         if (hw->mac.type >= ixgbe_mac_82599EB)
 717                         if (hw->mac.type == ixgbe_mac_82599EB)
 718                                 ixgbe->intr_throttling[0] =
 719                                     ixgbe->intr_throttling[0] & 0xFF8;

 721                         for (i = 0; i < MAX_INTR_VECTOR; i++)
 722                                 ixgbe->intr_throttling[i] =
 723                                     ixgbe->intr_throttling[0];

 725                         /* Set interrupt throttling rate */
```

```
726                         for (i = 0; i < ixgbe->intr_cnt; i++)
727                                 IXGBE_WRITE_REG(hw, IXGBE_EITR(i),
728                                     ixgbe->intr_throttling[i]);
729                 }
730                 return (err);
731         }
732         return (ENOTSUP);
733 }
```

***unchanged_portion_omitted_***

```
**********************************************************
  142881 Thu Jul 12 12:22:34 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_main.c
XXXX Intel X540 support
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright(c) 2007-2010 Intel Corporation. All rights reserved.
  24  */

  26 /*
  27  * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
  28  * Copyright (c) 2012, Joyent, Inc. All rights reserved.
  29  * Copyright 2012, Nexenta Systems, Inc. All rights reserved.
  30  */

  32 #include "ixgbe_sw.h"

  34 static char ixgbe_ident[] = "Intel 10Gb Ethernet";
  35 static char ixgbe_version[] = "ixgbe 1.1.7";

  37 /*
  38  * Local function protoypes
  39  */
  40 static int ixgbe_register_mac(ixgbe_t *);
  41 static int ixgbe_identify_hardware(ixgbe_t *);
  42 static int ixgbe_regs_map(ixgbe_t *);
  43 static void ixgbe_init_properties(ixgbe_t *);
  44 static int ixgbe_init_driver_settings(ixgbe_t *);
  45 static void ixgbe_init_locks(ixgbe_t *);
  46 static void ixgbe_destroy_locks(ixgbe_t *);
  47 static int ixgbe_init(ixgbe_t *);
  48 static int ixgbe_chip_start(ixgbe_t *);
  49 static void ixgbe_chip_stop(ixgbe_t *);
  50 static int ixgbe_reset(ixgbe_t *);
  51 static void ixgbe_tx_clean(ixgbe_t *);
  52 static boolean_t ixgbe_tx_drain(ixgbe_t *);
  53 static boolean_t ixgbe_rx_drain(ixgbe_t *);
  54 static int ixgbe_alloc_rings(ixgbe_t *);
  55 static void ixgbe_free_rings(ixgbe_t *);
  56 static int ixgbe_alloc_rx_data(ixgbe_t *);
  57 static void ixgbe_free_rx_data(ixgbe_t *);
  58 static void ixgbe_setup_rings(ixgbe_t *);
  59 static void ixgbe_setup_rx(ixgbe_t *);
  60 static void ixgbe_setup_tx(ixgbe_t *);
  61 static void ixgbe_setup_rx_ring(ixgbe_rx_ring_t *);
```



```
  62 static void ixgbe_setup_tx_ring(ixgbe_tx_ring_t *);
  63 static void ixgbe_setup_rss(ixgbe_t *);
  64 static void ixgbe_setup_vmdq(ixgbe_t *);
  65 static void ixgbe_setup_vmdq_rss(ixgbe_t *);
  66 static void ixgbe_init_unicst(ixgbe_t *);
  67 static int ixgbe_unicst_find(ixgbe_t *, const uint8_t *);
  68 static void ixgbe_setup_multicst(ixgbe_t *);
  69 static void ixgbe_get_hw_state(ixgbe_t *);
  70 static void ixgbe_setup_vmdq_rss_conf(ixgbe_t *ixgbe);
  71 static void ixgbe_get_conf(ixgbe_t *);
  72 static void ixgbe_init_params(ixgbe_t *);
  73 static int ixgbe_get_prop(ixgbe_t *, char *, int, int, int);
  74 static void ixgbe_driver_link_check(ixgbe_t *);
  75 static void ixgbe_sfp_check(void *);
  76 static void ixgbe_overtemp_check(void *);
  77 static void ixgbe_link_timer(void *);
  78 static void ixgbe_local_timer(void *);
  79 static void ixgbe_arm_watchdog_timer(ixgbe_t *);
  80 static void ixgbe_restart_watchdog_timer(ixgbe_t *);
  81 static void ixgbe_disable_adapter_interrupts(ixgbe_t *);
  82 static void ixgbe_enable_adapter_interrupts(ixgbe_t *);
  83 static boolean_t is_valid_mac_addr(uint8_t *);
  84 static boolean_t ixgbe_stall_check(ixgbe_t *);
  85 static boolean_t ixgbe_set_loopback_mode(ixgbe_t *, uint32_t);
  86 static void ixgbe_set_internal_mac_loopback(ixgbe_t *);
  87 static boolean_t ixgbe_find_mac_address(ixgbe_t *);
  88 static int ixgbe_alloc_intrs(ixgbe_t *);
  89 static int ixgbe_alloc_intr_handles(ixgbe_t *, int);
  90 static int ixgbe_add_intr_handlers(ixgbe_t *);
  91 static void ixgbe_map_rxring_to_vector(ixgbe_t *, int, int);
  92 static void ixgbe_map_txring_to_vector(ixgbe_t *, int, int);
  93 static void ixgbe_setup_ivar(ixgbe_t *, uint16_t, uint8_t, int8_t);
  94 static void ixgbe_enable_ivar(ixgbe_t *, uint16_t, int8_t);
  95 static void ixgbe_disable_ivar(ixgbe_t *, uint16_t, int8_t);
  96 static uint32_t ixgbe_get_hw_rx_index(ixgbe_t *ixgbe, uint32_t sw_rx_index);
  97 static int ixgbe_map_intrs_to_vectors(ixgbe_t *);
  98 static void ixgbe_setup_adapter_vector(ixgbe_t *);
  99 static void ixgbe_rem_intr_handlers(ixgbe_t *);
 100 static void ixgbe_rem_intrs(ixgbe_t *);
 101 static int ixgbe_enable_intrs(ixgbe_t *);
 102 static int ixgbe_disable_intrs(ixgbe_t *);
 103 static uint_t ixgbe_intr_legacy(void *, void *);
 104 static uint_t ixgbe_intr_msi(void *, void *);
 105 static uint_t ixgbe_intr_msix(void *, void *);
 106 static void ixgbe_intr_rx_work(ixgbe_rx_ring_t *);
 107 static void ixgbe_intr_tx_work(ixgbe_tx_ring_t *);
 108 static void ixgbe_intr_other_work(ixgbe_t *, uint32_t);
 109 static void ixgbe_get_driver_control(struct ixgbe_hw *);
 110 static int ixgbe_addmac(void *, const uint8_t *);
 111 static int ixgbe_remmac(void *, const uint8_t *);
 112 static void ixgbe_release_driver_control(struct ixgbe_hw *);

 114 static int ixgbe_attach(dev_info_t *, ddi_attach_cmd_t);
 115 static int ixgbe_detach(dev_info_t *, ddi_detach_cmd_t);
 116 static int ixgbe_resume(dev_info_t *);
 117 static int ixgbe_suspend(dev_info_t *);
 118 static void ixgbe_unconfigure(dev_info_t *, ixgbe_t *);
 119 static uint8_t *ixgbe_mc_table_itr(struct ixgbe_hw *, uint8_t **, uint32_t *);
 120 static int ixgbe_cbfunc(dev_info_t *, ddi_cb_action_t, void *, void *, void *);
 121 static int ixgbe_intr_cb_register(ixgbe_t *);
 122 static int ixgbe_intr_adjust(ixgbe_t *, ddi_cb_action_t, int);

 124 static int ixgbe_fm_error_cb(dev_info_t *dip, ddi_fm_error_t *err,
 125     const void *impl_data);
 126 static void ixgbe_fm_init(ixgbe_t *);
 127 static void ixgbe_fm_fini(ixgbe_t *);
```

```
129 char *ixgbe_priv_props[] = {
130         "_tx_copy_thresh",
131         "_tx_recycle_thresh",
132         "_tx_overload_thresh",
133         "_tx_resched_thresh",
134         "_rx_copy_thresh",
135         "_rx_limit_per_intr",
136         "_intr_throttling",
137         "_adv_pause_cap",
138         "_adv_asym_pause_cap",
139         NULL
140 };
```

*unchanged_portion_omitted_*

```
296 static adapter_info_t ixgbe_X540_cap = {
297         128,            /* maximum number of rx queues */
298         1,              /* minimum number of rx queues */
299         128,            /* default number of rx queues */
300         64,             /* maximum number of rx groups */
301         1,              /* minimum number of rx groups */
302         1,              /* default number of rx groups */
303         128,            /* maximum number of tx queues */
304         1,              /* minimum number of tx queues */
305         8,              /* default number of tx queues */
306         15500,          /* maximum MTU size */
307         0xFF8,          /* maximum interrupt throttle rate */
308         0,              /* minimum interrupt throttle rate */
309         200,            /* default interrupt throttle rate */
310         64,             /* maximum total msix vectors */
311         16,             /* maximum number of ring vectors */
312         2,              /* maximum number of other vectors */
313         /* XXX KEBE ASKS, Do we care about X540's SDP3? */
314         (IXGBE_EICR_LSC
315         | IXGBE_EICR_GPI_SDP0
316         | IXGBE_EICR_GPI_SDP1
317         | IXGBE_EICR_GPI_SDP2
318         /* | IXGBE_EICR_GPI_SDP3 */), /* "other" interrupt types handled */

320         (IXGBE_SDP0_GPIEN
321         | IXGBE_SDP1_GPIEN
322         /* | IXGBE_SDP2_GPIEN
323         | IXGBE_SDP3_GPIEN */), /* "other" interrupt types enable mask */

325         (IXGBE_FLAG_DCA_CAPABLE
326         | IXGBE_FLAG_RSS_CAPABLE
327         | IXGBE_FLAG_VMDQ_CAPABLE
328         | IXGBE_FLAG_RSC_CAPABLE
329         | IXGBE_FLAG_SFP_PLUG_CAPABLE) /* capability flags */
330         /* XXX KEBE ASKS, SFP_PLUG capable?!? */
331 };

333 /*
334  * Module Initialization Functions.
335  */

337 int
338 _init(void)
339 {
340         int status;

342         mac_init_ops(&ixgbe_dev_ops, MODULE_NAME);

344         status = mod_install(&ixgbe_modlinkage);

346         if (status != DDI_SUCCESS) {
```



```
347                 mac_fini_ops(&ixgbe_dev_ops);
348         }

350         return (status);
351 }
```

*unchanged_portion_omitted_*

```
853 /*
854  * ixgbe_identify_hardware - Identify the type of the chipset.
855  */
856 static int
857 ixgbe_identify_hardware(ixgbe_t *ixgbe)
858 {
859         struct ixgbe_hw *hw = &ixgbe->hw;
860         struct ixgbe_osdep *osdep = &ixgbe->osdep;

862         /*
863          * Get the device id
864          */
865         hw->vendor_id =
866             pci_config_get16(osdep->cfg_handle, PCI_CONF_VENID);
867         hw->device_id =
868             pci_config_get16(osdep->cfg_handle, PCI_CONF_DEVID);
869         hw->revision_id =
870             pci_config_get8(osdep->cfg_handle, PCI_CONF_REVID);
871         hw->subsystem_device_id =
872             pci_config_get16(osdep->cfg_handle, PCI_CONF_SUBSYSID);
873         hw->subsystem_vendor_id =
874             pci_config_get16(osdep->cfg_handle, PCI_CONF_SUBVENID);

876         /*
877          * Set the mac type of the adapter based on the device id
878          */
879         if (ixgbe_set_mac_type(hw) != IXGBE_SUCCESS) {
880                 return (IXGBE_FAILURE);
881         }

883         /*
884          * Install adapter capabilities
885          */
886         switch (hw->mac.type) {
887         case ixgbe_mac_82598EB:
888                 IXGBE_DEBUGLOG_0(ixgbe, "identify 82598 adapter\n");
889                 ixgbe->capab = &ixgbe_82598eb_cap;

891                 if (ixgbe_get_media_type(hw) == ixgbe_media_type_copper) {
892                         ixgbe->capab->flags |= IXGBE_FLAG_FAN_FAIL_CAPABLE;
893                         ixgbe->capab->other_intr |= IXGBE_EICR_GPI_SDP1;
894                         ixgbe->capab->other_gpie |= IXGBE_SDP1_GPIEN;
895                 }
896                 break;

898         case ixgbe_mac_82599EB:
899                 IXGBE_DEBUGLOG_0(ixgbe, "identify 82599 adapter\n");
900                 ixgbe->capab = &ixgbe_82599eb_cap;

902                 if (hw->device_id == IXGBE_DEV_ID_82599_T3_LOM) {
903                         ixgbe->capab->flags |= IXGBE_FLAG_TEMP_SENSOR_CAPABLE;
904                         ixgbe->capab->other_intr |= IXGBE_EICR_GPI_SDP0;
905                         ixgbe->capab->other_gpie |= IXGBE_SDP0_GPIEN;
906                 }
907                 break;

909         case ixgbe_mac_X540:
910                 IXGBE_DEBUGLOG_0(ixgbe, "identify X540 adapter\n");
911                 ixgbe->capab = &ixgbe_X540_cap;
```

```
                /*
                 * For now, X540 is all set in its capab structure.
                 * As other X540 variants show up, things can change here.
                 */
                break;

        default:
                IXGBE_DEBUGLOG_1(ixgbe,
                    "adapter not supported in ixgbe_identify_hardware(): %d\n",
                    hw->mac.type);
                return (IXGBE_FAILURE);
        }

        return (IXGBE_SUCCESS);
}
```

*unchanged_portion_omitted_*

```
/*
 * ixgbe_init - Initialize the device.
 */
static int
ixgbe_init(ixgbe_t *ixgbe)
{
        struct ixgbe_hw *hw = &ixgbe->hw;

        mutex_enter(&ixgbe->gen_lock);

        /*
         * Reset chipset to put the hardware in a known state
         * before we try to do anything with the eeprom.
         */
        if (ixgbe_reset_hw(hw) != IXGBE_SUCCESS) {
                ixgbe_fm_ereport(ixgbe, DDI_FM_DEVICE_INVAL_STATE);
                goto init_fail;
        }

        /*
         * Need to init eeprom before validating the checksum.
         */
        if (ixgbe_init_eeprom_params(hw) < 0) {
                ixgbe_error(ixgbe,
                    "Unable to intitialize the eeprom interface.");
                ixgbe_fm_ereport(ixgbe, DDI_FM_DEVICE_INVAL_STATE);
                goto init_fail;
        }

        /*
         * NVM validation
         */
        if (ixgbe_validate_eeprom_checksum(hw, NULL) < 0) {
                /*
                 * Some PCI-E parts fail the first check due to
                 * the link being in sleep state.  Call it again,
                 * if it fails a second time it's a real issue.
                 */
                if (ixgbe_validate_eeprom_checksum(hw, NULL) < 0) {
                        ixgbe_error(ixgbe,
                            "Invalid NVM checksum. Please contact "
                            "the vendor to update the NVM.");
                        ixgbe_fm_ereport(ixgbe, DDI_FM_DEVICE_INVAL_STATE);
                        goto init_fail;
                }
        }

        /*
         * Setup default flow control thresholds - enable/disable
```



```
         * & flow control type is controlled by ixgbe.conf
         */
        hw->fc.high_water[0] = DEFAULT_FCRTH;
        hw->fc.low_water[0] = DEFAULT_FCRTL;
        hw->fc.high_water = DEFAULT_FCRTH;
        hw->fc.low_water = DEFAULT_FCRTL;
        hw->fc.pause_time = DEFAULT_FCPAUSE;
        hw->fc.send_xon = B_TRUE;

        /*
         * Initialize link settings
         */
        (void) ixgbe_driver_setup_link(ixgbe, B_FALSE);

        /*
         * Initialize the chipset hardware
         */
        if (ixgbe_chip_start(ixgbe) != IXGBE_SUCCESS) {
                ixgbe_fm_ereport(ixgbe, DDI_FM_DEVICE_INVAL_STATE);
                goto init_fail;
        }

        if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) != DDI_FM_OK) {
                goto init_fail;
        }

        mutex_exit(&ixgbe->gen_lock);
        return (IXGBE_SUCCESS);

init_fail:
        /*
         * Reset PHY
         */
        (void) ixgbe_reset_phy(hw);

        mutex_exit(&ixgbe->gen_lock);
        ddi_fm_service_impact(ixgbe->dip, DDI_SERVICE_LOST);
        return (IXGBE_FAILURE);
}
```

*unchanged_portion_omitted_*

```
static void
ixgbe_setup_rx_ring(ixgbe_rx_ring_t *rx_ring)
{
        ixgbe_t *ixgbe = rx_ring->ixgbe;
        ixgbe_rx_data_t *rx_data = rx_ring->rx_data;
        struct ixgbe_hw *hw = &ixgbe->hw;
        rx_control_block_t *rcb;
        union ixgbe_adv_rx_desc *rbd;
        uint32_t size;
        uint32_t buf_low;
        uint32_t buf_high;
        uint32_t reg_val;
        int i;

        ASSERT(mutex_owned(&rx_ring->rx_lock));
        ASSERT(mutex_owned(&ixgbe->gen_lock));

        for (i = 0; i < ixgbe->rx_ring_size; i++) {
                rcb = rx_data->work_list[i];
                rbd = &rx_data->rbd_ring[i];

                rbd->read.pkt_addr = rcb->rx_buf.dma_address;
                rbd->read.hdr_addr = NULL;
        }
```

```
2108         /*
2109          * Initialize the length register
2110          */
2111         size = rx_data->ring_size * sizeof (union ixgbe_adv_rx_desc);
2112         IXGBE_WRITE_REG(hw, IXGBE_RDLEN(rx_ring->hw_index), size);

2114         /*
2115          * Initialize the base address registers
2116          */
2117         buf_low = (uint32_t)rx_data->rbd_area.dma_address;
2118         buf_high = (uint32_t)(rx_data->rbd_area.dma_address >> 32);
2119         IXGBE_WRITE_REG(hw, IXGBE_RDBAH(rx_ring->hw_index), buf_high);
2120         IXGBE_WRITE_REG(hw, IXGBE_RDBAL(rx_ring->hw_index), buf_low);

2122         /*
2123          * Setup head & tail pointers
2124          */
2125         IXGBE_WRITE_REG(hw, IXGBE_RDT(rx_ring->hw_index),
2126             rx_data->ring_size - 1);
2127         IXGBE_WRITE_REG(hw, IXGBE_RDH(rx_ring->hw_index), 0);

2129         rx_data->rbd_next = 0;
2130         rx_data->lro_first = 0;

2132         /*
2133          * Setup the Receive Descriptor Control Register (RXDCTL)
2134          * PTHRESH=32 descriptors (half the internal cache)
2135          * HTHRESH=0 descriptors (to minimize latency on fetch)
2136          * WTHRESH defaults to 1 (writeback each descriptor)
2137          */
2138         reg_val = IXGBE_READ_REG(hw, IXGBE_RXDCTL(rx_ring->hw_index));
2139         reg_val |= IXGBE_RXDCTL_ENABLE; /* enable queue */

2141         /* Not a valid value for 82599 */
2142         if (hw->mac.type < ixgbe_mac_82599EB) {
2143                 reg_val |= 0x0020;      /* pthresh */
2144         }
2145         IXGBE_WRITE_REG(hw, IXGBE_RXDCTL(rx_ring->hw_index), reg_val);

2147         if (hw->mac.type >= ixgbe_mac_82599EB) {
2099         if (hw->mac.type == ixgbe_mac_82599EB) {
2148                 reg_val = IXGBE_READ_REG(hw, IXGBE_RDRXCTL);
2149                 reg_val |= (IXGBE_RDRXCTL_CRCSTRIP | IXGBE_RDRXCTL_AGGDIS);
2150                 IXGBE_WRITE_REG(hw, IXGBE_RDRXCTL, reg_val);
2151         }

2153         /*
2154          * Setup the Split and Replication Receive Control Register.
2155          * Set the rx buffer size and the advanced descriptor type.
2156          */
2157         reg_val = (ixgbe->rx_buf_size >> IXGBE_SRRCTL_BSIZEPKT_SHIFT) |
2158             IXGBE_SRRCTL_DESCTYPE_ADV_ONEBUF;
2159         reg_val |= IXGBE_SRRCTL_DROP_EN;
2160         IXGBE_WRITE_REG(hw, IXGBE_SRRCTL(rx_ring->hw_index), reg_val);
2161 }
```

*unchanged_portion_omitted_*

```
2324 static void
2325 ixgbe_setup_tx_ring(ixgbe_tx_ring_t *tx_ring)
2326 {
2327         ixgbe_t *ixgbe = tx_ring->ixgbe;
2328         struct ixgbe_hw *hw = &ixgbe->hw;
2329         uint32_t size;
2330         uint32_t buf_low;
2331         uint32_t buf_high;
2332         uint32_t reg_val;
```



```
2334         ASSERT(mutex_owned(&tx_ring->tx_lock));
2335         ASSERT(mutex_owned(&ixgbe->gen_lock));

2337         /*
2338          * Initialize the length register
2339          */
2340         size = tx_ring->ring_size * sizeof (union ixgbe_adv_tx_desc);
2341         IXGBE_WRITE_REG(hw, IXGBE_TDLEN(tx_ring->index), size);

2343         /*
2344          * Initialize the base address registers
2345          */
2346         buf_low = (uint32_t)tx_ring->tbd_area.dma_address;
2347         buf_high = (uint32_t)(tx_ring->tbd_area.dma_address >> 32);
2348         IXGBE_WRITE_REG(hw, IXGBE_TDBAL(tx_ring->index), buf_low);
2349         IXGBE_WRITE_REG(hw, IXGBE_TDBAH(tx_ring->index), buf_high);

2351         /*
2352          * Setup head & tail pointers
2353          */
2354         IXGBE_WRITE_REG(hw, IXGBE_TDH(tx_ring->index), 0);
2355         IXGBE_WRITE_REG(hw, IXGBE_TDT(tx_ring->index), 0);

2357         /*
2358          * Setup head write-back
2359          */
2360         if (ixgbe->tx_head_wb_enable) {
2361                 /*
2362                  * The memory of the head write-back is allocated using
2363                  * the extra tbd beyond the tail of the tbd ring.
2364                  */
2365                 tx_ring->tbd_head_wb = (uint32_t *)
2366                     ((uintptr_t)tx_ring->tbd_area.address + size);
2367                 *tx_ring->tbd_head_wb = 0;

2369                 buf_low = (uint32_t)
2370                     (tx_ring->tbd_area.dma_address + size);
2371                 buf_high = (uint32_t)
2372                     ((tx_ring->tbd_area.dma_address + size) >> 32);

2374                 /* Set the head write-back enable bit */
2375                 buf_low |= IXGBE_TDWBAL_HEAD_WB_ENABLE;

2377                 IXGBE_WRITE_REG(hw, IXGBE_TDWBAL(tx_ring->index), buf_low);
2378                 IXGBE_WRITE_REG(hw, IXGBE_TDWBAH(tx_ring->index), buf_high);

2380                 /*
2381                  * Turn off relaxed ordering for head write back or it will
2382                  * cause problems with the tx recycling
2383                  */
2384 #if 0
2385                 /* XXX KEBE ASKS --> Should we do what FreeBSD does? */
2386                 reg_val = (hw->mac.type == ixgbe_mac_82598EB) ?
2387                     IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL(tx_ring->index)) :
2388                     IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL_82599(tx_ring->index));
2389                 reg_val &= ~IXGBE_DCA_TXCTRL_DESC_WRO_EN;
2390                 if (hw->mac.type == ixgbe_mac_82598EB) {
2391                         IXGBE_WRITE_REG(hw,
2392                             IXGBE_DCA_TXCTRL(tx_ring->index), reg_val);
2393                 } else {
2394                         IXGBE_WRITE_REG(hw,
2395                             IXGBE_DCA_TXCTRL_82599(tx_ring->index), reg_val);
2396                 }
2397 #else
2398                 /* XXX KEBE ASKS --> Or should we do what we've always done? */
```

```
2399                 reg_val = IXGBE_READ_REG(hw,
2400                     IXGBE_DCA_TXCTRL(tx_ring->index));
2401                 reg_val &= ~IXGBE_DCA_TXCTRL_DESC_WRO_EN;
2338                 reg_val &= ~IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
2402                 IXGBE_WRITE_REG(hw,
2403                     IXGBE_DCA_TXCTRL(tx_ring->index), reg_val);
2404 #endif
2405         } else {
2406                 tx_ring->tbd_head_wb = NULL;
2407 #if 0
2408                 /*
2409                  * XXX KEBE ASKS --> Should we do what FreeBSD does and
2410                  * twiddle TXCTRL_DESC_WR0_EN off anyway?
2411                  */
2412                 reg_val = (hw->mac.type == ixgbe_mac_82598EB) ?
2413                     IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL(tx_ring->index)) :
2414                     IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL_82599(tx_ring->index));
2415                 reg_val &= ~IXGBE_DCA_TXCTRL_DESC_WRO_EN;
2416                 if (hw->mac.type == ixgbe_mac_82598EB) {
2417                         IXGBE_WRITE_REG(hw,
2418                             IXGBE_DCA_TXCTRL(tx_ring->index), reg_val);
2419                 } else {
2420                         IXGBE_WRITE_REG(hw,
2421                             IXGBE_DCA_TXCTRL_82599(tx_ring->index), reg_val);
2422                 }
2423 #endif
2424         }

2426         tx_ring->tbd_head = 0;
2427         tx_ring->tbd_tail = 0;
2428         tx_ring->tbd_free = tx_ring->ring_size;

2430         if (ixgbe->tx_ring_init == B_TRUE) {
2431                 tx_ring->tcb_head = 0;
2432                 tx_ring->tcb_tail = 0;
2433                 tx_ring->tcb_free = tx_ring->free_list_size;
2434         }

2436         /*
2437          * Initialize the s/w context structure
2438          */
2439         bzero(&tx_ring->tx_context, sizeof (ixgbe_tx_context_t));
2440 }

2442 static void
2443 ixgbe_setup_tx(ixgbe_t *ixgbe)
2444 {
2445         struct ixgbe_hw *hw = &ixgbe->hw;
2446         ixgbe_tx_ring_t *tx_ring;
2447         uint32_t reg_val;
2448         uint32_t ring_mapping;
2449         int i;

2451         for (i = 0; i < ixgbe->num_tx_rings; i++) {
2452                 tx_ring = &ixgbe->tx_rings[i];
2453                 ixgbe_setup_tx_ring(tx_ring);
2454         }

2456         /*
2457          * Setup the per-ring statistics mapping.
2458          */
2459         ring_mapping = 0;
2460         for (i = 0; i < ixgbe->num_tx_rings; i++) {
2461                 ring_mapping |= (i & 0xF) << (8 * (i & 0x3));
2462                 if ((i & 0x3) == 0x3) {
2463                         switch (hw->mac.type) {
```



```
2464                         case ixgbe_mac_82598EB:
2465                                 IXGBE_WRITE_REG(hw, IXGBE_TQSMR(i >> 2),
2466                                     ring_mapping);
2467                                 break;

2469                         case ixgbe_mac_82599EB:
2470                         case ixgbe_mac_X540:
2471                                 IXGBE_WRITE_REG(hw, IXGBE_TQSM(i >> 2),
2472                                     ring_mapping);
2473                                 break;

2475                         default:
2476                                 break;
2477                         }

2479                         ring_mapping = 0;
2480                 }
2481         }
2482         if (i & 0x3) {
2483                 switch (hw->mac.type) {
2484                 case ixgbe_mac_82598EB:
2485                         IXGBE_WRITE_REG(hw, IXGBE_TQSMR(i >> 2), ring_mapping);
2486                         break;

2488                 case ixgbe_mac_82599EB:
2489                 case ixgbe_mac_X540:
2490                         IXGBE_WRITE_REG(hw, IXGBE_TQSM(i >> 2), ring_mapping);
2491                         break;

2493                 default:
2494                         break;
2495                 }
2496         }

2498         /*
2499          * Enable CRC appending and TX padding (for short tx frames)
2500          */
2501         reg_val = IXGBE_READ_REG(hw, IXGBE_HLREG0);
2502         reg_val |= IXGBE_HLREG0_TXCRCEN | IXGBE_HLREG0_TXPADEN;
2503         IXGBE_WRITE_REG(hw, IXGBE_HLREG0, reg_val);

2505         /*
2506          * enable DMA for 82599 and X540 parts
2423          * enable DMA for 82599 parts
2507          */
2508         if (hw->mac.type >= ixgbe_mac_82599EB) {
2425         if (hw->mac.type == ixgbe_mac_82599EB) {
2509         /* DMATXCTL.TE must be set after all Tx config is complete */
2510                 reg_val = IXGBE_READ_REG(hw, IXGBE_DMATXCTL);
2511                 reg_val |= IXGBE_DMATXCTL_TE;
2512                 IXGBE_WRITE_REG(hw, IXGBE_DMATXCTL, reg_val);
2513                 /* XXX KEBE SAYS - FreeBSD sets up MTQC.  Should we? */
2514         }

2516         /*
2517          * Enabling tx queues ..
2518          * For 82599 must be done after DMATXCTL.TE is set
2519          */
2520         for (i = 0; i < ixgbe->num_tx_rings; i++) {
2521                 tx_ring = &ixgbe->tx_rings[i];
2522                 reg_val = IXGBE_READ_REG(hw, IXGBE_TXDCTL(tx_ring->index));
2523                 reg_val |= IXGBE_TXDCTL_ENABLE;
2524                 IXGBE_WRITE_REG(hw, IXGBE_TXDCTL(tx_ring->index), reg_val);
2525         }
2526 }
```

_____***unchanged_portion_omitted_***

```
2588 /*
2589  * ixgbe_setup_vmdq - Setup MAC classification feature
2590  */
2591 static void
2592 ixgbe_setup_vmdq(ixgbe_t *ixgbe)
2593 {
2594         struct ixgbe_hw *hw = &ixgbe->hw;
2595         uint32_t vmdctl, i, vtctl;

2597         /*
2598          * Setup the VMDq Control register, enable VMDq based on
2599          * packet destination MAC address:
2600          */
2601         switch (hw->mac.type) {
2602         case ixgbe_mac_82598EB:
2603                 /*
2604                  * VMDq Enable = 1;
2605                  * VMDq Filter = 0; MAC filtering
2606                  * Default VMDq output index = 0;
2607                  */
2608                 vmdctl = IXGBE_VMD_CTL_VMDQ_EN;
2609                 IXGBE_WRITE_REG(hw, IXGBE_VMD_CTL, vmdctl);
2610                 break;

2612         case ixgbe_mac_82599EB:
2613         case ixgbe_mac_X540:
2614                 /*
2615                  * Enable VMDq-only.
2616                  */
2617                 vmdctl = IXGBE_MRQC_VMDQEN;
2618                 IXGBE_WRITE_REG(hw, IXGBE_MRQC, vmdctl);

2620                 for (i = 0; i < hw->mac.num_rar_entries; i++) {
2621                         IXGBE_WRITE_REG(hw, IXGBE_MPSAR_LO(i), 0);
2622                         IXGBE_WRITE_REG(hw, IXGBE_MPSAR_HI(i), 0);
2623                 }

2625                 /*
2626                  * Enable Virtualization and Replication.
2627                  */
2628                 vtctl = IXGBE_VT_CTL_VT_ENABLE | IXGBE_VT_CTL_REPLEN;
2629                 IXGBE_WRITE_REG(hw, IXGBE_VT_CTL, vtctl);

2631                 /*
2632                  * Enable receiving packets to all VFs
2633                  */
2634                 IXGBE_WRITE_REG(hw, IXGBE_VFRE(0), IXGBE_VFRE_ENABLE_ALL);
2635                 IXGBE_WRITE_REG(hw, IXGBE_VFRE(1), IXGBE_VFRE_ENABLE_ALL);
2636                 break;

2638         default:
2639                 break;
2640         }
2641 }

2643 /*
2644  * ixgbe_setup_vmdq_rss - Setup both vmdq feature and rss feature.
2645  */
2646 static void
2647 ixgbe_setup_vmdq_rss(ixgbe_t *ixgbe)
2648 {
2649         struct ixgbe_hw *hw = &ixgbe->hw;
2650         uint32_t i, mrqc, rxcsum;
2651         uint32_t random;
2652         uint32_t reta;
```



```
2653         uint32_t ring_per_group;
2654         uint32_t vmdctl, vtctl;

2656         /*
2657          * Fill out redirection table
2658          */
2659         reta = 0;
2660         ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;
2661         for (i = 0; i < 128; i++) {
2662                 reta = (reta << 8) | (i % ring_per_group) |
2663                     ((i % ring_per_group) << 4);
2664                 if ((i & 3) == 3)
2665                         IXGBE_WRITE_REG(hw, IXGBE_RETA(i >> 2), reta);
2666         }

2668         /*
2669          * Fill out hash function seeds with a random constant
2670          */
2671         for (i = 0; i < 10; i++) {
2672                 (void) random_get_pseudo_bytes((uint8_t *)&random,
2673                     sizeof (uint32_t));
2674                 IXGBE_WRITE_REG(hw, IXGBE_RSSRK(i), random);
2675         }

2677         /*
2678          * Enable and setup RSS and VMDq
2679          */
2680         switch (hw->mac.type) {
2681         case ixgbe_mac_82598EB:
2682                 /*
2683                  * Enable RSS & Setup RSS Hash functions
2684                  */
2685                 mrqc = IXGBE_MRQC_RSSEN |
2686                     IXGBE_MRQC_RSS_FIELD_IPV4 |
2687                     IXGBE_MRQC_RSS_FIELD_IPV4_TCP |
2688                     IXGBE_MRQC_RSS_FIELD_IPV4_UDP |
2689                     IXGBE_MRQC_RSS_FIELD_IPV6_EX_TCP |
2690                     IXGBE_MRQC_RSS_FIELD_IPV6_EX |
2691                     IXGBE_MRQC_RSS_FIELD_IPV6 |
2692                     IXGBE_MRQC_RSS_FIELD_IPV6_TCP |
2693                     IXGBE_MRQC_RSS_FIELD_IPV6_UDP |
2694                     IXGBE_MRQC_RSS_FIELD_IPV6_EX_UDP;
2695                 IXGBE_WRITE_REG(hw, IXGBE_MRQC, mrqc);

2697                 /*
2698                  * Enable and Setup VMDq
2699                  * VMDq Filter = 0; MAC filtering
2700                  * Default VMDq output index = 0;
2701                  */
2702                 vmdctl = IXGBE_VMD_CTL_VMDQ_EN;
2703                 IXGBE_WRITE_REG(hw, IXGBE_VMD_CTL, vmdctl);
2704                 break;

2706         case ixgbe_mac_82599EB:
2707         case ixgbe_mac_X540:
2708                 /*
2709                  * Enable RSS & Setup RSS Hash functions
2710                  */
2711                 mrqc = IXGBE_MRQC_RSS_FIELD_IPV4 |
2712                     IXGBE_MRQC_RSS_FIELD_IPV4_TCP |
2713                     IXGBE_MRQC_RSS_FIELD_IPV4_UDP |
2714                     IXGBE_MRQC_RSS_FIELD_IPV6_EX_TCP |
2715                     IXGBE_MRQC_RSS_FIELD_IPV6_EX |
2716                     IXGBE_MRQC_RSS_FIELD_IPV6 |
2717                     IXGBE_MRQC_RSS_FIELD_IPV6_TCP |
2718                     IXGBE_MRQC_RSS_FIELD_IPV6_UDP |
```

```
2719                        IXGBE_MRQC_RSS_FIELD_IPV6_EX_UDP;

2721                /*
2722                 * Enable VMDq+RSS.
2723                 */
2724                if (ixgbe->num_rx_groups > 32)  {
2725                        mrqc = mrqc | IXGBE_MRQC_VMDQRSS64EN;
2726                } else {
2727                        mrqc = mrqc | IXGBE_MRQC_VMDQRSS32EN;
2728                }

2730                IXGBE_WRITE_REG(hw, IXGBE_MRQC, mrqc);

2732                for (i = 0; i < hw->mac.num_rar_entries; i++) {
2733                        IXGBE_WRITE_REG(hw, IXGBE_MPSAR_LO(i), 0);
2734                        IXGBE_WRITE_REG(hw, IXGBE_MPSAR_HI(i), 0);
2735                }
2736                break;

2738        default:
2739                break;

2741        }

2743        /*
2744         * Disable Packet Checksum to enable RSS for multiple receive queues.
2745         * It is an adapter hardware limitation that Packet Checksum is
2746         * mutually exclusive with RSS.
2747         */
2748        rxcsum = IXGBE_READ_REG(hw, IXGBE_RXCSUM);
2749        rxcsum |= IXGBE_RXCSUM_PCSD;
2750        rxcsum &= ~IXGBE_RXCSUM_IPPCSE;
2751        IXGBE_WRITE_REG(hw, IXGBE_RXCSUM, rxcsum);

2753        if (hw->mac.type >= ixgbe_mac_82599EB) {
2667        if (hw->mac.type == ixgbe_mac_82599EB) {
2754                /*
2755                 * Enable Virtualization and Replication.
2756                 */
2757                vtctl = IXGBE_VT_CTL_VT_ENABLE | IXGBE_VT_CTL_REPLEN;
2758                IXGBE_WRITE_REG(hw, IXGBE_VT_CTL, vtctl);

2760                /*
2761                 * Enable receiving packets to all VFs
2762                 */
2763                IXGBE_WRITE_REG(hw, IXGBE_VFRE(0), IXGBE_VFRE_ENABLE_ALL);
2764                IXGBE_WRITE_REG(hw, IXGBE_VFRE(1), IXGBE_VFRE_ENABLE_ALL);
2765        }
2766 }
```

*unchanged_portion_omitted_*

```
2904 /*
2905  * ixgbe_setup_multicast - Setup multicast data structures.
2906  *
2907  * This routine initializes all of the multicast related structures
2908  * and save them in the hardware registers.
2909  */
2910 static void
2911 ixgbe_setup_multicst(ixgbe_t *ixgbe)
2912 {
2913         uint8_t *mc_addr_list;
2914         uint32_t mc_addr_count;
2915         struct ixgbe_hw *hw = &ixgbe->hw;

2917         ASSERT(mutex_owned(&ixgbe->gen_lock));
```



```
2919         ASSERT(ixgbe->mcast_count <= MAX_NUM_MULTICAST_ADDRESSES);

2921         mc_addr_list = (uint8_t *)ixgbe->mcast_table;
2922         mc_addr_count = ixgbe->mcast_count;

2924         /*
2925          * Update the multicast addresses to the MTA registers
2926          */
2927         (void) ixgbe_update_mc_addr_list(hw, mc_addr_list, mc_addr_count,
2928             ixgbe_mc_table_itr, TRUE);
2842             ixgbe_mc_table_itr);
2929 }

2931 /*
2932  * ixgbe_setup_vmdq_rss_conf - Configure vmdq and rss (number and mode).
2933  *
2934  * Configure the rx classification mode (vmdq & rss) and vmdq & rss numbers.
2935  * Different chipsets may have different allowed configuration of vmdq and rss.
2936  */
2937 static void
2938 ixgbe_setup_vmdq_rss_conf(ixgbe_t *ixgbe)
2939 {
2940         struct ixgbe_hw *hw = &ixgbe->hw;
2941         uint32_t ring_per_group;

2943         switch (hw->mac.type) {
2944         case ixgbe_mac_82598EB:
2945                 /*
2946                  * 82598 supports the following combination:
2947                  * vmdq no. x rss no.
2948                  * [5..16]  x 1
2949                  * [1..4]   x [1..16]
2950                  * However 8 rss queue per pool (vmdq) is sufficient for
2951                  * most cases.
2952                  */
2953                 ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;
2954                 if (ixgbe->num_rx_groups > 4) {
2955                         ixgbe->num_rx_rings = ixgbe->num_rx_groups;
2956                 } else {
2957                         ixgbe->num_rx_rings = ixgbe->num_rx_groups *
2958                             min(8, ring_per_group);
2959                 }

2961                 break;

2963         case ixgbe_mac_82599EB:
2964         case ixgbe_mac_X540:
2965                 /*
2966                  * 82599 supports the following combination:
2967                  * vmdq no. x rss no.
2968                  * [33..64] x [1..2]
2969                  * [2..32]  x [1..4]
2970                  * 1 x [1..16]
2971                  * However 8 rss queue per pool (vmdq) is sufficient for
2972                  * most cases.
2973                  *
2974                  * For now, treat X540 like the 82599.
2975                  */
2976                 ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;
2977                 if (ixgbe->num_rx_groups == 1) {
2978                         ixgbe->num_rx_rings = min(8, ring_per_group);
2979                 } else if (ixgbe->num_rx_groups <= 32) {
2980                         ixgbe->num_rx_rings = ixgbe->num_rx_groups *
2981                             min(4, ring_per_group);
2982                 } else if (ixgbe->num_rx_groups <= 64) {
2983                         ixgbe->num_rx_rings = ixgbe->num_rx_groups *
```

```
2984                            min(2, ring_per_group);
2985                }
2986                break;

2988        default:
2989                break;
2990        }

2992        ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;

2994        if (ixgbe->num_rx_groups == 1 && ring_per_group == 1) {
2995                ixgbe->classify_mode = IXGBE_CLASSIFY_NONE;
2996        } else if (ixgbe->num_rx_groups != 1 && ring_per_group == 1) {
2997                ixgbe->classify_mode = IXGBE_CLASSIFY_VMDQ;
2998        } else if (ixgbe->num_rx_groups != 1 && ring_per_group != 1) {
2999                ixgbe->classify_mode = IXGBE_CLASSIFY_VMDQ_RSS;
3000        } else {
3001                ixgbe->classify_mode = IXGBE_CLASSIFY_RSS;
3002        }

3004        IXGBE_DEBUGLOG_2(ixgbe, "rx group number:%d, rx ring number:%d",
3005            ixgbe->num_rx_groups, ixgbe->num_rx_rings);
3006 }

3008 /*
3009  * ixgbe_get_conf - Get driver configurations set in driver.conf.
3010  *
3011  * This routine gets user-configured values out of the configuration
3012  * file ixgbe.conf.
3013  *
3014  * For each configurable value, there is a minimum, a maximum, and a
3015  * default.
3016  * If user does not configure a value, use the default.
3017  * If user configures below the minimum, use the minumum.
3018  * If user configures above the maximum, use the maxumum.
3019  */
3020 static void
3021 ixgbe_get_conf(ixgbe_t *ixgbe)
3022 {
3023        struct ixgbe_hw *hw = &ixgbe->hw;
3024        uint32_t flow_control;

3026        /*
3027         * ixgbe driver supports the following user configurations:
3028         *
3029         * Jumbo frame configuration:
3030         *    default_mtu
3031         *
3032         * Ethernet flow control configuration:
3033         *    flow_control
3034         *
3035         * Multiple rings configurations:
3036         *    tx_queue_number
3037         *    tx_ring_size
3038         *    rx_queue_number
3039         *    rx_ring_size
3040         *
3041         * Call ixgbe_get_prop() to get the value for a specific
3042         * configuration parameter.
3043         */

3045        /*
3046         * Jumbo frame configuration - max_frame_size controls host buffer
3047         * allocation, so includes MTU, ethernet header, vlan tag and
3048         * frame check sequence.
3049         */
```



```
3050        ixgbe->default_mtu = ixgbe_get_prop(ixgbe, PROP_DEFAULT_MTU,
3051            MIN_MTU, ixgbe->capab->max_mtu, DEFAULT_MTU);

3053        ixgbe->max_frame_size = ixgbe->default_mtu +
3054            sizeof (struct ether_vlan_header) + ETHERFCSL;

3056        /*
3057         * Ethernet flow control configuration
3058         */
3059        flow_control = ixgbe_get_prop(ixgbe, PROP_FLOW_CONTROL,
3060            ixgbe_fc_none, 3, ixgbe_fc_none);
3061        if (flow_control == 3)
3062                flow_control = ixgbe_fc_default;

3064        /*
3065         * fc.requested mode is what the user requests.  After autoneg,
3066         * fc.current_mode will be the flow_control mode that was negotiated.
3067         */
3068        hw->fc.requested_mode = flow_control;

3070        /*
3071         * Multiple rings configurations
3072         */
3073        ixgbe->num_tx_rings = ixgbe_get_prop(ixgbe, PROP_TX_QUEUE_NUM,
3074            ixgbe->capab->min_tx_que_num,
3075            ixgbe->capab->max_tx_que_num,
3076            ixgbe->capab->def_tx_que_num);
3077        ixgbe->tx_ring_size = ixgbe_get_prop(ixgbe, PROP_TX_RING_SIZE,
3078            MIN_TX_RING_SIZE, MAX_TX_RING_SIZE, DEFAULT_TX_RING_SIZE);

3080        ixgbe->num_rx_rings = ixgbe_get_prop(ixgbe, PROP_RX_QUEUE_NUM,
3081            ixgbe->capab->min_rx_que_num,
3082            ixgbe->capab->max_rx_que_num,
3083            ixgbe->capab->def_rx_que_num);
3084        ixgbe->rx_ring_size = ixgbe_get_prop(ixgbe, PROP_RX_RING_SIZE,
3085            MIN_RX_RING_SIZE, MAX_RX_RING_SIZE, DEFAULT_RX_RING_SIZE);

3087        /*
3088         * Multiple groups configuration
3089         */
3090        ixgbe->num_rx_groups = ixgbe_get_prop(ixgbe, PROP_RX_GROUP_NUM,
3091            ixgbe->capab->min_rx_grp_num, ixgbe->capab->max_rx_grp_num,
3092            ixgbe->capab->def_rx_grp_num);

3094        ixgbe->mr_enable = ixgbe_get_prop(ixgbe, PROP_MR_ENABLE,
3095            0, 1, DEFAULT_MR_ENABLE);

3097        if (ixgbe->mr_enable == B_FALSE) {
3098                ixgbe->num_tx_rings = 1;
3099                ixgbe->num_rx_rings = 1;
3100                ixgbe->num_rx_groups = 1;
3101                ixgbe->classify_mode = IXGBE_CLASSIFY_NONE;
3102        } else {
3103                ixgbe->num_rx_rings = ixgbe->num_rx_groups *
3104                    max(ixgbe->num_rx_rings / ixgbe->num_rx_groups, 1);
3105                /*
3106                 * The combination of num_rx_rings and num_rx_groups
3107                 * may be not supported by h/w. We need to adjust
3108                 * them to appropriate values.
3109                 */
3110                ixgbe_setup_vmdq_rss_conf(ixgbe);
3111        }

3113        /*
3114         * Tunable used to force an interrupt type. The only use is
3115         * for testing of the lesser interrupt types.
```

```
3116          * 0 = don't force interrupt type
3117          * 1 = force interrupt type MSI-X
3118          * 2 = force interrupt type MSI
3119          * 3 = force interrupt type Legacy
3120          */
3121         ixgbe->intr_force = ixgbe_get_prop(ixgbe, PROP_INTR_FORCE,
3122             IXGBE_INTR_NONE, IXGBE_INTR_LEGACY, IXGBE_INTR_NONE);

3124         ixgbe->tx_hcksum_enable = ixgbe_get_prop(ixgbe, PROP_TX_HCKSUM_ENABLE,
3125             0, 1, DEFAULT_TX_HCKSUM_ENABLE);
3126         ixgbe->rx_hcksum_enable = ixgbe_get_prop(ixgbe, PROP_RX_HCKSUM_ENABLE,
3127             0, 1, DEFAULT_RX_HCKSUM_ENABLE);
3128         ixgbe->lso_enable = ixgbe_get_prop(ixgbe, PROP_LSO_ENABLE,
3129             0, 1, DEFAULT_LSO_ENABLE);
3130         ixgbe->lro_enable = ixgbe_get_prop(ixgbe, PROP_LRO_ENABLE,
3131             0, 1, DEFAULT_LRO_ENABLE);
3132         ixgbe->tx_head_wb_enable = ixgbe_get_prop(ixgbe, PROP_TX_HEAD_WB_ENABLE,
3133             0, 1, DEFAULT_TX_HEAD_WB_ENABLE);
3134         ixgbe->relax_order_enable = ixgbe_get_prop(ixgbe,
3135             PROP_RELAX_ORDER_ENABLE, 0, 1, DEFAULT_RELAX_ORDER_ENABLE);

3137         /* Head Write Back not recommended for 82599 and X540 */
3048         /* Head Write Back not recommended for 82599 */
3138         if (hw->mac.type >= ixgbe_mac_82599EB) {
3139                 ixgbe->tx_head_wb_enable = B_FALSE;
3140         }

3142         /*
3143          * ixgbe LSO needs the tx h/w checksum support.
3144          * LSO will be disabled if tx h/w checksum is not
3145          * enabled.
3146          */
3147         if (ixgbe->tx_hcksum_enable == B_FALSE) {
3148                 ixgbe->lso_enable = B_FALSE;
3149         }

3151         /*
3152          * ixgbe LRO needs the rx h/w checksum support.
3153          * LRO will be disabled if rx h/w checksum is not
3154          * enabled.
3155          */
3156         if (ixgbe->rx_hcksum_enable == B_FALSE) {
3157                 ixgbe->lro_enable = B_FALSE;
3158         }

3160         /*
3161          * ixgbe LRO only been supported by 82599 and X540 now
3072          * ixgbe LRO only been supported by 82599 now
3162          */
3163         if (hw->mac.type < ixgbe_mac_82599EB) {
3074         if (hw->mac.type != ixgbe_mac_82599EB) {
3164                 ixgbe->lro_enable = B_FALSE;
3165         }
3166         ixgbe->tx_copy_thresh = ixgbe_get_prop(ixgbe, PROP_TX_COPY_THRESHOLD,
3167             MIN_TX_COPY_THRESHOLD, MAX_TX_COPY_THRESHOLD,
3168             DEFAULT_TX_COPY_THRESHOLD);
3169         ixgbe->tx_recycle_thresh = ixgbe_get_prop(ixgbe,
3170             PROP_TX_RECYCLE_THRESHOLD, MIN_TX_RECYCLE_THRESHOLD,
3171             MAX_TX_RECYCLE_THRESHOLD, DEFAULT_TX_RECYCLE_THRESHOLD);
3172         ixgbe->tx_overload_thresh = ixgbe_get_prop(ixgbe,
3173             PROP_TX_OVERLOAD_THRESHOLD, MIN_TX_OVERLOAD_THRESHOLD,
3174             MAX_TX_OVERLOAD_THRESHOLD, DEFAULT_TX_OVERLOAD_THRESHOLD);
3175         ixgbe->tx_resched_thresh = ixgbe_get_prop(ixgbe,
3176             PROP_TX_RESCHED_THRESHOLD, MIN_TX_RESCHED_THRESHOLD,
3177             MAX_TX_RESCHED_THRESHOLD, DEFAULT_TX_RESCHED_THRESHOLD);
```



```
3179         ixgbe->rx_copy_thresh = ixgbe_get_prop(ixgbe, PROP_RX_COPY_THRESHOLD,
3180             MIN_RX_COPY_THRESHOLD, MAX_RX_COPY_THRESHOLD,
3181             DEFAULT_RX_COPY_THRESHOLD);
3182         ixgbe->rx_limit_per_intr = ixgbe_get_prop(ixgbe, PROP_RX_LIMIT_PER_INTR,
3183             MIN_RX_LIMIT_PER_INTR, MAX_RX_LIMIT_PER_INTR,
3184             DEFAULT_RX_LIMIT_PER_INTR);

3186         ixgbe->intr_throttling[0] = ixgbe_get_prop(ixgbe, PROP_INTR_THROTTLING,
3187             ixgbe->capab->min_intr_throttle,
3188             ixgbe->capab->max_intr_throttle,
3189             ixgbe->capab->def_intr_throttle);
3190         /*
3191          * 82599 and X540 require the interupt throttling rate is
3102          * 82599 requires the interupt throttling rate is
3192          * a multiple of 8. This is enforced by the register
3193          * definiton.
3194          */
3195         if (hw->mac.type >= ixgbe_mac_82599EB)
3106         if (hw->mac.type == ixgbe_mac_82599EB)
3196                 ixgbe->intr_throttling[0] = ixgbe->intr_throttling[0] & 0xFF8;
3197 }
______unchanged_portion_omitted_

3301 /*
3302  * ixgbe_driver_link_check - Link status processing.
3303  *
3304  * This function can be called in both kernel context and interrupt context
3305  */
3306 static void
3307 ixgbe_driver_link_check(ixgbe_t *ixgbe)
3308 {
3309         struct ixgbe_hw *hw = &ixgbe->hw;
3310         ixgbe_link_speed speed = IXGBE_LINK_SPEED_UNKNOWN;
3311         boolean_t link_up = B_FALSE;
3312         boolean_t link_changed = B_FALSE;

3314         ASSERT(mutex_owned(&ixgbe->gen_lock));

3316         (void) ixgbe_check_link(hw, &speed, &link_up, false);
3317         if (link_up) {
3318                 ixgbe->link_check_complete = B_TRUE;

3320                 /* Link is up, enable flow control settings */
3321                 (void) ixgbe_fc_enable(hw);
3232                 (void) ixgbe_fc_enable(hw, 0);

3323                 /*
3324                  * The Link is up, check whether it was marked as down earlier
3325                  */
3326                 if (ixgbe->link_state != LINK_STATE_UP) {
3327                         switch (speed) {
3328                         case IXGBE_LINK_SPEED_10GB_FULL:
3329                                 ixgbe->link_speed = SPEED_10GB;
3330                                 break;
3331                         case IXGBE_LINK_SPEED_1GB_FULL:
3332                                 ixgbe->link_speed = SPEED_1GB;
3333                                 break;
3334                         case IXGBE_LINK_SPEED_100_FULL:
3335                                 ixgbe->link_speed = SPEED_100;
3336                         }
3337                         ixgbe->link_duplex = LINK_DUPLEX_FULL;
3338                         ixgbe->link_state = LINK_STATE_UP;
3339                         link_changed = B_TRUE;
3340                 }
3341         } else {
3342                 if (ixgbe->link_check_complete == B_TRUE ||
```

```
3343                    (ixgbe->link_check_complete == B_FALSE &&
3344                    gethrtime() >= ixgbe->link_check_hrtime)) {
3345                        /*
3346                         * The link is really down
3347                         */
3348                        ixgbe->link_check_complete = B_TRUE;

3350                        if (ixgbe->link_state != LINK_STATE_DOWN) {
3351                                ixgbe->link_speed = 0;
3352                                ixgbe->link_duplex = LINK_DUPLEX_UNKNOWN;
3353                                ixgbe->link_state = LINK_STATE_DOWN;
3354                                link_changed = B_TRUE;
3355                        }
3356                }
3357        }

3359        /*
3360         * If we are in an interrupt context, need to re-enable the
3361         * interrupt, which was automasked
3362         */
3363        if (servicing_interrupt() != 0) {
3364                ixgbe->eims |= IXGBE_EICR_LSC;
3365                IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);
3366        }

3368        if (link_changed) {
3369                mac_link_update(ixgbe->mac_hdl, ixgbe->link_state);
3370        }
3371 }
```

*unchanged_portion_omitted_*

```
3784 /*
3785  * ixgbe_enable_adapter_interrupts - Enable all hardware interrupts.
3786  */
3787 static void
3788 ixgbe_enable_adapter_interrupts(ixgbe_t *ixgbe)
3789 {
3790        struct ixgbe_hw *hw = &ixgbe->hw;
3791        uint32_t eiac, eiam;
3792        uint32_t gpie = IXGBE_READ_REG(hw, IXGBE_GPIE);

3794        /* interrupt types to enable */
3795        ixgbe->eims = IXGBE_EIMS_ENABLE_MASK;   /* shared code default */
3796        ixgbe->eims &= ~IXGBE_EIMS_TCP_TIMER;   /* minus tcp timer */
3797        ixgbe->eims |= ixgbe->capab->other_intr; /* "other" interrupt types */

3799        /* enable automask on "other" causes that this adapter can generate */
3800        eiam = ixgbe->capab->other_intr;

3802        /*
3803         * msi-x mode
3804         */
3805        if (ixgbe->intr_type == DDI_INTR_TYPE_MSIX) {
3806                /* enable autoclear but not on bits 29:20 */
3807                eiac = (ixgbe->eims & ~IXGBE_OTHER_INTR);

3809                /* general purpose interrupt enable */
3810                gpie |= (IXGBE_GPIE_MSIX_MODE
3811                    | IXGBE_GPIE_PBA_SUPPORT
3812                    | IXGBE_GPIE_OCD
3813                    | IXGBE_GPIE_EIAME);
3814        /*
3815         * non-msi-x mode
3816         */
3817        } else {
```



```
3819                /* disable autoclear, leave gpie at default */
3820                eiac = 0;

3822                /*
3823                 * General purpose interrupt enable.
3824                 * For 82599, extended interrupt automask enable
3825                 * only in MSI or MSI-X mode
3826                 */
3827                if ((hw->mac.type < ixgbe_mac_82599EB) ||
3828                    (ixgbe->intr_type == DDI_INTR_TYPE_MSI)) {
3829                        gpie |= IXGBE_GPIE_EIAME;
3830                }
3831        }

3833        /* Enable specific "other" interrupt types */
3834        switch (hw->mac.type) {
3835        case ixgbe_mac_82598EB:
3836                gpie |= ixgbe->capab->other_gpie;
3837                break;

3839        case ixgbe_mac_82599EB:
3840        case ixgbe_mac_X540:
3841                gpie |= ixgbe->capab->other_gpie;

3843                /* Enable RSC Delay 8us when LRO enabled  */
3844                if (ixgbe->lro_enable) {
3845                        gpie |= (1 << IXGBE_GPIE_RSC_DELAY_SHIFT);
3846                }
3847                break;

3849        default:
3850                break;
3851        }

3853        /* write to interrupt control registers */
3854        IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);
3855        IXGBE_WRITE_REG(hw, IXGBE_EIAC, eiac);
3856        IXGBE_WRITE_REG(hw, IXGBE_EIAM, eiam);
3857        IXGBE_WRITE_REG(hw, IXGBE_GPIE, gpie);
3858        IXGBE_WRITE_FLUSH(hw);
3859 }
```

*unchanged_portion_omitted_*

```
3980 /*
3981  * ixgbe_set_internal_mac_loopback - Set the internal MAC loopback mode.
3982  */
3983 static void
3984 ixgbe_set_internal_mac_loopback(ixgbe_t *ixgbe)
3985 {
3986        struct ixgbe_hw *hw;
3987        uint32_t reg;
3988        uint8_t atlas;

3990        hw = &ixgbe->hw;

3992        /*
3993         * Setup MAC loopback
3994         */
3995        reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_HLREG0);
3996        reg |= IXGBE_HLREG0_LPBK;
3997        IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_HLREG0, reg);

3999        reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_AUTOC);
4000        reg &= ~IXGBE_AUTOC_LMS_MASK;
4001        IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_AUTOC, reg);
```

```
4003         /*
4004          * Disable Atlas Tx lanes to keep packets in loopback and not on wire
4005          */
4006         switch (hw->mac.type) {
4007         case ixgbe_mac_82598EB:
4008                 (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_LPBK,
4009                     &atlas);
4010                 atlas |= IXGBE_ATLAS_PDN_TX_REG_EN;
4011                 (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_LPBK,
4012                     atlas);

4014                 (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_10G,
4015                     &atlas);
4016                 atlas |= IXGBE_ATLAS_PDN_TX_10G_QL_ALL;
4017                 (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_10G,
4018                     atlas);

4020                 (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_1G,
4021                     &atlas);
4022                 atlas |= IXGBE_ATLAS_PDN_TX_1G_QL_ALL;
4023                 (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_1G,
4024                     atlas);

4026                 (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_AN,
4027                     &atlas);
4028                 atlas |= IXGBE_ATLAS_PDN_TX_AN_QL_ALL;
4029                 (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_AN,
4030                     atlas);
4031                 break;

4033         case ixgbe_mac_82599EB:
4034         case ixgbe_mac_X540:
4035                 reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_AUTOC);
4036                 reg |= (IXGBE_AUTOC_FLU |
4037                     IXGBE_AUTOC_10G_KX4);
4038                 IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_AUTOC, reg);

4040                 (void) ixgbe_setup_link(&ixgbe->hw, IXGBE_LINK_SPEED_10GB_FULL,
4041                     B_FALSE, B_TRUE);
4042                 break;

4044         default:
4045                 break;
4046         }
4047 }
```

*unchanged_portion_omitted_*

```
4168 /*
4169  * ixgbe_intr_legacy - Interrupt handler for legacy interrupts.
4170  */
4171 static uint_t
4172 ixgbe_intr_legacy(void *arg1, void *arg2)
4173 {
4174         ixgbe_t *ixgbe = (ixgbe_t *)arg1;
4175         struct ixgbe_hw *hw = &ixgbe->hw;
4176         ixgbe_tx_ring_t *tx_ring;
4177         ixgbe_rx_ring_t *rx_ring;
4178         uint32_t eicr;
4179         mblk_t *mp;
4180         boolean_t tx_reschedule;
4181         uint_t result;

4183         _NOTE(ARGUNUSED(arg2));

4185         mutex_enter(&ixgbe->gen_lock);
4186         if (ixgbe->ixgbe_state & IXGBE_SUSPENDED) {
```



```
4187                 mutex_exit(&ixgbe->gen_lock);
4188                 return (DDI_INTR_UNCLAIMED);
4189         }

4191         mp = NULL;
4192         tx_reschedule = B_FALSE;

4194         /*
4195          * Any bit set in eicr: claim this interrupt
4196          */
4197         eicr = IXGBE_READ_REG(hw, IXGBE_EICR);

4199         if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) != DDI_FM_OK) {
4200                 mutex_exit(&ixgbe->gen_lock);
4201                 ddi_fm_service_impact(ixgbe->dip, DDI_SERVICE_DEGRADED);
4202                 atomic_or_32(&ixgbe->ixgbe_state, IXGBE_ERROR);
4203                 return (DDI_INTR_CLAIMED);
4204         }

4206         if (eicr) {
4207                 /*
4208                  * For legacy interrupt, we have only one interrupt,
4209                  * so we have only one rx ring and one tx ring enabled.
4210                  */
4211                 ASSERT(ixgbe->num_rx_rings == 1);
4212                 ASSERT(ixgbe->num_tx_rings == 1);

4214                 /*
4215                  * For legacy interrupt, rx rings[0] will use RTxQ[0].
4216                  */
4217                 if (eicr & 0x1) {
4218                         ixgbe->eimc |= IXGBE_EICR_RTX_QUEUE;
4219                         IXGBE_WRITE_REG(hw, IXGBE_EIMC, ixgbe->eimc);
4220                         ixgbe->eims |= IXGBE_EICR_RTX_QUEUE;
4221                         /*
4222                          * Clean the rx descriptors
4223                          */
4224                         rx_ring = &ixgbe->rx_rings[0];
4225                         mp = ixgbe_ring_rx(rx_ring, IXGBE_POLL_NULL);
4226                 }

4228                 /*
4229                  * For legacy interrupt, tx rings[0] will use RTxQ[1].
4230                  */
4231                 if (eicr & 0x2) {
4232                         /*
4233                          * Recycle the tx descriptors
4234                          */
4235                         tx_ring = &ixgbe->tx_rings[0];
4236                         tx_ring->tx_recycle(tx_ring);

4238                         /*
4239                          * Schedule the re-transmit
4240                          */
4241                         tx_reschedule = (tx_ring->reschedule &&
4242                             (tx_ring->tbd_free >= ixgbe->tx_resched_thresh));
4243                 }

4245                 /* any interrupt type other than tx/rx */
4246                 if (eicr & ixgbe->capab->other_intr) {
4247                         switch (hw->mac.type) {
4248                         case ixgbe_mac_82598EB:
4249                                 ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4250                                 break;

4252                         case ixgbe_mac_82599EB:
```

```
4253                        case ixgbe_mac_X540:
4254                                ixgbe->eimc = IXGBE_82599_OTHER_INTR;
4255                                IXGBE_WRITE_REG(hw, IXGBE_EIMC, ixgbe->eimc);
4256                                break;

4258                        default:
4259                                break;
4260                        }
4261                        ixgbe_intr_other_work(ixgbe, eicr);
4262                        ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4263                }

4265                mutex_exit(&ixgbe->gen_lock);

4267                result = DDI_INTR_CLAIMED;
4268        } else {
4269                mutex_exit(&ixgbe->gen_lock);

4271                /*
4272                 * No interrupt cause bits set: don't claim this interrupt.
4273                 */
4274                result = DDI_INTR_UNCLAIMED;
4275        }

4277        /* re-enable the interrupts which were automasked */
4278        IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);

4280        /*
4281         * Do the following work outside of the gen_lock
4282         */
4283        if (mp != NULL) {
4284                mac_rx_ring(rx_ring->ixgbe->mac_hdl, rx_ring->ring_handle, mp,
4285                    rx_ring->ring_gen_num);
4286        }

4288        if (tx_reschedule)  {
4289                tx_ring->reschedule = B_FALSE;
4290                mac_tx_ring_update(ixgbe->mac_hdl, tx_ring->ring_handle);
4291                IXGBE_DEBUG_STAT(tx_ring->stat_reschedule);
4292        }

4294        return (result);
4295 }

4297 /*
4298  * ixgbe_intr_msi - Interrupt handler for MSI.
4299  */
4300 static uint_t
4301 ixgbe_intr_msi(void *arg1, void *arg2)
4302 {
4303        ixgbe_t *ixgbe = (ixgbe_t *)arg1;
4304        struct ixgbe_hw *hw = &ixgbe->hw;
4305        uint32_t eicr;

4307        _NOTE(ARGUNUSED(arg2));

4309        eicr = IXGBE_READ_REG(hw, IXGBE_EICR);

4311        if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) != DDI_FM_OK) {
4312                ddi_fm_service_impact(ixgbe->dip, DDI_SERVICE_DEGRADED);
4313                atomic_or_32(&ixgbe->ixgbe_state, IXGBE_ERROR);
4314                return (DDI_INTR_CLAIMED);
4315        }

4317        /*
4318         * For MSI interrupt, we have only one vector,
```



```
4319         * so we have only one rx ring and one tx ring enabled.
4320         */
4321        ASSERT(ixgbe->num_rx_rings == 1);
4322        ASSERT(ixgbe->num_tx_rings == 1);

4324        /*
4325         * For MSI interrupt, rx rings[0] will use RTxQ[0].
4326         */
4327        if (eicr & 0x1) {
4328                ixgbe_intr_rx_work(&ixgbe->rx_rings[0]);
4329        }

4331        /*
4332         * For MSI interrupt, tx rings[0] will use RTxQ[1].
4333         */
4334        if (eicr & 0x2) {
4335                ixgbe_intr_tx_work(&ixgbe->tx_rings[0]);
4336        }

4338        /* any interrupt type other than tx/rx */
4339        if (eicr & ixgbe->capab->other_intr) {
4340                mutex_enter(&ixgbe->gen_lock);
4341                switch (hw->mac.type) {
4342                case ixgbe_mac_82598EB:
4343                        ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4344                        break;

4346                case ixgbe_mac_82599EB:
4347                case ixgbe_mac_X540:
4348                        ixgbe->eimc = IXGBE_82599_OTHER_INTR;
4349                        IXGBE_WRITE_REG(hw, IXGBE_EIMC, ixgbe->eimc);
4350                        break;

4352                default:
4353                        break;
4354                }
4355                ixgbe_intr_other_work(ixgbe, eicr);
4356                ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4357                mutex_exit(&ixgbe->gen_lock);
4358        }

4360        /* re-enable the interrupts which were automasked */
4361        IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);

4363        return (DDI_INTR_CLAIMED);
4364 }

4366 /*
4367  * ixgbe_intr_msix - Interrupt handler for MSI-X.
4368  */
4369 static uint_t
4370 ixgbe_intr_msix(void *arg1, void *arg2)
4371 {
4372        ixgbe_intr_vector_t *vect = (ixgbe_intr_vector_t *)arg1;
4373        ixgbe_t *ixgbe = vect->ixgbe;
4374        struct ixgbe_hw *hw = &ixgbe->hw;
4375        uint32_t eicr;
4376        int r_idx = 0;

4378        _NOTE(ARGUNUSED(arg2));

4380        /*
4381         * Clean each rx ring that has its bit set in the map
4382         */
4383        r_idx = bt_getlowbit(vect->rx_map, 0, (ixgbe->num_rx_rings - 1));
4384        while (r_idx >= 0) {
```

```
4385                ixgbe_intr_rx_work(&ixgbe->rx_rings[r_idx]);
4386                r_idx = bt_getlowbit(vect->rx_map, (r_idx + 1),
4387                    (ixgbe->num_rx_rings - 1));
4388        }
4389
4390        /*
4391         * Clean each tx ring that has its bit set in the map
4392         */
4393        r_idx = bt_getlowbit(vect->tx_map, 0, (ixgbe->num_tx_rings - 1));
4394        while (r_idx >= 0) {
4395                ixgbe_intr_tx_work(&ixgbe->tx_rings[r_idx]);
4396                r_idx = bt_getlowbit(vect->tx_map, (r_idx + 1),
4397                    (ixgbe->num_tx_rings - 1));
4398        }
4399
4400
4401        /*
4402         * Clean other interrupt (link change) that has its bit set in the map
4403         */
4404        if (BT_TEST(vect->other_map, 0) == 1) {
4405                eicr = IXGBE_READ_REG(hw, IXGBE_EICR);
4406
4407                if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) !=
4408                    DDI_FM_OK) {
4409                        ddi_fm_service_impact(ixgbe->dip,
4410                            DDI_SERVICE_DEGRADED);
4411                        atomic_or_32(&ixgbe->ixgbe_state, IXGBE_ERROR);
4412                        return (DDI_INTR_CLAIMED);
4413                }
4414
4415                /*
4416                 * Check "other" cause bits: any interrupt type other than tx/rx
4417                 */
4418                if (eicr & ixgbe->capab->other_intr) {
4419                        mutex_enter(&ixgbe->gen_lock);
4420                        switch (hw->mac.type) {
4421                        case ixgbe_mac_82598EB:
4422                                ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4423                                ixgbe_intr_other_work(ixgbe, eicr);
4424                                break;
4425
4426                        case ixgbe_mac_82599EB:
4427                        case ixgbe_mac_X540:
4428                                ixgbe->eims |= IXGBE_EICR_RTX_QUEUE;
4429                                ixgbe_intr_other_work(ixgbe, eicr);
4430                                break;
4431
4432                        default:
4433                                break;
4434                        }
4435                        mutex_exit(&ixgbe->gen_lock);
4436                }
4437
4438                /* re-enable the interrupts which were automasked */
4439                IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);
4440        }
4441
4442        return (DDI_INTR_CLAIMED);
4443 }
```

*unchanged_portion_omitted*

```
4799 /*
4800  * ixgbe_setup_ivar - Set the given entry in the given interrupt vector
4801  * allocation register (IVAR).
4802  * cause:
4803  *   -1 : other cause
```



```
4804  *    0 : rx
4805  *    1 : tx
4806  */
4807 static void
4808 ixgbe_setup_ivar(ixgbe_t *ixgbe, uint16_t intr_alloc_entry, uint8_t msix_vector,
4809     int8_t cause)
4810 {
4811         struct ixgbe_hw *hw = &ixgbe->hw;
4812         u32 ivar, index;
4813
4814         switch (hw->mac.type) {
4815         case ixgbe_mac_82598EB:
4816                 msix_vector |= IXGBE_IVAR_ALLOC_VAL;
4817                 if (cause == -1) {
4818                         cause = 0;
4819                 }
4820                 index = (((cause * 64) + intr_alloc_entry) >> 2) & 0x1F;
4821                 ivar = IXGBE_READ_REG(hw, IXGBE_IVAR(index));
4822                 ivar &= ~(0xFF << (8 * (intr_alloc_entry & 0x3)));
4823                 ivar |= (msix_vector << (8 * (intr_alloc_entry & 0x3)));
4824                 IXGBE_WRITE_REG(hw, IXGBE_IVAR(index), ivar);
4825                 break;
4826
4827         case ixgbe_mac_82599EB:
4828         case ixgbe_mac_X540:
4829                 if (cause == -1) {
4830                         /* other causes */
4831                         msix_vector |= IXGBE_IVAR_ALLOC_VAL;
4832                         index = (intr_alloc_entry & 1) * 8;
4833                         ivar = IXGBE_READ_REG(hw, IXGBE_IVAR_MISC);
4834                         ivar &= ~(0xFF << index);
4835                         ivar |= (msix_vector << index);
4836                         IXGBE_WRITE_REG(hw, IXGBE_IVAR_MISC, ivar);
4837                 } else {
4838                         /* tx or rx causes */
4839                         msix_vector |= IXGBE_IVAR_ALLOC_VAL;
4840                         index = ((16 * (intr_alloc_entry & 1)) + (8 * cause));
4841                         ivar = IXGBE_READ_REG(hw,
4842                             IXGBE_IVAR(intr_alloc_entry >> 1));
4843                         ivar &= ~(0xFF << index);
4844                         ivar |= (msix_vector << index);
4845                         IXGBE_WRITE_REG(hw, IXGBE_IVAR(intr_alloc_entry >> 1),
4846                             ivar);
4847                 }
4848                 break;
4849
4850         default:
4851                 break;
4852         }
4853 }
4854
4855 /*
4856  * ixgbe_enable_ivar - Enable the given entry by setting the VAL bit of
4857  * given interrupt vector allocation register (IVAR).
4858  * cause:
4859  *   -1 : other cause
4860  *    0 : rx
4861  *    1 : tx
4862  */
4863 static void
4864 ixgbe_enable_ivar(ixgbe_t *ixgbe, uint16_t intr_alloc_entry, int8_t cause)
4865 {
4866         struct ixgbe_hw *hw = &ixgbe->hw;
4867         u32 ivar, index;
4868
4869         switch (hw->mac.type) {
```

```
4870         case ixgbe_mac_82598EB:
4871                 if (cause == -1) {
4872                         cause = 0;
4873                 }
4874                 index = (((cause * 64) + intr_alloc_entry) >> 2) & 0x1F;
4875                 ivar = IXGBE_READ_REG(hw, IXGBE_IVAR(index));
4876                 ivar |= (IXGBE_IVAR_ALLOC_VAL << (8 *
4877                     (intr_alloc_entry & 0x3)));
4878                 IXGBE_WRITE_REG(hw, IXGBE_IVAR(index), ivar);
4879                 break;

4881         case ixgbe_mac_82599EB:
4882         case ixgbe_mac_X540:
4883                 if (cause == -1) {
4884                         /* other causes */
4885                         index = (intr_alloc_entry & 1) * 8;
4886                         ivar = IXGBE_READ_REG(hw, IXGBE_IVAR_MISC);
4887                         ivar |= (IXGBE_IVAR_ALLOC_VAL << index);
4888                         IXGBE_WRITE_REG(hw, IXGBE_IVAR_MISC, ivar);
4889                 } else {
4890                         /* tx or rx causes */
4891                         index = ((16 * (intr_alloc_entry & 1)) + (8 * cause));
4892                         ivar = IXGBE_READ_REG(hw,
4893                             IXGBE_IVAR(intr_alloc_entry >> 1));
4894                         ivar |= (IXGBE_IVAR_ALLOC_VAL << index);
4895                         IXGBE_WRITE_REG(hw, IXGBE_IVAR(intr_alloc_entry >> 1),
4896                             ivar);
4897                 }
4898                 break;

4900         default:
4901                 break;
4902         }
4903 }

4905 /*
4906  * ixgbe_disable_ivar - Disble the given entry by clearing the VAL bit of
4907  * given interrupt vector allocation register (IVAR).
4908  * cause:
4909  *   -1 : other cause
4910  *    0 : rx
4911  *    1 : tx
4912  */
4913 static void
4914 ixgbe_disable_ivar(ixgbe_t *ixgbe, uint16_t intr_alloc_entry, int8_t cause)
4915 {
4916         struct ixgbe_hw *hw = &ixgbe->hw;
4917         u32 ivar, index;

4919         switch (hw->mac.type) {
4920         case ixgbe_mac_82598EB:
4921                 if (cause == -1) {
4922                         cause = 0;
4923                 }
4924                 index = (((cause * 64) + intr_alloc_entry) >> 2) & 0x1F;
4925                 ivar = IXGBE_READ_REG(hw, IXGBE_IVAR(index));
4926                 ivar &= ~(IXGBE_IVAR_ALLOC_VAL<< (8 *
4927                     (intr_alloc_entry & 0x3)));
4928                 IXGBE_WRITE_REG(hw, IXGBE_IVAR(index), ivar);
4929                 break;

4931         case ixgbe_mac_82599EB:
4932         case ixgbe_mac_X540:
4933                 if (cause == -1) {
4934                         /* other causes */
4935                         index = (intr_alloc_entry & 1) * 8;
```



```
4936                         ivar = IXGBE_READ_REG(hw, IXGBE_IVAR_MISC);
4937                         ivar &= ~(IXGBE_IVAR_ALLOC_VAL << index);
4938                         IXGBE_WRITE_REG(hw, IXGBE_IVAR_MISC, ivar);
4939                 } else {
4940                         /* tx or rx causes */
4941                         index = ((16 * (intr_alloc_entry & 1)) + (8 * cause));
4942                         ivar = IXGBE_READ_REG(hw,
4943                             IXGBE_IVAR(intr_alloc_entry >> 1));
4944                         ivar &= ~(IXGBE_IVAR_ALLOC_VAL << index);
4945                         IXGBE_WRITE_REG(hw, IXGBE_IVAR(intr_alloc_entry >> 1),
4946                             ivar);
4947                 }
4948                 break;

4950         default:
4951                 break;
4952         }
4953 }

4955 /*
4956  * Convert the rx ring index driver maintained to the rx ring index
4957  * in h/w.
4958  */
4959 static uint32_t
4960 ixgbe_get_hw_rx_index(ixgbe_t *ixgbe, uint32_t sw_rx_index)
4961 {

4963         struct ixgbe_hw *hw = &ixgbe->hw;
4964         uint32_t rx_ring_per_group, hw_rx_index;

4966         if (ixgbe->classify_mode == IXGBE_CLASSIFY_RSS ||
4967             ixgbe->classify_mode == IXGBE_CLASSIFY_NONE) {
4968                 return (sw_rx_index);
4969         } else if (ixgbe->classify_mode == IXGBE_CLASSIFY_VMDQ) {
4970                 switch (hw->mac.type) {
4971                 case ixgbe_mac_82598EB:
4972                         return (sw_rx_index);

4974                 case ixgbe_mac_82599EB:
4975                 case ixgbe_mac_X540:
4976                         return (sw_rx_index * 2);

4978                 default:
4979                         break;
4980                 }
4981         } else if (ixgbe->classify_mode == IXGBE_CLASSIFY_VMDQ_RSS) {
4982                 rx_ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;

4984                 switch (hw->mac.type) {
4985                 case ixgbe_mac_82598EB:
4986                         hw_rx_index = (sw_rx_index / rx_ring_per_group) *
4987                             16 + (sw_rx_index % rx_ring_per_group);
4988                         return (hw_rx_index);

4990                 case ixgbe_mac_82599EB:
4991                 case ixgbe_mac_X540:
4992                         if (ixgbe->num_rx_groups > 32) {
4993                                 hw_rx_index = (sw_rx_index /
4994                                     rx_ring_per_group) * 2 +
4995                                     (sw_rx_index % rx_ring_per_group);
4996                         } else {
4997                                 hw_rx_index = (sw_rx_index /
4998                                     rx_ring_per_group) * 4 +
4999                                     (sw_rx_index % rx_ring_per_group);
5000                         }
5001                         return (hw_rx_index);
```

```
5003                 default:
5004                         break;
5005                 }
5006         }

5008         /*
5009          * Should never reach. Just to make compiler happy.
5010          */
5011         return (sw_rx_index);
5012 }
______unchanged_portion_omitted_

5071 /*
5072  * ixgbe_setup_adapter_vector - Setup the adapter interrupt vector(s).
5073  *
5074  * This relies on ring/vector mapping already set up in the
5075  * vect_map[] structures
5076  */
5077 static void
5078 ixgbe_setup_adapter_vector(ixgbe_t *ixgbe)
5079 {
5080         struct ixgbe_hw *hw = &ixgbe->hw;
5081         ixgbe_intr_vector_t *vect;      /* vector bitmap */
5082         int r_idx;      /* ring index */
5083         int v_idx;      /* vector index */
5084         uint32_t hw_index;

5086         /*
5087          * Clear any previous entries
5088          */
5089         switch (hw->mac.type) {
5090         case ixgbe_mac_82598EB:
5091                 for (v_idx = 0; v_idx < 25; v_idx++)
5092                         IXGBE_WRITE_REG(hw, IXGBE_IVAR(v_idx), 0);
5093                 break;

5095         case ixgbe_mac_82599EB:
5096         case ixgbe_mac_X540:
5097                 for (v_idx = 0; v_idx < 64; v_idx++)
5098                         IXGBE_WRITE_REG(hw, IXGBE_IVAR(v_idx), 0);
5099                 IXGBE_WRITE_REG(hw, IXGBE_IVAR_MISC, 0);
5100                 break;

5102         default:
5103                 break;
5104         }

5106         /*
5107          * For non MSI-X interrupt, rx rings[0] will use RTxQ[0], and
5108          * tx rings[0] will use RTxQ[1].
5109          */
5110         if (ixgbe->intr_type != DDI_INTR_TYPE_MSIX) {
5111                 ixgbe_setup_ivar(ixgbe, 0, 0, 0);
5112                 ixgbe_setup_ivar(ixgbe, 0, 1, 1);
5113                 return;
5114         }

5116         /*
5117          * For MSI-X interrupt, "Other" is always on vector[0].
5118          */
5119         ixgbe_setup_ivar(ixgbe, IXGBE_IVAR_OTHER_CAUSES_INDEX, 0, -1);

5121         /*
5122          * For each interrupt vector, populate the IVAR table
5123          */
```



```
5124         for (v_idx = 0; v_idx < ixgbe->intr_cnt; v_idx++) {
5125                 vect = &ixgbe->vect_map[v_idx];

5127                 /*
5128                  * For each rx ring bit set
5129                  */
5130                 r_idx = bt_getlowbit(vect->rx_map, 0,
5131                     (ixgbe->num_rx_rings - 1));

5133                 while (r_idx >= 0) {
5134                         hw_index = ixgbe->rx_rings[r_idx].hw_index;
5135                         ixgbe_setup_ivar(ixgbe, hw_index, v_idx, 0);
5136                         r_idx = bt_getlowbit(vect->rx_map, (r_idx + 1),
5137                             (ixgbe->num_rx_rings - 1));
5138                 }

5140                 /*
5141                  * For each tx ring bit set
5142                  */
5143                 r_idx = bt_getlowbit(vect->tx_map, 0,
5144                     (ixgbe->num_tx_rings - 1));

5146                 while (r_idx >= 0) {
5147                         ixgbe_setup_ivar(ixgbe, r_idx, v_idx, 1);
5148                         r_idx = bt_getlowbit(vect->tx_map, (r_idx + 1),
5149                             (ixgbe->num_tx_rings - 1));
5150                 }
5151         }
5152 }
______unchanged_portion_omitted_
```

```
**********************************************************
   19731 Thu Jul 12 12:22:35 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_mbx.c
XXXX Intel X540 support
**********************************************************
/*******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

*******************************************************************************/
/*$FreeBSD$*/

#include "ixgbe_type.h"
#include "ixgbe_mbx.h"

/**
 *  ixgbe_read_mbx - Reads a message from the mailbox
 *  @hw: pointer to the HW structure
 *  @msg: The message buffer
 *  @size: Length of buffer
 *  @mbx_id: id of mailbox to read
 *
 *  returns SUCCESS if it successfuly read message from buffer
 **/
s32 ixgbe_read_mbx(struct ixgbe_hw *hw, u32 *msg, u16 size, u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        s32 ret_val = IXGBE_ERR_MBX;

        DEBUGFUNC("ixgbe_read_mbx");

        /* limit read to size of mailbox */
        if (size > mbx->size)
                size = mbx->size;

        if (mbx->ops.read)
                ret_val = mbx->ops.read(hw, msg, size, mbx_id);

        return ret_val;
```



```
}

/**
 *  ixgbe_write_mbx - Write a message to the mailbox
 *  @hw: pointer to the HW structure
 *  @msg: The message buffer
 *  @size: Length of buffer
 *  @mbx_id: id of mailbox to write
 *
 *  returns SUCCESS if it successfully copied message into the buffer
 **/
s32 ixgbe_write_mbx(struct ixgbe_hw *hw, u32 *msg, u16 size, u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        s32 ret_val = IXGBE_SUCCESS;

        DEBUGFUNC("ixgbe_write_mbx");

        if (size > mbx->size)
                ret_val = IXGBE_ERR_MBX;

        else if (mbx->ops.write)
                ret_val = mbx->ops.write(hw, msg, size, mbx_id);

        return ret_val;
}

/**
 *  ixgbe_check_for_msg - checks to see if someone sent us mail
 *  @hw: pointer to the HW structure
 *  @mbx_id: id of mailbox to check
 *
 *  returns SUCCESS if the Status bit was found or else ERR_MBX
 **/
s32 ixgbe_check_for_msg(struct ixgbe_hw *hw, u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        s32 ret_val = IXGBE_ERR_MBX;

        DEBUGFUNC("ixgbe_check_for_msg");

        if (mbx->ops.check_for_msg)
                ret_val = mbx->ops.check_for_msg(hw, mbx_id);

        return ret_val;
}

/**
 *  ixgbe_check_for_ack - checks to see if someone sent us ACK
 *  @hw: pointer to the HW structure
 *  @mbx_id: id of mailbox to check
 *
 *  returns SUCCESS if the Status bit was found or else ERR_MBX
 **/
s32 ixgbe_check_for_ack(struct ixgbe_hw *hw, u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        s32 ret_val = IXGBE_ERR_MBX;

        DEBUGFUNC("ixgbe_check_for_ack");

        if (mbx->ops.check_for_ack)
                ret_val = mbx->ops.check_for_ack(hw, mbx_id);

        return ret_val;
}
```

```
/**
 *  ixgbe_check_for_rst - checks to see if other side has reset
 *  @hw: pointer to the HW structure
 *  @mbx_id: id of mailbox to check
 *
 *  returns SUCCESS if the Status bit was found or else ERR_MBX
 **/
s32 ixgbe_check_for_rst(struct ixgbe_hw *hw, u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        s32 ret_val = IXGBE_ERR_MBX;

        DEBUGFUNC("ixgbe_check_for_rst");

        if (mbx->ops.check_for_rst)
                ret_val = mbx->ops.check_for_rst(hw, mbx_id);

        return ret_val;
}

/**
 *  ixgbe_poll_for_msg - Wait for message notification
 *  @hw: pointer to the HW structure
 *  @mbx_id: id of mailbox to write
 *
 *  returns SUCCESS if it successfully received a message notification
 **/
static s32 ixgbe_poll_for_msg(struct ixgbe_hw *hw, u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        int countdown = mbx->timeout;

        DEBUGFUNC("ixgbe_poll_for_msg");

        if (!countdown || !mbx->ops.check_for_msg)
                goto out;

        while (countdown && mbx->ops.check_for_msg(hw, mbx_id)) {
                countdown--;
                if (!countdown)
                        break;
                usec_delay(mbx->usec_delay);
        }

out:
        return countdown ? IXGBE_SUCCESS : IXGBE_ERR_MBX;
}

/**
 *  ixgbe_poll_for_ack - Wait for message acknowledgement
 *  @hw: pointer to the HW structure
 *  @mbx_id: id of mailbox to write
 *
 *  returns SUCCESS if it successfully received a message acknowledgement
 **/
static s32 ixgbe_poll_for_ack(struct ixgbe_hw *hw, u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        int countdown = mbx->timeout;

        DEBUGFUNC("ixgbe_poll_for_ack");

        if (!countdown || !mbx->ops.check_for_ack)
                goto out;
```



```
        while (countdown && mbx->ops.check_for_ack(hw, mbx_id)) {
                countdown--;
                if (!countdown)
                        break;
                usec_delay(mbx->usec_delay);
        }

out:
        return countdown ? IXGBE_SUCCESS : IXGBE_ERR_MBX;
}

/**
 *  ixgbe_read_posted_mbx - Wait for message notification and receive message
 *  @hw: pointer to the HW structure
 *  @msg: The message buffer
 *  @size: Length of buffer
 *  @mbx_id: id of mailbox to write
 *
 *  returns SUCCESS if it successfully received a message notification and
 *  copied it into the receive buffer.
 **/
s32 ixgbe_read_posted_mbx(struct ixgbe_hw *hw, u32 *msg, u16 size, u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        s32 ret_val = IXGBE_ERR_MBX;

        DEBUGFUNC("ixgbe_read_posted_mbx");

        if (!mbx->ops.read)
                goto out;

        ret_val = ixgbe_poll_for_msg(hw, mbx_id);

        /* if ack received read message, otherwise we timed out */
        if (!ret_val)
                ret_val = mbx->ops.read(hw, msg, size, mbx_id);
out:
        return ret_val;
}

/**
 *  ixgbe_write_posted_mbx - Write a message to the mailbox, wait for ack
 *  @hw: pointer to the HW structure
 *  @msg: The message buffer
 *  @size: Length of buffer
 *  @mbx_id: id of mailbox to write
 *
 *  returns SUCCESS if it successfully copied message into the buffer and
 *  received an ack to that message within delay * timeout period
 **/
s32 ixgbe_write_posted_mbx(struct ixgbe_hw *hw, u32 *msg, u16 size,
                           u16 mbx_id)
{
        struct ixgbe_mbx_info *mbx = &hw->mbx;
        s32 ret_val = IXGBE_ERR_MBX;

        DEBUGFUNC("ixgbe_write_posted_mbx");

        /* exit if either we can't write or there isn't a defined timeout */
        if (!mbx->ops.write || !mbx->timeout)
                goto out;

        /* send msg */
        ret_val = mbx->ops.write(hw, msg, size, mbx_id);

        /* if msg sent wait until we receive an ack */
```

```
260         if (!ret_val)
261                 ret_val = ixgbe_poll_for_ack(hw, mbx_id);
262 out:
263         return ret_val;
264 }

266 /**
267  *  ixgbe_init_mbx_ops_generic - Initialize MB function pointers
268  *  @hw: pointer to the HW structure
269  *
270  *  Setups up the mailbox read and write message function pointers
271  **/
272 void ixgbe_init_mbx_ops_generic(struct ixgbe_hw *hw)
273 {
274         struct ixgbe_mbx_info *mbx = &hw->mbx;

276         mbx->ops.read_posted = ixgbe_read_posted_mbx;
277         mbx->ops.write_posted = ixgbe_write_posted_mbx;
278 }

280 /**
281  *  ixgbe_read_v2p_mailbox - read v2p mailbox
282  *  @hw: pointer to the HW structure
283  *
284  *  This function is used to read the v2p mailbox without losing the read to
285  *  clear status bits.
286  **/
287 static u32 ixgbe_read_v2p_mailbox(struct ixgbe_hw *hw)
288 {
289         u32 v2p_mailbox = IXGBE_READ_REG(hw, IXGBE_VFMAILBOX);

291         v2p_mailbox |= hw->mbx.v2p_mailbox;
292         hw->mbx.v2p_mailbox |= v2p_mailbox & IXGBE_VFMAILBOX_R2C_BITS;

294         return v2p_mailbox;
295 }

297 /**
298  *  ixgbe_check_for_bit_vf - Determine if a status bit was set
299  *  @hw: pointer to the HW structure
300  *  @mask: bitmask for bits to be tested and cleared
301  *
302  *  This function is used to check for the read to clear bits within
303  *  the V2P mailbox.
304  **/
305 static s32 ixgbe_check_for_bit_vf(struct ixgbe_hw *hw, u32 mask)
306 {
307         u32 v2p_mailbox = ixgbe_read_v2p_mailbox(hw);
308         s32 ret_val = IXGBE_ERR_MBX;

310         if (v2p_mailbox & mask)
311                 ret_val = IXGBE_SUCCESS;

313         hw->mbx.v2p_mailbox &= ~mask;

315         return ret_val;
316 }

318 /**
319  *  ixgbe_check_for_msg_vf - checks to see if the PF has sent mail
320  *  @hw: pointer to the HW structure
321  *  @mbx_id: id of mailbox to check
322  *
323  *  returns SUCCESS if the PF has set the Status bit or else ERR_MBX
324  **/
325 static s32 ixgbe_check_for_msg_vf(struct ixgbe_hw *hw, u16 mbx_id)
```



```
326 {
327         s32 ret_val = IXGBE_ERR_MBX;

329         UNREFERENCED_1PARAMETER(mbx_id);
330         DEBUGFUNC("ixgbe_check_for_msg_vf");

332         if (!ixgbe_check_for_bit_vf(hw, IXGBE_VFMAILBOX_PFSTS)) {
333                 ret_val = IXGBE_SUCCESS;
334                 hw->mbx.stats.reqs++;
335         }

337         return ret_val;
338 }

340 /**
341  *  ixgbe_check_for_ack_vf - checks to see if the PF has ACK'd
342  *  @hw: pointer to the HW structure
343  *  @mbx_id: id of mailbox to check
344  *
345  *  returns SUCCESS if the PF has set the ACK bit or else ERR_MBX
346  **/
347 static s32 ixgbe_check_for_ack_vf(struct ixgbe_hw *hw, u16 mbx_id)
348 {
349         s32 ret_val = IXGBE_ERR_MBX;

351         UNREFERENCED_1PARAMETER(mbx_id);
352         DEBUGFUNC("ixgbe_check_for_ack_vf");

354         if (!ixgbe_check_for_bit_vf(hw, IXGBE_VFMAILBOX_PFACK)) {
355                 ret_val = IXGBE_SUCCESS;
356                 hw->mbx.stats.acks++;
357         }

359         return ret_val;
360 }

362 /**
363  *  ixgbe_check_for_rst_vf - checks to see if the PF has reset
364  *  @hw: pointer to the HW structure
365  *  @mbx_id: id of mailbox to check
366  *
367  *  returns TRUE if the PF has set the reset done bit or else FALSE
368  **/
369 static s32 ixgbe_check_for_rst_vf(struct ixgbe_hw *hw, u16 mbx_id)
370 {
371         s32 ret_val = IXGBE_ERR_MBX;

373         UNREFERENCED_1PARAMETER(mbx_id);
374         DEBUGFUNC("ixgbe_check_for_rst_vf");

376         if (!ixgbe_check_for_bit_vf(hw, (IXGBE_VFMAILBOX_RSTD |
377             IXGBE_VFMAILBOX_RSTI))) {
378                 ret_val = IXGBE_SUCCESS;
379                 hw->mbx.stats.rsts++;
380         }

382         return ret_val;
383 }

385 /**
386  *  ixgbe_obtain_mbx_lock_vf - obtain mailbox lock
387  *  @hw: pointer to the HW structure
388  *
389  *  return SUCCESS if we obtained the mailbox lock
390  **/
391 static s32 ixgbe_obtain_mbx_lock_vf(struct ixgbe_hw *hw)
```

```
392 {
393         s32 ret_val = IXGBE_ERR_MBX;
395         DEBUGFUNC("ixgbe_obtain_mbx_lock_vf");
397         /* Take ownership of the buffer */
398         IXGBE_WRITE_REG(hw, IXGBE_VFMAILBOX, IXGBE_VFMAILBOX_VFU);
400         /* reserve mailbox for vf use */
401         if (ixgbe_read_v2p_mailbox(hw) & IXGBE_VFMAILBOX_VFU)
402                 ret_val = IXGBE_SUCCESS;
404         return ret_val;
405 }
407 /**
408  *  ixgbe_write_mbx_vf - Write a message to the mailbox
409  *  @hw: pointer to the HW structure
410  *  @msg: The message buffer
411  *  @size: Length of buffer
412  *  @mbx_id: id of mailbox to write
413  *
414  *  returns SUCCESS if it successfully copied message into the buffer
415  **/
416 static s32 ixgbe_write_mbx_vf(struct ixgbe_hw *hw, u32 *msg, u16 size,
417                               u16 mbx_id)
418 {
419         s32 ret_val;
420         u16 i;
422         UNREFERENCED_1PARAMETER(mbx_id);
424         DEBUGFUNC("ixgbe_write_mbx_vf");
426         /* lock the mailbox to prevent pf/vf race condition */
427         ret_val = ixgbe_obtain_mbx_lock_vf(hw);
428         if (ret_val)
429                 goto out_no_write;
431         /* flush msg and acks as we are overwriting the message buffer */
432         ixgbe_check_for_msg_vf(hw, 0);
433         ixgbe_check_for_ack_vf(hw, 0);
435         /* copy the caller specified message to the mailbox memory buffer */
436         for (i = 0; i < size; i++)
437                 IXGBE_WRITE_REG_ARRAY(hw, IXGBE_VFMBMEM, i, msg[i]);
439         /* update stats */
440         hw->mbx.stats.msgs_tx++;
442         /* Drop VFU and interrupt the PF to tell it a message has been sent */
443         IXGBE_WRITE_REG(hw, IXGBE_VFMAILBOX, IXGBE_VFMAILBOX_REQ);
445 out_no_write:
446         return ret_val;
447 }
449 /**
450  *  ixgbe_read_mbx_vf - Reads a message from the inbox intended for vf
451  *  @hw: pointer to the HW structure
452  *  @msg: The message buffer
453  *  @size: Length of buffer
454  *  @mbx_id: id of mailbox to read
455  *
456  *  returns SUCCESS if it successfuly read message from buffer
457  **/
```



```
458 static s32 ixgbe_read_mbx_vf(struct ixgbe_hw *hw, u32 *msg, u16 size,
459                              u16 mbx_id)
460 {
461         s32 ret_val = IXGBE_SUCCESS;
462         u16 i;
464         DEBUGFUNC("ixgbe_read_mbx_vf");
465         UNREFERENCED_1PARAMETER(mbx_id);
467         /* lock the mailbox to prevent pf/vf race condition */
468         ret_val = ixgbe_obtain_mbx_lock_vf(hw);
469         if (ret_val)
470                 goto out_no_read;
472         /* copy the message from the mailbox memory buffer */
473         for (i = 0; i < size; i++)
474                 msg[i] = IXGBE_READ_REG_ARRAY(hw, IXGBE_VFMBMEM, i);
476         /* Acknowledge receipt and release mailbox, then we're done */
477         IXGBE_WRITE_REG(hw, IXGBE_VFMAILBOX, IXGBE_VFMAILBOX_ACK);
479         /* update stats */
480         hw->mbx.stats.msgs_rx++;
482 out_no_read:
483         return ret_val;
484 }
486 /**
487  *  ixgbe_init_mbx_params_vf - set initial values for vf mailbox
488  *  @hw: pointer to the HW structure
489  *
490  *  Initializes the hw->mbx struct to correct values for vf mailbox
491  */
492 void ixgbe_init_mbx_params_vf(struct ixgbe_hw *hw)
493 {
494         struct ixgbe_mbx_info *mbx = &hw->mbx;
496         /* start mailbox as timed out and let the reset_hw call set the timeout
497          * value to begin communications */
498         mbx->timeout = 0;
499         mbx->usec_delay = IXGBE_VF_MBX_INIT_DELAY;
501         mbx->size = IXGBE_VFMAILBOX_SIZE;
503         mbx->ops.read = ixgbe_read_mbx_vf;
504         mbx->ops.write = ixgbe_write_mbx_vf;
505         mbx->ops.read_posted = ixgbe_read_posted_mbx;
506         mbx->ops.write_posted = ixgbe_write_posted_mbx;
507         mbx->ops.check_for_msg = ixgbe_check_for_msg_vf;
508         mbx->ops.check_for_ack = ixgbe_check_for_ack_vf;
509         mbx->ops.check_for_rst = ixgbe_check_for_rst_vf;
511         mbx->stats.msgs_tx = 0;
512         mbx->stats.msgs_rx = 0;
513         mbx->stats.reqs = 0;
514         mbx->stats.acks = 0;
515         mbx->stats.rsts = 0;
516 }
518 static s32 ixgbe_check_for_bit_pf(struct ixgbe_hw *hw, u32 mask, s32 index)
519 {
520         u32 mbvficr = IXGBE_READ_REG(hw, IXGBE_MBVFICR(index));
521         s32 ret_val = IXGBE_ERR_MBX;
523         if (mbvficr & mask) {
```

```
524                 ret_val = IXGBE_SUCCESS;
525                 IXGBE_WRITE_REG(hw, IXGBE_MBVFICR(index), mask);
526         }

528         return ret_val;
529 }

531 /**
532  *  ixgbe_check_for_msg_pf - checks to see if the VF has sent mail
533  *  @hw: pointer to the HW structure
534  *  @vf_number: the VF index
535  *
536  *  returns SUCCESS if the VF has set the Status bit or else ERR_MBX
537  **/
538 static s32 ixgbe_check_for_msg_pf(struct ixgbe_hw *hw, u16 vf_number)
539 {
540         s32 ret_val = IXGBE_ERR_MBX;
541         s32 index = IXGBE_MBVFICR_INDEX(vf_number);
542         u32 vf_bit = vf_number % 16;

544         DEBUGFUNC("ixgbe_check_for_msg_pf");

546         if (!ixgbe_check_for_bit_pf(hw, IXGBE_MBVFICR_VFREQ_VF1 << vf_bit,
547                                     index)) {
548                 ret_val = IXGBE_SUCCESS;
549                 hw->mbx.stats.reqs++;
550         }

552         return ret_val;
553 }

555 /**
556  *  ixgbe_check_for_ack_pf - checks to see if the VF has ACKed
557  *  @hw: pointer to the HW structure
558  *  @vf_number: the VF index
559  *
560  *  returns SUCCESS if the VF has set the Status bit or else ERR_MBX
561  **/
562 static s32 ixgbe_check_for_ack_pf(struct ixgbe_hw *hw, u16 vf_number)
563 {
564         s32 ret_val = IXGBE_ERR_MBX;
565         s32 index = IXGBE_MBVFICR_INDEX(vf_number);
566         u32 vf_bit = vf_number % 16;

568         DEBUGFUNC("ixgbe_check_for_ack_pf");

570         if (!ixgbe_check_for_bit_pf(hw, IXGBE_MBVFICR_VFACK_VF1 << vf_bit,
571                                     index)) {
572                 ret_val = IXGBE_SUCCESS;
573                 hw->mbx.stats.acks++;
574         }

576         return ret_val;
577 }

579 /**
580  *  ixgbe_check_for_rst_pf - checks to see if the VF has reset
581  *  @hw: pointer to the HW structure
582  *  @vf_number: the VF index
583  *
584  *  returns SUCCESS if the VF has set the Status bit or else ERR_MBX
585  **/
586 static s32 ixgbe_check_for_rst_pf(struct ixgbe_hw *hw, u16 vf_number)
587 {
588         u32 reg_offset = (vf_number < 32) ? 0 : 1;
589         u32 vf_shift = vf_number % 32;
```



```
590         u32 vflre = 0;
591         s32 ret_val = IXGBE_ERR_MBX;

593         DEBUGFUNC("ixgbe_check_for_rst_pf");

595         switch (hw->mac.type) {
596         case ixgbe_mac_82599EB:
597                 vflre = IXGBE_READ_REG(hw, IXGBE_VFLRE(reg_offset));
598                 break;
599         case ixgbe_mac_X540:
600                 vflre = IXGBE_READ_REG(hw, IXGBE_VFLREC(reg_offset));
601                 break;
602         default:
603                 break;
604         }

606         if (vflre & (1 << vf_shift)) {
607                 ret_val = IXGBE_SUCCESS;
608                 IXGBE_WRITE_REG(hw, IXGBE_VFLREC(reg_offset), (1 << vf_shift));
609                 hw->mbx.stats.rsts++;
610         }

612         return ret_val;
613 }

615 /**
616  *  ixgbe_obtain_mbx_lock_pf - obtain mailbox lock
617  *  @hw: pointer to the HW structure
618  *  @vf_number: the VF index
619  *
620  *  return SUCCESS if we obtained the mailbox lock
621  **/
622 static s32 ixgbe_obtain_mbx_lock_pf(struct ixgbe_hw *hw, u16 vf_number)
623 {
624         s32 ret_val = IXGBE_ERR_MBX;
625         u32 p2v_mailbox;

627         DEBUGFUNC("ixgbe_obtain_mbx_lock_pf");

629         /* Take ownership of the buffer */
630         IXGBE_WRITE_REG(hw, IXGBE_PFMAILBOX(vf_number), IXGBE_PFMAILBOX_PFU);

632         /* reserve mailbox for vf use */
633         p2v_mailbox = IXGBE_READ_REG(hw, IXGBE_PFMAILBOX(vf_number));
634         if (p2v_mailbox & IXGBE_PFMAILBOX_PFU)
635                 ret_val = IXGBE_SUCCESS;

637         return ret_val;
638 }

640 /**
641  *  ixgbe_write_mbx_pf - Places a message in the mailbox
642  *  @hw: pointer to the HW structure
643  *  @msg: The message buffer
644  *  @size: Length of buffer
645  *  @vf_number: the VF index
646  *
647  *  returns SUCCESS if it successfully copied message into the buffer
648  **/
649 static s32 ixgbe_write_mbx_pf(struct ixgbe_hw *hw, u32 *msg, u16 size,
650                               u16 vf_number)
651 {
652         s32 ret_val;
653         u16 i;

655         DEBUGFUNC("ixgbe_write_mbx_pf");
```

```
657         /* lock the mailbox to prevent pf/vf race condition */
658         ret_val = ixgbe_obtain_mbx_lock_pf(hw, vf_number);
659         if (ret_val)
660                 goto out_no_write;

662         /* flush msg and acks as we are overwriting the message buffer */
663         ixgbe_check_for_msg_pf(hw, vf_number);
664         ixgbe_check_for_ack_pf(hw, vf_number);

666         /* copy the caller specified message to the mailbox memory buffer */
667         for (i = 0; i < size; i++)
668                 IXGBE_WRITE_REG_ARRAY(hw, IXGBE_PFMBMEM(vf_number), i, msg[i]);

670         /* Interrupt VF to tell it a message has been sent and release buffer*/
671         IXGBE_WRITE_REG(hw, IXGBE_PFMAILBOX(vf_number), IXGBE_PFMAILBOX_STS);

673         /* update stats */
674         hw->mbx.stats.msgs_tx++;

676 out_no_write:
677         return ret_val;

679 }

681 /**
682  *  ixgbe_read_mbx_pf - Read a message from the mailbox
683  *  @hw: pointer to the HW structure
684  *  @msg: The message buffer
685  *  @size: Length of buffer
686  *  @vf_number: the VF index
687  *
688  *  This function copies a message from the mailbox buffer to the caller's
689  *  memory buffer.  The presumption is that the caller knows that there was
690  *  a message due to a VF request so no polling for message is needed.
691  **/
692 static s32 ixgbe_read_mbx_pf(struct ixgbe_hw *hw, u32 *msg, u16 size,
693                              u16 vf_number)
694 {
695         s32 ret_val;
696         u16 i;

698         DEBUGFUNC("ixgbe_read_mbx_pf");

700         /* lock the mailbox to prevent pf/vf race condition */
701         ret_val = ixgbe_obtain_mbx_lock_pf(hw, vf_number);
702         if (ret_val)
703                 goto out_no_read;

705         /* copy the message to the mailbox memory buffer */
706         for (i = 0; i < size; i++)
707                 msg[i] = IXGBE_READ_REG_ARRAY(hw, IXGBE_PFMBMEM(vf_number), i);

709         /* Acknowledge the message and release buffer */
710         IXGBE_WRITE_REG(hw, IXGBE_PFMAILBOX(vf_number), IXGBE_PFMAILBOX_ACK);

712         /* update stats */
713         hw->mbx.stats.msgs_rx++;

715 out_no_read:
716         return ret_val;
717 }

719 /**
720  *  ixgbe_init_mbx_params_pf - set initial values for pf mailbox
721  *  @hw: pointer to the HW structure
```



```
722  *
723  *  Initializes the hw->mbx struct to correct values for pf mailbox
724  */
725 void ixgbe_init_mbx_params_pf(struct ixgbe_hw *hw)
726 {
727         struct ixgbe_mbx_info *mbx = &hw->mbx;

729         if (hw->mac.type != ixgbe_mac_82599EB &&
730             hw->mac.type != ixgbe_mac_X540)
731                 return;

733         mbx->timeout = 0;
734         mbx->usec_delay = 0;

736         mbx->size = IXGBE_VFMAILBOX_SIZE;

738         mbx->ops.read = ixgbe_read_mbx_pf;
739         mbx->ops.write = ixgbe_write_mbx_pf;
740         mbx->ops.read_posted = ixgbe_read_posted_mbx;
741         mbx->ops.write_posted = ixgbe_write_posted_mbx;
742         mbx->ops.check_for_msg = ixgbe_check_for_msg_pf;
743         mbx->ops.check_for_ack = ixgbe_check_for_ack_pf;
744         mbx->ops.check_for_rst = ixgbe_check_for_rst_pf;

746         mbx->stats.msgs_tx = 0;
747         mbx->stats.msgs_rx = 0;
748         mbx->stats.reqs = 0;
749         mbx->stats.acks = 0;
750         mbx->stats.rsts = 0;
751 }
```

```
**********
    5358 Thu Jul 12 12:22:36 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_mbx.h
XXXX Intel X540 support
**********
```

```
/*******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

*******************************************************************************/
/*$FreeBSD$*/

#ifndef _IXGBE_MBX_H_
#define _IXGBE_MBX_H_

#include "ixgbe_type.h"

#define IXGBE_VFMAILBOX_SIZE    16 /* 16 32 bit words - 64 bytes */
#define IXGBE_ERR_MBX           -100

#define IXGBE_VFMAILBOX         0x002FC
#define IXGBE_VFMBMEM           0x00200

/* Define mailbox register bits */
#define IXGBE_VFMAILBOX_REQ     0x00000001 /* Request for PF Ready bit */
#define IXGBE_VFMAILBOX_ACK     0x00000002 /* Ack PF message received */
#define IXGBE_VFMAILBOX_VFU     0x00000004 /* VF owns the mailbox buffer */
#define IXGBE_VFMAILBOX_PFU     0x00000008 /* PF owns the mailbox buffer */
#define IXGBE_VFMAILBOX_PFSTS   0x00000010 /* PF wrote a message in the MB */
#define IXGBE_VFMAILBOX_PFACK   0x00000020 /* PF ack the previous VF msg */
#define IXGBE_VFMAILBOX_RSTI    0x00000040 /* PF has reset indication */
#define IXGBE_VFMAILBOX_RSTD    0x00000080 /* PF has indicated reset done */
#define IXGBE_VFMAILBOX_R2C_BITS        0x000000B0 /* All read to clear bits */

#define IXGBE_PFMAILBOX_STS     0x00000001 /* Initiate message send to VF */
#define IXGBE_PFMAILBOX_ACK     0x00000002 /* Ack message recv'd from VF */
#define IXGBE_PFMAILBOX_VFU     0x00000004 /* VF owns the mailbox buffer */
#define IXGBE_PFMAILBOX_PFU     0x00000008 /* PF owns the mailbox buffer */
#define IXGBE_PFMAILBOX_RVFU    0x00000010 /* Reset VFU - used when VF stuck */
```



```
#define IXGBE_MBVFICR_VFREQ_MASK        0x0000FFFF /* bits for VF messages */
#define IXGBE_MBVFICR_VFREQ_VF1         0x00000001 /* bit for VF 1 message */
#define IXGBE_MBVFICR_VFACK_MASK        0xFFFF0000 /* bits for VF acks */
#define IXGBE_MBVFICR_VFACK_VF1         0x00010000 /* bit for VF 1 ack */


/* If it's a IXGBE_VF_* msg then it originates in the VF and is sent to the
 * PF.  The reverse is TRUE if it is IXGBE_PF_*.
 * Message ACK's are the value or'd with 0xF0000000
 */
#define IXGBE_VT_MSGTYPE_ACK    0x80000000 /* Messages below or'd with
                                            * this are the ACK */
#define IXGBE_VT_MSGTYPE_NACK   0x40000000 /* Messages below or'd with
                                            * this are the NACK */
#define IXGBE_VT_MSGTYPE_CTS    0x20000000 /* Indicates that VF is still
                                            * clear to send requests */
#define IXGBE_VT_MSGINFO_SHIFT  16
/* bits 23:16 are used for extra info for certain messages */
#define IXGBE_VT_MSGINFO_MASK   (0xFF << IXGBE_VT_MSGINFO_SHIFT)

#define IXGBE_VF_RESET          0x01 /* VF requests reset */
#define IXGBE_VF_SET_MAC_ADDR   0x02 /* VF requests PF to set MAC addr */
#define IXGBE_VF_SET_MULTICAST  0x03 /* VF requests PF to set MC addr */
#define IXGBE_VF_SET_VLAN       0x04 /* VF requests PF to set VLAN */
#define IXGBE_VF_SET_LPE        0x05 /* VF requests PF to set VMOLR.LPE */
#define IXGBE_VF_SET_MACVLAN    0x06 /* VF requests PF for unicast filter */

/* length of permanent address message returned from PF */
#define IXGBE_VF_PERMADDR_MSG_LEN       4
/* word in permanent address message with the current multicast type */
#define IXGBE_VF_MC_TYPE_WORD           3

#define IXGBE_PF_CONTROL_MSG            0x0100 /* PF control message */


#define IXGBE_VF_MBX_INIT_TIMEOUT       2000 /* number of retries on mailbox */
#define IXGBE_VF_MBX_INIT_DELAY         500  /* microseconds between retries */

s32 ixgbe_read_mbx(struct ixgbe_hw *, u32 *, u16, u16);
s32 ixgbe_write_mbx(struct ixgbe_hw *, u32 *, u16, u16);
s32 ixgbe_read_posted_mbx(struct ixgbe_hw *, u32 *, u16, u16);
s32 ixgbe_write_posted_mbx(struct ixgbe_hw *, u32 *, u16, u16);
s32 ixgbe_check_for_msg(struct ixgbe_hw *, u16);
s32 ixgbe_check_for_ack(struct ixgbe_hw *, u16);
s32 ixgbe_check_for_rst(struct ixgbe_hw *, u16);
void ixgbe_init_mbx_ops_generic(struct ixgbe_hw *hw);
void ixgbe_init_mbx_params_vf(struct ixgbe_hw *);
void ixgbe_init_mbx_params_pf(struct ixgbe_hw *);

#endif /* _IXGBE_MBX_H_ */
```

```
**********************************************************
    3898 Thu Jul 12 12:22:37 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_osdep.h
XXXX Intel X540 support
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright(c) 2007-2010 Intel Corporation. All rights reserved.
 */

/*
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 */

#ifndef _IXGBE_OSDEP_H
#define _IXGBE_OSDEP_H

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <sys/byteorder.h>
#include <sys/conf.h>
#include <sys/debug.h>
#include <sys/stropts.h>
#include <sys/stream.h>
#include <sys/strlog.h>
#include <sys/kmem.h>
#include <sys/stat.h>
#include <sys/kstat.h>
#include <sys/modctl.h>
#include <sys/errno.h>
#include <sys/ddi.h>
#include <sys/ddidmareq.h>
#include <sys/sunddi.h>
#include <sys/pci.h>
#include <sys/atomic.h>
#include <sys/note.h>
#include "ixgbe_debug.h"

/* Cheesy hack for EWARN() */
#define EWARN(H, W, S) printf(W)
```



```
/* function declarations */
struct ixgbe_hw;
uint16_t ixgbe_read_pci_cfg(struct ixgbe_hw *, uint32_t);
void ixgbe_write_pci_cfg(struct ixgbe_hw *, uint32_t, uint32_t);

#define usec_delay(x)           drv_usecwait(x)
#define msec_delay(x)           drv_usecwait(x * 1000)

#define OS_DEP(hw)              ((struct ixgbe_osdep *)((hw)->back))

#define false           B_FALSE
#define true            B_TRUE
#define FALSE           B_FALSE
#define TRUE            B_TRUE

#define IXGBE_READ_PCIE_WORD    ixgbe_read_pci_cfg
#define IXGBE_WRITE_PCIE_WORD   ixgbe_write_pci_cfg
#define CMD_MEM_WRT_INVALIDATE  0x0010  /* BIT_4 */
#define PCI_COMMAND_REGISTER    0x04
#define PCI_EX_CONF_CAP         0xE0
#define SPEED_10GB              10000
#define SPEED_1GB               1000
#define SPEED_100               100
#define FULL_DUPLEX             2

#define IXGBE_WRITE_FLUSH(a)    (void) IXGBE_READ_REG(a, IXGBE_STATUS)

#define IXGBE_WRITE_REG(a, reg, value)  \
        ddi_put32((OS_DEP(a))->reg_handle, \
            (uint32_t *)((uintptr_t)(a)->hw_addr + reg), (value))

#define IXGBE_WRITE_REG_ARRAY(a, reg, index, value)     \
        IXGBE_WRITE_REG(a, ((reg) + ((index) << 2)), (value))

#define IXGBE_READ_REG(a, reg)  \
        ddi_get32((OS_DEP(a))->reg_handle, \
            (uint32_t *)((uintptr_t)(a)->hw_addr + reg))

#define IXGBE_READ_REG_ARRAY(a, reg, index)     \
        IXGBE_READ_REG(a, ((reg) + ((index) << 2)))

#define msec_delay_irq  msec_delay
#define IXGBE_HTONL     htonl
#define IXGBE_NTOHL     ntohl
#define IXGBE_NTOHS     ntohs

#ifdef _BIG_ENDIAN
#define IXGBE_CPU_TO_LE32       BSWAP_32
#define IXGBE_LE32_TO_CPUS      BSWAP_32
#else
#define IXGBE_CPU_TO_LE32(x)    (x)
#define IXGBE_LE32_TO_CPUS(x)   (x)
#endif  /* _BIG_ENDIAN */

#define UNREFERENCED_PARAMETER(x)               _NOTE(ARGUNUSED(x))
#define UNREFERENCED_1PARAMETER(_p)             UNREFERENCED_PARAMETER(_p)
#define UNREFERENCED_2PARAMETER(_p, _q)         _NOTE(ARGUNUSED(_p, _q))
#define UNREFERENCED_3PARAMETER(_p, _q, _r)     _NOTE(ARGUNUSED(_p, _q, _r))
#define UNREFERENCED_4PARAMETER(_p, _q, _r, _s) _NOTE(ARGUNUSED(_p, _q,_r, _s))


typedef int8_t          s8;
typedef int16_t         s16;
typedef int32_t         s32;
```

```
 128 typedef int64_t         s64;
 129 typedef uint8_t         u8;
 130 typedef uint16_t        u16;
 131 typedef uint32_t        u32;
 132 typedef uint64_t        u64;
 133 typedef boolean_t       bool;

 135 struct ixgbe_osdep {
 136         ddi_acc_handle_t reg_handle;
 137         ddi_acc_handle_t cfg_handle;
 138         struct ixgbe *ixgbe;
 139 };
```

***unchanged_portion_omitted_***

```
**********************************************************
   51266 Thu Jul 12 12:22:37 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_phy.c
XXXX Intel X540 support
**********************************************************
/******************************************************************************
  Copyright (c) 2001-2012, Intel Corporation
  Copyright (c) 2001-2010, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_phy.c,v 1.13 2012/07/05 20:51:44 jfv Exp $*/
/*$FreeBSD$*/

#include "ixgbe_api.h"
#include "ixgbe_common.h"
#include "ixgbe_phy.h"

static void ixgbe_i2c_start(struct ixgbe_hw *hw);
static void ixgbe_i2c_stop(struct ixgbe_hw *hw);
static s32 ixgbe_clock_in_i2c_byte(struct ixgbe_hw *hw, u8 *data);
static s32 ixgbe_clock_out_i2c_byte(struct ixgbe_hw *hw, u8 data);
static s32 ixgbe_get_i2c_ack(struct ixgbe_hw *hw);
static s32 ixgbe_clock_in_i2c_bit(struct ixgbe_hw *hw, bool *data);
static s32 ixgbe_clock_out_i2c_bit(struct ixgbe_hw *hw, bool data);
static void ixgbe_raise_i2c_clk(struct ixgbe_hw *hw, u32 *i2cctl);
static s32 ixgbe_raise_i2c_clk(struct ixgbe_hw *hw, u32 *i2cctl);
static void ixgbe_lower_i2c_clk(struct ixgbe_hw *hw, u32 *i2cctl);
static s32 ixgbe_set_i2c_data(struct ixgbe_hw *hw, u32 *i2cctl, bool data);
static bool ixgbe_get_i2c_data(u32 *i2cctl);
void ixgbe_i2c_bus_clear(struct ixgbe_hw *hw);

/**
 *  ixgbe_init_phy_ops_generic - Inits PHY function ptrs
 *  @hw: pointer to the hardware structure
 *
 *  Initialize the function pointers.
 **/
s32 ixgbe_init_phy_ops_generic(struct ixgbe_hw *hw)
```



```
{
        struct ixgbe_phy_info *phy = &hw->phy;

        DEBUGFUNC("ixgbe_init_phy_ops_generic");

        /* PHY */
        phy->ops.identify = &ixgbe_identify_phy_generic;
        phy->ops.reset = &ixgbe_reset_phy_generic;
        phy->ops.read_reg = &ixgbe_read_phy_reg_generic;
        phy->ops.write_reg = &ixgbe_write_phy_reg_generic;
        phy->ops.setup_link = &ixgbe_setup_phy_link_generic;
        phy->ops.setup_link_speed = &ixgbe_setup_phy_link_speed_generic;
        phy->ops.check_link = NULL;
        phy->ops.get_firmware_version = ixgbe_get_phy_firmware_version_generic;
        phy->ops.read_i2c_byte = &ixgbe_read_i2c_byte_generic;
        phy->ops.write_i2c_byte = &ixgbe_write_i2c_byte_generic;
        phy->ops.read_i2c_eeprom = &ixgbe_read_i2c_eeprom_generic;
        phy->ops.write_i2c_eeprom = &ixgbe_write_i2c_eeprom_generic;
        phy->ops.i2c_bus_clear = &ixgbe_i2c_bus_clear;
        phy->ops.identify_sfp = &ixgbe_identify_module_generic;
        phy->ops.identify_sfp = &ixgbe_identify_sfp_module_generic;
        phy->sfp_type = ixgbe_sfp_type_unknown;
        phy->ops.check_overtemp = &ixgbe_tn_check_overtemp;
        return IXGBE_SUCCESS;
}

/**
 *  ixgbe_identify_phy_generic - Get physical layer module
 *  @hw: pointer to hardware structure
 *
 *  Determines the physical layer module found on the current adapter.
 **/
s32 ixgbe_identify_phy_generic(struct ixgbe_hw *hw)
{
        s32 status = IXGBE_ERR_PHY_ADDR_INVALID;
        u32 phy_addr;
        u16 ext_ability = 0;

        DEBUGFUNC("ixgbe_identify_phy_generic");

        if (hw->phy.type == ixgbe_phy_unknown) {
                for (phy_addr = 0; phy_addr < IXGBE_MAX_PHY_ADDR; phy_addr++) {
                        if (ixgbe_validate_phy_addr(hw, phy_addr)) {
                                hw->phy.addr = phy_addr;
                                ixgbe_get_phy_id(hw);
                                (void) ixgbe_get_phy_id(hw);
                                hw->phy.type =
                                        ixgbe_get_phy_type_from_id(hw->phy.id);

                                if (hw->phy.type == ixgbe_phy_unknown) {
                                        hw->phy.ops.read_reg(hw,
                                                  IXGBE_MDIO_PHY_EXT_ABILITY,
                                                  IXGBE_MDIO_PMA_PMD_DEV_TYPE,
                                                  &ext_ability);
                                        if (ext_ability &
                                            (IXGBE_MDIO_PHY_10GBASET_ABILITY |
                                             IXGBE_MDIO_PHY_1000BASET_ABILITY))
                                                hw->phy.type =
                                                        ixgbe_phy_cu_unknown;
                                        else
                                                hw->phy.type =
                                                        ixgbe_phy_generic;
                                }

                                status = IXGBE_SUCCESS;
                                break;
```

```
122                         }
123                 }
124                 /* clear value if nothing found */
125                 if (status != IXGBE_SUCCESS)
126                         hw->phy.addr = 0;
127         } else {
128                 status = IXGBE_SUCCESS;
129         }

131         return status;
132 }
_____unchanged_portion_omitted_

184 /**
185  *  ixgbe_get_phy_type_from_id - Get the phy type
186  *  @hw: pointer to hardware structure
187  *
188  **/
189 enum ixgbe_phy_type ixgbe_get_phy_type_from_id(u32 phy_id)
190 {
191         enum ixgbe_phy_type phy_type;

193         DEBUGFUNC("ixgbe_get_phy_type_from_id");

195         switch (phy_id) {
196         case TN1010_PHY_ID:
197                 phy_type = ixgbe_phy_tn;
198                 break;
199         case X540_PHY_ID:
200         case AQ1002_PHY_ID:
200                 phy_type = ixgbe_phy_aq;
201                 break;
202         case QT2022_PHY_ID:
203                 phy_type = ixgbe_phy_qt;
204                 break;
205         case ATH_PHY_ID:
206                 phy_type = ixgbe_phy_nl;
207                 break;
208         default:
209                 phy_type = ixgbe_phy_unknown;
210                 break;
211         }

213         DEBUGOUT1("phy type found is %d\n", phy_type);
214         return phy_type;
215 }
_____unchanged_portion_omitted_

272 /**
273  *  ixgbe_read_phy_reg_generic - Reads a value from a specified PHY register
274  *  @hw: pointer to hardware structure
275  *  @reg_addr: 32 bit address of PHY register to read
276  *  @phy_data: Pointer to read data from PHY register
277  **/
278 s32 ixgbe_read_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
279                                u32 device_type, u16 *phy_data)
280 {
281         u32 command;
282         u32 i;
283         u32 data;
284         s32 status = IXGBE_SUCCESS;
285         u16 gssr;

287         DEBUGFUNC("ixgbe_read_phy_reg_generic");

289         if (IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_LAN_ID_1)
```



```
290                 gssr = IXGBE_GSSR_PHY1_SM;
291         else
292                 gssr = IXGBE_GSSR_PHY0_SM;

294         if (hw->mac.ops.acquire_swfw_sync(hw, gssr) != IXGBE_SUCCESS)
295         if (ixgbe_acquire_swfw_sync(hw, gssr) != IXGBE_SUCCESS)
295                 status = IXGBE_ERR_SWFW_SYNC;

297         if (status == IXGBE_SUCCESS) {
298                 /* Setup and write the address cycle command */
299                 command = ((reg_addr << IXGBE_MSCA_NP_ADDR_SHIFT)  |
300                            (device_type << IXGBE_MSCA_DEV_TYPE_SHIFT) |
301                            (hw->phy.addr << IXGBE_MSCA_PHY_ADDR_SHIFT) |
302                            (IXGBE_MSCA_ADDR_CYCLE | IXGBE_MSCA_MDI_COMMAND));

304                 IXGBE_WRITE_REG(hw, IXGBE_MSCA, command);

306                 /*
307                  * Check every 10 usec to see if the address cycle completed.
308                  * The MDI Command bit will clear when the operation is
309                  * complete
310                  */
311                 for (i = 0; i < IXGBE_MDIO_COMMAND_TIMEOUT; i++) {
312                         usec_delay(10);

314                         command = IXGBE_READ_REG(hw, IXGBE_MSCA);

316                         if ((command & IXGBE_MSCA_MDI_COMMAND) == 0)
317                                 break;
318                 }

320                 if ((command & IXGBE_MSCA_MDI_COMMAND) != 0) {
321                         DEBUGOUT("PHY address command did not complete.\n");
322                         status = IXGBE_ERR_PHY;
323                 }

325                 if (status == IXGBE_SUCCESS) {
326                         /*
327                          * Address cycle complete, setup and write the read
328                          * command
329                          */
330                         command = ((reg_addr << IXGBE_MSCA_NP_ADDR_SHIFT)  |
331                                    (device_type << IXGBE_MSCA_DEV_TYPE_SHIFT) |
332                                    (hw->phy.addr << IXGBE_MSCA_PHY_ADDR_SHIFT) |
333                                    (IXGBE_MSCA_READ | IXGBE_MSCA_MDI_COMMAND));

335                         IXGBE_WRITE_REG(hw, IXGBE_MSCA, command);

337                         /*
338                          * Check every 10 usec to see if the address cycle
339                          * completed. The MDI Command bit will clear when the
340                          * operation is complete
341                          */
342                         for (i = 0; i < IXGBE_MDIO_COMMAND_TIMEOUT; i++) {
343                                 usec_delay(10);

345                                 command = IXGBE_READ_REG(hw, IXGBE_MSCA);

347                                 if ((command & IXGBE_MSCA_MDI_COMMAND) == 0)
348                                         break;
349                         }

351                         if ((command & IXGBE_MSCA_MDI_COMMAND) != 0) {
352                                 DEBUGOUT("PHY read command didn't complete\n");
353                                 status = IXGBE_ERR_PHY;
354                         } else {
```

```
355                                 /*
356                                  * Read operation is complete.  Get the data
357                                  * from MSRWD
358                                  */
359                                 data = IXGBE_READ_REG(hw, IXGBE_MSRWD);
360                                 data >>= IXGBE_MSRWD_READ_DATA_SHIFT;
361                                 *phy_data = (u16)(data);
362                         }
363                 }

365                 hw->mac.ops.release_swfw_sync(hw, gssr);
366                 ixgbe_release_swfw_sync(hw, gssr);
366         }

368         return status;
369 }

371 /**
372  *  ixgbe_write_phy_reg_generic - Writes a value to specified PHY register
373  *  @hw: pointer to hardware structure
374  *  @reg_addr: 32 bit PHY register to write
375  *  @device_type: 5 bit device type
376  *  @phy_data: Data to write to the PHY register
377  **/
378 s32 ixgbe_write_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
379                                 u32 device_type, u16 phy_data)
380 {
381         u32 command;
382         u32 i;
383         s32 status = IXGBE_SUCCESS;
384         u16 gssr;

386         DEBUGFUNC("ixgbe_write_phy_reg_generic");

388         if (IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_LAN_ID_1)
389                 gssr = IXGBE_GSSR_PHY1_SM;
390         else
391                 gssr = IXGBE_GSSR_PHY0_SM;

393         if (hw->mac.ops.acquire_swfw_sync(hw, gssr) != IXGBE_SUCCESS)
394         if (ixgbe_acquire_swfw_sync(hw, gssr) != IXGBE_SUCCESS)
394                 status = IXGBE_ERR_SWFW_SYNC;

396         if (status == IXGBE_SUCCESS) {
397                 /* Put the data in the MDI single read and write data register*/
398                 IXGBE_WRITE_REG(hw, IXGBE_MSRWD, (u32)phy_data);

400                 /* Setup and write the address cycle command */
401                 command = ((reg_addr << IXGBE_MSCA_NP_ADDR_SHIFT)  |
402                           (device_type << IXGBE_MSCA_DEV_TYPE_SHIFT) |
403                           (hw->phy.addr << IXGBE_MSCA_PHY_ADDR_SHIFT) |
404                           (IXGBE_MSCA_ADDR_CYCLE | IXGBE_MSCA_MDI_COMMAND));

406                 IXGBE_WRITE_REG(hw, IXGBE_MSCA, command);

408                 /*
409                  * Check every 10 usec to see if the address cycle completed.
410                  * The MDI Command bit will clear when the operation is
411                  * complete
412                  */
413                 for (i = 0; i < IXGBE_MDIO_COMMAND_TIMEOUT; i++) {
414                         usec_delay(10);

416                         command = IXGBE_READ_REG(hw, IXGBE_MSCA);

418                         if ((command & IXGBE_MSCA_MDI_COMMAND) == 0)
```



```
419                                 break;
420                 }

422                 if ((command & IXGBE_MSCA_MDI_COMMAND) != 0) {
423                         DEBUGOUT("PHY address cmd didn't complete\n");
424                         status = IXGBE_ERR_PHY;
425                 }

427                 if (status == IXGBE_SUCCESS) {
428                         /*
429                          * Address cycle complete, setup and write the write
430                          * command
431                          */
432                         command = ((reg_addr << IXGBE_MSCA_NP_ADDR_SHIFT)  |
433                                   (device_type << IXGBE_MSCA_DEV_TYPE_SHIFT) |
434                                   (hw->phy.addr << IXGBE_MSCA_PHY_ADDR_SHIFT) |
435                                   (IXGBE_MSCA_WRITE | IXGBE_MSCA_MDI_COMMAND));

437                         IXGBE_WRITE_REG(hw, IXGBE_MSCA, command);

439                         /*
440                          * Check every 10 usec to see if the address cycle
441                          * completed. The MDI Command bit will clear when the
442                          * operation is complete
443                          */
444                         for (i = 0; i < IXGBE_MDIO_COMMAND_TIMEOUT; i++) {
445                                 usec_delay(10);

447                                 command = IXGBE_READ_REG(hw, IXGBE_MSCA);

449                                 if ((command & IXGBE_MSCA_MDI_COMMAND) == 0)
450                                         break;
451                         }

453                         if ((command & IXGBE_MSCA_MDI_COMMAND) != 0) {
454                                 DEBUGOUT("PHY address cmd didn't complete\n");
455                                 status = IXGBE_ERR_PHY;
456                         }
457                 }

459                 hw->mac.ops.release_swfw_sync(hw, gssr);
460                 ixgbe_release_swfw_sync(hw, gssr);
460         }

462         return status;
463 }

465 /**
466  *  ixgbe_setup_phy_link_generic - Set and restart autoneg
467  *  @hw: pointer to hardware structure
468  *
469  *  Restart autonegotiation and PHY and waits for completion.
470  **/
471 s32 ixgbe_setup_phy_link_generic(struct ixgbe_hw *hw)
472 {
473         s32 status = IXGBE_SUCCESS;
474         u32 time_out;
475         u32 max_time_out = 10;
476         u16 autoneg_reg = IXGBE_MII_AUTONEG_REG;
477         bool autoneg = FALSE;
478         ixgbe_link_speed speed;

480         DEBUGFUNC("ixgbe_setup_phy_link_generic");

482         ixgbe_get_copper_link_capabilities_generic(hw, &speed, &autoneg);
483         (void) ixgbe_get_copper_link_capabilities_generic(hw, &speed, &autoneg);
```

```
484         if (speed & IXGBE_LINK_SPEED_10GB_FULL) {
485                 /* Set or unset auto-negotiation 10G advertisement */
486                 hw->phy.ops.read_reg(hw, IXGBE_MII_10GBASE_T_AUTONEG_CTRL_REG,
487                                      IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
488                                      &autoneg_reg);

490                 autoneg_reg &= ~IXGBE_MII_10GBASE_T_ADVERTISE;
491                 if (hw->phy.autoneg_advertised & IXGBE_LINK_SPEED_10GB_FULL)
492                         autoneg_reg |= IXGBE_MII_10GBASE_T_ADVERTISE;

494                 hw->phy.ops.write_reg(hw, IXGBE_MII_10GBASE_T_AUTONEG_CTRL_REG,
495                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
496                                       autoneg_reg);
497         }

499         if (speed & IXGBE_LINK_SPEED_1GB_FULL) {
500                 /* Set or unset auto-negotiation 1G advertisement */
501                 hw->phy.ops.read_reg(hw,
502                                      IXGBE_MII_AUTONEG_VENDOR_PROVISION_1_REG,
503                                      IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
504                                      &autoneg_reg);

506                 autoneg_reg &= ~IXGBE_MII_1GBASE_T_ADVERTISE;
507                 if (hw->phy.autoneg_advertised & IXGBE_LINK_SPEED_1GB_FULL)
508                         autoneg_reg |= IXGBE_MII_1GBASE_T_ADVERTISE;

510                 hw->phy.ops.write_reg(hw,
511                                       IXGBE_MII_AUTONEG_VENDOR_PROVISION_1_REG,
512                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
513                                       autoneg_reg);
514         }

516         if (speed & IXGBE_LINK_SPEED_100_FULL) {
517                 /* Set or unset auto-negotiation 100M advertisement */
518                 hw->phy.ops.read_reg(hw, IXGBE_MII_AUTONEG_ADVERTISE_REG,
519                                      IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
520                                      &autoneg_reg);

522                 autoneg_reg &= ~(IXGBE_MII_100BASE_T_ADVERTISE |
523                                  IXGBE_MII_100BASE_T_ADVERTISE_HALF);
523                 autoneg_reg &= ~IXGBE_MII_100BASE_T_ADVERTISE;
524                 if (hw->phy.autoneg_advertised & IXGBE_LINK_SPEED_100_FULL)
525                         autoneg_reg |= IXGBE_MII_100BASE_T_ADVERTISE;

527                 hw->phy.ops.write_reg(hw, IXGBE_MII_AUTONEG_ADVERTISE_REG,
528                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
529                                       autoneg_reg);
530         }

532         /* Restart PHY autonegotiation and wait for completion */
533         hw->phy.ops.read_reg(hw, IXGBE_MDIO_AUTO_NEG_CONTROL,
534                              IXGBE_MDIO_AUTO_NEG_DEV_TYPE, &autoneg_reg);

536         autoneg_reg |= IXGBE_MII_RESTART;

538         hw->phy.ops.write_reg(hw, IXGBE_MDIO_AUTO_NEG_CONTROL,
539                               IXGBE_MDIO_AUTO_NEG_DEV_TYPE, autoneg_reg);

541         /* Wait for autonegotiation to finish */
542         for (time_out = 0; time_out < max_time_out; time_out++) {
543                 usec_delay(10);
544                 /* Restart PHY autonegotiation and wait for completion */
545                 status = hw->phy.ops.read_reg(hw, IXGBE_MDIO_AUTO_NEG_STATUS,
546                                               IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
547                                               &autoneg_reg);
```



```
549                 autoneg_reg &= IXGBE_MII_AUTONEG_COMPLETE;
550                 if (autoneg_reg == IXGBE_MII_AUTONEG_COMPLETE)
550                 if (autoneg_reg == IXGBE_MII_AUTONEG_COMPLETE) {
551                         break;
552         }
553         }

554         if (time_out == max_time_out) {
555                 status = IXGBE_ERR_LINK_SETUP;
556                 DEBUGOUT("ixgbe_setup_phy_link_generic: time out");
557         }

559         return status;
560 }

562 /**
563  *  ixgbe_setup_phy_link_speed_generic - Sets the auto advertised capabilities
564  *  @hw: pointer to hardware structure
565  *  @speed: new link speed
566  *  @autoneg: TRUE if autonegotiation enabled
567  **/
568 s32 ixgbe_setup_phy_link_speed_generic(struct ixgbe_hw *hw,
569                                        ixgbe_link_speed speed,
570                                        bool autoneg,
571                                        bool autoneg_wait_to_complete)
572 {
573         UNREFERENCED_2PARAMETER(autoneg, autoneg_wait_to_complete);
574         UNREFERENCED_PARAMETER(autoneg);
575         UNREFERENCED_PARAMETER(autoneg_wait_to_complete);

575         DEBUGFUNC("ixgbe_setup_phy_link_speed_generic");

577         /*
578          * Clear autoneg_advertised and set new values based on input link
579          * speed.
580          */
581         hw->phy.autoneg_advertised = 0;

583         if (speed & IXGBE_LINK_SPEED_10GB_FULL)
584                 hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_10GB_FULL;

586         if (speed & IXGBE_LINK_SPEED_1GB_FULL)
587                 hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_1GB_FULL;

589         if (speed & IXGBE_LINK_SPEED_100_FULL)
590                 hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_100_FULL;

592         /* Setup link based on the new speed settings */
593         hw->phy.ops.setup_link(hw);

595         return IXGBE_SUCCESS;
596 }
______unchanged_portion_omitted_

634 /**
635  *  ixgbe_check_phy_link_tnx - Determine link and speed status
636  *  @hw: pointer to hardware structure
637  *
638  *  Reads the VS1 register to determine if link is up and the current speed for
639  *  the PHY.
640  **/
641 s32 ixgbe_check_phy_link_tnx(struct ixgbe_hw *hw, ixgbe_link_speed *speed,
642                              bool *link_up)
643 {
644         s32 status = IXGBE_SUCCESS;
```

```
645         u32 time_out;
646         u32 max_time_out = 10;
647         u16 phy_link = 0;
648         u16 phy_speed = 0;
649         u16 phy_data = 0;

651         DEBUGFUNC("ixgbe_check_phy_link_tnx");

653         /* Initialize speed and link to default case */
654         *link_up = FALSE;
655         *speed = IXGBE_LINK_SPEED_10GB_FULL;

657         /*
658          * Check current speed and link status of the PHY register.
659          * This is a vendor specific register and may have to
660          * be changed for other copper PHYs.
661          */
662         for (time_out = 0; time_out < max_time_out; time_out++) {
663                 usec_delay(10);
664                 status = hw->phy.ops.read_reg(hw,
665                                               IXGBE_MDIO_VENDOR_SPECIFIC_1_STATUS,
666                                               IXGBE_MDIO_VENDOR_SPECIFIC_1_DEV_TYPE,
667                                               &phy_data);
668                 phy_link = phy_data & IXGBE_MDIO_VENDOR_SPECIFIC_1_LINK_STATUS;
670                 phy_link = phy_data &
671                                 IXGBE_MDIO_VENDOR_SPECIFIC_1_LINK_STATUS;
669                 phy_speed = phy_data &
670                                 IXGBE_MDIO_VENDOR_SPECIFIC_1_SPEED_STATUS;
671                 if (phy_link == IXGBE_MDIO_VENDOR_SPECIFIC_1_LINK_STATUS) {
672                         *link_up = TRUE;
673                         if (phy_speed ==
674                             IXGBE_MDIO_VENDOR_SPECIFIC_1_SPEED_STATUS)
675                                 *speed = IXGBE_LINK_SPEED_1GB_FULL;
676                         break;
677                 }
678         }

680         return status;
681 }

683 /**
684  *      ixgbe_setup_phy_link_tnx - Set and restart autoneg
685  *      @hw: pointer to hardware structure
686  *
687  *      Restart autonegotiation and PHY and waits for completion.
688  **/
689 s32 ixgbe_setup_phy_link_tnx(struct ixgbe_hw *hw)
690 {
691         s32 status = IXGBE_SUCCESS;
692         u32 time_out;
693         u32 max_time_out = 10;
694         u16 autoneg_reg = IXGBE_MII_AUTONEG_REG;
695         bool autoneg = FALSE;
696         ixgbe_link_speed speed;

698         DEBUGFUNC("ixgbe_setup_phy_link_tnx");

700         ixgbe_get_copper_link_capabilities_generic(hw, &speed, &autoneg);
703         (void) ixgbe_get_copper_link_capabilities_generic(hw, &speed, &autoneg);

702         if (speed & IXGBE_LINK_SPEED_10GB_FULL) {
703                 /* Set or unset auto-negotiation 10G advertisement */
704                 hw->phy.ops.read_reg(hw, IXGBE_MII_10GBASE_T_AUTONEG_CTRL_REG,
705                                      IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
706                                      &autoneg_reg);
```



```
708                 autoneg_reg &= ~IXGBE_MII_10GBASE_T_ADVERTISE;
709                 if (hw->phy.autoneg_advertised & IXGBE_LINK_SPEED_10GB_FULL)
710                         autoneg_reg |= IXGBE_MII_10GBASE_T_ADVERTISE;

712                 hw->phy.ops.write_reg(hw, IXGBE_MII_10GBASE_T_AUTONEG_CTRL_REG,
713                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
714                                       autoneg_reg);
715         }

717         if (speed & IXGBE_LINK_SPEED_1GB_FULL) {
718                 /* Set or unset auto-negotiation 1G advertisement */
719                 hw->phy.ops.read_reg(hw, IXGBE_MII_AUTONEG_XNP_TX_REG,
720                                      IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
721                                      &autoneg_reg);

723                 autoneg_reg &= ~IXGBE_MII_1GBASE_T_ADVERTISE_XNP_TX;
724                 if (hw->phy.autoneg_advertised & IXGBE_LINK_SPEED_1GB_FULL)
725                         autoneg_reg |= IXGBE_MII_1GBASE_T_ADVERTISE_XNP_TX;

727                 hw->phy.ops.write_reg(hw, IXGBE_MII_AUTONEG_XNP_TX_REG,
728                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
729                                       autoneg_reg);
730         }

732         if (speed & IXGBE_LINK_SPEED_100_FULL) {
733                 /* Set or unset auto-negotiation 100M advertisement */
734                 hw->phy.ops.read_reg(hw, IXGBE_MII_AUTONEG_ADVERTISE_REG,
735                                      IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
736                                      &autoneg_reg);

738                 autoneg_reg &= ~IXGBE_MII_100BASE_T_ADVERTISE;
739                 if (hw->phy.autoneg_advertised & IXGBE_LINK_SPEED_100_FULL)
740                         autoneg_reg |= IXGBE_MII_100BASE_T_ADVERTISE;

742                 hw->phy.ops.write_reg(hw, IXGBE_MII_AUTONEG_ADVERTISE_REG,
743                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
744                                       autoneg_reg);
745         }

747         /* Restart PHY autonegotiation and wait for completion */
748         hw->phy.ops.read_reg(hw, IXGBE_MDIO_AUTO_NEG_CONTROL,
749                              IXGBE_MDIO_AUTO_NEG_DEV_TYPE, &autoneg_reg);

751         autoneg_reg |= IXGBE_MII_RESTART;

753         hw->phy.ops.write_reg(hw, IXGBE_MDIO_AUTO_NEG_CONTROL,
754                               IXGBE_MDIO_AUTO_NEG_DEV_TYPE, autoneg_reg);

756         /* Wait for autonegotiation to finish */
757         for (time_out = 0; time_out < max_time_out; time_out++) {
758                 usec_delay(10);
759                 /* Restart PHY autonegotiation and wait for completion */
760                 status = hw->phy.ops.read_reg(hw, IXGBE_MDIO_AUTO_NEG_STATUS,
761                                               IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
762                                               &autoneg_reg);

764                 autoneg_reg &= IXGBE_MII_AUTONEG_COMPLETE;
765                 if (autoneg_reg == IXGBE_MII_AUTONEG_COMPLETE)
768                 if (autoneg_reg == IXGBE_MII_AUTONEG_COMPLETE) {
766                         break;
767                 }
771                 }

769         if (time_out == max_time_out) {
770                 status = IXGBE_ERR_LINK_SETUP;
771                 DEBUGOUT("ixgbe_setup_phy_link_tnx: time out");
```

```
772         }

774         return status;
775 }
_____unchanged_portion_omitted_

815 /**
816  *  ixgbe_reset_phy_nl - Performs a PHY reset
817  *  @hw: pointer to hardware structure
818  **/
819 s32 ixgbe_reset_phy_nl(struct ixgbe_hw *hw)
820 {
821         u16 phy_offset, control, eword, edata, block_crc;
822         bool end_data = FALSE;
823         u16 list_offset, data_offset;
824         u16 phy_data = 0;
825         s32 ret_val = IXGBE_SUCCESS;
826         u32 i;

828         DEBUGFUNC("ixgbe_reset_phy_nl");

830         hw->phy.ops.read_reg(hw, IXGBE_MDIO_PHY_XS_CONTROL,
831                              IXGBE_MDIO_PHY_XS_DEV_TYPE, &phy_data);

833         /* reset the PHY and poll for completion */
834         hw->phy.ops.write_reg(hw, IXGBE_MDIO_PHY_XS_CONTROL,
835                               IXGBE_MDIO_PHY_XS_DEV_TYPE,
836                               (phy_data | IXGBE_MDIO_PHY_XS_RESET));

838         for (i = 0; i < 100; i++) {
839                 hw->phy.ops.read_reg(hw, IXGBE_MDIO_PHY_XS_CONTROL,
840                                      IXGBE_MDIO_PHY_XS_DEV_TYPE, &phy_data);
841                 if ((phy_data & IXGBE_MDIO_PHY_XS_RESET) == 0)
842                         break;
843                 msec_delay(10);
844         }

846         if ((phy_data & IXGBE_MDIO_PHY_XS_RESET) != 0) {
847                 DEBUGOUT("PHY reset did not complete.\n");
848                 ret_val = IXGBE_ERR_PHY;
849                 goto out;
850         }

852         /* Get init offsets */
853         ret_val = ixgbe_get_sfp_init_sequence_offsets(hw, &list_offset,
854                                                       &data_offset);
855         if (ret_val != IXGBE_SUCCESS)
856                 goto out;

858         ret_val = hw->eeprom.ops.read(hw, data_offset, &block_crc);
859         data_offset++;
860         while (!end_data) {
861                 /*
862                  * Read control word from PHY init contents offset
863                  */
864                 ret_val = hw->eeprom.ops.read(hw, data_offset, &eword);
865                 control = (eword & IXGBE_CONTROL_MASK_NL) >>
866                           IXGBE_CONTROL_SHIFT_NL;
867                 edata = eword & IXGBE_DATA_MASK_NL;
868                 switch (control) {
869                 case IXGBE_DELAY_NL:
870                         data_offset++;
871                         DEBUGOUT1("DELAY: %d MS\n", edata);
872                         msec_delay(edata);
873                         break;
874                 case IXGBE_DATA_NL:
```



```
875                         DEBUGOUT("DATA:\n");
879                         DEBUGOUT("DATA:  \n");
876                         data_offset++;
877                         hw->eeprom.ops.read(hw, data_offset++,
878                                             &phy_offset);
879                         for (i = 0; i < edata; i++) {
880                                 hw->eeprom.ops.read(hw, data_offset, &eword);
881                                 hw->phy.ops.write_reg(hw, phy_offset,
882                                                       IXGBE_TWINAX_DEV, eword);
883                                 DEBUGOUT2("Wrote %4.4x to %4.4x\n", eword,
884                                           phy_offset);
885                                 data_offset++;
886                                 phy_offset++;
887                         }
888                         break;
889                 case IXGBE_CONTROL_NL:
890                         data_offset++;
891                         DEBUGOUT("CONTROL:\n");
895                         DEBUGOUT("CONTROL: \n");
892                         if (edata == IXGBE_CONTROL_EOL_NL) {
893                                 DEBUGOUT("EOL\n");
894                                 end_data = TRUE;
895                         } else if (edata == IXGBE_CONTROL_SOL_NL) {
896                                 DEBUGOUT("SOL\n");
897                         } else {
898                                 DEBUGOUT("Bad control value\n");
899                                 ret_val = IXGBE_ERR_PHY;
900                                 goto out;
901                         }
902                         break;
903                 default:
904                         DEBUGOUT("Bad control type\n");
905                         ret_val = IXGBE_ERR_PHY;
906                         goto out;
907                 }
908         }

910 out:
911         return ret_val;
912 }

914 /**
915  *  ixgbe_identify_module_generic - Identifies module type
916  *  @hw: pointer to hardware structure
917  *
918  *  Determines HW type and calls appropriate function.
919  **/
920 s32 ixgbe_identify_module_generic(struct ixgbe_hw *hw)
921 {
922         s32 status = IXGBE_ERR_SFP_NOT_PRESENT;

924         DEBUGFUNC("ixgbe_identify_module_generic");

926         switch (hw->mac.ops.get_media_type(hw)) {
927         case ixgbe_media_type_fiber:
928                 status = ixgbe_identify_sfp_module_generic(hw);
929                 break;


932         default:
933                 hw->phy.sfp_type = ixgbe_sfp_type_not_present;
934                 status = IXGBE_ERR_SFP_NOT_PRESENT;
935                 break;
936         }

938         return status;
```

```
939 }
940
941 /**
942  *  ixgbe_identify_sfp_module_generic - Identifies SFP modules
943  *  @hw: pointer to hardware structure
944  *
945  *  Searches for and identifies the SFP module and assigns appropriate PHY type.
946  **/
947 s32 ixgbe_identify_sfp_module_generic(struct ixgbe_hw *hw)
948 {
949         s32 status = IXGBE_ERR_PHY_ADDR_INVALID;
950         u32 vendor_oui = 0;
951         enum ixgbe_sfp_type stored_sfp_type = hw->phy.sfp_type;
952         u8 identifier = 0;
953         u8 comp_codes_1g = 0;
954         u8 comp_codes_10g = 0;
955         u8 oui_bytes[3] = {0, 0, 0};
956         u8 cable_tech = 0;
957         u8 cable_spec = 0;
958         u16 enforce_sfp = 0;
959
960         DEBUGFUNC("ixgbe_identify_sfp_module_generic");
961
962         if (hw->mac.ops.get_media_type(hw) != ixgbe_media_type_fiber) {
963                 hw->phy.sfp_type = ixgbe_sfp_type_not_present;
964                 status = IXGBE_ERR_SFP_NOT_PRESENT;
965                 goto out;
966         }
967
968         status = hw->phy.ops.read_i2c_eeprom(hw,
969                                              IXGBE_SFF_IDENTIFIER,
970                                              &identifier);
971
972         if (status == IXGBE_ERR_SWFW_SYNC ||
973             status == IXGBE_ERR_I2C ||
974             status == IXGBE_ERR_SFP_NOT_PRESENT)
975                 goto err_read_i2c_eeprom;
976
977         /* LAN ID is needed for sfp_type determination */
978         hw->mac.ops.set_lan_id(hw);
979
980         if (identifier != IXGBE_SFF_IDENTIFIER_SFP) {
981                 hw->phy.type = ixgbe_phy_sfp_unsupported;
982                 status = IXGBE_ERR_SFP_NOT_SUPPORTED;
983         } else {
984                 status = hw->phy.ops.read_i2c_eeprom(hw,
985                                                      IXGBE_SFF_1GBE_COMP_CODES,
986                                                      &comp_codes_1g);
987
988                 if (status == IXGBE_ERR_SWFW_SYNC ||
989                     status == IXGBE_ERR_I2C ||
990                     status == IXGBE_ERR_SFP_NOT_PRESENT)
991                         goto err_read_i2c_eeprom;
992
993                 status = hw->phy.ops.read_i2c_eeprom(hw,
994                                                      IXGBE_SFF_10GBE_COMP_CODES,
995                                                      &comp_codes_10g);
996
997                 if (status == IXGBE_ERR_SWFW_SYNC ||
998                     status == IXGBE_ERR_I2C ||
999                     status == IXGBE_ERR_SFP_NOT_PRESENT)
1000                        goto err_read_i2c_eeprom;
1001                status = hw->phy.ops.read_i2c_eeprom(hw,
1002                                                     IXGBE_SFF_CABLE_TECHNOLOGY,
1003                                                     &cable_tech);
```



```
1005                if (status == IXGBE_ERR_SWFW_SYNC ||
1006                    status == IXGBE_ERR_I2C ||
1007                    status == IXGBE_ERR_SFP_NOT_PRESENT)
1008                        goto err_read_i2c_eeprom;
1009
1010                 /* ID Module
1011                  * =========
1012                  * 0   SFP_DA_CU
1013                  * 1   SFP_SR
1014                  * 2   SFP_LR
1015                  * 3   SFP_DA_CORE0 - 82599-specific
1016                  * 4   SFP_DA_CORE1 - 82599-specific
1017                  * 5   SFP_SR/LR_CORE0 - 82599-specific
1018                  * 6   SFP_SR/LR_CORE1 - 82599-specific
1019                  * 7   SFP_act_lmt_DA_CORE0 - 82599-specific
1020                  * 8   SFP_act_lmt_DA_CORE1 - 82599-specific
1021                  * 9   SFP_1g_cu_CORE0 - 82599-specific
1022                  * 10  SFP_1g_cu_CORE1 - 82599-specific
1023                  * 11  SFP_1g_sx_CORE0 - 82599-specific
1024                  * 12  SFP_1g_sx_CORE1 - 82599-specific
1025                  */
1026                if (hw->mac.type == ixgbe_mac_82598EB) {
1027                        if (cable_tech & IXGBE_SFF_DA_PASSIVE_CABLE)
1028                                hw->phy.sfp_type = ixgbe_sfp_type_da_cu;
1029                        else if (comp_codes_10g & IXGBE_SFF_10GBASESR_CAPABLE)
1030                                hw->phy.sfp_type = ixgbe_sfp_type_sr;
1031                        else if (comp_codes_10g & IXGBE_SFF_10GBASELR_CAPABLE)
1032                                hw->phy.sfp_type = ixgbe_sfp_type_lr;
1033                        else
1034                                hw->phy.sfp_type = ixgbe_sfp_type_unknown;
1035                } else if (hw->mac.type == ixgbe_mac_82599EB) {
1036                        if (cable_tech & IXGBE_SFF_DA_PASSIVE_CABLE) {
1037                                if (hw->bus.lan_id == 0)
1038                                        hw->phy.sfp_type =
1039                                                ixgbe_sfp_type_da_cu_core0;
1040                                else
1041                                        hw->phy.sfp_type =
1042                                                ixgbe_sfp_type_da_cu_core1;
1043                        } else if (cable_tech & IXGBE_SFF_DA_ACTIVE_CABLE) {
1044                                hw->phy.ops.read_i2c_eeprom(
1045                                                hw, IXGBE_SFF_CABLE_SPEC_COMP,
1046                                                &cable_spec);
1047                                if (cable_spec &
1048                                    IXGBE_SFF_DA_SPEC_ACTIVE_LIMITING) {
1049                                        if (hw->bus.lan_id == 0)
1050                                                hw->phy.sfp_type =
1051                                                ixgbe_sfp_type_da_act_lmt_core0;
1052                                        else
1053                                                hw->phy.sfp_type =
1054                                                ixgbe_sfp_type_da_act_lmt_core1;
1055                                } else {
1056                                        hw->phy.sfp_type =
1057                                                ixgbe_sfp_type_unknown;
1058                                }
1059                        } else if (comp_codes_10g &
1060                                   (IXGBE_SFF_10GBASESR_CAPABLE |
1061                                    IXGBE_SFF_10GBASELR_CAPABLE)) {
1062                                if (hw->bus.lan_id == 0)
1063                                        hw->phy.sfp_type =
1064                                                ixgbe_sfp_type_srlr_core0;
1065                                else
1066                                        hw->phy.sfp_type =
1067                                                ixgbe_sfp_type_srlr_core1;
1068                        } else if (comp_codes_1g & IXGBE_SFF_1GBASET_CAPABLE) {
1069                                if (hw->bus.lan_id == 0)
1070                                        hw->phy.sfp_type =
```

```
1071                                            ixgbe_sfp_type_1g_cu_core0;
1072                                    else
1073                                            hw->phy.sfp_type =
1074                                                    ixgbe_sfp_type_1g_cu_core1;
1075                            } else if (comp_codes_1g & IXGBE_SFF_1GBASESX_CAPABLE) {
1076                                    if (hw->bus.lan_id == 0)
1077                                            hw->phy.sfp_type =
1078                                                    ixgbe_sfp_type_1g_sx_core0;
1079                                    else
1080                                            hw->phy.sfp_type =
1081                                                    ixgbe_sfp_type_1g_sx_core1;
1082                            } else {
1083                                    hw->phy.sfp_type = ixgbe_sfp_type_unknown;
1084                            }
1085                    }

1087                    if (hw->phy.sfp_type != stored_sfp_type)
1088                            hw->phy.sfp_setup_needed = TRUE;

1090                    /* Determine if the SFP+ PHY is dual speed or not. */
1091                    hw->phy.multispeed_fiber = FALSE;
1092                    if (((comp_codes_1g & IXGBE_SFF_1GBASESX_CAPABLE) &&
1093                       (comp_codes_10g & IXGBE_SFF_10GBASESR_CAPABLE)) ||
1094                       ((comp_codes_1g & IXGBE_SFF_1GBASELX_CAPABLE) &&
1095                       (comp_codes_10g & IXGBE_SFF_10GBASELR_CAPABLE)))
1096                            hw->phy.multispeed_fiber = TRUE;

1098                    /* Determine PHY vendor */
1099                    if (hw->phy.type != ixgbe_phy_nl) {
1100                            hw->phy.id = identifier;
1101                            status = hw->phy.ops.read_i2c_eeprom(hw,
1102                                                    IXGBE_SFF_VENDOR_OUI_BYTE0,
1103                                                    &oui_bytes[0]);

1105                            if (status == IXGBE_ERR_SWFW_SYNC ||
1106                                status == IXGBE_ERR_I2C ||
1107                                status == IXGBE_ERR_SFP_NOT_PRESENT)
1108                                    goto err_read_i2c_eeprom;

1110                            status = hw->phy.ops.read_i2c_eeprom(hw,
1111                                                    IXGBE_SFF_VENDOR_OUI_BYTE1,
1112                                                    &oui_bytes[1]);

1114                            if (status == IXGBE_ERR_SWFW_SYNC ||
1115                                status == IXGBE_ERR_I2C ||
1116                                status == IXGBE_ERR_SFP_NOT_PRESENT)
1117                                    goto err_read_i2c_eeprom;

1119                            status = hw->phy.ops.read_i2c_eeprom(hw,
1120                                                    IXGBE_SFF_VENDOR_OUI_BYTE2,
1121                                                    &oui_bytes[2]);

1123                            if (status == IXGBE_ERR_SWFW_SYNC ||
1124                                status == IXGBE_ERR_I2C ||
1125                                status == IXGBE_ERR_SFP_NOT_PRESENT)
1126                                    goto err_read_i2c_eeprom;

1128                            vendor_oui =
1129                              ((oui_bytes[0] << IXGBE_SFF_VENDOR_OUI_BYTE0_SHIFT) |
1130                               (oui_bytes[1] << IXGBE_SFF_VENDOR_OUI_BYTE1_SHIFT) |
1131                               (oui_bytes[2] << IXGBE_SFF_VENDOR_OUI_BYTE2_SHIFT));

1133                            switch (vendor_oui) {
1134                            case IXGBE_SFF_VENDOR_OUI_TYCO:
1135                                    if (cable_tech & IXGBE_SFF_DA_PASSIVE_CABLE)
1136                                            hw->phy.type =
```



```
1137                                                    ixgbe_phy_sfp_passive_tyco;
1138                                    break;
1139                            case IXGBE_SFF_VENDOR_OUI_FTL:
1140                                    if (cable_tech & IXGBE_SFF_DA_ACTIVE_CABLE)
1141                                            hw->phy.type = ixgbe_phy_sfp_ftl_active;
1142                                    else
1143                                            hw->phy.type = ixgbe_phy_sfp_ftl;
1144                                    break;
1145                            case IXGBE_SFF_VENDOR_OUI_AVAGO:
1146                                    hw->phy.type = ixgbe_phy_sfp_avago;
1147                                    break;
1148                            case IXGBE_SFF_VENDOR_OUI_INTEL:
1149                                    hw->phy.type = ixgbe_phy_sfp_intel;
1150                                    break;
1151                            default:
1152                                    if (cable_tech & IXGBE_SFF_DA_PASSIVE_CABLE)
1153                                            hw->phy.type =
1154                                                    ixgbe_phy_sfp_passive_unknown;
1155                                    else if (cable_tech & IXGBE_SFF_DA_ACTIVE_CABLE)
1156                                            hw->phy.type =
1157                                                    ixgbe_phy_sfp_active_unknown;
1158                                    else
1159                                            hw->phy.type = ixgbe_phy_sfp_unknown;
1160                                    break;
1161                            }
1162                    }

1164                    /* Allow any DA cable vendor */
1165                    if (cable_tech & (IXGBE_SFF_DA_PASSIVE_CABLE |
1166                        IXGBE_SFF_DA_ACTIVE_CABLE)) {
1167                            status = IXGBE_SUCCESS;
1168                            goto out;
1169                    }

1171                    /* Verify supported 1G SFP modules */
1172                    if (comp_codes_10g == 0 &&
1173                        !(hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1 ||
1174                          hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0 ||
1175                          hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core0  ||
1176                          hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core1)) {
1142                          hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0)) {
1177                            hw->phy.type = ixgbe_phy_sfp_unsupported;
1178                            status = IXGBE_ERR_SFP_NOT_SUPPORTED;
1179                            goto out;
1180                    }

1182                    /* Anything else 82598-based is supported */
1183                    if (hw->mac.type == ixgbe_mac_82598EB) {
1184                            status = IXGBE_SUCCESS;
1185                            goto out;
1186                    }

1188                    ixgbe_get_device_caps(hw, &enforce_sfp);
1154                    (void) ixgbe_get_device_caps(hw, &enforce_sfp);
1189                    if (!(enforce_sfp & IXGBE_DEVICE_CAPS_ALLOW_ANY_SFP) &&
1190                        !((hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0) ||
1191                          (hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1) ||
1192                          (hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core0)  ||
1193                          (hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core1))) {
1157                          (hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1))) {
1194                            /* Make sure we're a supported PHY type */
1195                            if (hw->phy.type == ixgbe_phy_sfp_intel) {
1196                                    status = IXGBE_SUCCESS;
1197                            } else {
1198                                    if (hw->allow_unsupported_sfp == TRUE) {
1199                                            EWARN(hw, "WARNING: Intel (R) Network "
```

```
                                                "Connections are quality tested "
                                                "using Intel (R) Ethernet Optics."
                                                " Using untested modules is not "
                                                "supported and may cause unstable"
                                                " operation or damage to the "
                                                "module or the adapter. Intel "
                                                "Corporation is not responsible "
                                                "for any harm caused by using "
                                                "untested modules.\n", status);
                                        status = IXGBE_SUCCESS;
                                } else {
                                        DEBUGOUT("SFP+ module not supported\n");
                                        hw->phy.type =
                                                ixgbe_phy_sfp_unsupported;
                                hw->phy.type = ixgbe_phy_sfp_unsupported;
                                        status = IXGBE_ERR_SFP_NOT_SUPPORTED;
                                }
                        }
                } else {
                        status = IXGBE_SUCCESS;
                }
        }

out:
        return status;

err_read_i2c_eeprom:
        hw->phy.sfp_type = ixgbe_sfp_type_not_present;
        if (hw->phy.type != ixgbe_phy_nl) {
                hw->phy.id = 0;
                hw->phy.type = ixgbe_phy_unknown;
        }
        return IXGBE_ERR_SFP_NOT_PRESENT;
}



/**
 *  ixgbe_get_sfp_init_sequence_offsets - Provides offset of PHY init sequence
 *  @hw: pointer to hardware structure
 *  @list_offset: offset to the SFP ID list
 *  @data_offset: offset to the SFP data block
 *
 *  Checks the MAC's EEPROM to see if it supports a given SFP+ module type, if
 *  so it returns the offsets to the phy init sequence block.
 **/
s32 ixgbe_get_sfp_init_sequence_offsets(struct ixgbe_hw *hw,
                                        u16 *list_offset,
                                        u16 *data_offset)
{
        u16 sfp_id;
        u16 sfp_type = hw->phy.sfp_type;

        DEBUGFUNC("ixgbe_get_sfp_init_sequence_offsets");

        if (hw->phy.sfp_type == ixgbe_sfp_type_unknown)
                return IXGBE_ERR_SFP_NOT_SUPPORTED;

        if (hw->phy.sfp_type == ixgbe_sfp_type_not_present)
                return IXGBE_ERR_SFP_NOT_PRESENT;

        if ((hw->device_id == IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM) &&
            (hw->phy.sfp_type == ixgbe_sfp_type_da_cu))
                return IXGBE_ERR_SFP_NOT_SUPPORTED;

        /*
```



```
         * Limiting active cables and 1G Phys must be initialized as
         * SR modules
         */
        if (sfp_type == ixgbe_sfp_type_da_act_lmt_core0 ||
            sfp_type == ixgbe_sfp_type_1g_cu_core0 ||
            sfp_type == ixgbe_sfp_type_1g_sx_core0)
            sfp_type == ixgbe_sfp_type_1g_cu_core0)
                sfp_type = ixgbe_sfp_type_srlr_core0;
        else if (sfp_type == ixgbe_sfp_type_da_act_lmt_core1 ||
                 sfp_type == ixgbe_sfp_type_1g_cu_core1 ||
                 sfp_type == ixgbe_sfp_type_1g_sx_core1)
                 sfp_type == ixgbe_sfp_type_1g_cu_core1)
                sfp_type = ixgbe_sfp_type_srlr_core1;

        /* Read offset to PHY init contents */
        hw->eeprom.ops.read(hw, IXGBE_PHY_INIT_OFFSET_NL, list_offset);

        if ((!*list_offset) || (*list_offset == 0xFFFF))
                return IXGBE_ERR_SFP_NO_INIT_SEQ_PRESENT;

        /* Shift offset to first ID word */
        (*list_offset)++;

        /*
         * Find the matching SFP ID in the EEPROM
         * and program the init sequence
         */
        hw->eeprom.ops.read(hw, *list_offset, &sfp_id);

        while (sfp_id != IXGBE_PHY_INIT_END_NL) {
                if (sfp_id == sfp_type) {
                        (*list_offset)++;
                        hw->eeprom.ops.read(hw, *list_offset, data_offset);
                        if ((!*data_offset) || (*data_offset == 0xFFFF)) {
                                DEBUGOUT("SFP+ module not supported\n");
                                return IXGBE_ERR_SFP_NOT_SUPPORTED;
                        } else {
                                break;
                        }
                } else {
                        (*list_offset) += 2;
                        if (hw->eeprom.ops.read(hw, *list_offset, &sfp_id))
                                return IXGBE_ERR_PHY;
                }
        }

        if (sfp_id == IXGBE_PHY_INIT_END_NL) {
                DEBUGOUT("No matching SFP+ module found\n");
                return IXGBE_ERR_SFP_NOT_SUPPORTED;
        }

        return IXGBE_SUCCESS;
}
```

***_unchanged_portion_omitted_***

```
/**
 *  ixgbe_read_i2c_byte_generic - Reads 8 bit word over I2C
 *  @hw: pointer to hardware structure
 *  @byte_offset: byte offset to read
 *  @data: value read
 *
 *  Performs byte read operation to SFP module's EEPROM over I2C interface at
 *  a specified device address.
 *  a specified deivce address.
 **/
s32 ixgbe_read_i2c_byte_generic(struct ixgbe_hw *hw, u8 byte_offset,
```

```
1363                                u8 dev_addr, u8 *data)
1364 {
1365         s32 status = IXGBE_SUCCESS;
1366         u32 max_retry = 10;
1367         u32 retry = 0;
1368         u16 swfw_mask = 0;
1369         bool nack = 1;
1370         *data = 0;

1372         DEBUGFUNC("ixgbe_read_i2c_byte_generic");

1374         if (IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_LAN_ID_1)
1375                 swfw_mask = IXGBE_GSSR_PHY1_SM;
1376         else
1377                 swfw_mask = IXGBE_GSSR_PHY0_SM;

1379         do {
1380                 if (hw->mac.ops.acquire_swfw_sync(hw, swfw_mask)
1381                     != IXGBE_SUCCESS) {
1324                 if (ixgbe_acquire_swfw_sync(hw, swfw_mask) != IXGBE_SUCCESS) {
1382                         status = IXGBE_ERR_SWFW_SYNC;
1383                         goto read_byte_out;
1384                 }

1386                 ixgbe_i2c_start(hw);

1388                 /* Device Address and write indication */
1389                 status = ixgbe_clock_out_i2c_byte(hw, dev_addr);
1390                 if (status != IXGBE_SUCCESS)
1391                         goto fail;

1393                 status = ixgbe_get_i2c_ack(hw);
1394                 if (status != IXGBE_SUCCESS)
1395                         goto fail;

1397                 status = ixgbe_clock_out_i2c_byte(hw, byte_offset);
1398                 if (status != IXGBE_SUCCESS)
1399                         goto fail;

1401                 status = ixgbe_get_i2c_ack(hw);
1402                 if (status != IXGBE_SUCCESS)
1403                         goto fail;

1405                 ixgbe_i2c_start(hw);

1407                 /* Device Address and read indication */
1408                 status = ixgbe_clock_out_i2c_byte(hw, (dev_addr | 0x1));
1409                 if (status != IXGBE_SUCCESS)
1410                         goto fail;

1412                 status = ixgbe_get_i2c_ack(hw);
1413                 if (status != IXGBE_SUCCESS)
1414                         goto fail;

1416                 status = ixgbe_clock_in_i2c_byte(hw, data);
1417                 if (status != IXGBE_SUCCESS)
1418                         goto fail;

1420                 status = ixgbe_clock_out_i2c_bit(hw, nack);
1421                 if (status != IXGBE_SUCCESS)
1422                         goto fail;

1424                 ixgbe_i2c_stop(hw);
1425                 break;

1427 fail:
```



```
1428                 hw->mac.ops.release_swfw_sync(hw, swfw_mask);
1371                 ixgbe_release_swfw_sync(hw, swfw_mask);
1429                 msec_delay(100);
1430                 ixgbe_i2c_bus_clear(hw);
1431                 retry++;
1432                 if (retry < max_retry)
1433                         DEBUGOUT("I2C byte read error - Retrying.\n");
1434                 else
1435                         DEBUGOUT("I2C byte read error.\n");

1437         } while (retry < max_retry);

1439         hw->mac.ops.release_swfw_sync(hw, swfw_mask);
1382         ixgbe_release_swfw_sync(hw, swfw_mask);

1441 read_byte_out:
1442         return status;
1443 }

1445 /**
1446  *  ixgbe_write_i2c_byte_generic - Writes 8 bit word over I2C
1447  *  @hw: pointer to hardware structure
1448  *  @byte_offset: byte offset to write
1449  *  @data: value to write
1450  *
1451  *  Performs byte write operation to SFP module's EEPROM over I2C interface at
1452  *  a specified device address.
1453  **/
1454 s32 ixgbe_write_i2c_byte_generic(struct ixgbe_hw *hw, u8 byte_offset,
1455                                  u8 dev_addr, u8 data)
1456 {
1457         s32 status = IXGBE_SUCCESS;
1458         u32 max_retry = 1;
1459         u32 retry = 0;
1460         u16 swfw_mask = 0;

1462         DEBUGFUNC("ixgbe_write_i2c_byte_generic");

1464         if (IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_LAN_ID_1)
1465                 swfw_mask = IXGBE_GSSR_PHY1_SM;
1466         else
1467                 swfw_mask = IXGBE_GSSR_PHY0_SM;

1469         if (hw->mac.ops.acquire_swfw_sync(hw, swfw_mask) != IXGBE_SUCCESS) {
1412         if (ixgbe_acquire_swfw_sync(hw, swfw_mask) != IXGBE_SUCCESS) {
1470                 status = IXGBE_ERR_SWFW_SYNC;
1471                 goto write_byte_out;
1472         }

1474         do {
1475                 ixgbe_i2c_start(hw);

1477                 status = ixgbe_clock_out_i2c_byte(hw, dev_addr);
1478                 if (status != IXGBE_SUCCESS)
1479                         goto fail;

1481                 status = ixgbe_get_i2c_ack(hw);
1482                 if (status != IXGBE_SUCCESS)
1483                         goto fail;

1485                 status = ixgbe_clock_out_i2c_byte(hw, byte_offset);
1486                 if (status != IXGBE_SUCCESS)
1487                         goto fail;

1489                 status = ixgbe_get_i2c_ack(hw);
1490                 if (status != IXGBE_SUCCESS)
```

```
1491                        goto fail;

1493                status = ixgbe_clock_out_i2c_byte(hw, data);
1494                if (status != IXGBE_SUCCESS)
1495                        goto fail;

1497                status = ixgbe_get_i2c_ack(hw);
1498                if (status != IXGBE_SUCCESS)
1499                        goto fail;

1501                ixgbe_i2c_stop(hw);
1502                break;

1504 fail:
1505                ixgbe_i2c_bus_clear(hw);
1506                retry++;
1507                if (retry < max_retry)
1508                        DEBUGOUT("I2C byte write error - Retrying.\n");
1509                else
1510                        DEBUGOUT("I2C byte write error.\n");
1511        } while (retry < max_retry);

1513        hw->mac.ops.release_swfw_sync(hw, swfw_mask);
1456        ixgbe_release_swfw_sync(hw, swfw_mask);

1515 write_byte_out:
1516        return status;
1517 }

1519 /**
1520  *  ixgbe_i2c_start - Sets I2C start condition
1521  *  @hw: pointer to hardware structure
1522  *
1523  *  Sets I2C start condition (High -> Low on SDA while SCL is High)
1524  **/
1525 static void ixgbe_i2c_start(struct ixgbe_hw *hw)
1526 {
1527        u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);

1529        DEBUGFUNC("ixgbe_i2c_start");

1531        /* Start condition must begin with data and clock high */
1532        ixgbe_set_i2c_data(hw, &i2cctl, 1);
1533        ixgbe_raise_i2c_clk(hw, &i2cctl);
1475        (void) ixgbe_set_i2c_data(hw, &i2cctl, 1);
1476        (void) ixgbe_raise_i2c_clk(hw, &i2cctl);

1535        /* Setup time for start condition (4.7us) */
1536        usec_delay(IXGBE_I2C_T_SU_STA);

1538        ixgbe_set_i2c_data(hw, &i2cctl, 0);
1481        (void) ixgbe_set_i2c_data(hw, &i2cctl, 0);

1540        /* Hold time for start condition (4us) */
1541        usec_delay(IXGBE_I2C_T_HD_STA);

1543        ixgbe_lower_i2c_clk(hw, &i2cctl);

1545        /* Minimum low period of clock is 4.7 us */
1546        usec_delay(IXGBE_I2C_T_LOW);

1548 }

1550 /**
1551  *  ixgbe_i2c_stop - Sets I2C stop condition
1552  *  @hw: pointer to hardware structure
```



```
1553  *
1554  *  Sets I2C stop condition (Low -> High on SDA while SCL is High)
1555  **/
1556 static void ixgbe_i2c_stop(struct ixgbe_hw *hw)
1557 {
1558        u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);

1560        DEBUGFUNC("ixgbe_i2c_stop");

1562        /* Stop condition must begin with data low and clock high */
1563        ixgbe_set_i2c_data(hw, &i2cctl, 0);
1564        ixgbe_raise_i2c_clk(hw, &i2cctl);
1506        (void) ixgbe_set_i2c_data(hw, &i2cctl, 0);
1507        (void) ixgbe_raise_i2c_clk(hw, &i2cctl);

1566        /* Setup time for stop condition (4us) */
1567        usec_delay(IXGBE_I2C_T_SU_STO);

1569        ixgbe_set_i2c_data(hw, &i2cctl, 1);
1512        (void) ixgbe_set_i2c_data(hw, &i2cctl, 1);

1571        /* bus free time between stop and start (4.7us)*/
1572        usec_delay(IXGBE_I2C_T_BUF);
1573 }

1575 /**
1576  *  ixgbe_clock_in_i2c_byte - Clocks in one byte via I2C
1577  *  @hw: pointer to hardware structure
1578  *  @data: data byte to clock in
1579  *
1580  *  Clocks in one byte data via I2C data/clock
1581  **/
1582 static s32 ixgbe_clock_in_i2c_byte(struct ixgbe_hw *hw, u8 *data)
1583 {
1527        s32 status = IXGBE_SUCCESS;
1584        s32 i;
1585        bool bit = 0;

1587        DEBUGFUNC("ixgbe_clock_in_i2c_byte");

1589        for (i = 7; i >= 0; i--) {
1590                ixgbe_clock_in_i2c_bit(hw, &bit);
1534                status = ixgbe_clock_in_i2c_bit(hw, &bit);
1591                *data |= bit << i;

1537                if (status != IXGBE_SUCCESS)
1538                        break;
1592        }

1594        return IXGBE_SUCCESS;
1541        return status;
1595 }

1597 /**
1598  *  ixgbe_clock_out_i2c_byte - Clocks out one byte via I2C
1599  *  @hw: pointer to hardware structure
1600  *  @data: data byte clocked out
1601  *
1602  *  Clocks out one byte data via I2C data/clock
1603  **/
1604 static s32 ixgbe_clock_out_i2c_byte(struct ixgbe_hw *hw, u8 data)
1605 {
1606        s32 status = IXGBE_SUCCESS;
1607        s32 i;
1608        u32 i2cctl;
1609        bool bit = 0;
```

```
1611         DEBUGFUNC("ixgbe_clock_out_i2c_byte");

1613         for (i = 7; i >= 0; i--) {
1614                 bit = (data >> i) & 0x1;
1615                 status = ixgbe_clock_out_i2c_bit(hw, bit);

1617                 if (status != IXGBE_SUCCESS)
1618                         break;
1619         }

1621         /* Release SDA line (set high) */
1622         i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1623         i2cctl |= IXGBE_I2C_DATA_OUT;
1624         IXGBE_WRITE_REG(hw, IXGBE_I2CCTL, i2cctl);
1625         IXGBE_WRITE_FLUSH(hw);

1627         return status;
1628 }

1630 /**
1631  *  ixgbe_get_i2c_ack - Polls for I2C ACK
1632  *  @hw: pointer to hardware structure
1633  *
1634  *  Clocks in/out one bit via I2C data/clock
1635  **/
1636 static s32 ixgbe_get_i2c_ack(struct ixgbe_hw *hw)
1637 {
1638         s32 status = IXGBE_SUCCESS;
1584         s32 status;
1639         u32 i = 0;
1640         u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1641         u32 timeout = 10;
1642         bool ack = 1;

1644         DEBUGFUNC("ixgbe_get_i2c_ack");

1646         ixgbe_raise_i2c_clk(hw, &i2cctl);
1592         status = ixgbe_raise_i2c_clk(hw, &i2cctl);

1594         if (status != IXGBE_SUCCESS)
1595                 goto out;

1649         /* Minimum high period of clock is 4us */
1650         usec_delay(IXGBE_I2C_T_HIGH);

1652         /* Poll for ACK.  Note that ACK in I2C spec is
1653          * transition from 1 to 0 */
1654         for (i = 0; i < timeout; i++) {
1655                 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1656                 ack = ixgbe_get_i2c_data(&i2cctl);

1658                 usec_delay(1);
1659                 if (ack == 0)
1660                         break;
1661         }

1663         if (ack == 1) {
1664                 DEBUGOUT("I2C ack was not received.\n");
1665                 status = IXGBE_ERR_I2C;
1666         }

1668         ixgbe_lower_i2c_clk(hw, &i2cctl);

1670         /* Minimum low period of clock is 4.7 us */
1671         usec_delay(IXGBE_I2C_T_LOW);
```



```
1621 out:
1673         return status;
1674 }

1676 /**
1677  *  ixgbe_clock_in_i2c_bit - Clocks in one bit via I2C data/clock
1678  *  @hw: pointer to hardware structure
1679  *  @data: read data value
1680  *
1681  *  Clocks in one bit via I2C data/clock
1682  **/
1683 static s32 ixgbe_clock_in_i2c_bit(struct ixgbe_hw *hw, bool *data)
1684 {
1634         s32 status;
1685         u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);

1687         DEBUGFUNC("ixgbe_clock_in_i2c_bit");

1689         ixgbe_raise_i2c_clk(hw, &i2cctl);
1639         status = ixgbe_raise_i2c_clk(hw, &i2cctl);

1691         /* Minimum high period of clock is 4us */
1692         usec_delay(IXGBE_I2C_T_HIGH);

1694         i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1695         *data = ixgbe_get_i2c_data(&i2cctl);

1697         ixgbe_lower_i2c_clk(hw, &i2cctl);

1699         /* Minimum low period of clock is 4.7 us */
1700         usec_delay(IXGBE_I2C_T_LOW);

1702         return IXGBE_SUCCESS;
1652         return status;
1703 }

1705 /**
1706  *  ixgbe_clock_out_i2c_bit - Clocks in/out one bit via I2C data/clock
1707  *  @hw: pointer to hardware structure
1708  *  @data: data value to write
1709  *
1710  *  Clocks out one bit via I2C data/clock
1711  **/
1712 static s32 ixgbe_clock_out_i2c_bit(struct ixgbe_hw *hw, bool data)
1713 {
1714         s32 status;
1715         u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);

1717         DEBUGFUNC("ixgbe_clock_out_i2c_bit");

1719         status = ixgbe_set_i2c_data(hw, &i2cctl, data);
1720         if (status == IXGBE_SUCCESS) {
1721                 ixgbe_raise_i2c_clk(hw, &i2cctl);
1671                 status = ixgbe_raise_i2c_clk(hw, &i2cctl);

1723                 /* Minimum high period of clock is 4us */
1724                 usec_delay(IXGBE_I2C_T_HIGH);

1726                 ixgbe_lower_i2c_clk(hw, &i2cctl);

1728                 /* Minimum low period of clock is 4.7 us.
1729                  * This also takes care of the data hold time.
1730                  */
1731                 usec_delay(IXGBE_I2C_T_LOW);
1732         } else {
```

```
1733                 status = IXGBE_ERR_I2C;
1734                 DEBUGOUT1("I2C data was not set to %X\n", data);
1735         }

1737         return status;
1738 }
1739 /**
1740  *  ixgbe_raise_i2c_clk - Raises the I2C SCL clock
1741  *  @hw: pointer to hardware structure
1742  *  @i2cctl: Current value of I2CCTL register
1743  *
1744  *  Raises the I2C clock line '0'->'1'
1745  **/
1746 static void ixgbe_raise_i2c_clk(struct ixgbe_hw *hw, u32 *i2cctl)
1696 static s32 ixgbe_raise_i2c_clk(struct ixgbe_hw *hw, u32 *i2cctl)
1747 {
1748         u32 i = 0;
1749         u32 timeout = IXGBE_I2C_CLOCK_STRETCHING_TIMEOUT;
1750         u32 i2cctl_r = 0;
1698         s32 status = IXGBE_SUCCESS;

1752         DEBUGFUNC("ixgbe_raise_i2c_clk");

1754         for (i = 0; i < timeout; i++) {
1755                 *i2cctl |= IXGBE_I2C_CLK_OUT;

1757                 IXGBE_WRITE_REG(hw, IXGBE_I2CCTL, *i2cctl);
1758                 IXGBE_WRITE_FLUSH(hw);

1759                 /* SCL rise time (1000ns) */
1760                 usec_delay(IXGBE_I2C_T_RISE);

1762                 i2cctl_r = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1763                 if (i2cctl_r & IXGBE_I2C_CLK_IN)
1764                         break;
1765         }
1709         return status;
1766 }

1768 /**
1769  *  ixgbe_lower_i2c_clk - Lowers the I2C SCL clock
1770  *  @hw: pointer to hardware structure
1771  *  @i2cctl: Current value of I2CCTL register
1772  *
1773  *  Lowers the I2C clock line '1'->'0'
1774  **/
1775 static void ixgbe_lower_i2c_clk(struct ixgbe_hw *hw, u32 *i2cctl)
1776 {

1778         DEBUGFUNC("ixgbe_lower_i2c_clk");

1780         *i2cctl &= ~IXGBE_I2C_CLK_OUT;

1782         IXGBE_WRITE_REG(hw, IXGBE_I2CCTL, *i2cctl);
1783         IXGBE_WRITE_FLUSH(hw);

1785         /* SCL fall time (300ns) */
1786         usec_delay(IXGBE_I2C_T_FALL);
1787 }

1789 /**
1790  *  ixgbe_set_i2c_data - Sets the I2C data bit
1791  *  @hw: pointer to hardware structure
1792  *  @i2cctl: Current value of I2CCTL register
1793  *  @data: I2C data value (0 or 1) to set
1794  *
```



```
1795  *  Sets the I2C data bit
1796  **/
1797 static s32 ixgbe_set_i2c_data(struct ixgbe_hw *hw, u32 *i2cctl, bool data)
1798 {
1799         s32 status = IXGBE_SUCCESS;

1801         DEBUGFUNC("ixgbe_set_i2c_data");

1803         if (data)
1804                 *i2cctl |= IXGBE_I2C_DATA_OUT;
1805         else
1806                 *i2cctl &= ~IXGBE_I2C_DATA_OUT;

1808         IXGBE_WRITE_REG(hw, IXGBE_I2CCTL, *i2cctl);
1809         IXGBE_WRITE_FLUSH(hw);

1811         /* Data rise/fall (1000ns/300ns) and set-up time (250ns) */
1812         usec_delay(IXGBE_I2C_T_RISE + IXGBE_I2C_T_FALL + IXGBE_I2C_T_SU_DATA);

1814         /* Verify data was set correctly */
1815         *i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1816         if (data != ixgbe_get_i2c_data(i2cctl)) {
1817                 status = IXGBE_ERR_I2C;
1818                 DEBUGOUT1("Error - I2C data was not set to %X.\n", data);
1819         }

1821         return status;
1822 }
______unchanged_portion_omitted_

1845 /**
1846  *  ixgbe_i2c_bus_clear - Clears the I2C bus
1847  *  @hw: pointer to hardware structure
1848  *
1849  *  Clears the I2C bus by sending nine clock pulses.
1850  *  Used when data line is stuck low.
1851  **/
1852 void ixgbe_i2c_bus_clear(struct ixgbe_hw *hw)
1853 {
1854         u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1855         u32 i;

1857         DEBUGFUNC("ixgbe_i2c_bus_clear");

1859         ixgbe_i2c_start(hw);

1861         ixgbe_set_i2c_data(hw, &i2cctl, 1);
1803         (void) ixgbe_set_i2c_data(hw, &i2cctl, 1);

1863         for (i = 0; i < 9; i++) {
1864                 ixgbe_raise_i2c_clk(hw, &i2cctl);
1806                 (void) ixgbe_raise_i2c_clk(hw, &i2cctl);

1866                 /* Min high period of clock is 4us */
1867                 usec_delay(IXGBE_I2C_T_HIGH);

1869                 ixgbe_lower_i2c_clk(hw, &i2cctl);

1871                 /* Min low period of clock is 4.7us*/
1872                 usec_delay(IXGBE_I2C_T_LOW);
1873         }

1875         ixgbe_i2c_start(hw);

1877         /* Put the i2c bus back to default state */
1878         ixgbe_i2c_stop(hw);
```

```
1879 }

1881 /**
1882  *  ixgbe_tn_check_overtemp - Checks if an overtemp occurred.
1824  *  ixgbe_tn_check_overtemp - Checks if an overtemp occured.
1883  *  @hw: pointer to hardware structure
1884  *
1885  *  Checks if the LASI temp alarm status was triggered due to overtemp
1886  **/
1887 s32 ixgbe_tn_check_overtemp(struct ixgbe_hw *hw)
1888 {
1889         s32 status = IXGBE_SUCCESS;
1890         u16 phy_data = 0;

1892         DEBUGFUNC("ixgbe_tn_check_overtemp");

1894         if (hw->device_id != IXGBE_DEV_ID_82599_T3_LOM)
1895                 goto out;

1897         /* Check that the LASI temp alarm status was triggered */
1898         hw->phy.ops.read_reg(hw, IXGBE_TN_LASI_STATUS_REG,
1899                              IXGBE_MDIO_PMA_PMD_DEV_TYPE, &phy_data);

1901         if (!(phy_data & IXGBE_TN_LASI_STATUS_TEMP_ALARM))
1902                 goto out;

1904         status = IXGBE_ERR_OVERTEMP;
1905 out:
1906         return status;
1907 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5811 Thu Jul 12 12:22:38 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_phy.h
XXXX Intel X540 support
**********************************************************
```

```
/******************************************************************************
```



```
******************************************************************************/
/*$FreeBSD$*/

#ifndef _IXGBE_PHY_H_
#define _IXGBE_PHY_H_

#include "ixgbe_type.h"
#define IXGBE_I2C_EEPROM_DEV_ADDR    0xA0

/* EEPROM byte offsets */
#define IXGBE_SFF_IDENTIFIER            0x0
#define IXGBE_SFF_IDENTIFIER_SFP        0x3
#define IXGBE_SFF_VENDOR_OUI_BYTE0      0x25
#define IXGBE_SFF_VENDOR_OUI_BYTE1      0x26
#define IXGBE_SFF_VENDOR_OUI_BYTE2      0x27
#define IXGBE_SFF_1GBE_COMP_CODES       0x6
#define IXGBE_SFF_10GBE_COMP_CODES      0x3
#define IXGBE_SFF_CABLE_TECHNOLOGY      0x8
#define IXGBE_SFF_CABLE_SPEC_COMP       0x3C

/* Bitmasks */
#define IXGBE_SFF_DA_PASSIVE_CABLE      0x4
#define IXGBE_SFF_DA_ACTIVE_CABLE       0x8
#define IXGBE_SFF_DA_SPEC_ACTIVE_LIMITING       0x4
#define IXGBE_SFF_1GBASESX_CAPABLE      0x1
#define IXGBE_SFF_1GBASELX_CAPABLE      0x2
#define IXGBE_SFF_1GBASET_CAPABLE       0x8
#define IXGBE_SFF_10GBASESR_CAPABLE     0x10
#define IXGBE_SFF_10GBASELR_CAPABLE     0x20
```



```
#define IXGBE_I2C_EEPROM_READ_MASK      0x100
#define IXGBE_I2C_EEPROM_STATUS_MASK    0x3
#define IXGBE_I2C_EEPROM_STATUS_NO_OPERATION    0x0
#define IXGBE_I2C_EEPROM_STATUS_PASS    0x1
#define IXGBE_I2C_EEPROM_STATUS_FAIL    0x2
#define IXGBE_I2C_EEPROM_STATUS_IN_PROGRESS     0x3

/* Flow control defines */
#define IXGBE_TAF_SYM_PAUSE             0x400
#define IXGBE_TAF_ASM_PAUSE             0x800

/* Bit-shift macros */
#define IXGBE_SFF_VENDOR_OUI_BYTE0_SHIFT        24
#define IXGBE_SFF_VENDOR_OUI_BYTE1_SHIFT        16
#define IXGBE_SFF_VENDOR_OUI_BYTE2_SHIFT        8

/* Vendor OUIs: format of OUI is 0x[byte0][byte1][byte2][00] */
#define IXGBE_SFF_VENDOR_OUI_TYCO       0x00407600
#define IXGBE_SFF_VENDOR_OUI_FTL        0x00906500
#define IXGBE_SFF_VENDOR_OUI_AVAGO      0x00176A00
#define IXGBE_SFF_VENDOR_OUI_INTEL      0x001B2100

/* I2C SDA and SCL timing parameters for standard mode */
#define IXGBE_I2C_T_HD_STA      4
#define IXGBE_I2C_T_LOW         5
#define IXGBE_I2C_T_HIGH        4
#define IXGBE_I2C_T_SU_STA      5
#define IXGBE_I2C_T_HD_DATA     5
#define IXGBE_I2C_T_SU_DATA     1
#define IXGBE_I2C_T_RISE        1
#define IXGBE_I2C_T_FALL        1
#define IXGBE_I2C_T_SU_STO      4
#define IXGBE_I2C_T_BUF         5

#define IXGBE_TN_LASI_STATUS_REG        0x9005
#define IXGBE_TN_LASI_STATUS_TEMP_ALARM 0x0008

s32 ixgbe_init_phy_ops_generic(struct ixgbe_hw *hw);
bool ixgbe_validate_phy_addr(struct ixgbe_hw *hw, u32 phy_addr);
enum ixgbe_phy_type ixgbe_get_phy_type_from_id(u32 phy_id);
s32 ixgbe_get_phy_id(struct ixgbe_hw *hw);
s32 ixgbe_identify_phy_generic(struct ixgbe_hw *hw);
s32 ixgbe_reset_phy_generic(struct ixgbe_hw *hw);
s32 ixgbe_read_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
                               u32 device_type, u16 *phy_data);
s32 ixgbe_write_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
                                u32 device_type, u16 phy_data);
s32 ixgbe_setup_phy_link_generic(struct ixgbe_hw *hw);
s32 ixgbe_setup_phy_link_speed_generic(struct ixgbe_hw *hw,
                                       ixgbe_link_speed speed,
                                       bool autoneg,
                                       bool autoneg_wait_to_complete);
s32 ixgbe_get_copper_link_capabilities_generic(struct ixgbe_hw *hw,
                                               ixgbe_link_speed *speed,
                                               bool *autoneg);

/* PHY specific */
s32 ixgbe_check_phy_link_tnx(struct ixgbe_hw *hw,
                             ixgbe_link_speed *speed,
                             bool *link_up);
s32 ixgbe_setup_phy_link_tnx(struct ixgbe_hw *hw);
s32 ixgbe_get_phy_firmware_version_tnx(struct ixgbe_hw *hw,
                                       u16 *firmware_version);
s32 ixgbe_get_phy_firmware_version_generic(struct ixgbe_hw *hw,
                                           u16 *firmware_version);
```

```
127 s32 ixgbe_reset_phy_nl(struct ixgbe_hw *hw);
128 s32 ixgbe_identify_module_generic(struct ixgbe_hw *hw);
129 s32 ixgbe_identify_sfp_module_generic(struct ixgbe_hw *hw);
130 s32 ixgbe_get_sfp_init_sequence_offsets(struct ixgbe_hw *hw,
131                                         u16 *list_offset,
132                                         u16 *data_offset);
133 s32 ixgbe_tn_check_overtemp(struct ixgbe_hw *hw);
134 s32 ixgbe_read_i2c_byte_generic(struct ixgbe_hw *hw, u8 byte_offset,
135                                 u8 dev_addr, u8 *data);
136 s32 ixgbe_write_i2c_byte_generic(struct ixgbe_hw *hw, u8 byte_offset,
137                                  u8 dev_addr, u8 data);
138 s32 ixgbe_read_i2c_eeprom_generic(struct ixgbe_hw *hw, u8 byte_offset,
139                                   u8 *eeprom_data);
140 s32 ixgbe_write_i2c_eeprom_generic(struct ixgbe_hw *hw, u8 byte_offset,
141                                    u8 eeprom_data);
142 void ixgbe_i2c_bus_clear(struct ixgbe_hw *hw);
143 #endif /* _IXGBE_PHY_H_ */
```

```
**********************************************************
   24248 Thu Jul 12 12:22:39 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_stat.c
XXXX Intel X540 support
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright(c) 2007-2010 Intel Corporation. All rights reserved.
 */

/*
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 */

#include "ixgbe_sw.h"

/*
 * Update driver private statistics.
 */
static int
ixgbe_update_stats(kstat_t *ks, int rw)
{
        ixgbe_t *ixgbe;
        struct ixgbe_hw *hw;
        ixgbe_stat_t *ixgbe_ks;
        int i;

        if (rw == KSTAT_WRITE)
                return (EACCES);

        ixgbe = (ixgbe_t *)ks->ks_private;
        ixgbe_ks = (ixgbe_stat_t *)ks->ks_data;
        hw = &ixgbe->hw;

        mutex_enter(&ixgbe->gen_lock);

        /*
         * Basic information
         */
        ixgbe_ks->link_speed.value.ui64 = ixgbe->link_speed;
        ixgbe_ks->reset_count.value.ui64 = ixgbe->reset_count;
        ixgbe_ks->lroc.value.ui64 = ixgbe->lro_pkt_count;

#ifdef IXGBE_DEBUG
        ixgbe_ks->rx_frame_error.value.ui64 = 0;
        ixgbe_ks->rx_cksum_error.value.ui64 = 0;
```



```
        ixgbe_ks->rx_exceed_pkt.value.ui64 = 0;
        for (i = 0; i < ixgbe->num_rx_rings; i++) {
                ixgbe_ks->rx_frame_error.value.ui64 +=
                    ixgbe->rx_rings[i].stat_frame_error;
                ixgbe_ks->rx_cksum_error.value.ui64 +=
                    ixgbe->rx_rings[i].stat_cksum_error;
                ixgbe_ks->rx_exceed_pkt.value.ui64 +=
                    ixgbe->rx_rings[i].stat_exceed_pkt;
        }

        ixgbe_ks->tx_overload.value.ui64 = 0;
        ixgbe_ks->tx_fail_no_tbd.value.ui64 = 0;
        ixgbe_ks->tx_fail_no_tcb.value.ui64 = 0;
        ixgbe_ks->tx_fail_dma_bind.value.ui64 = 0;
        ixgbe_ks->tx_reschedule.value.ui64 = 0;
        for (i = 0; i < ixgbe->num_tx_rings; i++) {
                ixgbe_ks->tx_overload.value.ui64 +=
                    ixgbe->tx_rings[i].stat_overload;
                ixgbe_ks->tx_fail_no_tbd.value.ui64 +=
                    ixgbe->tx_rings[i].stat_fail_no_tbd;
                ixgbe_ks->tx_fail_no_tcb.value.ui64 +=
                    ixgbe->tx_rings[i].stat_fail_no_tcb;
                ixgbe_ks->tx_fail_dma_bind.value.ui64 +=
                    ixgbe->tx_rings[i].stat_fail_dma_bind;
                ixgbe_ks->tx_reschedule.value.ui64 +=
                    ixgbe->tx_rings[i].stat_reschedule;
        }
#endif

        /*
         * Hardware calculated statistics.
         */
        ixgbe_ks->gprc.value.ui64 = 0;
        ixgbe_ks->gptc.value.ui64 = 0;
        ixgbe_ks->tor.value.ui64 = 0;
        ixgbe_ks->tot.value.ui64 = 0;
        for (i = 0; i < 16; i++) {
                ixgbe_ks->qprc[i].value.ui64 +=
                    IXGBE_READ_REG(hw, IXGBE_QPRC(i));
                ixgbe_ks->gprc.value.ui64 += ixgbe_ks->qprc[i].value.ui64;
                ixgbe_ks->qptc[i].value.ui64 +=
                    IXGBE_READ_REG(hw, IXGBE_QPTC(i));
                ixgbe_ks->gptc.value.ui64 += ixgbe_ks->qptc[i].value.ui64;
                ixgbe_ks->qbrc[i].value.ui64 +=
                    IXGBE_READ_REG(hw, IXGBE_QBRC(i));
                ixgbe_ks->tor.value.ui64 += ixgbe_ks->qbrc[i].value.ui64;
                switch (hw->mac.type) {
                case ixgbe_mac_82598EB:
                        ixgbe_ks->qbtc[i].value.ui64 +=
                            IXGBE_READ_REG(hw, IXGBE_QBTC(i));
                        break;

                case ixgbe_mac_82599EB:
                case ixgbe_mac_X540:
                        ixgbe_ks->qbtc[i].value.ui64 +=
                            IXGBE_READ_REG(hw, IXGBE_QBTC_L(i));
                        ixgbe_ks->qbtc[i].value.ui64 +=
                            ((uint64_t)((IXGBE_READ_REG(hw,
                            IXGBE_QBTC_H(i))) & 0xF) << 32);
                        break;

                default:
                        break;
                }
                ixgbe_ks->tot.value.ui64 += ixgbe_ks->qbtc[i].value.ui64;
        }
```

```
128         /*
129          * This is a Workaround:
130          * Currently h/w GORCH, GOTCH, TORH registers are not
131          * correctly implemented. We found that the values in
132          * these registers are same as those in corresponding
133          * *L registers (i.e. GORCL, GOTCL, and TORL). Here the
134          * gor and got stat data will not be retrieved through
135          * GORC{H/L} and GOTC{H/L} registers but be obtained by
136          * simply assigning tor/tot stat data, so the gor/got
137          * stat data will not be accurate.
138          */
139         ixgbe_ks->gor.value.ui64 = ixgbe_ks->tor.value.ui64;
140         ixgbe_ks->got.value.ui64 = ixgbe_ks->tot.value.ui64;

142         ixgbe_ks->prc64.value.ul += IXGBE_READ_REG(hw, IXGBE_PRC64);
143         ixgbe_ks->prc127.value.ul += IXGBE_READ_REG(hw, IXGBE_PRC127);
144         ixgbe_ks->prc255.value.ul += IXGBE_READ_REG(hw, IXGBE_PRC255);
145         ixgbe_ks->prc511.value.ul += IXGBE_READ_REG(hw, IXGBE_PRC511);
146         ixgbe_ks->prc1023.value.ul += IXGBE_READ_REG(hw, IXGBE_PRC1023);
147         ixgbe_ks->prc1522.value.ul += IXGBE_READ_REG(hw, IXGBE_PRC1522);
148         ixgbe_ks->ptc64.value.ul += IXGBE_READ_REG(hw, IXGBE_PTC64);
149         ixgbe_ks->ptc127.value.ul += IXGBE_READ_REG(hw, IXGBE_PTC127);
150         ixgbe_ks->ptc255.value.ul += IXGBE_READ_REG(hw, IXGBE_PTC255);
151         ixgbe_ks->ptc511.value.ul += IXGBE_READ_REG(hw, IXGBE_PTC511);
152         ixgbe_ks->ptc1023.value.ul += IXGBE_READ_REG(hw, IXGBE_PTC1023);
153         ixgbe_ks->ptc1522.value.ul += IXGBE_READ_REG(hw, IXGBE_PTC1522);

155         ixgbe_ks->mspdc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_MSPDC);
156         for (i = 0; i < 8; i++)
157                 ixgbe_ks->mpc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_MPC(i));
158         ixgbe_ks->mlfc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_MLFC);
159         ixgbe_ks->mrfc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_MRFC);
160         ixgbe_ks->rlec.value.ui64 += IXGBE_READ_REG(hw, IXGBE_RLEC);
161         ixgbe_ks->lxontxc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_LXONTXC);
162         switch (hw->mac.type) {
163         case ixgbe_mac_82598EB:
164                 ixgbe_ks->lxonrxc.value.ui64 += IXGBE_READ_REG(hw,
165                     IXGBE_LXONRXC);
166                 break;

168         case ixgbe_mac_82599EB:
169         case ixgbe_mac_X540:
170                 ixgbe_ks->lxonrxc.value.ui64 += IXGBE_READ_REG(hw,
171                     IXGBE_LXONRXCNT);
172                 break;

174         default:
175                 break;
176         }
177         ixgbe_ks->lxofftxc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_LXOFFTXC);
178         switch (hw->mac.type) {
179         case ixgbe_mac_82598EB:
180                 ixgbe_ks->lxoffrxc.value.ui64 += IXGBE_READ_REG(hw,
181                     IXGBE_LXOFFRXC);
182                 break;

184         case ixgbe_mac_82599EB:
185         case ixgbe_mac_X540:
186                 ixgbe_ks->lxoffrxc.value.ui64 += IXGBE_READ_REG(hw,
187                     IXGBE_LXOFFRXCNT);
188                 break;

190         default:
191                 break;
192         }
193         ixgbe_ks->ruc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_RUC);
```



```
194         ixgbe_ks->rfc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_RFC);
195         ixgbe_ks->roc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_ROC);
196         ixgbe_ks->rjc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_RJC);

198         mutex_exit(&ixgbe->gen_lock);

200         if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) != DDI_FM_OK)
201                 ddi_fm_service_impact(ixgbe->dip, DDI_SERVICE_UNAFFECTED);

203         return (0);
204 }
```

*unchanged_portion_omitted_*

```
469 /*
470  * Retrieve a value for one of the statistics.
471  */
472 int
473 ixgbe_m_stat(void *arg, uint_t stat, uint64_t *val)
474 {
475         ixgbe_t *ixgbe = (ixgbe_t *)arg;
476         struct ixgbe_hw *hw = &ixgbe->hw;
477         ixgbe_stat_t *ixgbe_ks;
478         int i;

480         ixgbe_ks = (ixgbe_stat_t *)ixgbe->ixgbe_ks->ks_data;

482         mutex_enter(&ixgbe->gen_lock);

484         if (ixgbe->ixgbe_state & IXGBE_SUSPENDED) {
485                 mutex_exit(&ixgbe->gen_lock);
486                 return (ECANCELED);
487         }

489         switch (stat) {
490         case MAC_STAT_IFSPEED:
491                 *val = ixgbe->link_speed * 1000000ull;
492                 break;

494         case MAC_STAT_MULTIRCV:
495                 ixgbe_ks->mprc.value.ui64 +=
496                     IXGBE_READ_REG(hw, IXGBE_MPRC);
497                 *val = ixgbe_ks->mprc.value.ui64;
498                 break;

500         case MAC_STAT_BRDCSTRCV:
501                 ixgbe_ks->bprc.value.ui64 +=
502                     IXGBE_READ_REG(hw, IXGBE_BPRC);
503                 *val = ixgbe_ks->bprc.value.ui64;
504                 break;

506         case MAC_STAT_MULTIXMT:
507                 ixgbe_ks->mptc.value.ui64 +=
508                     IXGBE_READ_REG(hw, IXGBE_MPTC);
509                 *val = ixgbe_ks->mptc.value.ui64;
510                 break;

512         case MAC_STAT_BRDCSTXMT:
513                 ixgbe_ks->bptc.value.ui64 +=
514                     IXGBE_READ_REG(hw, IXGBE_BPTC);
515                 *val = ixgbe_ks->bptc.value.ui64;
516                 break;

518         case MAC_STAT_NORCVBUF:
519                 for (i = 0; i < 8; i++) {
520                         ixgbe_ks->rnbc.value.ui64 +=
521                             IXGBE_READ_REG(hw, IXGBE_RNBC(i));
```

```
522                 }
523                 *val = ixgbe_ks->rnbc.value.ui64;
524                 break;

526         case MAC_STAT_IERRORS:
527                 ixgbe_ks->crcerrs.value.ui64 +=
528                     IXGBE_READ_REG(hw, IXGBE_CRCERRS);
529                 ixgbe_ks->illerrc.value.ui64 +=
530                     IXGBE_READ_REG(hw, IXGBE_ILLERRC);
531                 ixgbe_ks->errbc.value.ui64 +=
532                     IXGBE_READ_REG(hw, IXGBE_ERRBC);
533                 ixgbe_ks->rlec.value.ui64 +=
534                     IXGBE_READ_REG(hw, IXGBE_RLEC);
535                 *val = ixgbe_ks->crcerrs.value.ui64 +
536                     ixgbe_ks->illerrc.value.ui64 +
537                     ixgbe_ks->errbc.value.ui64 +
538                     ixgbe_ks->rlec.value.ui64;
539                 break;

541         case MAC_STAT_RBYTES:
542                 ixgbe_ks->tor.value.ui64 = 0;
543                 for (i = 0; i < 16; i++) {
544                         ixgbe_ks->qbrc[i].value.ui64 +=
545                             IXGBE_READ_REG(hw, IXGBE_QBRC(i));
546                         ixgbe_ks->tor.value.ui64 +=
547                             ixgbe_ks->qbrc[i].value.ui64;
548                 }
549                 *val = ixgbe_ks->tor.value.ui64;
550                 break;

552         case MAC_STAT_OBYTES:
553                 ixgbe_ks->tot.value.ui64 = 0;
554                 for (i = 0; i < 16; i++) {
555                         switch (hw->mac.type) {
556                         case ixgbe_mac_82598EB:
557                                 ixgbe_ks->qbtc[i].value.ui64 +=
558                                     IXGBE_READ_REG(hw, IXGBE_QBTC(i));
559                                 break;

561                         case ixgbe_mac_82599EB:
562                         case ixgbe_mac_X540:
563                                 ixgbe_ks->qbtc[i].value.ui64 +=
564                                     IXGBE_READ_REG(hw, IXGBE_QBTC_L(i));
565                                 ixgbe_ks->qbtc[i].value.ui64 +=
566                                     ((uint64_t)((IXGBE_READ_REG(hw,
567                                     IXGBE_QBTC_H(i))) & 0xF) << 32);
568                                 break;

570                         default:
571                                 break;
572                         }
573                         ixgbe_ks->tot.value.ui64 +=
574                             ixgbe_ks->qbtc[i].value.ui64;
575                 }
576                 *val = ixgbe_ks->tot.value.ui64;
577                 break;

579         case MAC_STAT_IPACKETS:
580                 ixgbe_ks->tpr.value.ui64 +=
581                     IXGBE_READ_REG(hw, IXGBE_TPR);
582                 *val = ixgbe_ks->tpr.value.ui64;
583                 break;

585         case MAC_STAT_OPACKETS:
586                 ixgbe_ks->tpt.value.ui64 +=
587                     IXGBE_READ_REG(hw, IXGBE_TPT);
```



```
588                 *val = ixgbe_ks->tpt.value.ui64;
589                 break;

591         /* RFC 1643 stats */
592         case ETHER_STAT_FCS_ERRORS:
593                 ixgbe_ks->crcerrs.value.ui64 +=
594                     IXGBE_READ_REG(hw, IXGBE_CRCERRS);
595                 *val = ixgbe_ks->crcerrs.value.ui64;
596                 break;

598         case ETHER_STAT_TOOLONG_ERRORS:
599                 ixgbe_ks->roc.value.ui64 +=
600                     IXGBE_READ_REG(hw, IXGBE_ROC);
601                 *val = ixgbe_ks->roc.value.ui64;
602                 break;

604         case ETHER_STAT_MACRCV_ERRORS:
605                 ixgbe_ks->crcerrs.value.ui64 +=
606                     IXGBE_READ_REG(hw, IXGBE_CRCERRS);
607                 ixgbe_ks->illerrc.value.ui64 +=
608                     IXGBE_READ_REG(hw, IXGBE_ILLERRC);
609                 ixgbe_ks->errbc.value.ui64 +=
610                     IXGBE_READ_REG(hw, IXGBE_ERRBC);
611                 ixgbe_ks->rlec.value.ui64 +=
612                     IXGBE_READ_REG(hw, IXGBE_RLEC);
613                 *val = ixgbe_ks->crcerrs.value.ui64 +
614                     ixgbe_ks->illerrc.value.ui64 +
615                     ixgbe_ks->errbc.value.ui64 +
616                     ixgbe_ks->rlec.value.ui64;
617                 break;

619         /* MII/GMII stats */
620         case ETHER_STAT_XCVR_ADDR:
621                 /* The Internal PHY's MDI address for each MAC is 1 */
622                 *val = 1;
623                 break;

625         case ETHER_STAT_XCVR_ID:
626                 *val = hw->phy.id;
627                 break;

629         case ETHER_STAT_XCVR_INUSE:
630                 switch (ixgbe->link_speed) {
631                 case IXGBE_LINK_SPEED_1GB_FULL:
632                         *val =
633                             (hw->phy.media_type == ixgbe_media_type_copper) ?
634                             XCVR_1000T : XCVR_1000X;
635                         break;
636                 case IXGBE_LINK_SPEED_100_FULL:
637                         *val = (hw->phy.media_type == ixgbe_media_type_copper) ?
638                             XCVR_100T2 : XCVR_100X;
639                         break;
640                 default:
641                         *val = XCVR_NONE;
642                         break;
643                 }
644                 break;

646         case ETHER_STAT_CAP_10GFDX:
647                 *val = 1;
648                 break;

650         case ETHER_STAT_CAP_1000FDX:
651                 *val = 1;
652                 break;
```

```
654         case ETHER_STAT_CAP_100FDX:
655                 *val = 1;
656                 break;

658         case ETHER_STAT_CAP_ASMPAUSE:
659                 *val = ixgbe->param_asym_pause_cap;
660                 break;

662         case ETHER_STAT_CAP_PAUSE:
663                 *val = ixgbe->param_pause_cap;
664                 break;

666         case ETHER_STAT_CAP_AUTONEG:
667                 *val = 1;
668                 break;

670         case ETHER_STAT_ADV_CAP_10GFDX:
671                 *val = ixgbe->param_adv_10000fdx_cap;
672                 break;

674         case ETHER_STAT_ADV_CAP_1000FDX:
675                 *val = ixgbe->param_adv_1000fdx_cap;
676                 break;

678         case ETHER_STAT_ADV_CAP_100FDX:
679                 *val = ixgbe->param_adv_100fdx_cap;
680                 break;

682         case ETHER_STAT_ADV_CAP_ASMPAUSE:
683                 *val = ixgbe->param_adv_asym_pause_cap;
684                 break;

686         case ETHER_STAT_ADV_CAP_PAUSE:
687                 *val = ixgbe->param_adv_pause_cap;
688                 break;

690         case ETHER_STAT_ADV_CAP_AUTONEG:
691                 *val = ixgbe->param_adv_autoneg_cap;
692                 break;

694         case ETHER_STAT_LP_CAP_10GFDX:
695                 *val = ixgbe->param_lp_10000fdx_cap;
696                 break;

698         case ETHER_STAT_LP_CAP_1000FDX:
699                 *val = ixgbe->param_lp_1000fdx_cap;
700                 break;

702         case ETHER_STAT_LP_CAP_100FDX:
703                 *val = ixgbe->param_lp_100fdx_cap;
704                 break;

706         case ETHER_STAT_LP_CAP_ASMPAUSE:
707                 *val = ixgbe->param_lp_asym_pause_cap;
708                 break;

710         case ETHER_STAT_LP_CAP_PAUSE:
711                 *val = ixgbe->param_lp_pause_cap;
712                 break;

714         case ETHER_STAT_LP_CAP_AUTONEG:
715                 *val = ixgbe->param_lp_autoneg_cap;
716                 break;

718         case ETHER_STAT_LINK_ASMPAUSE:
719                 *val = ixgbe->param_asym_pause_cap;
```



```
720                 break;

722         case ETHER_STAT_LINK_PAUSE:
723                 *val = ixgbe->param_pause_cap;
724                 break;

726         case ETHER_STAT_LINK_AUTONEG:
727                 *val = ixgbe->param_adv_autoneg_cap;
728                 break;

730         case ETHER_STAT_LINK_DUPLEX:
731                 *val = ixgbe->link_duplex;
732                 break;

734         case ETHER_STAT_TOOSHORT_ERRORS:
735                 ixgbe_ks->ruc.value.ui64 +=
736                     IXGBE_READ_REG(hw, IXGBE_RUC);
737                 *val = ixgbe_ks->ruc.value.ui64;
738                 break;

740         case ETHER_STAT_CAP_REMFAULT:
741                 *val = ixgbe->param_rem_fault;
742                 break;

744         case ETHER_STAT_ADV_REMFAULT:
745                 *val = ixgbe->param_adv_rem_fault;
746                 break;

748         case ETHER_STAT_LP_REMFAULT:
749                 *val = ixgbe->param_lp_rem_fault;
750                 break;

752         case ETHER_STAT_JABBER_ERRORS:
753                 ixgbe_ks->rjc.value.ui64 +=
754                     IXGBE_READ_REG(hw, IXGBE_RJC);
755                 *val = ixgbe_ks->rjc.value.ui64;
756                 break;

758         default:
759                 mutex_exit(&ixgbe->gen_lock);
760                 return (ENOTSUP);
761         }

763         mutex_exit(&ixgbe->gen_lock);

765         if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) != DDI_FM_OK) {
766                 ddi_fm_service_impact(ixgbe->dip, DDI_SERVICE_DEGRADED);
767                 return (EIO);
768         }

770         return (0);
771 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   41329 Thu Jul 12 12:22:40 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_tx.c
XXXX Intel X540 support
**********************************************************
______unchanged_portion_omitted_

 961 /*
 962  * ixgbe_tx_fill_ring
 963  *
 964  * Fill the tx descriptor ring with the data
 965  */
 966 static int
 967 ixgbe_tx_fill_ring(ixgbe_tx_ring_t *tx_ring, link_list_t *pending_list,
 968     ixgbe_tx_context_t *ctx, size_t mbsize)
 969 {
 970         struct ixgbe_hw *hw = &tx_ring->ixgbe->hw;
 971         boolean_t load_context;
 972         uint32_t index, tcb_index, desc_num;
 973         union ixgbe_adv_tx_desc *tbd, *first_tbd;
 974         tx_control_block_t *tcb, *first_tcb;
 975         uint32_t hcksum_flags;
 976         int i;

 978         ASSERT(mutex_owned(&tx_ring->tx_lock));

 980         tbd = NULL;
 981         first_tbd = NULL;
 982         first_tcb = NULL;
 983         desc_num = 0;
 984         hcksum_flags = 0;
 985         load_context = B_FALSE;

 987         /*
 988          * Get the index of the first tx descriptor that will be filled,
 989          * and the index of the first work list item that will be attached
 990          * with the first used tx control block in the pending list.
 991          * Note: the two indexes are the same.
 992          */
 993         index = tx_ring->tbd_tail;
 994         tcb_index = tx_ring->tbd_tail;

 996         if (ctx != NULL) {
 997                 hcksum_flags = ctx->hcksum_flags;

 999                 /*
1000                  * Check if a new context descriptor is needed for this packet
1001                  */
1002                 load_context = ixgbe_check_context(tx_ring, ctx);

1004                 if (load_context) {
1005                         tbd = &tx_ring->tbd_ring[index];

1007                         /*
1008                          * Fill the context descriptor with the
1009                          * hardware checksum offload informations.
1010                          */
1011                         ixgbe_fill_context(
1012                             (struct ixgbe_adv_tx_context_desc *)tbd, ctx);

1014                         index = NEXT_INDEX(index, 1, tx_ring->ring_size);
1015                         desc_num++;

1017                         /*
1018                          * Store the checksum context data if
1019                          * a new context descriptor is added
```



```
1020                          */
1021                         tx_ring->tx_context = *ctx;
1022                 }
1023         }

1025         first_tbd = &tx_ring->tbd_ring[index];

1027         /*
1028          * Fill tx data descriptors with the data saved in the pending list.
1029          * The tx control blocks in the pending list are added to the work list
1030          * at the same time.
1031          *
1032          * The work list is strictly 1:1 corresponding to the descriptor ring.
1033          * One item of the work list corresponds to one tx descriptor. Because
1034          * one tx control block can span multiple tx descriptors, the tx
1035          * control block will be added to the first work list item that
1036          * corresponds to the first tx descriptor generated from that tx
1037          * control block.
1038          */
1039         tcb = (tx_control_block_t *)LIST_POP_HEAD(pending_list);
1040         first_tcb = tcb;
1041         while (tcb != NULL) {

1043                 for (i = 0; i < tcb->desc_num; i++) {
1044                         tbd = &tx_ring->tbd_ring[index];

1046                         tbd->read.buffer_addr = tcb->desc[i].address;
1047                         tbd->read.cmd_type_len = tcb->desc[i].length;

1049                         tbd->read.cmd_type_len |= IXGBE_ADVTXD_DCMD_DEXT
1050                             | IXGBE_ADVTXD_DTYP_DATA;

1052                         tbd->read.olinfo_status = 0;

1054                         index = NEXT_INDEX(index, 1, tx_ring->ring_size);
1055                         desc_num++;
1056                 }

1058                 /*
1059                  * Add the tx control block to the work list
1060                  */
1061                 ASSERT(tx_ring->work_list[tcb_index] == NULL);
1062                 tx_ring->work_list[tcb_index] = tcb;

1064                 tcb_index = index;
1065                 tcb = (tx_control_block_t *)LIST_POP_HEAD(pending_list);
1066         }

1068         if (load_context) {
1069                 /*
1070                  * Count the context descriptor for
1071                  * the first tx control block.
1072                  */
1073                 first_tcb->desc_num++;
1074         }
1075         first_tcb->last_index = PREV_INDEX(index, 1, tx_ring->ring_size);

1077         /*
1078          * The Insert Ethernet CRC (IFCS) bit and the checksum fields are only
1079          * valid in the first descriptor of the packet.
1080          * Setting paylen in every first_tbd for all parts.
1081          * 82599 and X540 require the packet length in paylen field with or
1082          * without LSO and 82598 will ignore it in non-LSO mode.
1081          * 82599 requires the packet length in paylen field with or without
1082          * LSO and 82598 will ignore it in non-LSO mode.
1083          */
```

```
1084         ASSERT(first_tbd != NULL);
1085         first_tbd->read.cmd_type_len |= IXGBE_ADVTXD_DCMD_IFCS;

1087         switch (hw->mac.type) {
1088         case ixgbe_mac_82598EB:
1089                 if (ctx != NULL && ctx->lso_flag) {
1090                         first_tbd->read.cmd_type_len |= IXGBE_ADVTXD_DCMD_TSE;
1091                         first_tbd->read.olinfo_status |=
1092                             (mbsize - ctx->mac_hdr_len - ctx->ip_hdr_len
1093                             - ctx->l4_hdr_len) << IXGBE_ADVTXD_PAYLEN_SHIFT;
1094                 }
1095                 break;

1097         case ixgbe_mac_82599EB:
1098         case ixgbe_mac_X540:
1099                 if (ctx != NULL && ctx->lso_flag) {
1100                         first_tbd->read.cmd_type_len |= IXGBE_ADVTXD_DCMD_TSE;
1101                         first_tbd->read.olinfo_status |=
1102                             (mbsize - ctx->mac_hdr_len - ctx->ip_hdr_len
1103                             - ctx->l4_hdr_len) << IXGBE_ADVTXD_PAYLEN_SHIFT;
1104                 } else {
1105                         first_tbd->read.olinfo_status |=
1106                             (mbsize << IXGBE_ADVTXD_PAYLEN_SHIFT);
1107                 }
1108                 break;

1110         default:
1111                 break;
1112         }

1114         /* Set hardware checksum bits */
1115         if (hcksum_flags != 0) {
1116                 if (hcksum_flags & HCK_IPV4_HDRCKSUM)
1117                         first_tbd->read.olinfo_status |=
1118                             IXGBE_ADVTXD_POPTS_IXSM;
1119                 if (hcksum_flags & HCK_PARTIALCKSUM)
1120                         first_tbd->read.olinfo_status |=
1121                             IXGBE_ADVTXD_POPTS_TXSM;
1122         }

1124         /*
1125          * The last descriptor of packet needs End Of Packet (EOP),
1126          * and Report Status (RS) bits set
1127          */
1128         ASSERT(tbd != NULL);
1129         tbd->read.cmd_type_len |=
1130             IXGBE_ADVTXD_DCMD_EOP | IXGBE_ADVTXD_DCMD_RS;

1132         /*
1133          * Sync the DMA buffer of the tx descriptor ring
1134          */
1135         DMA_SYNC(&tx_ring->tbd_area, DDI_DMA_SYNC_FORDEV);

1137         /*
1138          * Update the number of the free tx descriptors.
1139          * The mutual exclusion between the transmission and the recycling
1140          * (for the tx descriptor ring and the work list) is implemented
1141          * with the atomic operation on the number of the free tx descriptors.
1142          *
1143          * Note: we should always decrement the counter tbd_free before
1144          * advancing the hardware TDT pointer to avoid the race condition -
1145          * before the counter tbd_free is decremented, the transmit of the
1146          * tx descriptors has done and the counter tbd_free is increased by
1147          * the tx recycling.
1148          */
1149         i = ixgbe_atomic_reserve(&tx_ring->tbd_free, desc_num);
```



```
1150         ASSERT(i >= 0);

1152         tx_ring->tbd_tail = index;

1154         /*
1155          * Advance the hardware TDT pointer of the tx descriptor ring
1156          */
1157         IXGBE_WRITE_REG(hw, IXGBE_TDT(tx_ring->index), index);

1159         if (ixgbe_check_acc_handle(tx_ring->ixgbe->osdep.reg_handle) !=
1160             DDI_FM_OK) {
1161                 ddi_fm_service_impact(tx_ring->ixgbe->dip,
1162                     DDI_SERVICE_DEGRADED);
1163                 atomic_or_32(&tx_ring->ixgbe->ixgbe_state, IXGBE_ERROR);
1164         }

1166         return (desc_num);
1167 }
```

*unchanged_portion_omitted_*

```
**********************************************************
  124904 Thu Jul 12 12:22:40 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_type.h
XXXX Intel X540 support
**********************************************************
```

```
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_type.h,v 1.14 2012/07/05 20:51:44 jfv Exp $*

#ifndef _IXGBE_TYPE_H_
#define _IXGBE_TYPE_H_

#include "ixgbe_osdep.h"


/* Vendor ID */
#define IXGBE_INTEL_VENDOR_ID                   0x8086

/* Device IDs */
#define IXGBE_DEV_ID_82598                      0x10B6
#define IXGBE_DEV_ID_82598_BX                   0x1508
#define IXGBE_DEV_ID_82598AF_DUAL_PORT          0x10C6
#define IXGBE_DEV_ID_82598AF_SINGLE_PORT        0x10C7
#define IXGBE_DEV_ID_82598AT                    0x10C8
#define IXGBE_DEV_ID_82598AT2                   0x150B
#define IXGBE_DEV_ID_82598EB_SFP_LOM            0x10DB
#define IXGBE_DEV_ID_82598EB_CX4                0x10DD
#define IXGBE_DEV_ID_82598_CX4_DUAL_PORT        0x10EC
#define IXGBE_DEV_ID_82598_DA_DUAL_PORT         0x10F1
#define IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM      0x10E1
#define IXGBE_DEV_ID_82598EB_XF_LR              0x10F4
#define IXGBE_DEV_ID_82599_KX4                  0x10F7
#define IXGBE_DEV_ID_82599_KX4_MEZZ             0x1514
#define IXGBE_DEV_ID_82599_KR                   0x1517
#define IXGBE_DEV_ID_82599_COMBO_BACKPLANE      0x10F8
#define IXGBE_SUBDEV_ID_82599_KX4_KR_MEZZ       0x000C
```



```
#define IXGBE_DEV_ID_82599_CX4                  0x10F9
#define IXGBE_DEV_ID_82599_SFP                  0x10FB
#define IXGBE_SUBDEV_ID_82599_SFP               0x11A9
#define IXGBE_SUBDEV_ID_82599_560FLR            0x17D0
#define IXGBE_DEV_ID_82599_BACKPLANE_FCOE       0x152A
#define IXGBE_DEV_ID_82599_SFP_FCOE             0x1529
#define IXGBE_DEV_ID_82599_SFP_EM               0x1507
#define IXGBE_DEV_ID_82599_SFP_SF2              0x154D
#define IXGBE_DEV_ID_82599EN_SFP                0x1557
#define IXGBE_DEV_ID_82599_XAUI_LOM             0x10FC
#define IXGBE_DEV_ID_82599_T3_LOM               0x151C
#define IXGBE_DEV_ID_82599_VF                   0x10ED
#define IXGBE_DEV_ID_X540_VF                    0x1515
#define IXGBE_DEV_ID_X540T                      0x1528
#define IXGBE_DEV_ID_X540T1                     0x1560

/* General Registers */
#define IXGBE_CTRL              0x00000
#define IXGBE_STATUS            0x00008
#define IXGBE_CTRL_EXT          0x00018
#define IXGBE_ESDP              0x00020
#define IXGBE_EODSDP            0x00028
#define IXGBE_I2CCTL            0x00028
#define IXGBE_PHY_GPIO          0x00028
#define IXGBE_MAC_GPIO          0x00030
#define IXGBE_PHYINT_STATUS0    0x00100
#define IXGBE_PHYINT_STATUS1    0x00104
#define IXGBE_PHYINT_STATUS2    0x00108
#define IXGBE_LEDCTL            0x00200
#define IXGBE_FRTIMER           0x00048
#define IXGBE_TCPTIMER          0x0004C
#define IXGBE_CORESPARE         0x00600
#define IXGBE_EXVET             0x05078

/* NVM Registers */
#define IXGBE_EEC       0x10010
#define IXGBE_EERD      0x10014
#define IXGBE_EEWR      0x10018
#define IXGBE_FLA       0x1001C
#define IXGBE_EEMNGCTL  0x10110
#define IXGBE_EEMNGDATA 0x10114
#define IXGBE_FLMNGCTL  0x10118
#define IXGBE_FLMNGDATA 0x1011C
#define IXGBE_FLMNGCNT  0x10120
#define IXGBE_FLOP      0x1013C
#define IXGBE_GRC       0x10200
#define IXGBE_SRAMREL   0x10210
#define IXGBE_PHYDBG    0x10218

/* General Receive Control */
#define IXGBE_GRC_MNG   0x00000001 /* Manageability Enable */
#define IXGBE_GRC_APME  0x00000002 /* APM enabled in EEPROM */

#define IXGBE_VPDDIAG0  0x10204
#define IXGBE_VPDDIAG1  0x10208

/* I2CCTL Bit Masks */
#define IXGBE_I2C_CLK_IN        0x00000001
#define IXGBE_I2C_CLK_OUT       0x00000002
#define IXGBE_I2C_DATA_IN       0x00000004
#define IXGBE_I2C_DATA_OUT      0x00000008
#define IXGBE_I2C_CLOCK_STRETCHING_TIMEOUT      500


/* Interrupt Registers */
#define IXGBE_EICR              0x00800
```

```
#define IXGBE_EICS              0x00808
#define IXGBE_EIMS              0x00880
#define IXGBE_EIMC              0x00888
#define IXGBE_EIAC              0x00810
#define IXGBE_EIAM              0x00890
#define IXGBE_EICS_EX(_i)       (0x00A90 + (_i) * 4)
#define IXGBE_EIMS_EX(_i)       (0x00AA0 + (_i) * 4)
#define IXGBE_EIMC_EX(_i)       (0x00AB0 + (_i) * 4)
#define IXGBE_EIAM_EX(_i)       (0x00AD0 + (_i) * 4)
/* 82599 EITR is only 12 bits, with the lower 3 always zero */
/*
 * 82598 EITR is 16 bits but set the limits based on the max
 * supported by all ixgbe hardware
 */
#define IXGBE_MAX_INT_RATE      488281
#define IXGBE_MIN_INT_RATE      956
#define IXGBE_MAX_EITR          0x00000FF8
#define IXGBE_MIN_EITR          8
#define IXGBE_EITR(_i)          (((_i) <= 23) ? (0x00820 + ((_i) * 4)) : \
                                 (0x012300 + (((_i) - 24) * 4)))
#define IXGBE_EITR_ITR_INT_MASK 0x00000FF8
#define IXGBE_EITR_LLI_MOD      0x00008000
#define IXGBE_EITR_CNT_WDIS     0x80000000
#define IXGBE_IVAR(_i)          (0x00900 + ((_i) * 4)) /* 24 at 0x900-0x960 */
#define IXGBE_IVAR_MISC         0x00A00 /* misc MSI-X interrupt causes */
#define IXGBE_EITRSEL           0x00894
#define IXGBE_MSIXT             0x00000 /* MSI-X Table. 0x0000 - 0x01C */
#define IXGBE_MSIXPBA           0x02000 /* MSI-X Pending bit array */
#define IXGBE_PBACL(_i) (((_i) == 0) ? (0x11068) : (0x110C0 + ((_i) * 4)))
#define IXGBE_GPIE              0x00898

/* Flow Control Registers */
#define IXGBE_FCADBUL           0x03210
#define IXGBE_FCADBUH           0x03214
#define IXGBE_FCAMACL           0x04328
#define IXGBE_FCAMACH           0x0432C
#define IXGBE_FCRTH_82599(_i)   (0x03260 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_FCRTL_82599(_i)   (0x03220 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_PFCTOP            0x03008
#define IXGBE_FCTTV(_i)         (0x03200 + ((_i) * 4)) /* 4 of these (0-3) */
#define IXGBE_FCRTL(_i)         (0x03220 + ((_i) * 8)) /* 8 of these (0-7) */
#define IXGBE_FCRTH(_i)         (0x03260 + ((_i) * 8)) /* 8 of these (0-7) */
#define IXGBE_FCRTV             0x032A0
#define IXGBE_FCCFG             0x03D00
#define IXGBE_TFCS              0x0CE00

/* Receive DMA Registers */
#define IXGBE_RDBAL(_i) (((_i) < 64) ? (0x01000 + ((_i) * 0x40)) : \
                         (0x0D000 + (((_i) - 64) * 0x40)))
                         (0x0D000 + ((_i - 64) * 0x40)))
#define IXGBE_RDBAH(_i) (((_i) < 64) ? (0x01004 + ((_i) * 0x40)) : \
                         (0x0D004 + (((_i) - 64) * 0x40)))
                         (0x0D004 + ((_i - 64) * 0x40)))
#define IXGBE_RDLEN(_i) (((_i) < 64) ? (0x01008 + ((_i) * 0x40)) : \
                         (0x0D008 + (((_i) - 64) * 0x40)))
                         (0x0D008 + ((_i - 64) * 0x40)))
#define IXGBE_RDH(_i)   (((_i) < 64) ? (0x01010 + ((_i) * 0x40)) : \
                         (0x0D010 + (((_i) - 64) * 0x40)))
                         (0x0D010 + ((_i - 64) * 0x40)))
#define IXGBE_RDT(_i)   (((_i) < 64) ? (0x01018 + ((_i) * 0x40)) : \
                         (0x0D018 + (((_i) - 64) * 0x40)))
                         (0x0D018 + ((_i - 64) * 0x40)))
#define IXGBE_RXDCTL(_i)        (((_i) < 64) ? (0x01028 + ((_i) * 0x40)) : \
                                 (0x0D028 + (((_i) - 64) * 0x40)))
                          (0x0D028 + ((_i - 64) * 0x40)))
#define IXGBE_RSCCTL(_i)        (((_i) < 64) ? (0x0102C + ((_i) * 0x40)) : \
```



```
                                 (0x0D02C + (((_i) - 64) * 0x40)))
                           (0x0D02C + ((_i - 64) * 0x40)))
#define IXGBE_RSCDBU    0x03028
#define IXGBE_RDDCC     0x02F20
#define IXGBE_RXMEMWRAP 0x03190
#define IXGBE_STARCTRL  0x03024
/*
 * Split and Replication Receive Control Registers
 * 00-15 : 0x02100 + n*4
 * 16-64 : 0x01014 + n*0x40
 * 64-127: 0x0D014 + (n-64)*0x40
 */
#define IXGBE_SRRCTL(_i)        (((_i) <= 15) ? (0x02100 + ((_i) * 4)) : \
                                 (((_i) < 64) ? (0x01014 + ((_i) * 0x40)) : \
                                 (0x0D014 + (((_i) - 64) * 0x40))))
                           (0x0D014 + ((_i - 64) * 0x40))))
/*
 * Rx DCA Control Register:
 * 00-15 : 0x02200 + n*4
 * 16-64 : 0x0100C + n*0x40
 * 64-127: 0x0D00C + (n-64)*0x40
 */
#define IXGBE_DCA_RXCTRL(_i)    (((_i) <= 15) ? (0x02200 + ((_i) * 4)) : \
                                 (((_i) < 64) ? (0x0100C + ((_i) * 0x40)) : \
                                 (0x0D00C + (((_i) - 64) * 0x40))))
                                 (0x0D00C + ((_i - 64) * 0x40))))
#define IXGBE_RDRXCTL           0x02F00
#define IXGBE_RDRXCTL_RSC_PUSH  0x80
/* 8 of these 0x03C00 - 0x03C1C */
#define IXGBE_RXPBSIZE(_i)      (0x03C00 + ((_i) * 4))
                                                /* 8 of these 0x03C00 - 0x03C1C */
#define IXGBE_RXCTRL            0x03000
#define IXGBE_DROPEN            0x03D04
#define IXGBE_RXPBSIZE_SHIFT    10

/* Receive Registers */
#define IXGBE_RXCSUM            0x05000
#define IXGBE_RFCTL             0x05008
#define IXGBE_DRECCCTL          0x02F08
#define IXGBE_DRECCCTL_DISABLE  0
#define IXGBE_DRECCCTL2         0x02F8C

/* Multicast Table Array - 128 entries */
#define IXGBE_MTA(_i)           (0x05200 + ((_i) * 4))
#define IXGBE_RAL(_i)           (((_i) <= 15) ? (0x05400 + ((_i) * 8)) : \
                                 (0x0A200 + ((_i) * 8)))
#define IXGBE_RAH(_i)           (((_i) <= 15) ? (0x05404 + ((_i) * 8)) : \
                                 (0x0A204 + ((_i) * 8)))
#define IXGBE_MPSAR_LO(_i)      (0x0A600 + ((_i) * 8))
#define IXGBE_MPSAR_HI(_i)      (0x0A604 + ((_i) * 8))
/* Packet split receive type */
#define IXGBE_PSRTYPE(_i)       (((_i) <= 15) ? (0x05480 + ((_i) * 4)) : \
                                 (0x0EA00 + ((_i) * 4)))
/* array of 4096 1-bit vlan filters */
#define IXGBE_VFTA(_i)          (0x0A000 + ((_i) * 4))
/*array of 4096 4-bit vlan vmdq indices */
#define IXGBE_VFTAVIND(_j, _i)  (0x0A200 + ((_j) * 0x200) + ((_i) * 4))
#define IXGBE_FCTRL             0x05080
#define IXGBE_VLNCTRL           0x05088
#define IXGBE_MCSTCTRL          0x05090
#define IXGBE_MRQC              0x05818
#define IXGBE_SAQF(_i)  (0x0E000 + ((_i) * 4)) /* Source Address Queue Filter */
#define IXGBE_DAQF(_i)  (0x0E200 + ((_i) * 4)) /* Dest. Address Queue Filter */
#define IXGBE_SDPQF(_i) (0x0E400 + ((_i) * 4)) /* Src Dest. Addr Queue Filter */
#define IXGBE_FTQF(_i)  (0x0E600 + ((_i) * 4)) /* Five Tuple Queue Filter */
#define IXGBE_ETQF(_i)  (0x05128 + ((_i) * 4)) /* EType Queue Filter */
```

```
250 #define IXGBE_ETQS(_i)  (0x0EC00 + ((_i) * 4)) /* EType Queue Select */
251 #define IXGBE_SYNQF     0x0EC30 /* SYN Packet Queue Filter */
252 #define IXGBE_RQTC      0x0EC70
253 #define IXGBE_MTQC      0x08120
254 #define IXGBE_VLVF(_i)  (0x0F100 + ((_i) * 4))  /* 64 of these (0-63) */
255 #define IXGBE_VLVFB(_i) (0x0F200 + ((_i) * 4))  /* 128 of these (0-127) */
256 #define IXGBE_VMVIR(_i) (0x08000 + ((_i) * 4))  /* 64 of these (0-63) */
257 #define IXGBE_VT_CTL            0x051B0
258 #define IXGBE_PFMAILBOX(_i)     (0x04B00 + (4 * (_i))) /* 64 total */
259 /* 64 Mailboxes, 16 DW each */
260 #define IXGBE_PFMBMEM(_i)       (0x13000 + (64 * (_i)))
261 #define IXGBE_PFMBICR(_i)       (0x00710 + (4 * (_i))) /* 4 total */
262 #define IXGBE_PFMBIMR(_i)       (0x00720 + (4 * (_i))) /* 4 total */
263 #define IXGBE_VFRE(_i)          (0x051E0 + ((_i) * 4))
264 #define IXGBE_VFTE(_i)          (0x08110 + ((_i) * 4))
265 #define IXGBE_VMECM(_i)         (0x08790 + ((_i) * 4))
266 #define IXGBE_QDE               0x2F04
267 #define IXGBE_VMTXSW(_i)        (0x05180 + ((_i) * 4)) /* 2 total */
268 #define IXGBE_VMOLR(_i)         (0x0F000 + ((_i) * 4)) /* 64 total */
269 #define IXGBE_UTA(_i)           (0x0F400 + ((_i) * 4))
270 #define IXGBE_MRCTL(_i)         (0x0F600 + ((_i) * 4))
249 #define IXGBE_VMRCTL(_i)        (0x0F600 + ((_i) * 4))
271 #define IXGBE_VMRVLAN(_i)       (0x0F610 + ((_i) * 4))
272 #define IXGBE_VMRVM(_i)         (0x0F630 + ((_i) * 4))
273 #define IXGBE_L34T_IMIR(_i)     (0x0E800 + ((_i) * 4)) /*128 of these (0-127)*/
274 #define IXGBE_RXFECCERR0        0x051B8
275 #define IXGBE_LLITHRESH         0x0EC90
276 #define IXGBE_IMIR(_i)          (0x05A80 + ((_i) * 4))  /* 8 of these (0-7) */
277 #define IXGBE_IMIREXT(_i)       (0x05AA0 + ((_i) * 4))  /* 8 of these (0-7) */
278 #define IXGBE_IMIRVP            0x05AC0
279 #define IXGBE_VMD_CTL           0x0581C
280 #define IXGBE_RETA(_i)          (0x05C00 + ((_i) * 4))  /* 32 of these (0-31) */
281 #define IXGBE_RSSRK(_i)         (0x05C80 + ((_i) * 4))  /* 10 of these (0-9) */

283 /* Flow Director registers */
284 #define IXGBE_FDIRCTRL  0x0EE00
285 #define IXGBE_FDIRHKEY  0x0EE68
286 #define IXGBE_FDIRSKEY  0x0EE6C
287 #define IXGBE_FDIRDIP4M 0x0EE3C
288 #define IXGBE_FDIRSIP4M 0x0EE40
289 #define IXGBE_FDIRTCPM  0x0EE44
290 #define IXGBE_FDIRUDPM  0x0EE48
291 #define IXGBE_FDIRIP6M  0x0EE74
292 #define IXGBE_FDIRM     0x0EE70

294 /* Flow Director Stats registers */
295 #define IXGBE_FDIRFREE  0x0EE38
296 #define IXGBE_FDIRLEN   0x0EE4C
297 #define IXGBE_FDIRUSTAT 0x0EE50
298 #define IXGBE_FDIRFSTAT 0x0EE54
299 #define IXGBE_FDIRMATCH 0x0EE58
300 #define IXGBE_FDIRMISS  0x0EE5C

302 /* Flow Director Programming registers */
303 #define IXGBE_FDIRSIPv6(_i) (0x0EE0C + ((_i) * 4)) /* 3 of these (0-2) */
304 #define IXGBE_FDIRIPSA  0x0EE18
305 #define IXGBE_FDIRIPDA  0x0EE1C
306 #define IXGBE_FDIRPORT  0x0EE20
307 #define IXGBE_FDIRVLAN  0x0EE24
308 #define IXGBE_FDIRHASH  0x0EE28
309 #define IXGBE_FDIRCMD   0x0EE2C

311 /* Transmit DMA registers */
312 #define IXGBE_TDBAL(_i)         (0x06000 + ((_i) * 0x40)) /* 32 of them (0-31)*/
290 #define IXGBE_TDBAL(_i) (0x06000 + ((_i) * 0x40)) /* 32 of these (0-31)*/
313 #define IXGBE_TDBAH(_i)         (0x06004 + ((_i) * 0x40))
```



```
314 #define IXGBE_TDLEN(_i)         (0x06008 + ((_i) * 0x40))
315 #define IXGBE_TDH(_i)           (0x06010 + ((_i) * 0x40))
316 #define IXGBE_TDT(_i)           (0x06018 + ((_i) * 0x40))
317 #define IXGBE_TXDCTL(_i)        (0x06028 + ((_i) * 0x40))
318 #define IXGBE_TDWBAL(_i)        (0x06038 + ((_i) * 0x40))
319 #define IXGBE_TDWBAH(_i)        (0x0603C + ((_i) * 0x40))
320 #define IXGBE_DTXCTL            0x07E00

322 #define IXGBE_DMATXCTL          0x04A80
323 #define IXGBE_PFVFSPOOF(_i)     (0x08200 + ((_i) * 4)) /* 8 of these 0 - 7 */
324 #define IXGBE_PFDTXGSWC         0x08220
325 #define IXGBE_DTXMXSZRQ         0x08100
326 #define IXGBE_DTXTCPFLGL        0x04A88
327 #define IXGBE_DTXTCPFLGH        0x04A8C
328 #define IXGBE_LBDRPEN           0x0CA00
329 #define IXGBE_TXPBTHRESH(_i)    (0x04950 + ((_i) * 4)) /* 8 of these 0 - 7 */

331 #define IXGBE_DMATXCTL_TE       0x1 /* Transmit Enable */
332 #define IXGBE_DMATXCTL_NS       0x2 /* No Snoop LSO hdr buffer */
333 #define IXGBE_DMATXCTL_GDV      0x8 /* Global Double VLAN */
334 #define IXGBE_DMATXCTL_VT_SHIFT 16  /* VLAN EtherType */

336 #define IXGBE_PFDTXGSWC_VT_LBEN 0x1 /* Local L2 VT switch enable */

338 /* Anti-spoofing defines */
339 #define IXGBE_SPOOF_MACAS_MASK          0xFF
340 #define IXGBE_SPOOF_VLANAS_MASK         0xFF00
341 #define IXGBE_SPOOF_VLANAS_SHIFT        8
342 #define IXGBE_PFVFSPOOF_REG_COUNT       8
343 /* 16 of these (0-15) */
344 #define IXGBE_DCA_TXCTRL(_i)            (0x07200 + ((_i) * 4))
321 #define IXGBE_DCA_TXCTRL(_i)    (0x07200 + ((_i) * 4)) /* 16 of these (0-15) */
345 /* Tx DCA Control register : 128 of these (0-127) */
346 #define IXGBE_DCA_TXCTRL_82599(_i)      (0x0600C + ((_i) * 0x40))
347 #define IXGBE_TIPG                      0x0CB00
348 #define IXGBE_TXPBSIZE(_i)              (0x0CC00 + ((_i) * 4)) /* 8 of these */
349 #define IXGBE_MNGTXMAP                  0x0CD10
350 #define IXGBE_TIPG_FIBER_DEFAULT        3
351 #define IXGBE_TXPBSIZE_SHIFT            10

353 /* Wake up registers */
354 #define IXGBE_WUC       0x05800
355 #define IXGBE_WUFC      0x05808
356 #define IXGBE_WUS       0x05810
357 #define IXGBE_IPAV      0x05838
358 #define IXGBE_IP4AT     0x05840 /* IPv4 table 0x5840-0x5858 */
359 #define IXGBE_IP6AT     0x05880 /* IPv6 table 0x5880-0x588F */

361 #define IXGBE_WUPL      0x05900
362 #define IXGBE_WUPM      0x05A00 /* wake up pkt memory 0x5A00-0x5A7C */
363 #define IXGBE_FHFT(_n)  (0x09000 + (_n * 0x100)) /* Flex host filter table */
364 /* Ext Flexible Host Filter Table */
365 #define IXGBE_FHFT_EXT(_n)      (0x09800 + (_n * 0x100))
341 #define IXGBE_FHFT_EXT(_n) (0x09800 + (_n * 0x100)) /* Ext Flexible Host
342                                                      * Filter Table */

367 #define IXGBE_FLEXIBLE_FILTER_COUNT_MAX         4
368 #define IXGBE_EXT_FLEXIBLE_FILTER_COUNT_MAX     2

370 /* Each Flexible Filter is at most 128 (0x80) bytes in length */
371 #define IXGBE_FLEXIBLE_FILTER_SIZE_MAX          128
372 #define IXGBE_FHFT_LENGTH_OFFSET                0xFC  /* Length byte in FHFT */
373 #define IXGBE_FHFT_LENGTH_MASK                  0x0FF /* Length in lower byte */

375 /* Definitions for power management and wakeup registers */
376 /* Wake Up Control */
```

```
377 #define IXGBE_WUC_PME_EN        0x00000002 /* PME Enable */
378 #define IXGBE_WUC_PME_STATUS    0x00000004 /* PME Status */
379 #define IXGBE_WUC_WKEN          0x00000010 /* Enable PE_WAKE_N pin assertion  */

381 /* Wake Up Filter Control */
382 #define IXGBE_WUFC_LNKC 0x00000001 /* Link Status Change Wakeup Enable */
383 #define IXGBE_WUFC_MAG  0x00000002 /* Magic Packet Wakeup Enable */
384 #define IXGBE_WUFC_EX   0x00000004 /* Directed Exact Wakeup Enable */
385 #define IXGBE_WUFC_MC   0x00000008 /* Directed Multicast Wakeup Enable */
386 #define IXGBE_WUFC_BC   0x00000010 /* Broadcast Wakeup Enable */
387 #define IXGBE_WUFC_ARP  0x00000020 /* ARP Request Packet Wakeup Enable */
388 #define IXGBE_WUFC_IPV4 0x00000040 /* Directed IPv4 Packet Wakeup Enable */
389 #define IXGBE_WUFC_IPV6 0x00000080 /* Directed IPv6 Packet Wakeup Enable */
390 #define IXGBE_WUFC_MNG  0x00000100 /* Directed Mgmt Packet Wakeup Enable */

392 #define IXGBE_WUFC_IGNORE_TCO   0x00008000 /* Ignore WakeOn TCO packets */
393 #define IXGBE_WUFC_FLX0 0x00010000 /* Flexible Filter 0 Enable */
394 #define IXGBE_WUFC_FLX1 0x00020000 /* Flexible Filter 1 Enable */
395 #define IXGBE_WUFC_FLX2 0x00040000 /* Flexible Filter 2 Enable */
396 #define IXGBE_WUFC_FLX3 0x00080000 /* Flexible Filter 3 Enable */
397 #define IXGBE_WUFC_FLX4 0x00100000 /* Flexible Filter 4 Enable */
398 #define IXGBE_WUFC_FLX5 0x00200000 /* Flexible Filter 5 Enable */
399 #define IXGBE_WUFC_FLX_FILTERS  0x000F0000 /* Mask for 4 flex filters */
400 /* Mask for Ext. flex filters */
401 #define IXGBE_WUFC_EXT_FLX_FILTERS      0x00300000
377 #define IXGBE_WUFC_EXT_FLX_FILTERS 0x00300000 /* Mask for Ext. flex filters */
402 #define IXGBE_WUFC_ALL_FILTERS  0x003F00FF /* Mask for all wakeup filters */
403 #define IXGBE_WUFC_FLX_OFFSET   16 /* Offset to the Flexible Filters bits */

405 /* Wake Up Status */
406 #define IXGBE_WUS_LNKC          IXGBE_WUFC_LNKC
407 #define IXGBE_WUS_MAG           IXGBE_WUFC_MAG
408 #define IXGBE_WUS_EX            IXGBE_WUFC_EX
409 #define IXGBE_WUS_MC            IXGBE_WUFC_MC
410 #define IXGBE_WUS_BC            IXGBE_WUFC_BC
411 #define IXGBE_WUS_ARP           IXGBE_WUFC_ARP
412 #define IXGBE_WUS_IPV4          IXGBE_WUFC_IPV4
413 #define IXGBE_WUS_IPV6          IXGBE_WUFC_IPV6
414 #define IXGBE_WUS_MNG           IXGBE_WUFC_MNG
415 #define IXGBE_WUS_FLX0          IXGBE_WUFC_FLX0
416 #define IXGBE_WUS_FLX1          IXGBE_WUFC_FLX1
417 #define IXGBE_WUS_FLX2          IXGBE_WUFC_FLX2
418 #define IXGBE_WUS_FLX3          IXGBE_WUFC_FLX3
419 #define IXGBE_WUS_FLX4          IXGBE_WUFC_FLX4
420 #define IXGBE_WUS_FLX5          IXGBE_WUFC_FLX5
421 #define IXGBE_WUS_FLX_FILTERS   IXGBE_WUFC_FLX_FILTERS

423 /* Wake Up Packet Length */
424 #define IXGBE_WUPL_LENGTH_MASK  0xFFFF

426 /* DCB registers */
427 #define IXGBE_DCB_MAX_TRAFFIC_CLASS     8
428 #define IXGBE_RMCS              0x03D00
429 #define IXGBE_DPMCS             0x07F40
430 #define IXGBE_PDPMCS            0x0CD00
431 #define IXGBE_RUPPBMR           0x050A0
432 #define IXGBE_RT2CR(_i)         (0x03C20 + ((_i) * 4)) /* 8 of these (0-7) */
433 #define IXGBE_RT2SR(_i)         (0x03C40 + ((_i) * 4)) /* 8 of these (0-7) */
434 #define IXGBE_TDTQ2TCCR(_i)     (0x0602C + ((_i) * 0x40)) /* 8 of these (0-7) */
435 #define IXGBE_TDTQ2TCSR(_i)     (0x0622C + ((_i) * 0x40)) /* 8 of these (0-7) */
436 #define IXGBE_TDPT2TCCR(_i)     (0x0CD20 + ((_i) * 4)) /* 8 of these (0-7) */
437 #define IXGBE_TDPT2TCSR(_i)     (0x0CD40 + ((_i) * 4)) /* 8 of these (0-7) */


440 /* Security Control Registers */
441 #define IXGBE_SECTXCTRL         0x08800
```



```
442 #define IXGBE_SECTXSTAT         0x08804
443 #define IXGBE_SECTXBUFFAF       0x08808
444 #define IXGBE_SECTXMINIFG       0x08810
420 #define IXGBE_SECTXSTAT         0x08804
445 #define IXGBE_SECRXCTRL         0x08D00
446 #define IXGBE_SECRXSTAT         0x08D04

448 /* Security Bit Fields and Masks */
449 #define IXGBE_SECTXCTRL_SECTX_DIS       0x00000001
450 #define IXGBE_SECTXCTRL_TX_DIS          0x00000002
451 #define IXGBE_SECTXCTRL_STORE_FORWARD   0x00000004

453 #define IXGBE_SECTXSTAT_SECTX_RDY       0x00000001
454 #define IXGBE_SECTXSTAT_ECC_TXERR       0x00000002

456 #define IXGBE_SECRXCTRL_SECRX_DIS       0x00000001
457 #define IXGBE_SECRXCTRL_RX_DIS          0x00000002

459 #define IXGBE_SECRXSTAT_SECRX_RDY       0x00000001
460 #define IXGBE_SECRXSTAT_ECC_RXERR       0x00000002

462 /* LinkSec (MacSec) Registers */
463 #define IXGBE_LSECTXCAP         0x08A00
464 #define IXGBE_LSECRXCAP         0x08F00
465 #define IXGBE_LSECTXCTRL        0x08A04
466 #define IXGBE_LSECTXSCL         0x08A08 /* SCI Low */
467 #define IXGBE_LSECTXSCH         0x08A0C /* SCI High */
468 #define IXGBE_LSECTXSA          0x08A10
469 #define IXGBE_LSECTXPN0         0x08A14
470 #define IXGBE_LSECTXPN1         0x08A18
471 #define IXGBE_LSECTXKEY0(_n)    (0x08A1C + (4 * (_n))) /* 4 of these (0-3) */
472 #define IXGBE_LSECTXKEY1(_n)    (0x08A2C + (4 * (_n))) /* 4 of these (0-3) */
473 #define IXGBE_LSECRXCTRL        0x08F04
474 #define IXGBE_LSECRXSCL         0x08F08
475 #define IXGBE_LSECRXSCH         0x08F0C
476 #define IXGBE_LSECRXSA(_i)      (0x08F10 + (4 * (_i))) /* 2 of these (0-1) */
477 #define IXGBE_LSECRXPN(_i)      (0x08F18 + (4 * (_i))) /* 2 of these (0-1) */
478 #define IXGBE_LSECRXKEY(_n, _m) (0x08F20 + ((0x10 * (_n)) + (4 * (_m))))
479 #define IXGBE_LSECTXUT          0x08A3C /* OutPktsUntagged */
480 #define IXGBE_LSECTXPKTE        0x08A40 /* OutPktsEncrypted */
481 #define IXGBE_LSECTXPKTP        0x08A44 /* OutPktsProtected */
482 #define IXGBE_LSECTXOCTE        0x08A48 /* OutOctetsEncrypted */
483 #define IXGBE_LSECTXOCTP        0x08A4C /* OutOctetsProtected */
484 #define IXGBE_LSECRXUT          0x08F40 /* InPktsUntagged/InPktsNoTag */
485 #define IXGBE_LSECRXOCTD        0x08F44 /* InOctetsDecrypted */
486 #define IXGBE_LSECRXOCTV        0x08F48 /* InOctetsValidated */
487 #define IXGBE_LSECRXBAD         0x08F4C /* InPktsBadTag */
488 #define IXGBE_LSECRXNOSCI       0x08F50 /* InPktsNoSci */
489 #define IXGBE_LSECRXUNSCI       0x08F54 /* InPktsUnknownSci */
490 #define IXGBE_LSECRXUNCH        0x08F58 /* InPktsUnchecked */
491 #define IXGBE_LSECRXDELAY       0x08F5C /* InPktsDelayed */
492 #define IXGBE_LSECRXLATE        0x08F60 /* InPktsLate */
493 #define IXGBE_LSECRXOK(_n)      (0x08F64 + (0x04 * (_n))) /* InPktsOk */
494 #define IXGBE_LSECRXINV(_n)     (0x08F6C + (0x04 * (_n))) /* InPktsInvalid */
495 #define IXGBE_LSECRXNV(_n)      (0x08F74 + (0x04 * (_n))) /* InPktsNotValid */
496 #define IXGBE_LSECRXUNSA        0x08F7C /* InPktsUnusedSa */
497 #define IXGBE_LSECRXNUSA        0x08F80 /* InPktsNotUsingSa */

499 /* LinkSec (MacSec) Bit Fields and Masks */
500 #define IXGBE_LSECTXCAP_SUM_MASK        0x00FF0000
501 #define IXGBE_LSECTXCAP_SUM_SHIFT       16
502 #define IXGBE_LSECRXCAP_SUM_MASK        0x00FF0000
503 #define IXGBE_LSECRXCAP_SUM_SHIFT       16

505 #define IXGBE_LSECTXCTRL_EN_MASK        0x00000003
506 #define IXGBE_LSECTXCTRL_DISABLE        0x0
```

```
507 #define IXGBE_LSECTXCTRL_AUTH           0x1
508 #define IXGBE_LSECTXCTRL_AUTH_ENCRYPT   0x2
509 #define IXGBE_LSECTXCTRL_AISCI          0x00000020
510 #define IXGBE_LSECTXCTRL_PNTHRSH_MASK   0xFFFFFF00
511 #define IXGBE_LSECTXCTRL_RSV_MASK       0x000000D8

513 #define IXGBE_LSECRXCTRL_EN_MASK        0x0000000C
514 #define IXGBE_LSECRXCTRL_EN_SHIFT       2
515 #define IXGBE_LSECRXCTRL_DISABLE        0x0
516 #define IXGBE_LSECRXCTRL_CHECK          0x1
517 #define IXGBE_LSECRXCTRL_STRICT         0x2
518 #define IXGBE_LSECRXCTRL_DROP           0x3
519 #define IXGBE_LSECRXCTRL_PLSH           0x00000040
520 #define IXGBE_LSECRXCTRL_RP             0x00000080
521 #define IXGBE_LSECRXCTRL_RSV_MASK       0xFFFFFF33

523 /* IpSec Registers */
524 #define IXGBE_IPSTXIDX          0x08900
525 #define IXGBE_IPSTXSALT         0x08904
526 #define IXGBE_IPSTXKEY(_i)      (0x08908 + (4 * (_i))) /* 4 of these (0-3) */
527 #define IXGBE_IPSRXIDX          0x08E00
528 #define IXGBE_IPSRXIPADDR(_i)   (0x08E04 + (4 * (_i))) /* 4 of these (0-3) */
529 #define IXGBE_IPSRXSPI          0x08E14
530 #define IXGBE_IPSRXIPIDX        0x08E18
531 #define IXGBE_IPSRXKEY(_i)      (0x08E1C + (4 * (_i))) /* 4 of these (0-3) */
532 #define IXGBE_IPSRXSALT         0x08E2C
533 #define IXGBE_IPSRXMOD          0x08E30

535 #define IXGBE_SECTXCTRL_STORE_FORWARD_ENABLE    0x4

537 /* DCB registers */
538 #define IXGBE_RTRPCS            0x02430
539 #define IXGBE_RTTDCS            0x04900
540 #define IXGBE_RTTDCS_ARBDIS     0x00000040 /* DCB arbiter disable */
541 #define IXGBE_RTTPCS            0x0CD00
542 #define IXGBE_RTRUP2TC          0x03020
543 #define IXGBE_RTTUP2TC          0x0C800
544 #define IXGBE_RTRPT4C(_i)       (0x02140 + ((_i) * 4)) /* 8 of these (0-7) */
545 #define IXGBE_TXLLQ(_i)         (0x082E0 + ((_i) * 4)) /* 4 of these (0-3) */
546 #define IXGBE_RTRPT4S(_i)       (0x02160 + ((_i) * 4)) /* 8 of these (0-7) */
547 #define IXGBE_RTTDT2C(_i)       (0x04910 + ((_i) * 4)) /* 8 of these (0-7) */
548 #define IXGBE_RTTDT2S(_i)       (0x04930 + ((_i) * 4)) /* 8 of these (0-7) */
549 #define IXGBE_RTTPT2C(_i)       (0x0CD20 + ((_i) * 4)) /* 8 of these (0-7) */
550 #define IXGBE_RTTPT2S(_i)       (0x0CD40 + ((_i) * 4)) /* 8 of these (0-7) */
551 #define IXGBE_RTTDQSEL          0x04904
552 #define IXGBE_RTTDT1C           0x04908
553 #define IXGBE_RTTDT1S           0x0490C
554 #define IXGBE_RTTDTECC          0x04990
555 #define IXGBE_RTTDTECC_NO_BCN   0x00000100

557 #define IXGBE_RTTBCNRC                  0x04984
558 #define IXGBE_RTTBCNRC_RS_ENA           0x80000000
559 #define IXGBE_RTTBCNRC_RF_DEC_MASK      0x00003FFF
560 #define IXGBE_RTTBCNRC_RF_INT_SHIFT     14
561 #define IXGBE_RTTBCNRC_RF_INT_MASK \
562         (IXGBE_RTTBCNRC_RF_DEC_MASK << IXGBE_RTTBCNRC_RF_INT_SHIFT)
563 #define IXGBE_RTTBCNRM  0x04980

565 /* BCN (for DCB) Registers */
540 #define IXGBE_RTTBCNRM    0x04980
566 #define IXGBE_RTTBCNRS  0x04988
567 #define IXGBE_RTTBCNCR  0x08B00
568 #define IXGBE_RTTBCNACH 0x08B04
569 #define IXGBE_RTTBCNACL 0x08B08
570 #define IXGBE_RTTBCNTG  0x04A90
571 #define IXGBE_RTTBCNIDX 0x08B0C
```



```
572 #define IXGBE_RTTBCNCP  0x08B10
573 #define IXGBE_RTFRTIMER 0x08B14
574 #define IXGBE_RTTBCNRTT 0x05150
575 #define IXGBE_RTTBCNRD  0x0498C

577 /* FCoE DMA Context Registers */
578 #define IXGBE_FCPTRL            0x02410 /* FC User Desc. PTR Low */
579 #define IXGBE_FCPTRH            0x02414 /* FC USer Desc. PTR High */
580 #define IXGBE_FCBUFF            0x02418 /* FC Buffer Control */
581 #define IXGBE_FCDMARW           0x02420 /* FC Receive DMA RW */
582 #define IXGBE_FCINVST0          0x03FC0 /* FC Invalid DMA Context Status Reg 0*/
557 #define IXGBE_FCINVST0  0x03FC0 /* FC Invalid DMA Context Status Reg 0 */
583 #define IXGBE_FCINVST(_i)       (IXGBE_FCINVST0 + ((_i) * 4))
584 #define IXGBE_FCBUFF_VALID      (1 << 0)   /* DMA Context Valid */
585 #define IXGBE_FCBUFF_BUFFSIZE   (3 << 3)   /* User Buffer Size */
586 #define IXGBE_FCBUFF_WRCONTX    (1 << 7)   /* 0: Initiator, 1: Target */
587 #define IXGBE_FCBUFF_BUFFCNT    0x0000ff00 /* Number of User Buffers */
588 #define IXGBE_FCBUFF_OFFSET     0xffff0000 /* User Buffer Offset */
589 #define IXGBE_FCBUFF_BUFFSIZE_SHIFT     3
590 #define IXGBE_FCBUFF_BUFFCNT_SHIFT      8
591 #define IXGBE_FCBUFF_OFFSET_SHIFT       16
592 #define IXGBE_FCDMARW_WE                (1 << 14)   /* Write enable */
593 #define IXGBE_FCDMARW_RE                (1 << 15)   /* Read enable */
594 #define IXGBE_FCDMARW_FCOESEL           0x000001ff  /* FC X_ID: 11 bits */
595 #define IXGBE_FCDMARW_LASTSIZE          0xffff0000  /* Last User Buffer Size */
596 #define IXGBE_FCDMARW_LASTSIZE_SHIFT    16
597 /* FCoE SOF/EOF */
598 #define IXGBE_TEOFF             0x04A94 /* Tx FC EOF */
599 #define IXGBE_TSOFF             0x04A98 /* Tx FC SOF */
600 #define IXGBE_REOFF             0x05158 /* Rx FC EOF */
601 #define IXGBE_RSOFF             0x051F8 /* Rx FC SOF */
602 /* FCoE Filter Context Registers */
603 #define IXGBE_FCFLT             0x05108 /* FC FLT Context */
604 #define IXGBE_FCFLTRW           0x05110 /* FC Filter RW Control */
605 #define IXGBE_FCPARAM           0x051d8 /* FC Offset Parameter */
606 #define IXGBE_FCFLT_VALID       (1 << 0)   /* Filter Context Valid */
607 #define IXGBE_FCFLT_FIRST       (1 << 1)   /* Filter First */
608 #define IXGBE_FCFLT_SEQID       0x00ff0000 /* Sequence ID */
609 #define IXGBE_FCFLT_SEQCNT      0xff000000 /* Sequence Count */
610 #define IXGBE_FCFLTRW_RVALDT    (1 << 13)  /* Fast Re-Validation */
611 #define IXGBE_FCFLTRW_WE        (1 << 14)  /* Write Enable */
612 #define IXGBE_FCFLTRW_RE        (1 << 15)  /* Read Enable */
613 /* FCoE Receive Control */
614 #define IXGBE_FCRXCTRL          0x05100 /* FC Receive Control */
615 #define IXGBE_FCRXCTRL_FCOELLI  (1 << 0)   /* Low latency interrupt */
616 #define IXGBE_FCRXCTRL_SAVBAD   (1 << 1)   /* Save Bad Frames */
617 #define IXGBE_FCRXCTRL_FRSTRDH  (1 << 2)   /* EN 1st Read Header */
618 #define IXGBE_FCRXCTRL_LASTSEQH (1 << 3)   /* EN Last Header in Seq */
619 #define IXGBE_FCRXCTRL_ALLH     (1 << 4)   /* EN All Headers */
620 #define IXGBE_FCRXCTRL_FRSTSEQH (1 << 5)   /* EN 1st Seq. Header */
621 #define IXGBE_FCRXCTRL_ICRC     (1 << 6)   /* Ignore Bad FC CRC */
622 #define IXGBE_FCRXCTRL_FCCRCBO  (1 << 7)   /* FC CRC Byte Ordering */
623 #define IXGBE_FCRXCTRL_FCOEVER  0x00000f00 /* FCoE Version: 4 bits */
624 #define IXGBE_FCRXCTRL_FCOEVER_SHIFT    8
625 /* FCoE Redirection */
626 #define IXGBE_FCRECTL           0x0ED00 /* FC Redirection Control */
627 #define IXGBE_FCRETA0           0x0ED10 /* FC Redirection Table 0 */
628 #define IXGBE_FCRETA(_i)        (IXGBE_FCRETA0 + ((_i) * 4)) /* FCoE Redir */
629 #define IXGBE_FCRECTL_ENA       0x1 /* FCoE Redir Table Enable */
630 #define IXGBE_FCRETASEL_ENA     0x2 /* FCoE FCRETASEL bit */
631 #define IXGBE_FCRETA_SIZE       8 /* Max entries in FCRETA */
632 #define IXGBE_FCRETA_ENTRY_MASK 0x0000007f /* 7 bits for the queue index */

634 /* Stats registers */
635 #define IXGBE_CRCERRS   0x04000
636 #define IXGBE_ILLERRC   0x04004
```

```
637 #define IXGBE_ERRBC     0x04008
638 #define IXGBE_MSPDC     0x04010
639 #define IXGBE_MPC(_i)   (0x03FA0 + ((_i) * 4)) /* 8 of these 3FA0-3FBC*/
640 #define IXGBE_MLFC      0x04034
641 #define IXGBE_MRFC      0x04038
642 #define IXGBE_RLEC      0x04040
643 #define IXGBE_LXONTXC   0x03F60
644 #define IXGBE_LXONRXC   0x0CF60
645 #define IXGBE_LXOFFTXC  0x03F68
646 #define IXGBE_LXOFFRXC  0x0CF68
647 #define IXGBE_LXONRXCNT         0x041A4
648 #define IXGBE_LXOFFRXCNT        0x041A8
649 #define IXGBE_PXONRXCNT(_i)     (0x04140 + ((_i) * 4)) /* 8 of these */
650 #define IXGBE_PXOFFRXCNT(_i)    (0x04160 + ((_i) * 4)) /* 8 of these */
651 #define IXGBE_PXON2OFFCNT(_i)   (0x03240 + ((_i) * 4)) /* 8 of these */
652 #define IXGBE_PXONTXC(_i)       (0x03F00 + ((_i) * 4)) /* 8 of these 3F00-3F1C*/
653 #define IXGBE_PXONRXC(_i)       (0x0CF00 + ((_i) * 4)) /* 8 of these CF00-CF1C*/
654 #define IXGBE_PXOFFTXC(_i)      (0x03F20 + ((_i) * 4)) /* 8 of these 3F20-3F3C*/
655 #define IXGBE_PXOFFRXC(_i)      (0x0CF20 + ((_i) * 4)) /* 8 of these CF20-CF3C*/
656 #define IXGBE_PRC64             0x0405C
657 #define IXGBE_PRC127            0x04060
658 #define IXGBE_PRC255            0x04064
659 #define IXGBE_PRC511            0x04068
660 #define IXGBE_PRC1023           0x0406C
661 #define IXGBE_PRC1522           0x04070
662 #define IXGBE_GPRC              0x04074
663 #define IXGBE_BPRC              0x04078
664 #define IXGBE_MPRC              0x0407C
665 #define IXGBE_GPTC              0x04080
666 #define IXGBE_GORCL             0x04088
667 #define IXGBE_GORCH             0x0408C
668 #define IXGBE_GOTCL             0x04090
669 #define IXGBE_GOTCH             0x04094
670 #define IXGBE_RNBC(_i)          (0x03FC0 + ((_i) * 4)) /* 8 of these 3FC0-3FDC*/
671 #define IXGBE_RUC               0x040A4
672 #define IXGBE_RFC               0x040A8
673 #define IXGBE_ROC               0x040AC
674 #define IXGBE_RJC               0x040B0
675 #define IXGBE_MNGPRC            0x040B4
676 #define IXGBE_MNGPDC            0x040B8
677 #define IXGBE_MNGPTC            0x0CF90
678 #define IXGBE_TORL              0x040C0
679 #define IXGBE_TORH              0x040C4
680 #define IXGBE_TPR               0x040D0
681 #define IXGBE_TPT               0x040D4
682 #define IXGBE_PTC64             0x040D8
683 #define IXGBE_PTC127            0x040DC
684 #define IXGBE_PTC255            0x040E0
685 #define IXGBE_PTC511            0x040E4
686 #define IXGBE_PTC1023           0x040E8
687 #define IXGBE_PTC1522           0x040EC
688 #define IXGBE_MPTC              0x040F0
689 #define IXGBE_BPTC              0x040F4
690 #define IXGBE_XEC               0x04120
691 #define IXGBE_SSVPC             0x08780

693 #define IXGBE_RQSMR(_i) (0x02300 + ((_i) * 4))
694 #define IXGBE_TQSMR(_i) (((_i) <= 7) ? (0x07300 + ((_i) * 4)) : \
695                          (0x08600 + ((_i) * 4)))
696 #define IXGBE_TQSM(_i)  (0x08600 + ((_i) * 4))

698 #define IXGBE_QPRC(_i)  (0x01030 + ((_i) * 0x40)) /* 16 of these */
699 #define IXGBE_QPTC(_i)  (0x06030 + ((_i) * 0x40)) /* 16 of these */
700 #define IXGBE_QBRC(_i)  (0x01034 + ((_i) * 0x40)) /* 16 of these */
701 #define IXGBE_QBTC(_i)  (0x06034 + ((_i) * 0x40)) /* 16 of these */
702 #define IXGBE_QBRC_L(_i)        (0x01034 + ((_i) * 0x40)) /* 16 of these */
```



```
703 #define IXGBE_QBRC_H(_i)        (0x01038 + ((_i) * 0x40)) /* 16 of these */
704 #define IXGBE_QPRDC(_i)         (0x01430 + ((_i) * 0x40)) /* 16 of these */
705 #define IXGBE_QBTC_L(_i)        (0x08700 + ((_i) * 0x8)) /* 16 of these */
706 #define IXGBE_QBTC_H(_i)        (0x08704 + ((_i) * 0x8)) /* 16 of these */
707 #define IXGBE_FCCRC             0x05118 /* Num of Good Eth CRC w/ Bad FC CRC */
681 #define IXGBE_FCCRC     0x05118 /* Count of Good Eth CRC w/ Bad FC CRC */
708 #define IXGBE_FCOERPDC          0x0241C /* FCoE Rx Packets Dropped Count */
709 #define IXGBE_FCLAST            0x02424 /* FCoE Last Error Count */
710 #define IXGBE_FCOEPRC           0x02428 /* Number of FCoE Packets Received */
711 #define IXGBE_FCOEDWRC          0x0242C /* Number of FCoE DWords Received */
712 #define IXGBE_FCOEPTC           0x08784 /* Number of FCoE Packets Transmitted */
713 #define IXGBE_FCOEDWTC          0x08788 /* Number of FCoE DWords Transmitted */
714 #define IXGBE_FCCRC_CNT_MASK    0x0000FFFF /* CRC_CNT: bit 0 - 15 */
715 #define IXGBE_FCLAST_CNT_MASK   0x0000FFFF /* Last_CNT: bit 0 - 15 */
716 #define IXGBE_O2BGPTC           0x041C4
717 #define IXGBE_O2BSPC            0x087B0
718 #define IXGBE_B2OSPC            0x041C0
719 #define IXGBE_B2OGPRC           0x02F90
720 #define IXGBE_BUPRC             0x04180
721 #define IXGBE_BMPRC             0x04184
722 #define IXGBE_BBPRC             0x04188
723 #define IXGBE_BUPTC             0x0418C
724 #define IXGBE_BMPTC             0x04190
725 #define IXGBE_BBPTC             0x04194
726 #define IXGBE_BCRCERRS          0x04198
727 #define IXGBE_BXONRXC           0x0419C
728 #define IXGBE_BXOFFRXC          0x041E0
729 #define IXGBE_BXONTXC           0x041E4
730 #define IXGBE_BXOFFTXC          0x041E8
731 #define IXGBE_PCRC8ECL          0x0E810
732 #define IXGBE_PCRC8ECH          0x0E811
733 #define IXGBE_PCRC8ECH_MASK     0x1F
734 #define IXGBE_LDPCECL           0x0E820
735 #define IXGBE_LDPCECH           0x0E821

737 /* Management */
738 #define IXGBE_MAVTV(_i)         (0x05010 + ((_i) * 4)) /* 8 of these (0-7) */
739 #define IXGBE_MFUTP(_i)         (0x05030 + ((_i) * 4)) /* 8 of these (0-7) */
740 #define IXGBE_MANC              0x05820
741 #define IXGBE_MFVAL             0x05824
742 #define IXGBE_MANC2H            0x05860
743 #define IXGBE_MDEF(_i)          (0x05890 + ((_i) * 4)) /* 8 of these (0-7) */
744 #define IXGBE_MIPAF             0x058B0
745 #define IXGBE_MMAL(_i)          (0x05910 + ((_i) * 8)) /* 4 of these (0-3) */
746 #define IXGBE_MMAH(_i)          (0x05914 + ((_i) * 8)) /* 4 of these (0-3) */
747 #define IXGBE_FTFT              0x09400 /* 0x9400-0x97FC */
748 #define IXGBE_METF(_i)          (0x05190 + ((_i) * 4)) /* 4 of these (0-3) */
749 #define IXGBE_MDEF_EXT(_i)      (0x05160 + ((_i) * 4)) /* 8 of these (0-7) */
750 #define IXGBE_LSWFW             0x15014
751 #define IXGBE_BMCIP(_i)         (0x05050 + ((_i) * 4)) /* 0x5050-0x505C */
752 #define IXGBE_BMCIPVAL          0x05060
753 #define IXGBE_BMCIP_IPADDR_TYPE 0x00000001
754 #define IXGBE_BMCIP_IPADDR_VALID        0x00000002

756 /* Management Bit Fields and Masks */
757 #define IXGBE_MANC_EN_BMC2OS    0x10000000 /* Ena BMC2OS and OS2BMC traffic */
758 #define IXGBE_MANC_EN_BMC2OS_SHIFT      28

760 /* Firmware Semaphore Register */
761 #define IXGBE_FWSM_MODE_MASK    0xE

763 /* ARC Subsystem registers */
764 #define IXGBE_HICR              0x15F00
765 #define IXGBE_FWSTS             0x15F0C
766 #define IXGBE_HSMC0R            0x15F04
767 #define IXGBE_HSMC1R            0x15F08
```

```
768 #define IXGBE_SWSR              0x15F10
769 #define IXGBE_HFDR              0x15FE8
770 #define IXGBE_FLEX_MNG          0x15800 /* 0x15800 - 0x15EFC */

772 #define IXGBE_HICR_EN           0x01  /* Enable bit - RO */
773 /* Driver sets this bit when done to put command in RAM */
774 #define IXGBE_HICR_C            0x02
775 #define IXGBE_HICR_SV           0x04  /* Status Validity */
776 #define IXGBE_HICR_FW_RESET_ENABLE      0x40
777 #define IXGBE_HICR_FW_RESET     0x80

779 /* PCI-E registers */
780 #define IXGBE_GCR               0x11000
781 #define IXGBE_GTV               0x11004
782 #define IXGBE_FUNCTAG           0x11008
783 #define IXGBE_GLT               0x1100C
784 #define IXGBE_PCIEPIPEADR       0x11004
785 #define IXGBE_PCIEPIPEDAT       0x11008
786 #define IXGBE_GSCL_1            0x11010
787 #define IXGBE_GSCL_2            0x11014
788 #define IXGBE_GSCL_3            0x11018
789 #define IXGBE_GSCL_4            0x1101C
790 #define IXGBE_GSCN_0            0x11020
791 #define IXGBE_GSCN_1            0x11024
792 #define IXGBE_GSCN_2            0x11028
793 #define IXGBE_GSCN_3            0x1102C
794 #define IXGBE_FACTPS            0x10150
795 #define IXGBE_PCIEANACTL        0x11040
796 #define IXGBE_SWSM              0x10140
797 #define IXGBE_FWSM              0x10148
798 #define IXGBE_GSSR              0x10160
799 #define IXGBE_MREVID            0x11064
800 #define IXGBE_DCA_ID            0x11070
801 #define IXGBE_DCA_CTRL          0x11074
802 #define IXGBE_SWFW_SYNC         IXGBE_GSSR

804 /* PCI-E registers 82599-Specific */
805 #define IXGBE_GCR_EXT           0x11050
806 #define IXGBE_GSCL_5_82599      0x11030
807 #define IXGBE_GSCL_6_82599      0x11034
808 #define IXGBE_GSCL_7_82599      0x11038
809 #define IXGBE_GSCL_8_82599      0x1103C
810 #define IXGBE_PHYADR_82599      0x11040
811 #define IXGBE_PHYDAT_82599      0x11044
812 #define IXGBE_PHYCTL_82599      0x11048
813 #define IXGBE_PBACLR_82599      0x11068
814 #define IXGBE_CIAA_82599        0x11088
815 #define IXGBE_CIAD_82599        0x1108C
816 #define IXGBE_PICAUSE           0x110B0
817 #define IXGBE_PIENA             0x110B8
750 #define IXGBE_INTRPT_CSR_82599  0x110B0
751 #define IXGBE_INTRPT_MASK_82599 0x110B8
818 #define IXGBE_CDQ_MBR_82599     0x110B4
819 #define IXGBE_PCIESPARE         0x110BC
820 #define IXGBE_MISC_REG_82599    0x110F0
821 #define IXGBE_ECC_CTRL_0_82599  0x11100
822 #define IXGBE_ECC_CTRL_1_82599  0x11104
823 #define IXGBE_ECC_STATUS_82599  0x110E0
824 #define IXGBE_BAR_CTRL_82599    0x110F4

826 /* PCI Express Control */
827 #define IXGBE_GCR_CMPL_TMOUT_MASK       0x0000F000
828 #define IXGBE_GCR_CMPL_TMOUT_10ms       0x00001000
829 #define IXGBE_GCR_CMPL_TMOUT_RESEND     0x00010000
830 #define IXGBE_GCR_CAP_VER2              0x00040000
```



```
832 #define IXGBE_GCR_EXT_MSIX_EN           0x80000000
833 #define IXGBE_GCR_EXT_BUFFERS_CLEAR     0x40000000
834 #define IXGBE_GCR_EXT_VT_MODE_16        0x00000001
835 #define IXGBE_GCR_EXT_VT_MODE_32        0x00000002
836 #define IXGBE_GCR_EXT_VT_MODE_64        0x00000003
837 #define IXGBE_GCR_EXT_SRIOV             (IXGBE_GCR_EXT_MSIX_EN | \
838                                          IXGBE_GCR_EXT_VT_MODE_64)
839 #define IXGBE_GCR_EXT_VT_MODE_MASK      0x00000003
840 /* Time Sync Registers */
841 #define IXGBE_TSYNCRXCTL        0x05188 /* Rx Time Sync Control register - RW */
842 #define IXGBE_TSYNCTXCTL        0x08C00 /* Tx Time Sync Control register - RW */
843 #define IXGBE_RXSTMPL   0x051E8 /* Rx timestamp Low - RO */
844 #define IXGBE_RXSTMPH   0x051A4 /* Rx timestamp High - RO */
845 #define IXGBE_RXSATRL   0x051A0 /* Rx timestamp attribute low - RO */
846 #define IXGBE_RXSATRH   0x051A8 /* Rx timestamp attribute high - RO */
847 #define IXGBE_RXMTRL    0x05120 /* RX message type register low - RW */
848 #define IXGBE_TXSTMPL   0x08C04 /* Tx timestamp value Low - RO */
849 #define IXGBE_TXSTMPH   0x08C08 /* Tx timestamp value High - RO */
850 #define IXGBE_SYSTIML   0x08C0C /* System time register Low - RO */
851 #define IXGBE_SYSTIMH   0x08C10 /* System time register High - RO */
852 #define IXGBE_TIMINCA   0x08C14 /* Increment attributes register - RW */
853 #define IXGBE_TIMADJL   0x08C18 /* Time Adjustment Offset register Low - RW */
854 #define IXGBE_TIMADJH   0x08C1C /* Time Adjustment Offset register High - RW */
855 #define IXGBE_TSAUXC    0x08C20 /* TimeSync Auxiliary Control register - RW */
856 #define IXGBE_TRGTTIML0 0x08C24 /* Target Time Register 0 Low - RW */
857 #define IXGBE_TRGTTIMH0 0x08C28 /* Target Time Register 0 High - RW */
858 #define IXGBE_TRGTTIML1 0x08C2C /* Target Time Register 1 Low - RW */
859 #define IXGBE_TRGTTIMH1 0x08C30 /* Target Time Register 1 High - RW */
860 #define IXGBE_CLKTIML   0x08C34 /* Clock Out Time Register Low - RW */
861 #define IXGBE_CLKTIMH   0x08C38 /* Clock Out Time Register High - RW */
862 #define IXGBE_FREQOUT0  0x08C34 /* Frequency Out 0 Control register - RW */
863 #define IXGBE_FREQOUT1  0x08C38 /* Frequency Out 1 Control register - RW */
864 #define IXGBE_AUXSTMPL0 0x08C3C /* Auxiliary Time Stamp 0 register Low - RO */
865 #define IXGBE_AUXSTMPH0 0x08C40 /* Auxiliary Time Stamp 0 register High - RO */
866 #define IXGBE_AUXSTMPL1 0x08C44 /* Auxiliary Time Stamp 1 register Low - RO */
867 #define IXGBE_AUXSTMPH1 0x08C48 /* Auxiliary Time Stamp 1 register High - RO */
784 #define IXGBE_RXUDP      0x08C1C /* Time Sync Rx UDP Port - RW */

869 /* Diagnostic Registers */
870 #define IXGBE_RDSTATCTL         0x02C20
871 #define IXGBE_RDSTAT(_i)        (0x02C00 + ((_i) * 4)) /* 0x02C00-0x02C1C */
872 #define IXGBE_RDHMPN            0x02F08
873 #define IXGBE_RIC_DW(_i)        (0x02F10 + ((_i) * 4))
874 #define IXGBE_RDPROBE           0x02F20
875 #define IXGBE_RDMAM             0x02F30
876 #define IXGBE_RDMAD             0x02F34
877 #define IXGBE_TDSTATCTL         0x07C20
878 #define IXGBE_TDSTAT(_i)        (0x07C00 + ((_i) * 4)) /* 0x07C00 - 0x07C1C */
879 #define IXGBE_TDHMPN            0x07F08
880 #define IXGBE_TDHMPN2           0x082FC
881 #define IXGBE_TXDESCIC          0x082CC
882 #define IXGBE_TIC_DW(_i)        (0x07F10 + ((_i) * 4))
883 #define IXGBE_TIC_DW2(_i)       (0x082B0 + ((_i) * 4))
884 #define IXGBE_TDPROBE           0x07F20
885 #define IXGBE_TXBUFCTRL         0x0C600
886 #define IXGBE_TXBUFDATA0        0x0C610
887 #define IXGBE_TXBUFDATA1        0x0C614
888 #define IXGBE_TXBUFDATA2        0x0C618
889 #define IXGBE_TXBUFDATA3        0x0C61C
890 #define IXGBE_RXBUFCTRL         0x03600
891 #define IXGBE_RXBUFDATA0        0x03610
892 #define IXGBE_RXBUFDATA1        0x03614
893 #define IXGBE_RXBUFDATA2        0x03618
894 #define IXGBE_RXBUFDATA3        0x0361C
895 #define IXGBE_PCIE_DIAG(_i)     (0x11090 + ((_i) * 4)) /* 8 of these */
896 #define IXGBE_RFVAL             0x050A4
```

```
#define IXGBE_MDFTC1            0x042B8
#define IXGBE_MDFTC2            0x042C0
#define IXGBE_MDFTFIFO1         0x042C4
#define IXGBE_MDFTFIFO2         0x042C8
#define IXGBE_MDFTS             0x042CC
#define IXGBE_RXDATAWRPTR(_i)   (0x03700 + ((_i) * 4)) /* 8 of these 3700-370C*/
#define IXGBE_RXDESCWRPTR(_i)   (0x03710 + ((_i) * 4)) /* 8 of these 3710-371C*/
#define IXGBE_RXDATARDPTR(_i)   (0x03720 + ((_i) * 4)) /* 8 of these 3720-372C*/
#define IXGBE_RXDESCRDPTR(_i)   (0x03730 + ((_i) * 4)) /* 8 of these 3730-373C*/
#define IXGBE_TXDATAWRPTR(_i)   (0x0C700 + ((_i) * 4)) /* 8 of these C700-C70C*/
#define IXGBE_TXDESCWRPTR(_i)   (0x0C710 + ((_i) * 4)) /* 8 of these C710-C71C*/
#define IXGBE_TXDATARDPTR(_i)   (0x0C720 + ((_i) * 4)) /* 8 of these C720-C72C*/
#define IXGBE_TXDESCRDPTR(_i)   (0x0C730 + ((_i) * 4)) /* 8 of these C730-C73C*/
#define IXGBE_PCIEECCCTL        0x1106C
#define IXGBE_RXWRPTR(_i)       (0x03100 + ((_i) * 4)) /* 8 of these 3100-310C*/
#define IXGBE_RXUSED(_i)        (0x03120 + ((_i) * 4)) /* 8 of these 3120-312C*/
#define IXGBE_RXRDPTR(_i)       (0x03140 + ((_i) * 4)) /* 8 of these 3140-314C*/
#define IXGBE_RXRDWRPTR(_i)     (0x03160 + ((_i) * 4)) /* 8 of these 3160-310C*/
#define IXGBE_TXWRPTR(_i)       (0x0C100 + ((_i) * 4)) /* 8 of these C100-C10C*/
#define IXGBE_TXUSED(_i)        (0x0C120 + ((_i) * 4)) /* 8 of these C120-C12C*/
#define IXGBE_TXRDPTR(_i)       (0x0C140 + ((_i) * 4)) /* 8 of these C140-C14C*/
#define IXGBE_TXRDWRPTR(_i)     (0x0C160 + ((_i) * 4)) /* 8 of these C160-C10C*/
#define IXGBE_PCIEECCCTL0       0x11100
#define IXGBE_PCIEECCCTL1       0x11104
#define IXGBE_RXDBUECC          0x03F70
#define IXGBE_TXDBUECC          0x0CF70
#define IXGBE_RXDBUEST          0x03F74
#define IXGBE_TXDBUEST          0x0CF74
#define IXGBE_PBTXECC           0x0C300
#define IXGBE_PBRXECC           0x03300
#define IXGBE_GHECCR            0x110B0

/* MAC Registers */
#define IXGBE_PCS1GCFIG         0x04200
#define IXGBE_PCS1GLCTL         0x04208
#define IXGBE_PCS1GLSTA         0x0420C
#define IXGBE_PCS1GDBG0         0x04210
#define IXGBE_PCS1GDBG1         0x04214
#define IXGBE_PCS1GANA          0x04218
#define IXGBE_PCS1GANLP         0x0421C
#define IXGBE_PCS1GANNP         0x04220
#define IXGBE_PCS1GANLPNP       0x04224
#define IXGBE_HLREG0            0x04240
#define IXGBE_HLREG1            0x04244
#define IXGBE_PAP               0x04248
#define IXGBE_MACA              0x0424C
#define IXGBE_APAE              0x04250
#define IXGBE_ARD               0x04254
#define IXGBE_AIS               0x04258
#define IXGBE_MSCA              0x0425C
#define IXGBE_MSRWD             0x04260
#define IXGBE_MLADD             0x04264
#define IXGBE_MHADD             0x04268
#define IXGBE_MAXFRS            0x04268
#define IXGBE_TREG              0x0426C
#define IXGBE_PCSS1             0x04288
#define IXGBE_PCSS2             0x0428C
#define IXGBE_XPCSS             0x04290
#define IXGBE_MFLCN             0x04294
#define IXGBE_SERDESC           0x04298
#define IXGBE_MACS              0x0429C
#define IXGBE_AUTOC             0x042A0
#define IXGBE_LINKS             0x042A4
#define IXGBE_LINKS2            0x04324
#define IXGBE_AUTOC2            0x042A8
#define IXGBE_AUTOC3            0x042AC
```



```
#define IXGBE_ANLP1             0x042B0
#define IXGBE_ANLP2             0x042B4
#define IXGBE_MACC              0x04330
#define IXGBE_ATLASCTL          0x04800
#define IXGBE_MMNGC             0x042D0
#define IXGBE_ANLPNP1           0x042D4
#define IXGBE_ANLPNP2           0x042D8
#define IXGBE_KRPCSFC           0x042E0
#define IXGBE_KRPCSS            0x042E4
#define IXGBE_FECS1             0x042E8
#define IXGBE_FECS2             0x042EC
#define IXGBE_SMADARCTL         0x14F10
#define IXGBE_MPVC              0x04318
#define IXGBE_SGMIIC            0x04314

/* Statistics Registers */
#define IXGBE_RXNFGPC           0x041B0
#define IXGBE_RXNFGBCL          0x041B4
#define IXGBE_RXNFGBCH          0x041B8
#define IXGBE_RXDGPC            0x02F50
#define IXGBE_RXDGBCL           0x02F54
#define IXGBE_RXDGBCH           0x02F58
#define IXGBE_RXDDGPC           0x02F5C
#define IXGBE_RXDDGBCL          0x02F60
#define IXGBE_RXDDGBCH          0x02F64
#define IXGBE_RXLPBKGPC         0x02F68
#define IXGBE_RXLPBKGBCL        0x02F6C
#define IXGBE_RXLPBKGBCH        0x02F70
#define IXGBE_RXDLPBKGPC        0x02F74
#define IXGBE_RXDLPBKGBCL       0x02F78
#define IXGBE_RXDLPBKGBCH       0x02F7C
#define IXGBE_TXDGPC            0x087A0
#define IXGBE_TXDGBCL           0x087A4
#define IXGBE_TXDGBCH           0x087A8

#define IXGBE_RXDSTATCTRL       0x02F40

/* Copper Pond 2 link timeout */
#define IXGBE_VALIDATE_LINK_READY_TIMEOUT 50

/* Omer CORECTL */
#define IXGBE_CORECTL                   0x014F00
/* BARCTRL */
#define IXGBE_BARCTRL                   0x110F4
#define IXGBE_BARCTRL_FLSIZE            0x0700
#define IXGBE_BARCTRL_FLSIZE_SHIFT      8
#define IXGBE_BARCTRL_CSRSIZE           0x2000

/* RSCCTL Bit Masks */
#define IXGBE_RSCCTL_RSCEN      0x01
#define IXGBE_RSCCTL_MAXDESC_1  0x00
#define IXGBE_RSCCTL_MAXDESC_4  0x04
#define IXGBE_RSCCTL_MAXDESC_8  0x08
#define IXGBE_RSCCTL_MAXDESC_16 0x0C

/* RSCDBU Bit Masks */
#define IXGBE_RSCDBU_RSCSMALDIS_MASK    0x0000007F
#define IXGBE_RSCDBU_RSCACKDIS          0x00000080

/* RDRXCTL Bit Masks */
#define IXGBE_RDRXCTL_RDMTS_1_2         0x00000000 /* Rx Desc Min THLD Size */
#define IXGBE_RDRXCTL_RDMTS_1_2     0x00000000 /* Rx Desc Min Threshold Size */
#define IXGBE_RDRXCTL_CRCSTRIP          0x00000002 /* CRC Strip */
#define IXGBE_RDRXCTL_MVMEN             0x00000020
#define IXGBE_RDRXCTL_DMAIDONE          0x00000008 /* DMA init cycle done */
#define IXGBE_RDRXCTL_AGGDIS            0x00010000 /* Aggregation disable */
```

```
#define IXGBE_RDRXCTL_RSCFRSTSIZE       0x003E0000 /* RSC First packet size */
#define IXGBE_RDRXCTL_RSCLLIDIS         0x00800000 /* Disabl RSC compl on LLI */
#define IXGBE_RDRXCTL_RSCACKC           0x02000000 /* must set 1 when RSC ena */
#define IXGBE_RDRXCTL_FCOE_WRFIX        0x04000000 /* must set 1 when RSC ena */
#define IXGBE_RDRXCTL_RSCLLIDIS     0x00800000 /* Disable RSC compl on LLI */
#define IXGBE_RDRXCTL_RSCACKC       0x02000000 /* must set 1 when RSC enabled */
#define IXGBE_RDRXCTL_FCOE_WRFIX    0x04000000 /* must set 1 when RSC enabled */

/* RQTC Bit Masks and Shifts */
#define IXGBE_RQTC_SHIFT_TC(_i) ((_i) * 4)
#define IXGBE_RQTC_TC0_MASK     (0x7 << 0)
#define IXGBE_RQTC_TC1_MASK     (0x7 << 4)
#define IXGBE_RQTC_TC2_MASK     (0x7 << 8)
#define IXGBE_RQTC_TC3_MASK     (0x7 << 12)
#define IXGBE_RQTC_TC4_MASK     (0x7 << 16)
#define IXGBE_RQTC_TC5_MASK     (0x7 << 20)
#define IXGBE_RQTC_TC6_MASK     (0x7 << 24)
#define IXGBE_RQTC_TC7_MASK     (0x7 << 28)

/* PSRTYPE.RQPL Bit masks and shift */
#define IXGBE_PSRTYPE_RQPL_MASK         0x7
#define IXGBE_PSRTYPE_RQPL_SHIFT        29

/* CTRL Bit Masks */
#define IXGBE_CTRL_GIO_DIS      0x00000004 /* Global IO Master Disable bit */
#define IXGBE_CTRL_LNK_RST      0x00000008 /* Link Reset. Resets everything. */
#define IXGBE_CTRL_RST          0x04000000 /* Reset (SW) */
#define IXGBE_CTRL_RST_MASK     (IXGBE_CTRL_LNK_RST | IXGBE_CTRL_RST)

/* FACTPS */
#define IXGBE_FACTPS_LFS        0x40000000 /* LAN Function Select */

/* MHADD Bit Masks */
#define IXGBE_MHADD_MFS_MASK    0xFFFF0000
#define IXGBE_MHADD_MFS_SHIFT   16

/* Extended Device Control */
#define IXGBE_CTRL_EXT_PFRSTD   0x00004000 /* Physical Function Reset Done */
#define IXGBE_CTRL_EXT_NS_DIS   0x00010000 /* No Snoop disable */
#define IXGBE_CTRL_EXT_RO_DIS   0x00020000 /* Relaxed Ordering disable */
#define IXGBE_CTRL_EXT_DRV_LOAD 0x10000000 /* Driver loaded bit for FW */

/* Direct Cache Access (DCA) definitions */
#define IXGBE_DCA_CTRL_DCA_ENABLE       0x00000000 /* DCA Enable */
#define IXGBE_DCA_CTRL_DCA_DISABLE      0x00000001 /* DCA Disable */

#define IXGBE_DCA_CTRL_DCA_MODE_CB1     0x00 /* DCA Mode CB1 */
#define IXGBE_DCA_CTRL_DCA_MODE_CB2     0x02 /* DCA Mode CB2 */

#define IXGBE_DCA_RXCTRL_CPUID_MASK     0x0000001F /* Rx CPUID Mask */
#define IXGBE_DCA_RXCTRL_CPUID_MASK_82599       0xFF000000 /* Rx CPUID Mask */
#define IXGBE_DCA_RXCTRL_CPUID_SHIFT_82599      24 /* Rx CPUID Shift */
#define IXGBE_DCA_RXCTRL_DESC_DCA_EN    (1 << 5) /* Rx Desc enable */
#define IXGBE_DCA_RXCTRL_HEAD_DCA_EN    (1 << 6) /* Rx Desc header ena */
#define IXGBE_DCA_RXCTRL_DATA_DCA_EN    (1 << 7) /* Rx Desc payload ena */
#define IXGBE_DCA_RXCTRL_DESC_RRO_EN    (1 << 9) /* Rx rd Desc Relax Order */
#define IXGBE_DCA_RXCTRL_DATA_WRO_EN    (1 << 13) /* Rx wr data Relax Order */
#define IXGBE_DCA_RXCTRL_HEAD_WRO_EN    (1 << 15) /* Rx wr header RO */
#define IXGBE_DCA_RXCTRL_DESC_DCA_EN (1 << 5) /* DCA Rx Desc enable */
#define IXGBE_DCA_RXCTRL_HEAD_DCA_EN (1 << 6) /* DCA Rx Desc header enable */
#define IXGBE_DCA_RXCTRL_DATA_DCA_EN (1 << 7) /* DCA Rx Desc payload enable */
#define IXGBE_DCA_RXCTRL_DESC_RRO_EN (1 << 9) /* DCA Rx rd Desc Relax Order */
#define IXGBE_DCA_RXCTRL_DESC_WRO_EN (1 << 13) /* DCA Rx wr Desc Relax Order */
#define IXGBE_DCA_RXCTRL_DESC_HSRO_EN (1 << 15) /* DCA Rx Split Header RO */

#define IXGBE_DCA_TXCTRL_CPUID_MASK     0x0000001F /* Tx CPUID Mask */
```



```
#define IXGBE_DCA_TXCTRL_CPUID_MASK_82599       0xFF000000 /* Tx CPUID Mask */
#define IXGBE_DCA_TXCTRL_CPUID_SHIFT_82599      24 /* Tx CPUID Shift */
#define IXGBE_DCA_TXCTRL_DESC_DCA_EN    (1 << 5) /* DCA Tx Desc enable */
#define IXGBE_DCA_TXCTRL_DESC_RRO_EN    (1 << 9) /* Tx rd Desc Relax Order */
#define IXGBE_DCA_TXCTRL_DESC_WRO_EN    (1 << 11) /* Tx Desc writeback RO bit */
#define IXGBE_DCA_TXCTRL_DATA_RRO_EN    (1 << 13) /* Tx rd data Relax Order */
#define IXGBE_DCA_TXCTRL_TX_WB_RO_EN (1 << 11) /* Tx Desc writeback RO bit */
#define IXGBE_DCA_MAX_QUEUES_82598      16 /* DCA regs only on 16 queues */

/* MSCA Bit Masks */
#define IXGBE_MSCA_NP_ADDR_MASK         0x0000FFFF /* MDI Addr (new prot) */
#define IXGBE_MSCA_NP_ADDR_MASK      0x0000FFFF /* MDI Address (new protocol) */
#define IXGBE_MSCA_NP_ADDR_SHIFT        0
#define IXGBE_MSCA_DEV_TYPE_MASK        0x001F0000 /* Dev Type (new prot) */
#define IXGBE_MSCA_DEV_TYPE_SHIFT       16 /* Register Address (old prot */
#define IXGBE_MSCA_DEV_TYPE_MASK     0x001F0000 /* Device Type (new protocol) */
#define IXGBE_MSCA_DEV_TYPE_SHIFT    16 /* Register Address (old protocol */
#define IXGBE_MSCA_PHY_ADDR_MASK        0x03E00000 /* PHY Address mask */
#define IXGBE_MSCA_PHY_ADDR_SHIFT       21 /* PHY Address shift*/
#define IXGBE_MSCA_OP_CODE_MASK         0x0C000000 /* OP CODE mask */
#define IXGBE_MSCA_OP_CODE_SHIFT        26 /* OP CODE shift */
#define IXGBE_MSCA_ADDR_CYCLE           0x00000000 /* OP CODE 00 (addr cycle) */
#define IXGBE_MSCA_WRITE                0x04000000 /* OP CODE 01 (wr) */
#define IXGBE_MSCA_READ                 0x0C000000 /* OP CODE 11 (rd) */
#define IXGBE_MSCA_READ_AUTOINC         0x08000000 /* OP CODE 10 (rd auto inc)*/
#define IXGBE_MSCA_WRITE             0x04000000 /* OP CODE 01 (write) */
#define IXGBE_MSCA_READ              0x0C000000 /* OP CODE 11 (read) */
#define IXGBE_MSCA_READ_AUTOINC      0x08000000 /* OP CODE 10 (read, auto inc)*/
#define IXGBE_MSCA_ST_CODE_MASK         0x30000000 /* ST Code mask */
#define IXGBE_MSCA_ST_CODE_SHIFT        28 /* ST Code shift */
#define IXGBE_MSCA_NEW_PROTOCOL         0x00000000 /* ST CODE 00 (new prot) */
#define IXGBE_MSCA_OLD_PROTOCOL         0x10000000 /* ST CODE 01 (old prot) */
#define IXGBE_MSCA_NEW_PROTOCOL      0x00000000 /* ST CODE 00 (new protocol) */
#define IXGBE_MSCA_OLD_PROTOCOL      0x10000000 /* ST CODE 01 (old protocol) */
#define IXGBE_MSCA_MDI_COMMAND          0x40000000 /* Initiate MDI command */
#define IXGBE_MSCA_MDI_IN_PROG_EN       0x80000000 /* MDI in progress ena */
#define IXGBE_MSCA_MDI_IN_PROG_EN    0x80000000 /* MDI in progress enable */

/* MSRWD bit masks */
#define IXGBE_MSRWD_WRITE_DATA_MASK     0x0000FFFF
#define IXGBE_MSRWD_WRITE_DATA_SHIFT    0
#define IXGBE_MSRWD_READ_DATA_MASK      0xFFFF0000
#define IXGBE_MSRWD_READ_DATA_SHIFT     16

/* Atlas registers */
#define IXGBE_ATLAS_PDN_LPBK            0x24
#define IXGBE_ATLAS_PDN_10G             0xB
#define IXGBE_ATLAS_PDN_1G              0xC
#define IXGBE_ATLAS_PDN_AN              0xD

/* Atlas bit masks */
#define IXGBE_ATLASCTL_WRITE_CMD        0x00010000
#define IXGBE_ATLAS_PDN_TX_REG_EN       0x10
#define IXGBE_ATLAS_PDN_TX_10G_QL_ALL   0xF0
#define IXGBE_ATLAS_PDN_TX_1G_QL_ALL    0xF0
#define IXGBE_ATLAS_PDN_TX_AN_QL_ALL    0xF0

/* Omer bit masks */
#define IXGBE_CORECTL_WRITE_CMD         0x00010000

/* Device Type definitions for new protocol MDIO commands */
#define IXGBE_MDIO_PMA_PMD_DEV_TYPE             0x1
#define IXGBE_MDIO_PCS_DEV_TYPE                 0x3
#define IXGBE_MDIO_PHY_XS_DEV_TYPE              0x4
#define IXGBE_MDIO_AUTO_NEG_DEV_TYPE            0x7
#define IXGBE_MDIO_VENDOR_SPECIFIC_1_DEV_TYPE   0x1E   /* Device 30 */
```

```
1141 #define IXGBE_TWINAX_DEV                        1

1143 #define IXGBE_MDIO_COMMAND_TIMEOUT      100 /* PHY Timeout for 1 GB mode */

1145 #define IXGBE_MDIO_VENDOR_SPECIFIC_1_CONTROL            0x0 /* VS1 Ctrl Reg */
1036 #define IXGBE_MDIO_VENDOR_SPECIFIC_1_CONTROL      0x0    /* VS1 Control Reg */
1146 #define IXGBE_MDIO_VENDOR_SPECIFIC_1_STATUS             0x1 /* VS1 Status Reg */
1147 #define IXGBE_MDIO_VENDOR_SPECIFIC_1_LINK_STATUS        0x0008 /* 1 = Link Up */
1148 #define IXGBE_MDIO_VENDOR_SPECIFIC_1_SPEED_STATUS       0x0010 /* 0-10G, 1-1G */
1039 #define IXGBE_MDIO_VENDOR_SPECIFIC_1_SPEED_STATUS 0x0010 /* 0 - 10G, 1 - 1G */
1149 #define IXGBE_MDIO_VENDOR_SPECIFIC_1_10G_SPEED          0x0018
1150 #define IXGBE_MDIO_VENDOR_SPECIFIC_1_1G_SPEED           0x0010

1152 #define IXGBE_MDIO_AUTO_NEG_CONTROL     0x0 /* AUTO_NEG Control Reg */
1153 #define IXGBE_MDIO_AUTO_NEG_STATUS      0x1 /* AUTO_NEG Status Reg */
1154 #define IXGBE_MDIO_AUTO_NEG_ADVT        0x10 /* AUTO_NEG Advt Reg */
1155 #define IXGBE_MDIO_AUTO_NEG_LP          0x13 /* AUTO_NEG LP Status Reg */
1156 #define IXGBE_MDIO_PHY_XS_CONTROL       0x0 /* PHY_XS Control Reg */
1157 #define IXGBE_MDIO_PHY_XS_RESET         0x8000 /* PHY_XS Reset */
1158 #define IXGBE_MDIO_PHY_ID_HIGH          0x2 /* PHY ID High Reg*/
1159 #define IXGBE_MDIO_PHY_ID_LOW           0x3 /* PHY ID Low Reg*/
1160 #define IXGBE_MDIO_PHY_SPEED_ABILITY    0x4 /* Speed Ability Reg */
1161 #define IXGBE_MDIO_PHY_SPEED_10G        0x0001 /* 10G capable */
1162 #define IXGBE_MDIO_PHY_SPEED_1G         0x0010 /* 1G capable */
1163 #define IXGBE_MDIO_PHY_SPEED_100M       0x0020 /* 100M capable */
1164 #define IXGBE_MDIO_PHY_EXT_ABILITY      0xB /* Ext Ability Reg */
1165 #define IXGBE_MDIO_PHY_10GBASET_ABILITY         0x0004 /* 10GBaseT capable */
1166 #define IXGBE_MDIO_PHY_1000BASET_ABILITY        0x0020 /* 1000BaseT capable */
1167 #define IXGBE_MDIO_PHY_100BASETX_ABILITY        0x0080 /* 100BaseTX capable */
1168 #define IXGBE_MDIO_PHY_SET_LOW_POWER_MODE       0x0800 /* Set low power mode */

1170 #define IXGBE_MDIO_PMA_PMD_CONTROL_ADDR 0x0000 /* PMA/PMD Control Reg */
1171 #define IXGBE_MDIO_PMA_PMD_SDA_SCL_ADDR 0xC30A /* PHY_XS SDA/SCL Addr Reg */
1172 #define IXGBE_MDIO_PMA_PMD_SDA_SCL_DATA 0xC30B /* PHY_XS SDA/SCL Data Reg */
1173 #define IXGBE_MDIO_PMA_PMD_SDA_SCL_STAT 0xC30C /* PHY_XS SDA/SCL Status Reg */

1175 /* MII clause 22/28 definitions */
1176 #define IXGBE_MDIO_PHY_LOW_POWER_MODE   0x0800

1178 #define IXGBE_MII_10GBASE_T_AUTONEG_CTRL_REG    0x20   /* 10G Control Reg */
1179 #define IXGBE_MII_AUTONEG_VENDOR_PROVISION_1_REG 0xC400 /* 1G Provisioning 1 */
1180 #define IXGBE_MII_AUTONEG_XNP_TX_REG            0x17   /* 1G XNP Transmit */
1181 #define IXGBE_MII_AUTONEG_ADVERTISE_REG         0x10   /* 100M Advertisement */
1182 #define IXGBE_MII_10GBASE_T_ADVERTISE           0x1000 /* full duplex, bit:12*/
1183 #define IXGBE_MII_1GBASE_T_ADVERTISE_XNP_TX     0x4000 /* full duplex, bit:14*/
1184 #define IXGBE_MII_1GBASE_T_ADVERTISE            0x8000 /* full duplex, bit:15*/
1185 #define IXGBE_MII_100BASE_T_ADVERTISE           0x0100 /* full duplex, bit:8 */
1186 #define IXGBE_MII_100BASE_T_ADVERTISE_HALF      0x0080 /* half duplex, bit:7 */
1187 #define IXGBE_MII_RESTART                       0x200
1188 #define IXGBE_MII_AUTONEG_COMPLETE              0x20
1189 #define IXGBE_MII_AUTONEG_LINK_UP               0x04
1190 #define IXGBE_MII_AUTONEG_REG                   0x0

1192 #define IXGBE_PHY_REVISION_MASK         0xFFFFFFF0
1193 #define IXGBE_MAX_PHY_ADDR              32

1195 /* PHY IDs*/
1196 #define TN1010_PHY_ID   0x00A19410
1197 #define TNX_FW_REV      0xB
1198 #define X540_PHY_ID     0x01540200
1088 #define AQ1002_PHY_ID    0x03A1B420
1199 #define AQ_FW_REV       0x20
1200 #define QT2022_PHY_ID   0x0043A400
1201 #define ATH_PHY_ID      0x03429050

1203 /* PHY Types */
```



```
1204 #define IXGBE_M88E1145_E_PHY_ID 0x01410CD0

1206 /* Special PHY Init Routine */
1207 #define IXGBE_PHY_INIT_OFFSET_NL        0x002B
1208 #define IXGBE_PHY_INIT_END_NL           0xFFFF
1209 #define IXGBE_CONTROL_MASK_NL           0xF000
1210 #define IXGBE_DATA_MASK_NL              0x0FFF
1211 #define IXGBE_CONTROL_SHIFT_NL          12
1212 #define IXGBE_DELAY_NL                  0
1213 #define IXGBE_DATA_NL                   1
1214 #define IXGBE_CONTROL_NL                0x000F
1215 #define IXGBE_CONTROL_EOL_NL            0x0FFF
1216 #define IXGBE_CONTROL_SOL_NL            0x0000

1218 /* General purpose Interrupt Enable */
1219 #define IXGBE_SDP0_GPIEN        0x00000001 /* SDP0 */
1220 #define IXGBE_SDP1_GPIEN        0x00000002 /* SDP1 */
1221 #define IXGBE_SDP2_GPIEN        0x00000004 /* SDP2 */
1222 #define IXGBE_GPIE_MSIX_MODE    0x00000010 /* MSI-X mode */
1223 #define IXGBE_GPIE_OCD          0x00000020 /* Other Clear Disable */
1224 #define IXGBE_GPIE_EIMEN        0x00000040 /* Immediate Interrupt Enable */
1225 #define IXGBE_GPIE_EIAME        0x40000000
1226 #define IXGBE_GPIE_PBA_SUPPORT  0x80000000
1227 #define IXGBE_GPIE_RSC_DELAY_SHIFT      11
1228 #define IXGBE_GPIE_VTMODE_MASK  0x0000C000 /* VT Mode Mask */
1229 #define IXGBE_GPIE_VTMODE_16    0x00004000 /* 16 VFs 8 queues per VF */
1230 #define IXGBE_GPIE_VTMODE_32    0x00008000 /* 32 VFs 4 queues per VF */
1231 #define IXGBE_GPIE_VTMODE_64    0x0000C000 /* 64 VFs 2 queues per VF */

1233 /* Packet Buffer Initialization */
1234 #define IXGBE_MAX_PACKET_BUFFERS        8

1236 #define IXGBE_TXPBSIZE_20KB     0x00005000 /* 20KB Packet Buffer */
1237 #define IXGBE_TXPBSIZE_40KB     0x0000A000 /* 40KB Packet Buffer */
1238 #define IXGBE_RXPBSIZE_48KB     0x0000C000 /* 48KB Packet Buffer */
1239 #define IXGBE_RXPBSIZE_64KB     0x00010000 /* 64KB Packet Buffer */
1240 #define IXGBE_RXPBSIZE_80KB     0x00014000 /* 80KB Packet Buffer */
1241 #define IXGBE_RXPBSIZE_128KB    0x00020000 /* 128KB Packet Buffer */
1242 #define IXGBE_RXPBSIZE_MAX      0x00080000 /* 512KB Packet Buffer */
1243 #define IXGBE_TXPBSIZE_MAX      0x00028000 /* 160KB Packet Buffer */

1245 #define IXGBE_TXPKT_SIZE_MAX    0xA /* Max Tx Packet size */
1246 #define IXGBE_MAX_PB            8

1248 /* Packet buffer allocation strategies */
1249 enum {
1250         PBA_STRATEGY_EQUAL      = 0, /* Distribute PB space equally */
1251 #define PBA_STRATEGY_EQUAL      PBA_STRATEGY_EQUAL
1252         PBA_STRATEGY_WEIGHTED   = 1, /* Weight front half of TCs */
1253 #define PBA_STRATEGY_WEIGHTED   PBA_STRATEGY_WEIGHTED
1254 };

1256 /* Transmit Flow Control status */
1257 #define IXGBE_TFCS_TXOFF        0x00000001
1258 #define IXGBE_TFCS_TXOFF0       0x00000100
1259 #define IXGBE_TFCS_TXOFF1       0x00000200
1260 #define IXGBE_TFCS_TXOFF2       0x00000400
1261 #define IXGBE_TFCS_TXOFF3       0x00000800
1262 #define IXGBE_TFCS_TXOFF4       0x00001000
1263 #define IXGBE_TFCS_TXOFF5       0x00002000
1264 #define IXGBE_TFCS_TXOFF6       0x00004000
1265 #define IXGBE_TFCS_TXOFF7       0x00008000

1267 /* TCP Timer */
1268 #define IXGBE_TCPTIMER_KS               0x00000100
1269 #define IXGBE_TCPTIMER_COUNT_ENABLE     0x00000200
```

```
1270 #define IXGBE_TCPTIMER_COUNT_FINISH     0x00000400
1271 #define IXGBE_TCPTIMER_LOOP             0x00000800
1272 #define IXGBE_TCPTIMER_DURATION_MASK    0x000000FF

1274 /* HLREG0 Bit Masks */
1275 #define IXGBE_HLREG0_TXCRCEN            0x00000001 /* bit  0 */
1276 #define IXGBE_HLREG0_RXCRCSTRP          0x00000002 /* bit  1 */
1277 #define IXGBE_HLREG0_JUMBOEN            0x00000004 /* bit  2 */
1278 #define IXGBE_HLREG0_TXPADEN            0x00000400 /* bit 10 */
1279 #define IXGBE_HLREG0_TXPAUSEEN          0x00001000 /* bit 12 */
1280 #define IXGBE_HLREG0_RXPAUSEEN          0x00004000 /* bit 14 */
1281 #define IXGBE_HLREG0_LPBK               0x00008000 /* bit 15 */
1282 #define IXGBE_HLREG0_MDCSPD             0x00010000 /* bit 16 */
1283 #define IXGBE_HLREG0_CONTMDC            0x00020000 /* bit 17 */
1284 #define IXGBE_HLREG0_CTRLFLTR           0x00040000 /* bit 18 */
1285 #define IXGBE_HLREG0_PREPEND            0x00F00000 /* bits 20-23 */
1286 #define IXGBE_HLREG0_PRIPAUSEEN         0x01000000 /* bit 24 */
1287 #define IXGBE_HLREG0_RXPAUSERECDA       0x06000000 /* bits 25-26 */
1288 #define IXGBE_HLREG0_RXLNGTHERREN       0x08000000 /* bit 27 */
1289 #define IXGBE_HLREG0_RXPADSTRIPEN       0x10000000 /* bit 28 */

1291 /* VMD_CTL bitmasks */
1292 #define IXGBE_VMD_CTL_VMDQ_EN           0x00000001
1293 #define IXGBE_VMD_CTL_VMDQ_FILTER       0x00000002

1295 /* VT_CTL bitmasks */
1296 #define IXGBE_VT_CTL_DIS_DEFPL          0x20000000 /* disable default pool */
1297 #define IXGBE_VT_CTL_REPLEN             0x40000000 /* replication enabled */
1298 #define IXGBE_VT_CTL_VT_ENABLE          0x00000001  /* Enable VT Mode */
1299 #define IXGBE_VT_CTL_POOL_SHIFT         7
1300 #define IXGBE_VT_CTL_POOL_MASK          (0x3F << IXGBE_VT_CTL_POOL_SHIFT)

1302 /* VMOLR bitmasks */
1303 #define IXGBE_VMOLR_AUPE        0x01000000 /* accept untagged packets */
1304 #define IXGBE_VMOLR_ROMPE       0x02000000 /* accept packets in MTA tbl */
1305 #define IXGBE_VMOLR_ROPE        0x04000000 /* accept packets in UC tbl */
1306 #define IXGBE_VMOLR_BAM         0x08000000 /* accept broadcast packets */
1307 #define IXGBE_VMOLR_MPE         0x10000000 /* multicast promiscuous */

1309 /* VFRE bitmask */
1310 #define IXGBE_VFRE_ENABLE_ALL   0xFFFFFFFF

1312 #define IXGBE_VF_INIT_TIMEOUT   200 /* Number of retries to clear RSTI */

1314 /* RDHMPN and TDHMPN bitmasks */
1315 #define IXGBE_RDHMPN_RDICADDR           0x007FF800
1316 #define IXGBE_RDHMPN_RDICRDREQ          0x00800000
1317 #define IXGBE_RDHMPN_RDICADDR_SHIFT     11
1318 #define IXGBE_TDHMPN_TDICADDR           0x003FF800
1319 #define IXGBE_TDHMPN_TDICRDREQ          0x00800000
1320 #define IXGBE_TDHMPN_TDICADDR_SHIFT     11

1322 #define IXGBE_RDMAM_MEM_SEL_SHIFT               13
1323 #define IXGBE_RDMAM_DWORD_SHIFT                 9
1324 #define IXGBE_RDMAM_DESC_COMP_FIFO              1
1325 #define IXGBE_RDMAM_DFC_CMD_FIFO                2
1326 #define IXGBE_RDMAM_RSC_HEADER_ADDR             3
1327 #define IXGBE_RDMAM_TCN_STATUS_RAM              4
1328 #define IXGBE_RDMAM_WB_COLL_FIFO                5
1329 #define IXGBE_RDMAM_QSC_CNT_RAM                 6
1330 #define IXGBE_RDMAM_QSC_FCOE_RAM                7
1331 #define IXGBE_RDMAM_QSC_QUEUE_CNT               8
1332 #define IXGBE_RDMAM_QSC_QUEUE_RAM               0xA
1333 #define IXGBE_RDMAM_QSC_RSC_RAM                 0xB
1334 #define IXGBE_RDMAM_DESC_COM_FIFO_RANGE         135
1335 #define IXGBE_RDMAM_DESC_COM_FIFO_COUNT         4
```



```
1336 #define IXGBE_RDMAM_DFC_CMD_FIFO_RANGE          48
1337 #define IXGBE_RDMAM_DFC_CMD_FIFO_COUNT          7
1338 #define IXGBE_RDMAM_RSC_HEADER_ADDR_RANGE       32
1339 #define IXGBE_RDMAM_RSC_HEADER_ADDR_COUNT       4
1340 #define IXGBE_RDMAM_TCN_STATUS_RAM_RANGE        256
1341 #define IXGBE_RDMAM_TCN_STATUS_RAM_COUNT        9
1342 #define IXGBE_RDMAM_WB_COLL_FIFO_RANGE          8
1343 #define IXGBE_RDMAM_WB_COLL_FIFO_COUNT          4
1344 #define IXGBE_RDMAM_QSC_CNT_RAM_RANGE           64
1345 #define IXGBE_RDMAM_QSC_CNT_RAM_COUNT           4
1346 #define IXGBE_RDMAM_QSC_FCOE_RAM_RANGE          512
1347 #define IXGBE_RDMAM_QSC_FCOE_RAM_COUNT          5
1348 #define IXGBE_RDMAM_QSC_QUEUE_CNT_RANGE         32
1349 #define IXGBE_RDMAM_QSC_QUEUE_CNT_COUNT         4
1350 #define IXGBE_RDMAM_QSC_QUEUE_RAM_RANGE         128
1351 #define IXGBE_RDMAM_QSC_QUEUE_RAM_COUNT         8
1352 #define IXGBE_RDMAM_QSC_RSC_RAM_RANGE           32
1353 #define IXGBE_RDMAM_QSC_RSC_RAM_COUNT           8

1355 #define IXGBE_TXDESCIC_READY    0x80000000

1357 /* Receive Checksum Control */
1358 #define IXGBE_RXCSUM_IPPCSE     0x00001000 /* IP payload checksum enable */
1359 #define IXGBE_RXCSUM_PCSD       0x00002000 /* packet checksum disabled */

1361 /* FCRTL Bit Masks */
1362 #define IXGBE_FCRTL_XONE        0x80000000 /* XON enable */
1363 #define IXGBE_FCRTH_FCEN        0x80000000 /* Packet buffer fc enable */

1365 /* PAP bit masks*/
1366 #define IXGBE_PAP_TXPAUSECNT_MASK       0x0000FFFF /* Pause counter mask */

1368 /* RMCS Bit Masks */
1369 #define IXGBE_RMCS_RRM                  0x00000002 /* Rx Recycle Mode enable */
1236 #define IXGBE_RMCS_RRM          0x00000002 /* Receive Recycle Mode enable */
1370 /* Receive Arbitration Control: 0 Round Robin, 1 DFP */
1371 #define IXGBE_RMCS_RAC                  0x00000004
1372 /* Deficit Fixed Prio ena */
1373 #define IXGBE_RMCS_DFP                  IXGBE_RMCS_RAC
1239 #define IXGBE_RMCS_DFP          IXGBE_RMCS_RAC /* Deficit Fixed Priority ena */
1374 #define IXGBE_RMCS_TFCE_802_3X          0x00000008 /* Tx Priority FC ena */
1375 #define IXGBE_RMCS_TFCE_PRIORITY        0x00000010 /* Tx Priority FC ena */
1376 #define IXGBE_RMCS_ARBDIS               0x00000040 /* Arbitration disable bit */

1378 /* FCCFG Bit Masks */
1379 #define IXGBE_FCCFG_TFCE_802_3X         0x00000008 /* Tx link FC enable */
1380 #define IXGBE_FCCFG_TFCE_PRIORITY       0x00000010 /* Tx priority FC enable */

1382 /* Interrupt register bitmasks */

1384 /* Extended Interrupt Cause Read */
1385 #define IXGBE_EICR_RTX_QUEUE    0x0000FFFF /* RTx Queue Interrupt */
1386 #define IXGBE_EICR_FLOW_DIR     0x00010000 /* FDir Exception */
1387 #define IXGBE_EICR_RX_MISS      0x00020000 /* Packet Buffer Overrun */
1388 #define IXGBE_EICR_PCI          0x00040000 /* PCI Exception */
1389 #define IXGBE_EICR_MAILBOX      0x00080000 /* VF to PF Mailbox Interrupt */
1390 #define IXGBE_EICR_LSC          0x00100000 /* Link Status Change */
1391 #define IXGBE_EICR_LINKSEC      0x00200000 /* PN Threshold */
1392 #define IXGBE_EICR_MNG          0x00400000 /* Manageability Event Interrupt */
1393 #define IXGBE_EICR_TS           0x00800000 /* Thermal Sensor Event */
1394 #define IXGBE_EICR_TIMESYNC     0x01000000 /* Timesync Event */
1395 #define IXGBE_EICR_GPI_SDP0     0x01000000 /* Gen Purpose Interrupt on SDP0 */
1396 #define IXGBE_EICR_GPI_SDP1     0x02000000 /* Gen Purpose Interrupt on SDP1 */
1397 #define IXGBE_EICR_GPI_SDP2     0x04000000 /* Gen Purpose Interrupt on SDP2 */
1398 #define IXGBE_EICR_ECC          0x10000000 /* ECC Error */
1399 #define IXGBE_EICR_PBUR         0x10000000 /* Packet Buffer Handler Error */
```

```
#define IXGBE_EICR_DHER         0x20000000 /* Descriptor Handler Error */
#define IXGBE_EICR_TCP_TIMER    0x40000000 /* TCP Timer */
#define IXGBE_EICR_OTHER        0x80000000 /* Interrupt Cause Active */

/* Extended Interrupt Cause Set */
#define IXGBE_EICS_RTX_QUEUE    IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
#define IXGBE_EICS_FLOW_DIR     IXGBE_EICR_FLOW_DIR  /* FDir Exception */
#define IXGBE_EICS_RX_MISS      IXGBE_EICR_RX_MISS   /* Pkt Buffer Overrun */
#define IXGBE_EICS_PCI          IXGBE_EICR_PCI /* PCI Exception */
#define IXGBE_EICS_MAILBOX      IXGBE_EICR_MAILBOX   /* VF to PF Mailbox Int */
#define IXGBE_EICS_LSC          IXGBE_EICR_LSC /* Link Status Change */
#define IXGBE_EICS_MNG          IXGBE_EICR_MNG /* MNG Event Interrupt */
#define IXGBE_EICS_TIMESYNC     IXGBE_EICR_TIMESYNC /* Timesync Event */
#define IXGBE_EICS_GPI_SDP0     IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
#define IXGBE_EICS_GPI_SDP1     IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
#define IXGBE_EICS_GPI_SDP2     IXGBE_EICR_GPI_SDP2 /* SDP2 Gen Purpose Int */
#define IXGBE_EICS_ECC          IXGBE_EICR_ECC /* ECC Error */
#define IXGBE_EICS_PBUR         IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
#define IXGBE_EICS_DHER         IXGBE_EICR_DHER /* Desc Handler Error */
#define IXGBE_EICS_TCP_TIMER    IXGBE_EICR_TCP_TIMER /* TCP Timer */
#define IXGBE_EICS_OTHER        IXGBE_EICR_OTHER /* INT Cause Active */

/* Extended Interrupt Mask Set */
#define IXGBE_EIMS_RTX_QUEUE    IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
#define IXGBE_EIMS_FLOW_DIR     IXGBE_EICR_FLOW_DIR /* FDir Exception */
#define IXGBE_EIMS_RX_MISS      IXGBE_EICR_RX_MISS /* Packet Buffer Overrun */
#define IXGBE_EIMS_PCI          IXGBE_EICR_PCI /* PCI Exception */
#define IXGBE_EIMS_MAILBOX      IXGBE_EICR_MAILBOX   /* VF to PF Mailbox Int */
#define IXGBE_EIMS_LSC          IXGBE_EICR_LSC /* Link Status Change */
#define IXGBE_EIMS_MNG          IXGBE_EICR_MNG /* MNG Event Interrupt */
#define IXGBE_EIMS_TS           IXGBE_EICR_TS /* Thermal Sensor Event */
#define IXGBE_EIMS_TIMESYNC     IXGBE_EICR_TIMESYNC /* Timesync Event */
#define IXGBE_EIMS_GPI_SDP0     IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
#define IXGBE_EIMS_GPI_SDP1     IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
#define IXGBE_EIMS_GPI_SDP2     IXGBE_EICR_GPI_SDP2 /* SDP2 Gen Purpose Int */
#define IXGBE_EIMS_ECC          IXGBE_EICR_ECC /* ECC Error */
#define IXGBE_EIMS_PBUR         IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
#define IXGBE_EIMS_DHER         IXGBE_EICR_DHER /* Descr Handler Error */
#define IXGBE_EIMS_TCP_TIMER    IXGBE_EICR_TCP_TIMER /* TCP Timer */
#define IXGBE_EIMS_OTHER        IXGBE_EICR_OTHER /* INT Cause Active */

/* Extended Interrupt Mask Clear */
#define IXGBE_EIMC_RTX_QUEUE    IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
#define IXGBE_EIMC_FLOW_DIR     IXGBE_EICR_FLOW_DIR /* FDir Exception */
#define IXGBE_EIMC_RX_MISS      IXGBE_EICR_RX_MISS /* Packet Buffer Overrun */
#define IXGBE_EIMC_PCI          IXGBE_EICR_PCI /* PCI Exception */
#define IXGBE_EIMC_MAILBOX      IXGBE_EICR_MAILBOX /* VF to PF Mailbox Int */
#define IXGBE_EIMC_LSC          IXGBE_EICR_LSC /* Link Status Change */
#define IXGBE_EIMC_MNG          IXGBE_EICR_MNG /* MNG Event Interrupt */
#define IXGBE_EIMC_TIMESYNC     IXGBE_EICR_TIMESYNC /* Timesync Event */
#define IXGBE_EIMC_GPI_SDP0     IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
#define IXGBE_EIMC_GPI_SDP1     IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
#define IXGBE_EIMC_GPI_SDP2     IXGBE_EICR_GPI_SDP2  /* SDP2 Gen Purpose Int */
#define IXGBE_EIMC_ECC          IXGBE_EICR_ECC /* ECC Error */
#define IXGBE_EIMC_PBUR         IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
#define IXGBE_EIMC_DHER         IXGBE_EICR_DHER /* Desc Handler Err */
#define IXGBE_EIMC_TCP_TIMER    IXGBE_EICR_TCP_TIMER /* TCP Timer */
#define IXGBE_EIMC_OTHER        IXGBE_EICR_OTHER /* INT Cause Active */

#define IXGBE_EIMS_ENABLE_MASK ( \
                                IXGBE_EIMS_RTX_QUEUE    | \
                                IXGBE_EIMS_LSC          | \
                                IXGBE_EIMS_TCP_TIMER    | \
                                IXGBE_EIMS_OTHER)

/* Immediate Interrupt Rx (A.K.A. Low Latency Interrupt) */
```



```
#define IXGBE_IMIR_PORT_IM_EN   0x00010000  /* TCP port enable */
#define IXGBE_IMIR_PORT_BP      0x00020000  /* TCP port check bypass */
#define IXGBE_IMIREXT_SIZE_BP   0x00001000  /* Packet size bypass */
#define IXGBE_IMIREXT_CTRL_URG  0x00002000  /* Check URG bit in header */
#define IXGBE_IMIREXT_CTRL_ACK  0x00004000  /* Check ACK bit in header */
#define IXGBE_IMIREXT_CTRL_PSH  0x00008000  /* Check PSH bit in header */
#define IXGBE_IMIREXT_CTRL_RST  0x00010000  /* Check RST bit in header */
#define IXGBE_IMIREXT_CTRL_SYN  0x00020000  /* Check SYN bit in header */
#define IXGBE_IMIREXT_CTRL_FIN  0x00040000  /* Check FIN bit in header */
#define IXGBE_IMIREXT_CTRL_BP   0x00080000  /* Bypass check of control bits */
#define IXGBE_IMIR_SIZE_BP_82599        0x00001000 /* Packet size bypass */
#define IXGBE_IMIR_CTRL_URG_82599       0x00002000 /* Check URG bit in header */
#define IXGBE_IMIR_CTRL_ACK_82599       0x00004000 /* Check ACK bit in header */
#define IXGBE_IMIR_CTRL_PSH_82599       0x00008000 /* Check PSH bit in header */
#define IXGBE_IMIR_CTRL_RST_82599       0x00010000 /* Check RST bit in header */
#define IXGBE_IMIR_CTRL_SYN_82599       0x00020000 /* Check SYN bit in header */
#define IXGBE_IMIR_CTRL_FIN_82599       0x00040000 /* Check FIN bit in header */
#define IXGBE_IMIR_CTRL_BP_82599        0x00080000 /* Bypass chk of ctrl bits */
#define IXGBE_IMIR_CTRL_BP_82599  0x00080000 /* Bypass check of control bits */
#define IXGBE_IMIR_LLI_EN_82599         0x00100000 /* Enables low latency Int */
#define IXGBE_IMIR_RX_QUEUE_MASK_82599  0x0000007F /* Rx Queue Mask */
#define IXGBE_IMIR_RX_QUEUE_SHIFT_82599 21 /* Rx Queue Shift */
#define IXGBE_IMIRVP_PRIORITY_MASK      0x00000007 /* VLAN priority mask */
#define IXGBE_IMIRVP_PRIORITY_EN        0x00000008 /* VLAN priority enable */

#define IXGBE_MAX_FTQF_FILTERS          128
#define IXGBE_FTQF_PROTOCOL_MASK        0x00000003
#define IXGBE_FTQF_PROTOCOL_TCP         0x00000000
#define IXGBE_FTQF_PROTOCOL_UDP         0x00000001
#define IXGBE_FTQF_PROTOCOL_SCTP        2
#define IXGBE_FTQF_PRIORITY_MASK        0x00000007
#define IXGBE_FTQF_PRIORITY_SHIFT       2
#define IXGBE_FTQF_POOL_MASK            0x0000003F
#define IXGBE_FTQF_POOL_SHIFT           8
#define IXGBE_FTQF_5TUPLE_MASK_MASK     0x0000001F
#define IXGBE_FTQF_5TUPLE_MASK_SHIFT    25
#define IXGBE_FTQF_SOURCE_ADDR_MASK     0x1E
#define IXGBE_FTQF_DEST_ADDR_MASK       0x1D
#define IXGBE_FTQF_SOURCE_PORT_MASK     0x1B
#define IXGBE_FTQF_DEST_PORT_MASK       0x17
#define IXGBE_FTQF_PROTOCOL_COMP_MASK   0x0F
#define IXGBE_FTQF_POOL_MASK_EN         0x40000000
#define IXGBE_FTQF_QUEUE_ENABLE         0x80000000

/* Interrupt clear mask */
#define IXGBE_IRQ_CLEAR_MASK    0xFFFFFFFF

/* Interrupt Vector Allocation Registers */
#define IXGBE_IVAR_REG_NUM              25
#define IXGBE_IVAR_REG_NUM_82599        64
#define IXGBE_IVAR_TXRX_ENTRY           96
#define IXGBE_IVAR_RX_ENTRY             64
#define IXGBE_IVAR_RX_QUEUE(_i)         (0 + (_i))
#define IXGBE_IVAR_TX_QUEUE(_i)         (64 + (_i))
#define IXGBE_IVAR_TX_ENTRY             32

#define IXGBE_IVAR_TCP_TIMER_INDEX      96 /* 0 based index */
#define IXGBE_IVAR_OTHER_CAUSES_INDEX   97 /* 0 based index */

#define IXGBE_MSIX_VECTOR(_i)           (0 + (_i))

#define IXGBE_IVAR_ALLOC_VAL            0x80 /* Interrupt Allocation valid */

/* ETYPE Queue Filter/Select Bit Masks */
#define IXGBE_MAX_ETQF_FILTERS          8
#define IXGBE_ETQF_FCOE                 0x08000000 /* bit 27 */
```

```
#define IXGBE_ETQF_BCN                  0x10000000 /* bit 28 */
#define IXGBE_ETQF_1588                 0x40000000 /* bit 30 */
#define IXGBE_ETQF_FILTER_EN            0x80000000 /* bit 31 */
#define IXGBE_ETQF_POOL_ENABLE          (1 << 26) /* bit 26 */
#define IXGBE_ETQF_POOL_SHIFT           20

#define IXGBE_ETQS_RX_QUEUE             0x007F0000 /* bits 22:16 */
#define IXGBE_ETQS_RX_QUEUE_SHIFT       16
#define IXGBE_ETQS_LLI                  0x20000000 /* bit 29 */
#define IXGBE_ETQS_QUEUE_EN             0x80000000 /* bit 31 */

/*
 * ETQF filter list: one static filter per filter consumer. This is
 *                 to avoid filter collisions later. Add new filters
 *                 here!!
 *
 * Current filters:
 *      EAPOL 802.1x (0x888e): Filter 0
 *      FCoE (0x8906):   Filter 2
 *      1588 (0x88f7):   Filter 3
 *      FIP  (0x8914):   Filter 4
 */
#define IXGBE_ETQF_FILTER_EAPOL         0
#define IXGBE_ETQF_FILTER_FCOE          2
#define IXGBE_ETQF_FILTER_1588          3
#define IXGBE_ETQF_FILTER_FIP           4
/* VLAN Control Bit Masks */
#define IXGBE_VLNCTRL_VET               0x0000FFFF  /* bits 0-15 */
#define IXGBE_VLNCTRL_CFI               0x10000000  /* bit 28 */
#define IXGBE_VLNCTRL_CFIEN             0x20000000  /* bit 29 */
#define IXGBE_VLNCTRL_VFE               0x40000000  /* bit 30 */
#define IXGBE_VLNCTRL_VME               0x80000000  /* bit 31 */

/* VLAN pool filtering masks */
#define IXGBE_VLVF_VIEN                 0x80000000  /* filter is valid */
#define IXGBE_VLVF_ENTRIES              64
#define IXGBE_VLVF_VLANID_MASK          0x00000FFF
/* Per VF Port VLAN insertion rules */
#define IXGBE_VMVIR_VLANA_DEFAULT       0x40000000 /* Always use default VLAN */
#define IXGBE_VMVIR_VLANA_NEVER         0x80000000 /* Never insert VLAN tag */

#define IXGBE_ETHERNET_IEEE_VLAN_TYPE   0x8100  /* 802.1q protocol */

/* STATUS Bit Masks */
#define IXGBE_STATUS_LAN_ID             0x0000000C /* LAN ID */
#define IXGBE_STATUS_LAN_ID_SHIFT       2 /* LAN ID Shift*/
#define IXGBE_STATUS_GIO                0x00080000 /* GIO Master Ena Status */
#define IXGBE_STATUS_GIO            0x00080000 /* GIO Master Enable Status */

#define IXGBE_STATUS_LAN_ID_0   0x00000000 /* LAN ID 0 */
#define IXGBE_STATUS_LAN_ID_1   0x00000004 /* LAN ID 1 */

/* ESDP Bit Masks */
#define IXGBE_ESDP_SDP0         0x00000001 /* SDP0 Data Value */
#define IXGBE_ESDP_SDP1         0x00000002 /* SDP1 Data Value */
#define IXGBE_ESDP_SDP2         0x00000004 /* SDP2 Data Value */
#define IXGBE_ESDP_SDP3         0x00000008 /* SDP3 Data Value */
#define IXGBE_ESDP_SDP4         0x00000010 /* SDP4 Data Value */
#define IXGBE_ESDP_SDP5         0x00000020 /* SDP5 Data Value */
#define IXGBE_ESDP_SDP6         0x00000040 /* SDP6 Data Value */
#define IXGBE_ESDP_SDP7         0x00000080 /* SDP7 Data Value */
#define IXGBE_ESDP_SDP0_DIR     0x00000100 /* SDP0 IO direction */
#define IXGBE_ESDP_SDP1_DIR     0x00000200 /* SDP1 IO direction */
#define IXGBE_ESDP_SDP3_DIR     0x00000800 /* SDP3 IO direction */
#define IXGBE_ESDP_SDP4_DIR     0x00001000 /* SDP4 IO direction */
#define IXGBE_ESDP_SDP4_DIR     0x00000004 /* SDP4 IO direction */
```



```
#define IXGBE_ESDP_SDP5_DIR     0x00002000 /* SDP5 IO direction */
#define IXGBE_ESDP_SDP6_DIR     0x00004000 /* SDP6 IO direction */
#define IXGBE_ESDP_SDP7_DIR     0x00008000 /* SDP7 IO direction */
#define IXGBE_ESDP_SDP0_NATIVE  0x00010000 /* SDP0 IO mode */
#define IXGBE_ESDP_SDP1_NATIVE  0x00020000 /* SDP1 IO mode */


/* LEDCTL Bit Masks */
#define IXGBE_LED_IVRT_BASE             0x00000040
#define IXGBE_LED_BLINK_BASE            0x00000080
#define IXGBE_LED_MODE_MASK_BASE        0x0000000F
#define IXGBE_LED_OFFSET(_base, _i)     (_base << (8 * (_i)))
#define IXGBE_LED_MODE_SHIFT(_i)        (8*(_i))
#define IXGBE_LED_IVRT(_i)      IXGBE_LED_OFFSET(IXGBE_LED_IVRT_BASE, _i)
#define IXGBE_LED_BLINK(_i)     IXGBE_LED_OFFSET(IXGBE_LED_BLINK_BASE, _i)
#define IXGBE_LED_MODE_MASK(_i) IXGBE_LED_OFFSET(IXGBE_LED_MODE_MASK_BASE, _i)

/* LED modes */
#define IXGBE_LED_LINK_UP       0x0
#define IXGBE_LED_LINK_10G      0x1
#define IXGBE_LED_MAC           0x2
#define IXGBE_LED_FILTER        0x3
#define IXGBE_LED_LINK_ACTIVE   0x4
#define IXGBE_LED_LINK_1G       0x5
#define IXGBE_LED_ON            0xE
#define IXGBE_LED_OFF           0xF

/* AUTOC Bit Masks */
#define IXGBE_AUTOC_KX4_KX_SUPP_MASK 0xC0000000
#define IXGBE_AUTOC_KX4_SUPP    0x80000000
#define IXGBE_AUTOC_KX_SUPP     0x40000000
#define IXGBE_AUTOC_PAUSE       0x30000000
#define IXGBE_AUTOC_ASM_PAUSE   0x20000000
#define IXGBE_AUTOC_SYM_PAUSE   0x10000000
#define IXGBE_AUTOC_RF          0x08000000
#define IXGBE_AUTOC_PD_TMR      0x06000000
#define IXGBE_AUTOC_AN_RX_LOOSE 0x01000000
#define IXGBE_AUTOC_AN_RX_DRIFT 0x00800000
#define IXGBE_AUTOC_AN_RX_ALIGN 0x007C0000
#define IXGBE_AUTOC_FECA        0x00040000
#define IXGBE_AUTOC_FECR        0x00020000
#define IXGBE_AUTOC_KR_SUPP     0x00010000
#define IXGBE_AUTOC_AN_RESTART  0x00001000
#define IXGBE_AUTOC_FLU         0x00000001
#define IXGBE_AUTOC_LMS_SHIFT   13
#define IXGBE_AUTOC_LMS_10G_SERIAL      (0x3 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_KX4_KX_KR       (0x4 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_SGMII_1G_100M   (0x5 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN (0x6 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_KX4_KX_KR_SGMII (0x7 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_MASK            (0x7 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_1G_LINK_NO_AN   (0x0 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_10G_LINK_NO_AN  (0x1 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_1G_AN           (0x2 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_KX4_AN          (0x4 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_KX4_AN_1G_AN    (0x6 << IXGBE_AUTOC_LMS_SHIFT)
#define IXGBE_AUTOC_LMS_ATTACH_TYPE     (0x7 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)

#define IXGBE_AUTOC_1G_PMA_PMD_MASK     0x00000200
#define IXGBE_AUTOC_1G_PMA_PMD_SHIFT    9
#define IXGBE_AUTOC_10G_PMA_PMD_MASK    0x00000180
#define IXGBE_AUTOC_10G_PMA_PMD_SHIFT   7
#define IXGBE_AUTOC_10G_XAUI    (0x0 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
#define IXGBE_AUTOC_10G_KX4     (0x1 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
#define IXGBE_AUTOC_10G_CX4     (0x2 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
#define IXGBE_AUTOC_1G_BX       (0x0 << IXGBE_AUTOC_1G_PMA_PMD_SHIFT)
```

```
1661 #define IXGBE_AUTOC_1G_KX       (0x1 << IXGBE_AUTOC_1G_PMA_PMD_SHIFT)
1662 #define IXGBE_AUTOC_1G_SFI      (0x0 << IXGBE_AUTOC_1G_PMA_PMD_SHIFT)
1663 #define IXGBE_AUTOC_1G_KX_BX    (0x1 << IXGBE_AUTOC_1G_PMA_PMD_SHIFT)

1665 #define IXGBE_AUTOC2_UPPER_MASK 0xFFFF0000
1666 #define IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_MASK    0x00030000
1667 #define IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_SHIFT   16
1668 #define IXGBE_AUTOC2_10G_KR     (0x0 << IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_SHIFT)
1669 #define IXGBE_AUTOC2_10G_XFI    (0x1 << IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_SHIFT)
1670 #define IXGBE_AUTOC2_10G_SFI    (0x2 << IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_SHIFT)

1672 #define IXGBE_MACC_FLU          0x00000001
1673 #define IXGBE_MACC_FSV_10G      0x00030000
1674 #define IXGBE_MACC_FS           0x00040000
1675 #define IXGBE_MAC_RX2TX_LPBK    0x00000002

1677 /* LINKS Bit Masks */
1678 #define IXGBE_LINKS_KX_AN_COMP  0x80000000
1679 #define IXGBE_LINKS_UP          0x40000000
1680 #define IXGBE_LINKS_SPEED       0x20000000
1681 #define IXGBE_LINKS_MODE        0x18000000
1682 #define IXGBE_LINKS_RX_MODE     0x06000000
1683 #define IXGBE_LINKS_TX_MODE     0x01800000
1684 #define IXGBE_LINKS_XGXS_EN     0x00400000
1685 #define IXGBE_LINKS_SGMII_EN    0x02000000
1686 #define IXGBE_LINKS_PCS_1G_EN   0x00200000
1687 #define IXGBE_LINKS_1G_AN_EN    0x00100000
1688 #define IXGBE_LINKS_KX_AN_IDLE  0x00080000
1689 #define IXGBE_LINKS_1G_SYNC     0x00040000
1690 #define IXGBE_LINKS_10G_ALIGN   0x00020000
1691 #define IXGBE_LINKS_10G_LANE_SYNC       0x00017000
1692 #define IXGBE_LINKS_TL_FAULT            0x00001000
1693 #define IXGBE_LINKS_SIGNAL              0x00000F00

1695 #define IXGBE_LINKS_SPEED_82599         0x30000000
1696 #define IXGBE_LINKS_SPEED_10G_82599     0x30000000
1697 #define IXGBE_LINKS_SPEED_1G_82599      0x20000000
1698 #define IXGBE_LINKS_SPEED_100_82599     0x10000000
1699 #define IXGBE_LINK_UP_TIME              90 /* 9.0 Seconds */
1700 #define IXGBE_AUTO_NEG_TIME             45 /* 4.5 Seconds */

1702 #define IXGBE_LINKS2_AN_SUPPORTED       0x00000040

1704 /* PCS1GLSTA Bit Masks */
1705 #define IXGBE_PCS1GLSTA_LINK_OK         1
1706 #define IXGBE_PCS1GLSTA_SYNK_OK         0x10
1707 #define IXGBE_PCS1GLSTA_AN_COMPLETE     0x10000
1708 #define IXGBE_PCS1GLSTA_AN_PAGE_RX      0x20000
1709 #define IXGBE_PCS1GLSTA_AN_TIMED_OUT    0x40000
1710 #define IXGBE_PCS1GLSTA_AN_REMOTE_FAULT 0x80000
1711 #define IXGBE_PCS1GLSTA_AN_ERROR_RWS    0x100000

1713 #define IXGBE_PCS1GANA_SYM_PAUSE        0x80
1714 #define IXGBE_PCS1GANA_ASM_PAUSE        0x100

1716 /* PCS1GLCTL Bit Masks */
1717 #define IXGBE_PCS1GLCTL_AN_1G_TIMEOUT_EN 0x00040000 /* PCS 1G autoneg to en */
1718 #define IXGBE_PCS1GLCTL_FLV_LINK_UP     1
1719 #define IXGBE_PCS1GLCTL_FORCE_LINK      0x20
1720 #define IXGBE_PCS1GLCTL_LOW_LINK_LATCH  0x40
1721 #define IXGBE_PCS1GLCTL_AN_ENABLE       0x10000
1722 #define IXGBE_PCS1GLCTL_AN_RESTART      0x20000

1724 /* ANLP1 Bit Masks */
1725 #define IXGBE_ANLP1_PAUSE               0x0C00
1726 #define IXGBE_ANLP1_SYM_PAUSE           0x0400
```



```
1727 #define IXGBE_ANLP1_ASM_PAUSE           0x0800
1728 #define IXGBE_ANLP1_AN_STATE_MASK       0x000f0000

1730 /* SW Semaphore Register bitmasks */
1731 #define IXGBE_SWSM_SMBI         0x00000001 /* Driver Semaphore bit */
1732 #define IXGBE_SWSM_SWESMBI      0x00000002 /* FW Semaphore bit */
1733 #define IXGBE_SWSM_WMNG         0x00000004 /* Wake MNG Clock */
1734 #define IXGBE_SWFW_REGSMP       0x80000000 /* Register Semaphore bit 31 */

1736 /* SW_FW_SYNC/GSSR definitions */
1737 #define IXGBE_GSSR_EEP_SM       0x0001
1738 #define IXGBE_GSSR_PHY0_SM      0x0002
1739 #define IXGBE_GSSR_PHY1_SM      0x0004
1740 #define IXGBE_GSSR_MAC_CSR_SM   0x0008
1741 #define IXGBE_GSSR_FLASH_SM     0x0010
1742 #define IXGBE_GSSR_SW_MNG_SM    0x0400

1744 /* FW Status register bitmask */
1745 #define IXGBE_FWSTS_FWRI        0x00000200 /* Firmware Reset Indication */

1747 /* EEC Register */
1748 #define IXGBE_EEC_SK            0x00000001 /* EEPROM Clock */
1749 #define IXGBE_EEC_CS            0x00000002 /* EEPROM Chip Select */
1750 #define IXGBE_EEC_DI            0x00000004 /* EEPROM Data In */
1751 #define IXGBE_EEC_DO            0x00000008 /* EEPROM Data Out */
1752 #define IXGBE_EEC_FWE_MASK      0x00000030 /* FLASH Write Enable */
1753 #define IXGBE_EEC_FWE_DIS       0x00000010 /* Disable FLASH writes */
1754 #define IXGBE_EEC_FWE_EN        0x00000020 /* Enable FLASH writes */
1755 #define IXGBE_EEC_FWE_SHIFT     4
1756 #define IXGBE_EEC_REQ           0x00000040 /* EEPROM Access Request */
1757 #define IXGBE_EEC_GNT           0x00000080 /* EEPROM Access Grant */
1758 #define IXGBE_EEC_PRES          0x00000100 /* EEPROM Present */
1759 #define IXGBE_EEC_ARD           0x00000200 /* EEPROM Auto Read Done */
1760 #define IXGBE_EEC_FLUP          0x00800000 /* Flash update command */
1761 #define IXGBE_EEC_SEC1VAL       0x02000000 /* Sector 1 Valid */
1762 #define IXGBE_EEC_FLUDONE       0x04000000 /* Flash update done */
1763 /* EEPROM Addressing bits based on type (0-small, 1-large) */
1764 #define IXGBE_EEC_ADDR_SIZE     0x00000400
1765 #define IXGBE_EEC_SIZE          0x00007800 /* EEPROM Size */
1766 #define IXGBE_EERD_MAX_ADDR     0x00003FFF /* EERD alows 14 bits for addr. */

1768 #define IXGBE_EEC_SIZE_SHIFT            11
1769 #define IXGBE_EEPROM_WORD_SIZE_SHIFT    6
1610 #define IXGBE_EEPROM_WORD_SIZE_BASE_SHIFT  6
1770 #define IXGBE_EEPROM_OPCODE_BITS        8

1772 /* Part Number String Length */
1773 #define IXGBE_PBANUM_LENGTH     11

1775 /* Checksum and EEPROM pointers */
1776 #define IXGBE_PBANUM_PTR_GUARD  0xFAFA
1777 #define IXGBE_EEPROM_CHECKSUM   0x3F
1778 #define IXGBE_EEPROM_SUM        0xBABA
1779 #define IXGBE_PCIE_ANALOG_PTR   0x03
1780 #define IXGBE_ATLAS0_CONFIG_PTR 0x04
1781 #define IXGBE_PHY_PTR           0x04
1782 #define IXGBE_ATLAS1_CONFIG_PTR 0x05
1783 #define IXGBE_OPTION_ROM_PTR    0x05
1784 #define IXGBE_PCIE_GENERAL_PTR  0x06
1785 #define IXGBE_PCIE_CONFIG0_PTR  0x07
1786 #define IXGBE_PCIE_CONFIG1_PTR  0x08
1787 #define IXGBE_CORE0_PTR         0x09
1788 #define IXGBE_CORE1_PTR         0x0A
1789 #define IXGBE_MAC0_PTR          0x0B
1790 #define IXGBE_MAC1_PTR          0x0C
1791 #define IXGBE_CSR0_CONFIG_PTR   0x0D
```

```
#define IXGBE_CSR1_CONFIG_PTR   0x0E
#define IXGBE_FW_PTR            0x0F
#define IXGBE_PBANUM0_PTR       0x15
#define IXGBE_PBANUM1_PTR       0x16
#define IXGBE_ALT_MAC_ADDR_PTR  0x37
#define IXGBE_FREE_SPACE_PTR    0X3E

#define IXGBE_SAN_MAC_ADDR_PTR          0x28
#define IXGBE_DEVICE_CAPS               0x2C
#define IXGBE_SERIAL_NUMBER_MAC_ADDR    0x11
#define IXGBE_PCIE_MSIX_82599_CAPS      0x72
#define IXGBE_MAX_MSIX_VECTORS_82599    0x40
#define IXGBE_PCIE_MSIX_82598_CAPS      0x62
#define IXGBE_MAX_MSIX_VECTORS_82598    0x13

/* MSI-X capability fields masks */
#define IXGBE_PCIE_MSIX_TBL_SZ_MASK     0x7FF

/* Legacy EEPROM word offsets */
#define IXGBE_ISCSI_BOOT_CAPS           0x0033
#define IXGBE_ISCSI_SETUP_PORT_0        0x0030
#define IXGBE_ISCSI_SETUP_PORT_1        0x0034

/* EEPROM Commands - SPI */
#define IXGBE_EEPROM_MAX_RETRY_SPI      5000 /* Max wait 5ms for RDY signal */
#define IXGBE_EEPROM_STATUS_RDY_SPI     0x01
#define IXGBE_EEPROM_READ_OPCODE_SPI    0x03  /* EEPROM read opcode */
#define IXGBE_EEPROM_WRITE_OPCODE_SPI   0x02  /* EEPROM write opcode */
#define IXGBE_EEPROM_A8_OPCODE_SPI      0x08  /* opcode bit-3 = addr bit-8 */
#define IXGBE_EEPROM_WREN_OPCODE_SPI    0x06  /* EEPROM set Write Ena latch */
/* EEPROM reset Write Enable latch */
#define IXGBE_EEPROM_WRDI_OPCODE_SPI    0x04
#define IXGBE_EEPROM_RDSR_OPCODE_SPI    0x05  /* EEPROM read Status reg */
#define IXGBE_EEPROM_WRSR_OPCODE_SPI    0x01  /* EEPROM write Status reg */
#define IXGBE_EEPROM_ERASE4K_OPCODE_SPI 0x20  /* EEPROM ERASE 4KB */
#define IXGBE_EEPROM_ERASE64K_OPCODE_SPI        0xD8  /* EEPROM ERASE 64KB */
#define IXGBE_EEPROM_ERASE256_OPCODE_SPI        0xDB  /* EEPROM ERASE 256B */

/* EEPROM Read Register */
#define IXGBE_EEPROM_RW_REG_DATA        16 /* data offset in EEPROM read reg */
#define IXGBE_EEPROM_RW_REG_DONE        2 /* Offset to READ done bit */
#define IXGBE_EEPROM_RW_REG_START       1 /* First bit to start operation */
#define IXGBE_EEPROM_RW_ADDR_SHIFT      2 /* Shift to the address bits */
#define IXGBE_NVM_POLL_WRITE            1 /* Flag for polling for wr complete */
#define IXGBE_NVM_POLL_READ             0 /* Flag for polling for rd complete */
#define IXGBE_NVM_POLL_WRITE       1  /* Flag for polling for write complete */
#define IXGBE_NVM_POLL_READ        0  /* Flag for polling for read complete */

#define IXGBE_ETH_LENGTH_OF_ADDRESS     6

#define IXGBE_EEPROM_PAGE_SIZE_MAX      128
#define IXGBE_EEPROM_RD_BUFFER_MAX_COUNT        512 /* words rd in burst */
#define IXGBE_EEPROM_WR_BUFFER_MAX_COUNT        256 /* words wr in burst */

#ifndef IXGBE_EEPROM_GRANT_ATTEMPTS
#define IXGBE_EEPROM_GRANT_ATTEMPTS     1000 /* EEPROM attempts to gain grant */
#define IXGBE_EEPROM_GRANT_ATTEMPTS 1000 /* EEPROM # attempts to gain grant */
#endif

/* Number of 5 microseconds we wait for EERD read and
 * EERW write to complete */
#define IXGBE_EERD_EEWR_ATTEMPTS        100000

/* # attempts we wait for flush update to complete */
#define IXGBE_FLUDONE_ATTEMPTS          20000
```



```
#define IXGBE_PCIE_CTRL2                0x5   /* PCIe Control 2 Offset */
#define IXGBE_PCIE_CTRL2_DUMMY_ENABLE   0x8   /* Dummy Function Enable */
#define IXGBE_PCIE_CTRL2_LAN_DISABLE    0x2   /* LAN PCI Disable */
#define IXGBE_PCIE_CTRL2_DISABLE_SELECT 0x1   /* LAN Disable Select */

#define IXGBE_SAN_MAC_ADDR_PORT0_OFFSET         0x0
#define IXGBE_SAN_MAC_ADDR_PORT1_OFFSET         0x3
#define IXGBE_DEVICE_CAPS_ALLOW_ANY_SFP         0x1
#define IXGBE_DEVICE_CAPS_FCOE_OFFLOADS         0x2
#define IXGBE_FW_LESM_PARAMETERS_PTR            0x2
#define IXGBE_FW_LESM_STATE_1                   0x1
#define IXGBE_FW_LESM_STATE_ENABLED             0x8000 /* LESM Enable bit */
#define IXGBE_FW_PASSTHROUGH_PATCH_CONFIG_PTR   0x4
#define IXGBE_FW_PATCH_VERSION_4                0x7
#define IXGBE_FCOE_IBA_CAPS_BLK_PTR             0x33 /* iSCSI/FCOE block */
#define IXGBE_FCOE_IBA_CAPS_FCOE                0x20 /* FCOE flags */
#define IXGBE_ISCSI_FCOE_BLK_PTR                0x17 /* iSCSI/FCOE block */
#define IXGBE_ISCSI_FCOE_FLAGS_OFFSET           0x0 /* FCOE flags */
#define IXGBE_ISCSI_FCOE_FLAGS_ENABLE           0x1 /* FCOE flags enable bit */
#define IXGBE_ALT_SAN_MAC_ADDR_BLK_PTR          0x27 /* Alt. SAN MAC block */
#define IXGBE_ALT_SAN_MAC_ADDR_CAPS_OFFSET      0x0 /* Alt SAN MAC capability */
#define IXGBE_ALT_SAN_MAC_ADDR_PORT0_OFFSET     0x1 /* Alt SAN MAC 0 offset */
#define IXGBE_ALT_SAN_MAC_ADDR_PORT1_OFFSET     0x4 /* Alt SAN MAC 1 offset */
#define IXGBE_ALT_SAN_MAC_ADDR_WWNN_OFFSET      0x7 /* Alt WWNN prefix offset */
#define IXGBE_ALT_SAN_MAC_ADDR_WWPN_OFFSET      0x8 /* Alt WWPN prefix offset */
#define IXGBE_ALT_SAN_MAC_ADDR_CAPS_SANMAC      0x0 /* Alt SAN MAC exists */
#define IXGBE_ALT_SAN_MAC_ADDR_CAPS_ALTWWN      0x1 /* Alt WWN base exists */
#define IXGBE_ALT_SAN_MAC_ADDR_CAPS_OFFSET  0x0  /* Alt. SAN MAC capability */
#define IXGBE_ALT_SAN_MAC_ADDR_PORT0_OFFSET 0x1  /* Alt. SAN MAC 0 offset */
#define IXGBE_ALT_SAN_MAC_ADDR_PORT1_OFFSET 0x4  /* Alt. SAN MAC 1 offset */
#define IXGBE_ALT_SAN_MAC_ADDR_WWNN_OFFSET  0x7  /* Alt. WWNN prefix offset */
#define IXGBE_ALT_SAN_MAC_ADDR_WWPN_OFFSET  0x8  /* Alt. WWPN prefix offset */
#define IXGBE_ALT_SAN_MAC_ADDR_CAPS_SANMAC  0x0  /* Alt. SAN MAC exists */
#define IXGBE_ALT_SAN_MAC_ADDR_CAPS_ALTWWN  0x1  /* Alt. WWN base exists */

#define IXGBE_DEVICE_CAPS_WOL_PORT0_1   0x4 /* WoL supported on ports 0 & 1 */
#define IXGBE_DEVICE_CAPS_WOL_PORT0     0x8 /* WoL supported on port 0 */
#define IXGBE_DEVICE_CAPS_WOL_MASK      0xC /* Mask for WoL capabilities */

/* PCI Bus Info */
#define IXGBE_PCI_DEVICE_STATUS         0xAA
#define IXGBE_PCI_DEVICE_STATUS_TRANSACTION_PENDING     0x0020
#define IXGBE_PCI_LINK_STATUS           0xB2
#define IXGBE_PCI_DEVICE_CONTROL2       0xC8
#define IXGBE_PCI_LINK_WIDTH            0x3F0
#define IXGBE_PCI_LINK_WIDTH_1          0x10
#define IXGBE_PCI_LINK_WIDTH_2          0x20
#define IXGBE_PCI_LINK_WIDTH_4          0x40
#define IXGBE_PCI_LINK_WIDTH_8          0x80
#define IXGBE_PCI_LINK_SPEED            0xF
#define IXGBE_PCI_LINK_SPEED_2500       0x1
#define IXGBE_PCI_LINK_SPEED_5000       0x2
#define IXGBE_PCI_LINK_SPEED_8000       0x3
#define IXGBE_PCI_HEADER_TYPE_REGISTER  0x0E
#define IXGBE_PCI_HEADER_TYPE_MULTIFUNC 0x80
#define IXGBE_PCI_DEVICE_CONTROL2_16ms  0x0005

/* Number of 100 microseconds we wait for PCI Express master disable */
#define IXGBE_PCI_MASTER_DISABLE_TIMEOUT        800

/* Check whether address is multicast. This is little-endian specific check.*/
#define IXGBE_IS_MULTICAST(Address) \
                (bool)(((u8 *)(Address))[0] & ((u8)0x01))

/* Check whether an address is broadcast. */
#define IXGBE_IS_BROADCAST(Address) \
```

```
1914                (((u8 *)(Address))[0] == ((u8)0xff)) && \
1915                (((u8 *)(Address))[1] == ((u8)0xff)))

1917 /* RAH */
1918 #define IXGBE_RAH_VIND_MASK     0x003C0000
1919 #define IXGBE_RAH_VIND_SHIFT    18
1920 #define IXGBE_RAH_AV            0x80000000
1921 #define IXGBE_CLEAR_VMDQ_ALL    0xFFFFFFFF

1923 /* Header split receive */
1924 #define IXGBE_RFCTL_ISCSI_DIS           0x00000001
1925 #define IXGBE_RFCTL_ISCSI_DWC_MASK      0x0000003E
1926 #define IXGBE_RFCTL_ISCSI_DWC_SHIFT     1
1927 #define IXGBE_RFCTL_RSC_DIS             0x00000010
1928 #define IXGBE_RFCTL_NFSW_DIS            0x00000040
1929 #define IXGBE_RFCTL_NFSR_DIS            0x00000080
1930 #define IXGBE_RFCTL_NFS_VER_MASK        0x00000300
1931 #define IXGBE_RFCTL_NFS_VER_SHIFT       8
1932 #define IXGBE_RFCTL_NFS_VER_2           0
1933 #define IXGBE_RFCTL_NFS_VER_3           1
1934 #define IXGBE_RFCTL_NFS_VER_4           2
1935 #define IXGBE_RFCTL_IPV6_DIS            0x00000400
1936 #define IXGBE_RFCTL_IPV6_XSUM_DIS       0x00000800
1937 #define IXGBE_RFCTL_IPFRSP_DIS          0x00004000
1938 #define IXGBE_RFCTL_IPV6_EX_DIS         0x00010000
1939 #define IXGBE_RFCTL_NEW_IPV6_EXT_DIS    0x00020000

1941 /* Transmit Config masks */
1942 #define IXGBE_TXDCTL_ENABLE             0x02000000 /* Ena specific Tx Queue */
1943 #define IXGBE_TXDCTL_SWFLSH             0x04000000 /* Tx Desc. wr-bk flushing */
1768 #define IXGBE_TXDCTL_ENABLE     0x02000000 /* Enable specific Tx Queue */
1769 #define IXGBE_TXDCTL_SWFLSH     0x04000000 /* Tx Desc. write-back flushing */
1944 #define IXGBE_TXDCTL_WTHRESH_SHIFT      16 /* shift to WTHRESH bits */
1945 /* Enable short packet padding to 64 bytes */
1946 #define IXGBE_TX_PAD_ENABLE             0x00000400
1947 #define IXGBE_JUMBO_FRAME_ENABLE        0x00000004  /* Allow jumbo frames */
1948 /* This allows for 16K packets + 4k for vlan */
1949 #define IXGBE_MAX_FRAME_SZ              0x40040000

1951 #define IXGBE_TDWBAL_HEAD_WB_ENABLE     0x1 /* Tx head write-back enable */
1952 #define IXGBE_TDWBAL_SEQNUM_WB_ENABLE   0x2 /* Tx seq# write-back enable */

1954 /* Receive Config masks */
1955 #define IXGBE_RXCTRL_RXEN               0x00000001 /* Enable Receiver */
1956 #define IXGBE_RXCTRL_DMBYPS             0x00000002 /* Desc Monitor Bypass */
1957 #define IXGBE_RXDCTL_ENABLE             0x02000000 /* Ena specific Rx Queue */
1958 #define IXGBE_RXDCTL_SWFLSH             0x04000000 /* Rx Desc wr-bk flushing */
1959 #define IXGBE_RXDCTL_RLPMLMASK          0x00003FFF /* X540 supported only */
1960 #define IXGBE_RXDCTL_RLPML_EN           0x00008000
1782 #define IXGBE_RXCTRL_DMBYPS     0x00000002  /* Descriptor Monitor Bypass */
1783 #define IXGBE_RXDCTL_ENABLE     0x02000000  /* Enable specific Rx Queue */
1961 #define IXGBE_RXDCTL_VME                0x40000000 /* VLAN mode enable */

1963 #define IXGBE_TSAUXC_EN_CLK             0x00000004
1964 #define IXGBE_TSAUXC_SYNCLK             0x00000008
1965 #define IXGBE_TSAUXC_SDP0_INT           0x00000040

1967 #define IXGBE_TSYNCTXCTL_VALID          0x00000001 /* Tx timestamp valid */
1968 #define IXGBE_TSYNCTXCTL_ENABLED        0x00000010 /* Tx timestamping enabled */

1970 #define IXGBE_TSYNCRXCTL_VALID          0x00000001 /* Rx timestamp valid */
1971 #define IXGBE_TSYNCRXCTL_TYPE_MASK      0x0000000E /* Rx type mask */
1972 #define IXGBE_TSYNCRXCTL_TYPE_L2_V2     0x00
1973 #define IXGBE_TSYNCRXCTL_TYPE_L4_V1     0x02
1974 #define IXGBE_TSYNCRXCTL_TYPE_L2_L4_V2  0x04
1975 #define IXGBE_TSYNCRXCTL_TYPE_EVENT_V2  0x0A
```



```
1976 #define IXGBE_TSYNCRXCTL_ENABLED        0x00000010 /* Rx Timestamping enabled */

1978 #define IXGBE_RXMTRL_V1_CTRLT_MASK      0x000000FF
1979 #define IXGBE_RXMTRL_V1_SYNC_MSG        0x00
1980 #define IXGBE_RXMTRL_V1_DELAY_REQ_MSG   0x01
1981 #define IXGBE_RXMTRL_V1_FOLLOWUP_MSG    0x02
1982 #define IXGBE_RXMTRL_V1_DELAY_RESP_MSG  0x03
1983 #define IXGBE_RXMTRL_V1_MGMT_MSG        0x04

1985 #define IXGBE_RXMTRL_V2_MSGID_MASK      0x0000FF00
1986 #define IXGBE_RXMTRL_V2_SYNC_MSG        0x0000
1987 #define IXGBE_RXMTRL_V2_DELAY_REQ_MSG   0x0100
1988 #define IXGBE_RXMTRL_V2_PDELAY_REQ_MSG  0x0200
1989 #define IXGBE_RXMTRL_V2_PDELAY_RESP_MSG 0x0300
1990 #define IXGBE_RXMTRL_V2_FOLLOWUP_MSG    0x0800
1991 #define IXGBE_RXMTRL_V2_DELAY_RESP_MSG  0x0900
1992 #define IXGBE_RXMTRL_V2_PDELAY_FOLLOWUP_MSG 0x0A00
1993 #define IXGBE_RXMTRL_V2_ANNOUNCE_MSG    0x0B00
1994 #define IXGBE_RXMTRL_V2_SIGNALLING_MSG  0x0C00
1995 #define IXGBE_RXMTRL_V2_MGMT_MSG        0x0D00

1997 #define IXGBE_FCTRL_SBP         0x00000002 /* Store Bad Packet */
1998 #define IXGBE_FCTRL_MPE         0x00000100 /* Multicast Promiscuous Ena*/
1999 #define IXGBE_FCTRL_UPE         0x00000200 /* Unicast Promiscuous Ena */
2000 #define IXGBE_FCTRL_BAM         0x00000400 /* Broadcast Accept Mode */
2001 #define IXGBE_FCTRL_PMCF        0x00001000 /* Pass MAC Control Frames */
2002 #define IXGBE_FCTRL_DPF         0x00002000 /* Discard Pause Frame */
2003 /* Receive Priority Flow Control Enable */
2004 #define IXGBE_FCTRL_RPFCE       0x00004000
2005 #define IXGBE_FCTRL_RFCE        0x00008000 /* Receive Flow Control Ena */
2006 #define IXGBE_MFLCN_PMCF        0x00000001 /* Pass MAC Control Frames */
2007 #define IXGBE_MFLCN_DPF         0x00000002 /* Discard Pause Frame */
2008 #define IXGBE_MFLCN_RPFCE       0x00000004 /* Receive Priority FC Enable */
2009 #define IXGBE_MFLCN_RFCE        0x00000008 /* Receive FC Enable */
2010 #define IXGBE_MFLCN_RPFCE_MASK  0x00000FF4 /* Rx Priority FC bitmap mask */
2011 #define IXGBE_MFLCN_RPFCE_SHIFT 4 /* Rx Priority FC bitmap shift */

2013 /* Multiple Receive Queue Control */
2014 #define IXGBE_MRQC_RSSEN        0x00000001  /* RSS Enable */
2015 #define IXGBE_MRQC_MRQE_MASK    0xF /* Bits 3:0 */
2016 #define IXGBE_MRQC_RT8TCEN      0x00000002 /* 8 TC no RSS */
2017 #define IXGBE_MRQC_RT4TCEN      0x00000003 /* 4 TC no RSS */
2018 #define IXGBE_MRQC_RTRSS8TCEN   0x00000004 /* 8 TC w/ RSS */
2019 #define IXGBE_MRQC_RTRSS4TCEN   0x00000005 /* 4 TC w/ RSS */
2020 #define IXGBE_MRQC_VMDQEN       0x00000008 /* VMDq2 64 pools no RSS */
2021 #define IXGBE_MRQC_VMDQRSS32EN  0x0000000A /* VMDq2 32 pools w/ RSS */
2022 #define IXGBE_MRQC_VMDQRSS64EN  0x0000000B /* VMDq2 64 pools w/ RSS */
2023 #define IXGBE_MRQC_VMDQRT8TCEN  0x0000000C /* VMDq2/RT 16 pool 8 TC */
2024 #define IXGBE_MRQC_VMDQRT4TCEN  0x0000000D /* VMDq2/RT 32 pool 4 TC */
2025 #define IXGBE_MRQC_RSS_FIELD_MASK       0xFFFF0000
2026 #define IXGBE_MRQC_RSS_FIELD_IPV4_TCP   0x00010000
2027 #define IXGBE_MRQC_RSS_FIELD_IPV4       0x00020000
2028 #define IXGBE_MRQC_RSS_FIELD_IPV6_EX_TCP 0x00040000
2029 #define IXGBE_MRQC_RSS_FIELD_IPV6_EX    0x00080000
2030 #define IXGBE_MRQC_RSS_FIELD_IPV6       0x00100000
2031 #define IXGBE_MRQC_RSS_FIELD_IPV6_TCP   0x00200000
2032 #define IXGBE_MRQC_RSS_FIELD_IPV4_UDP   0x00400000
2033 #define IXGBE_MRQC_RSS_FIELD_IPV6_UDP   0x00800000
2034 #define IXGBE_MRQC_RSS_FIELD_IPV6_EX_UDP 0x01000000
2035 #define IXGBE_MRQC_L3L4TXSWEN           0x00008000

2037 /* Queue Drop Enable */
2038 #define IXGBE_QDE_ENABLE        0x00000001
2039 #define IXGBE_QDE_IDX_MASK      0x00007F00
2040 #define IXGBE_QDE_IDX_SHIFT     8
2041 #define IXGBE_QDE_WRITE         0x00010000
```

```
2042 #define IXGBE_QDE_READ          0x00020000

2044 #define IXGBE_TXD_POPTS_IXSM    0x01 /* Insert IP checksum */
2045 #define IXGBE_TXD_POPTS_TXSM    0x02 /* Insert TCP/UDP checksum */
2046 #define IXGBE_TXD_CMD_EOP       0x01000000 /* End of Packet */
2047 #define IXGBE_TXD_CMD_IFCS      0x02000000 /* Insert FCS (Ethernet CRC) */
2048 #define IXGBE_TXD_CMD_IC        0x04000000 /* Insert Checksum */
2049 #define IXGBE_TXD_CMD_RS        0x08000000 /* Report Status */
2050 #define IXGBE_TXD_CMD_DEXT      0x20000000 /* Desc extension (0 = legacy) */
1835 #define IXGBE_TXD_CMD_DEXT   0x20000000 /* Descriptor extension (0 = legacy) */
2051 #define IXGBE_TXD_CMD_VLE       0x40000000 /* Add VLAN tag */
2052 #define IXGBE_TXD_STAT_DD       0x00000001 /* Descriptor Done */

2054 #define IXGBE_RXDADV_IPSEC_STATUS_SECP          0x00020000
2055 #define IXGBE_RXDADV_IPSEC_ERROR_INVALID_PROTOCOL 0x08000000
2056 #define IXGBE_RXDADV_IPSEC_ERROR_INVALID_LENGTH 0x10000000
2057 #define IXGBE_RXDADV_IPSEC_ERROR_AUTH_FAILED    0x18000000
2058 #define IXGBE_RXDADV_IPSEC_ERROR_BIT_MASK       0x18000000
2059 /* Multiple Transmit Queue Command Register */
2060 #define IXGBE_MTQC_RT_ENA       0x1 /* DCB Enable */
2061 #define IXGBE_MTQC_VT_ENA       0x2 /* VMDQ2 Enable */
2062 #define IXGBE_MTQC_64Q_1PB      0x0 /* 64 queues 1 pack buffer */
2063 #define IXGBE_MTQC_32VF         0x8 /* 4 TX Queues per pool w/32VF's */
2064 #define IXGBE_MTQC_64VF         0x4 /* 2 TX Queues per pool w/64VF's */
2065 #define IXGBE_MTQC_4TC_4TQ      0x8 /* 4 TC if RT_ENA and VT_ENA */
2066 #define IXGBE_MTQC_8TC_8TQ      0xC /* 8 TC if RT_ENA or 8 TQ if VT_ENA */

2068 /* Receive Descriptor bit definitions */
2069 #define IXGBE_RXD_STAT_DD       0x01 /* Descriptor Done */
2070 #define IXGBE_RXD_STAT_EOP      0x02 /* End of Packet */
2071 #define IXGBE_RXD_STAT_FLM      0x04 /* FDir Match */
2072 #define IXGBE_RXD_STAT_VP       0x08 /* IEEE VLAN Packet */
2073 #define IXGBE_RXDADV_NEXTP_MASK 0x000FFFF0 /* Next Descriptor Index */
2074 #define IXGBE_RXDADV_NEXTP_SHIFT        0x00000004
2075 #define IXGBE_RXD_STAT_UDPCS    0x10 /* UDP xsum calculated */
2076 #define IXGBE_RXD_STAT_L4CS     0x20 /* L4 xsum calculated */
2077 #define IXGBE_RXD_STAT_IPCS     0x40 /* IP xsum calculated */
2078 #define IXGBE_RXD_STAT_PIF      0x80 /* passed in-exact filter */
2079 #define IXGBE_RXD_STAT_CRCV     0x100 /* Speculative CRC Valid */
2080 #define IXGBE_RXD_STAT_VEXT     0x200 /* 1st VLAN found */
2081 #define IXGBE_RXD_STAT_UDPV     0x400 /* Valid UDP checksum */
2082 #define IXGBE_RXD_STAT_DYNINT   0x800 /* Pkt caused INT via DYNINT */
2083 #define IXGBE_RXD_STAT_LLINT    0x800 /* Pkt caused Low Latency Interrupt */
2084 #define IXGBE_RXD_STAT_TS       0x10000 /* Time Stamp */
2085 #define IXGBE_RXD_STAT_SECP     0x20000 /* Security Processing */
2086 #define IXGBE_RXD_STAT_LB       0x40000 /* Loopback Status */
2087 #define IXGBE_RXD_STAT_ACK      0x8000 /* ACK Packet indication */
2088 #define IXGBE_RXD_ERR_CE        0x01 /* CRC Error */
2089 #define IXGBE_RXD_ERR_LE        0x02 /* Length Error */
2090 #define IXGBE_RXD_ERR_PE        0x08 /* Packet Error */
2091 #define IXGBE_RXD_ERR_OSE       0x10 /* Oversize Error */
2092 #define IXGBE_RXD_ERR_USE       0x20 /* Undersize Error */
2093 #define IXGBE_RXD_ERR_TCPE      0x40 /* TCP/UDP Checksum Error */
2094 #define IXGBE_RXD_ERR_IPE       0x80 /* IP Checksum Error */
2095 #define IXGBE_RXDADV_ERR_MASK           0xfff00000 /* RDESC.ERRORS mask */
2096 #define IXGBE_RXDADV_ERR_SHIFT          20 /* RDESC.ERRORS shift */
2097 #define IXGBE_RXDADV_ERR_RXE            0x20000000 /* Any MAC Error */
2098 #define IXGBE_RXDADV_ERR_FCEOFE         0x80000000 /* FCoEFe/IPE */
2099 #define IXGBE_RXDADV_ERR_FCERR          0x00700000 /* FCERR/FDIRERR */
2100 #define IXGBE_RXDADV_ERR_FDIR_LEN       0x00100000 /* FDIR Length error */
2101 #define IXGBE_RXDADV_ERR_FDIR_DROP      0x00200000 /* FDIR Drop error */
2102 #define IXGBE_RXDADV_ERR_FDIR_COLL      0x00400000 /* FDIR Collision error */
2103 #define IXGBE_RXDADV_ERR_HBO    0x00800000 /*Header Buffer Overflow */
2104 #define IXGBE_RXDADV_ERR_CE     0x01000000 /* CRC Error */
2105 #define IXGBE_RXDADV_ERR_LE     0x02000000 /* Length Error */
2106 #define IXGBE_RXDADV_ERR_PE     0x08000000 /* Packet Error */
```



```
2107 #define IXGBE_RXDADV_ERR_OSE    0x10000000 /* Oversize Error */
2108 #define IXGBE_RXDADV_ERR_USE    0x20000000 /* Undersize Error */
2109 #define IXGBE_RXDADV_ERR_TCPE   0x40000000 /* TCP/UDP Checksum Error */
2110 #define IXGBE_RXDADV_ERR_IPE    0x80000000 /* IP Checksum Error */
2111 #define IXGBE_RXD_VLAN_ID_MASK  0x0FFF  /* VLAN ID is in lower 12 bits */
2112 #define IXGBE_RXD_PRI_MASK      0xE000  /* Priority is in upper 3 bits */
2113 #define IXGBE_RXD_PRI_SHIFT     13
2114 #define IXGBE_RXD_CFI_MASK      0x1000  /* CFI is bit 12 */
2115 #define IXGBE_RXD_CFI_SHIFT     12

2117 #define IXGBE_RXDADV_STAT_DD            IXGBE_RXD_STAT_DD  /* Done */
2118 #define IXGBE_RXDADV_STAT_EOP           IXGBE_RXD_STAT_EOP /* End of Packet */
2119 #define IXGBE_RXDADV_STAT_FLM           IXGBE_RXD_STAT_FLM /* FDir Match */
2120 #define IXGBE_RXDADV_STAT_VP            IXGBE_RXD_STAT_VP  /* IEEE VLAN Pkt */
2121 #define IXGBE_RXDADV_STAT_MASK          0x000fffff /* Stat/NEXTP: bit 0-19 */
2122 #define IXGBE_RXDADV_STAT_FCEOFS        0x00000040 /* FCoE EOF/SOF Stat */
2123 #define IXGBE_RXDADV_STAT_FCSTAT        0x00000030 /* FCoE Pkt Stat */
2124 #define IXGBE_RXDADV_STAT_FCSTAT_NOMTCH 0x00000000 /* 00: No Ctxt Match */
2125 #define IXGBE_RXDADV_STAT_FCSTAT_NODDP  0x00000010 /* 01: Ctxt w/o DDP */
2126 #define IXGBE_RXDADV_STAT_FCSTAT_FCPRSP 0x00000020 /* 10: Recv. FCP_RSP */
2127 #define IXGBE_RXDADV_STAT_FCSTAT_DDP    0x00000030 /* 11: Ctxt w/ DDP */
2128 #define IXGBE_RXDADV_STAT_TS            0x00010000 /* IEEE1588 Time Stamp */

2130 /* PSRTYPE bit definitions */
2131 #define IXGBE_PSRTYPE_TCPHDR    0x00000010
2132 #define IXGBE_PSRTYPE_UDPHDR    0x00000020
2133 #define IXGBE_PSRTYPE_IPV4HDR   0x00000100
2134 #define IXGBE_PSRTYPE_IPV6HDR   0x00000200
2135 #define IXGBE_PSRTYPE_L2HDR     0x00001000

2137 /* SRRCTL bit definitions */
2138 #define IXGBE_SRRCTL_BSIZEPKT_SHIFT     10 /* so many KBs */
2139 #define IXGBE_SRRCTL_RDMTS_SHIFT        22
2140 #define IXGBE_SRRCTL_RDMTS_MASK         0x01C00000
2141 #define IXGBE_SRRCTL_DROP_EN            0x10000000
2142 #define IXGBE_SRRCTL_BSIZEPKT_MASK      0x0000007F
2143 #define IXGBE_SRRCTL_BSIZEHDR_MASK      0x00003F00
2144 #define IXGBE_SRRCTL_DESCTYPE_LEGACY    0x00000000
2145 #define IXGBE_SRRCTL_DESCTYPE_ADV_ONEBUF 0x02000000
2146 #define IXGBE_SRRCTL_DESCTYPE_HDR_SPLIT 0x04000000
2147 #define IXGBE_SRRCTL_DESCTYPE_HDR_REPLICATION_LARGE_PKT 0x08000000
2148 #define IXGBE_SRRCTL_DESCTYPE_HDR_SPLIT_ALWAYS 0x0A000000
2149 #define IXGBE_SRRCTL_DESCTYPE_MASK      0x0E000000

2151 #define IXGBE_RXDPS_HDRSTAT_HDRSP       0x00008000
2152 #define IXGBE_RXDPS_HDRSTAT_HDRLEN_MASK 0x000003FF

2154 #define IXGBE_RXDADV_RSSTYPE_MASK       0x0000000F
2155 #define IXGBE_RXDADV_PKTTYPE_MASK       0x0000FFF0
2156 #define IXGBE_RXDADV_PKTTYPE_MASK_EX    0x0001FFF0
2157 #define IXGBE_RXDADV_HDRBUFLEN_MASK     0x00007FE0
2158 #define IXGBE_RXDADV_RSCCNT_MASK        0x001E0000
2159 #define IXGBE_RXDADV_RSCCNT_SHIFT       17
2160 #define IXGBE_RXDADV_HDRBUFLEN_SHIFT    5
2161 #define IXGBE_RXDADV_SPLITHEADER_EN     0x00001000
2162 #define IXGBE_RXDADV_SPH                0x8000

2164 /* RSS Hash results */
2165 #define IXGBE_RXDADV_RSSTYPE_NONE       0x00000000
2166 #define IXGBE_RXDADV_RSSTYPE_IPV4_TCP   0x00000001
2167 #define IXGBE_RXDADV_RSSTYPE_IPV4       0x00000002
2168 #define IXGBE_RXDADV_RSSTYPE_IPV6_TCP   0x00000003
2169 #define IXGBE_RXDADV_RSSTYPE_IPV6_EX    0x00000004
2170 #define IXGBE_RXDADV_RSSTYPE_IPV6       0x00000005
2171 #define IXGBE_RXDADV_RSSTYPE_IPV6_TCP_EX 0x00000006
2172 #define IXGBE_RXDADV_RSSTYPE_IPV4_UDP   0x00000007
```

```
2173 #define IXGBE_RXDADV_RSSTYPE_IPV6_UDP   0x00000008
2174 #define IXGBE_RXDADV_RSSTYPE_IPV6_UDP_EX 0x00000009

2176 /* RSS Packet Types as indicated in the receive descriptor. */
2177 #define IXGBE_RXDADV_PKTTYPE_NONE       0x00000000
2178 #define IXGBE_RXDADV_PKTTYPE_IPV4       0x00000010 /* IPv4 hdr present */
2179 #define IXGBE_RXDADV_PKTTYPE_IPV4_EX    0x00000020 /* IPv4 hdr + extensions */
2180 #define IXGBE_RXDADV_PKTTYPE_IPV6       0x00000040 /* IPv6 hdr present */
2181 #define IXGBE_RXDADV_PKTTYPE_IPV6_EX    0x00000080 /* IPv6 hdr + extensions */
2182 #define IXGBE_RXDADV_PKTTYPE_TCP        0x00000100 /* TCP hdr present */
2183 #define IXGBE_RXDADV_PKTTYPE_UDP        0x00000200 /* UDP hdr present */
2184 #define IXGBE_RXDADV_PKTTYPE_SCTP       0x00000400 /* SCTP hdr present */
2185 #define IXGBE_RXDADV_PKTTYPE_NFS        0x00000800 /* NFS hdr present */
2186 #define IXGBE_RXDADV_PKTTYPE_IPSEC_ESP  0x00001000 /* IPSec ESP */
2187 #define IXGBE_RXDADV_PKTTYPE_IPSEC_AH   0x00002000 /* IPSec AH */
2188 #define IXGBE_RXDADV_PKTTYPE_LINKSEC    0x00004000 /* LinkSec Encap */
2189 #define IXGBE_RXDADV_PKTTYPE_ETQF       0x00008000 /* PKTTYPE is ETQF index */
2190 #define IXGBE_RXDADV_PKTTYPE_ETQF_MASK  0x00000070 /* ETQF has 8 indices */
2191 #define IXGBE_RXDADV_PKTTYPE_ETQF_SHIFT 4 /* Right-shift 4 bits */

2193 /* Security Processing bit Indication */
2194 #define IXGBE_RXDADV_LNKSEC_STATUS_SECP         0x00020000
2195 #define IXGBE_RXDADV_LNKSEC_ERROR_NO_SA_MATCH   0x08000000
2196 #define IXGBE_RXDADV_LNKSEC_ERROR_REPLAY_ERROR  0x10000000
2197 #define IXGBE_RXDADV_LNKSEC_ERROR_BIT_MASK      0x18000000
2198 #define IXGBE_RXDADV_LNKSEC_ERROR_BAD_SIG       0x18000000

2200 /* Masks to determine if packets should be dropped due to frame errors */
2201 #define IXGBE_RXD_ERR_FRAME_ERR_MASK ( \
2202                                 IXGBE_RXD_ERR_CE | \
2203                                 IXGBE_RXD_ERR_LE | \
2204                                 IXGBE_RXD_ERR_PE | \
2205                                 IXGBE_RXD_ERR_OSE | \
2206                                 IXGBE_RXD_ERR_USE)

2208 #define IXGBE_RXDADV_ERR_FRAME_ERR_MASK ( \
2209                                 IXGBE_RXDADV_ERR_CE | \
2210                                 IXGBE_RXDADV_ERR_LE | \
2211                                 IXGBE_RXDADV_ERR_PE | \
2212                                 IXGBE_RXDADV_ERR_OSE | \
2213                                 IXGBE_RXDADV_ERR_USE)

2215 #define IXGBE_RXDADV_ERR_FRAME_ERR_MASK_82599   IXGBE_RXDADV_ERR_RXE

2217 /* Multicast bit mask */
2218 #define IXGBE_MCSTCTRL_MFE      0x4

2220 /* Number of Transmit and Receive Descriptors must be a multiple of 8 */
2221 #define IXGBE_REQ_TX_DESCRIPTOR_MULTIPLE        8
2222 #define IXGBE_REQ_RX_DESCRIPTOR_MULTIPLE        8
2223 #define IXGBE_REQ_TX_BUFFER_GRANULARITY         1024

2225 /* Vlan-specific macros */
2226 #define IXGBE_RX_DESC_SPECIAL_VLAN_MASK 0x0FFF /* VLAN ID in lower 12 bits */
2227 #define IXGBE_RX_DESC_SPECIAL_PRI_MASK  0xE000 /* Priority in upper 3 bits */
2228 #define IXGBE_RX_DESC_SPECIAL_PRI_SHIFT 0x000D /* Priority in upper 3 of 16 */
2229 #define IXGBE_TX_DESC_SPECIAL_PRI_SHIFT IXGBE_RX_DESC_SPECIAL_PRI_SHIFT

2231 /* SR-IOV specific macros */
2232 #define IXGBE_MBVFICR_INDEX(vf_number)  (vf_number >> 4)
2233 #define IXGBE_MBVFICR(_i)               (0x00710 + ((_i) * 4))
2014 #define IXGBE_MBVFICR(_i)                (0x00710 + (_i * 4))
2234 #define IXGBE_VFLRE(_i)                 (((_i & 1) ? 0x001C0 : 0x00600))
2235 #define IXGBE_VFLREC(_i)                 (0x00700 + ((_i) * 4))
2016 #define IXGBE_VFLREC(_i)                 (0x00700 + (_i * 4))
```



```
2237 /* Little Endian defines */
2238 #ifndef __le16
2239 #define __le16  u16
2240 #endif
2241 #ifndef __le32
2242 #define __le32  u32
2243 #endif
2244 #ifndef __le64
2245 #define __le64  u64

2247 #endif
2248 #ifndef __be16
2249 /* Big Endian defines */
2250 #define __be16  u16
2251 #define __be32  u32
2252 #define __be64  u64

2254 #endif
2255 enum ixgbe_fdir_pballoc_type {
2256         IXGBE_FDIR_PBALLOC_NONE = 0,
2257         IXGBE_FDIR_PBALLOC_64K  = 1,
2258         IXGBE_FDIR_PBALLOC_128K = 2,
2259         IXGBE_FDIR_PBALLOC_256K = 3,
2037         IXGBE_FDIR_PBALLOC_64K = 0,
2038         IXGBE_FDIR_PBALLOC_128K,
2039         IXGBE_FDIR_PBALLOC_256K,
2260 };
2041 #define IXGBE_FDIR_PBALLOC_SIZE_SHIFT           16

2262 /* Flow Director register values */
2263 #define IXGBE_FDIRCTRL_PBALLOC_64K              0x00000001
2264 #define IXGBE_FDIRCTRL_PBALLOC_128K             0x00000002
2265 #define IXGBE_FDIRCTRL_PBALLOC_256K             0x00000003
2266 #define IXGBE_FDIRCTRL_INIT_DONE                0x00000008
2267 #define IXGBE_FDIRCTRL_PERFECT_MATCH            0x00000010
2268 #define IXGBE_FDIRCTRL_REPORT_STATUS            0x00000020
2269 #define IXGBE_FDIRCTRL_REPORT_STATUS_ALWAYS     0x00000080
2270 #define IXGBE_FDIRCTRL_DROP_Q_SHIFT             8
2271 #define IXGBE_FDIRCTRL_FLEX_SHIFT               16
2272 #define IXGBE_FDIRCTRL_SEARCHLIM                0x00800000
2273 #define IXGBE_FDIRCTRL_MAX_LENGTH_SHIFT         24
2274 #define IXGBE_FDIRCTRL_FULL_THRESH_MASK         0xF0000000
2275 #define IXGBE_FDIRCTRL_FULL_THRESH_SHIFT        28

2277 #define IXGBE_FDIRTCPM_DPORTM_SHIFT             16
2278 #define IXGBE_FDIRUDPM_DPORTM_SHIFT             16
2279 #define IXGBE_FDIRIP6M_DIPM_SHIFT               16
2280 #define IXGBE_FDIRM_VLANID                      0x00000001
2281 #define IXGBE_FDIRM_VLANP                       0x00000002
2282 #define IXGBE_FDIRM_POOL                        0x00000004
2283 #define IXGBE_FDIRM_L4P                         0x00000008
2284 #define IXGBE_FDIRM_FLEX                        0x00000010
2285 #define IXGBE_FDIRM_DIPv6                       0x00000020

2287 #define IXGBE_FDIRFREE_FREE_MASK                0xFFFF
2288 #define IXGBE_FDIRFREE_FREE_SHIFT               0
2289 #define IXGBE_FDIRFREE_COLL_MASK                0x7FFF0000
2290 #define IXGBE_FDIRFREE_COLL_SHIFT               16
2291 #define IXGBE_FDIRLEN_MAXLEN_MASK               0x3F
2292 #define IXGBE_FDIRLEN_MAXLEN_SHIFT              0
2293 #define IXGBE_FDIRLEN_MAXHASH_MASK              0x7FFF0000
2294 #define IXGBE_FDIRLEN_MAXHASH_SHIFT             16
2295 #define IXGBE_FDIRUSTAT_ADD_MASK                0xFFFF
2296 #define IXGBE_FDIRUSTAT_ADD_SHIFT               0
2297 #define IXGBE_FDIRUSTAT_REMOVE_MASK             0xFFFF0000
2298 #define IXGBE_FDIRUSTAT_REMOVE_SHIFT            16
```

```
2299 #define IXGBE_FDIRFSTAT_FADD_MASK               0x00FF
2300 #define IXGBE_FDIRFSTAT_FADD_SHIFT              0
2301 #define IXGBE_FDIRFSTAT_FREMOVE_MASK            0xFF00
2302 #define IXGBE_FDIRFSTAT_FREMOVE_SHIFT           8
2303 #define IXGBE_FDIRPORT_DESTINATION_SHIFT        16
2304 #define IXGBE_FDIRVLAN_FLEX_SHIFT               16
2305 #define IXGBE_FDIRHASH_BUCKET_VALID_SHIFT       15
2306 #define IXGBE_FDIRHASH_SIG_SW_INDEX_SHIFT       16

2308 #define IXGBE_FDIRCMD_CMD_MASK                  0x00000003
2309 #define IXGBE_FDIRCMD_CMD_ADD_FLOW              0x00000001
2310 #define IXGBE_FDIRCMD_CMD_REMOVE_FLOW           0x00000002
2311 #define IXGBE_FDIRCMD_CMD_QUERY_REM_FILT        0x00000003
2312 #define IXGBE_FDIRCMD_FILTER_VALID              0x00000004
2093 #define IXGBE_FDIRCMD_CMD_QUERY_REM_HASH        0x00000007
2313 #define IXGBE_FDIRCMD_FILTER_UPDATE             0x00000008
2314 #define IXGBE_FDIRCMD_IPv6DMATCH                0x00000010
2315 #define IXGBE_FDIRCMD_L4TYPE_UDP                0x00000020
2316 #define IXGBE_FDIRCMD_L4TYPE_TCP                0x00000040
2317 #define IXGBE_FDIRCMD_L4TYPE_SCTP               0x00000060
2318 #define IXGBE_FDIRCMD_IPV6                      0x00000080
2319 #define IXGBE_FDIRCMD_CLEARHT                   0x00000100
2320 #define IXGBE_FDIRCMD_DROP                      0x00000200
2321 #define IXGBE_FDIRCMD_INT                       0x00000400
2322 #define IXGBE_FDIRCMD_LAST                      0x00000800
2323 #define IXGBE_FDIRCMD_COLLISION                 0x00001000
2324 #define IXGBE_FDIRCMD_QUEUE_EN                  0x00008000
2325 #define IXGBE_FDIRCMD_FLOW_TYPE_SHIFT           5
2326 #define IXGBE_FDIRCMD_RX_QUEUE_SHIFT            16
2327 #define IXGBE_FDIRCMD_VT_POOL_SHIFT             24
2328 #define IXGBE_FDIR_INIT_DONE_POLL               10
2329 #define IXGBE_FDIRCMD_CMD_POLL                  10

2331 #define IXGBE_FDIR_DROP_QUEUE                   127

2333 #define IXGBE_STATUS_OVERHEATING_BIT            20 /* STATUS overtemp bit num */

2335 /* Manageablility Host Interface defines */
2336 #define IXGBE_HI_MAX_BLOCK_BYTE_LENGTH  1792 /* Num of bytes in range */
2337 #define IXGBE_HI_MAX_BLOCK_DWORD_LENGTH 448 /* Num of dwords in range */
2338 #define IXGBE_HI_COMMAND_TIMEOUT        500 /* Process HI command limit */

2340 /* CEM Support */
2341 #define FW_CEM_HDR_LEN                  0x4
2342 #define FW_CEM_CMD_DRIVER_INFO          0xDD
2343 #define FW_CEM_CMD_DRIVER_INFO_LEN      0x5
2344 #define FW_CEM_CMD_RESERVED             0X0
2345 #define FW_CEM_UNUSED_VER               0x0
2346 #define FW_CEM_MAX_RETRIES              3
2347 #define FW_CEM_RESP_STATUS_SUCCESS      0x1

2349 /* Host Interface Command Structures */

2351 struct ixgbe_hic_hdr {
2352         u8 cmd;
2353         u8 buf_len;
2354         union {
2355                 u8 cmd_resv;
2356                 u8 ret_status;
2357         } cmd_or_resp;
2358         u8 checksum;
2359 };

2361 struct ixgbe_hic_drv_info {
2362         struct ixgbe_hic_hdr hdr;
2363         u8 port_num;
```



```
2364         u8 ver_sub;
2365         u8 ver_build;
2366         u8 ver_min;
2367         u8 ver_maj;
2368         u8 pad; /* end spacing to ensure length is mult. of dword */
2369         u16 pad2; /* end spacing to ensure length is mult. of dword2 */
2370 };

2372 /* Transmit Descriptor - Legacy */
2373 struct ixgbe_legacy_tx_desc {
2374         u64 buffer_addr; /* Address of the descriptor's data buffer */
2375         union {
2376                 __le32 data;
2377                 struct {
2378                         __le16 length; /* Data buffer length */
2379                         u8 cso; /* Checksum offset */
2380                         u8 cmd; /* Descriptor control */
2381                 } flags;
2382         } lower;
2383         union {
2384                 __le32 data;
2385                 struct {
2386                         u8 status; /* Descriptor status */
2387                         u8 css; /* Checksum start */
2388                         __le16 vlan;
2389                 } fields;
2390         } upper;
2391 };
______unchanged_portion_omitted_

2456 /* Adv Transmit Descriptor Config Masks */
2457 #define IXGBE_ADVTXD_DTALEN_MASK        0x0000FFFF /* Data buf length(bytes) */
2458 #define IXGBE_ADVTXD_MAC_LINKSEC        0x00040000 /* Insert LinkSec */
2459 #define IXGBE_ADVTXD_MAC_TSTAMP         0x00080000 /* IEEE1588 time stamp */
2460 #define IXGBE_ADVTXD_IPSEC_SA_INDEX_MASK 0x000003FF /* IPSec SA index */
2461 #define IXGBE_ADVTXD_IPSEC_ESP_LEN_MASK 0x000001FF /* IPSec ESP length */
2462 #define IXGBE_ADVTXD_DTYP_MASK          0x00F00000 /* DTYP mask */
2463 #define IXGBE_ADVTXD_DTYP_CTXT          0x00200000 /* Adv Context Desc */
2464 #define IXGBE_ADVTXD_DTYP_DATA          0x00300000 /* Adv Data Descriptor */
2202 #define IXGBE_ADVTXD_DTYP_CTXT  0x00200000 /* Advanced Context Desc */
2203 #define IXGBE_ADVTXD_DTYP_DATA  0x00300000 /* Advanced Data Descriptor */
2465 #define IXGBE_ADVTXD_DCMD_EOP           IXGBE_TXD_CMD_EOP  /* End of Packet */
2466 #define IXGBE_ADVTXD_DCMD_IFCS          IXGBE_TXD_CMD_IFCS /* Insert FCS */
2467 #define IXGBE_ADVTXD_DCMD_RS            IXGBE_TXD_CMD_RS /* Report Status */
2468 #define IXGBE_ADVTXD_DCMD_DDTYP_ISCSI   0x10000000 /* DDP hdr type or iSCSI */
2469 #define IXGBE_ADVTXD_DCMD_DEXT          IXGBE_TXD_CMD_DEXT /* Desc ext 1=Adv */
2208 #define IXGBE_ADVTXD_DCMD_DEXT  IXGBE_TXD_CMD_DEXT /* Desc ext (1=Adv) */
2470 #define IXGBE_ADVTXD_DCMD_VLE           IXGBE_TXD_CMD_VLE  /* VLAN pkt enable */
2471 #define IXGBE_ADVTXD_DCMD_TSE           0x80000000 /* TCP Seg enable */
2472 #define IXGBE_ADVTXD_STAT_DD            IXGBE_TXD_STAT_DD  /* Descriptor Done */
2473 #define IXGBE_ADVTXD_STAT_SN_CRC        0x00000002 /* NXTSEQ/SEED pres in WB */
2474 #define IXGBE_ADVTXD_STAT_RSV           0x0000000C /* STA Reserved */
2475 #define IXGBE_ADVTXD_IDX_SHIFT          4 /* Adv desc Index shift */
2476 #define IXGBE_ADVTXD_CC                 0x00000080 /* Check Context */
2477 #define IXGBE_ADVTXD_POPTS_SHIFT        8  /* Adv desc POPTS shift */
2478 #define IXGBE_ADVTXD_POPTS_IXSM         (IXGBE_TXD_POPTS_IXSM << \
2479                                          IXGBE_ADVTXD_POPTS_SHIFT)
2480 #define IXGBE_ADVTXD_POPTS_TXSM         (IXGBE_TXD_POPTS_TXSM << \
2481                                          IXGBE_ADVTXD_POPTS_SHIFT)
2482 #define IXGBE_ADVTXD_POPTS_ISCO_1ST     0x00000000 /* 1st TSO of iSCSI PDU */
2483 #define IXGBE_ADVTXD_POPTS_ISCO_MDL     0x00000800 /* Middle TSO of iSCSI PDU */
2484 #define IXGBE_ADVTXD_POPTS_ISCO_LAST    0x00001000 /* Last TSO of iSCSI PDU */
2485 /* 1st&Last TSO-full iSCSI PDU */
2486 #define IXGBE_ADVTXD_POPTS_ISCO_FULL    0x00001800
2224 #define IXGBE_ADVTXD_POPTS_ISCO_FULL 0x00001800 /* 1st&Last TSO-full iSCSI PDU *
2487 #define IXGBE_ADVTXD_POPTS_RSV          0x00002000 /* POPTS Reserved */
```

```
#define IXGBE_ADVTXD_PAYLEN_SHIFT       14 /* Adv desc PAYLEN shift */
#define IXGBE_ADVTXD_MACLEN_SHIFT       9  /* Adv ctxt desc mac len shift */
#define IXGBE_ADVTXD_VLAN_SHIFT         16  /* Adv ctxt vlan tag shift */
#define IXGBE_ADVTXD_TUCMD_IPV4         0x00000400 /* IP Packet Type: 1=IPv4 */
#define IXGBE_ADVTXD_TUCMD_IPV6         0x00000000 /* IP Packet Type: 0=IPv6 */
#define IXGBE_ADVTXD_TUCMD_L4T_UDP      0x00000000 /* L4 Packet TYPE of UDP */
#define IXGBE_ADVTXD_TUCMD_L4T_TCP      0x00000800 /* L4 Packet TYPE of TCP */
#define IXGBE_ADVTXD_TUCMD_L4T_SCTP     0x00001000 /* L4 Packet TYPE of SCTP */
#define IXGBE_ADVTXD_TUCMD_MKRREQ       0x00002000 /* req Markers and CRC */
#define IXGBE_ADVTXD_TUCMD_MKRREQ    0x00002000 /*Req requires Markers and CRC*/
#define IXGBE_ADVTXD_POPTS_IPSEC        0x00000400 /* IPSec offload request */
#define IXGBE_ADVTXD_TUCMD_IPSEC_TYPE_ESP 0x00002000 /* IPSec Type ESP */
#define IXGBE_ADVTXD_TUCMD_IPSEC_ENCRYPT_EN 0x00004000/* ESP Encrypt Enable */
#define IXGBE_ADVTXT_TUCMD_FCOE         0x00008000 /* FCoE Frame Type */
#define IXGBE_ADVTXD_FCOEF_EOF_MASK     (0x3 << 10) /* FC EOF index */
#define IXGBE_ADVTXD_FCOEF_SOF          ((1 << 2) << 10) /* FC SOF index */
#define IXGBE_ADVTXD_FCOEF_PARINC       ((1 << 3) << 10) /* Rel_Off in F_CTL */
#define IXGBE_ADVTXD_FCOEF_ORIE         ((1 << 4) << 10) /* Orientation End */
#define IXGBE_ADVTXD_FCOEF_ORIS         ((1 << 5) << 10) /* Orientation Start */
#define IXGBE_ADVTXD_FCOEF_ORIE      ((1 << 4) << 10) /* Orientation: End */
#define IXGBE_ADVTXD_FCOEF_ORIS      ((1 << 5) << 10) /* Orientation: Start */
#define IXGBE_ADVTXD_FCOEF_EOF_N        (0x0 << 10) /* 00: EOFn */
#define IXGBE_ADVTXD_FCOEF_EOF_T        (0x1 << 10) /* 01: EOFt */
#define IXGBE_ADVTXD_FCOEF_EOF_NI       (0x2 << 10) /* 10: EOFni */
#define IXGBE_ADVTXD_FCOEF_EOF_A        (0x3 << 10) /* 11: EOFa */
#define IXGBE_ADVTXD_L4LEN_SHIFT        8  /* Adv ctxt L4LEN shift */
#define IXGBE_ADVTXD_MSS_SHIFT          16  /* Adv ctxt MSS shift */

/* Autonegotiation advertised speeds */
typedef u32 ixgbe_autoneg_advertised;
/* Link speed */
typedef u32 ixgbe_link_speed;
#define IXGBE_LINK_SPEED_UNKNOWN        0
#define IXGBE_LINK_SPEED_100_FULL       0x0008
#define IXGBE_LINK_SPEED_1GB_FULL       0x0020
#define IXGBE_LINK_SPEED_10GB_FULL      0x0080
#define IXGBE_LINK_SPEED_82598_AUTONEG  (IXGBE_LINK_SPEED_1GB_FULL | \
                                         IXGBE_LINK_SPEED_10GB_FULL)
#define IXGBE_LINK_SPEED_82599_AUTONEG  (IXGBE_LINK_SPEED_100_FULL | \
                                         IXGBE_LINK_SPEED_1GB_FULL | \
                                         IXGBE_LINK_SPEED_10GB_FULL)


/* Physical layer type */
typedef u32 ixgbe_physical_layer;
#define IXGBE_PHYSICAL_LAYER_UNKNOWN            0
#define IXGBE_PHYSICAL_LAYER_10GBASE_T          0x0001
#define IXGBE_PHYSICAL_LAYER_1000BASE_T         0x0002
#define IXGBE_PHYSICAL_LAYER_100BASE_TX         0x0004
#define IXGBE_PHYSICAL_LAYER_SFP_PLUS_CU        0x0008
#define IXGBE_PHYSICAL_LAYER_10GBASE_LR         0x0010
#define IXGBE_PHYSICAL_LAYER_10GBASE_LRM        0x0020
#define IXGBE_PHYSICAL_LAYER_10GBASE_SR         0x0040
#define IXGBE_PHYSICAL_LAYER_10GBASE_KX4        0x0080
#define IXGBE_PHYSICAL_LAYER_10GBASE_CX4        0x0100
#define IXGBE_PHYSICAL_LAYER_1000BASE_KX        0x0200
#define IXGBE_PHYSICAL_LAYER_1000BASE_BX        0x0400
#define IXGBE_PHYSICAL_LAYER_10GBASE_KR         0x0800
#define IXGBE_PHYSICAL_LAYER_10GBASE_XAUI       0x1000
#define IXGBE_PHYSICAL_LAYER_SFP_ACTIVE_DA      0x2000
#define IXGBE_PHYSICAL_LAYER_1000BASE_SX        0x4000

/* Flow Control Data Sheet defined values
 * Calculation and defines taken from 802.1bb Annex O
 */
/* Flow Control Macros */
```



```
#define PAUSE_RTT       8
#define PAUSE_MTU(MTU)  ((MTU + 1024 - 1) / 1024)

/* BitTimes (BT) conversion */
#define IXGBE_BT2KB(BT)         ((BT + (8 * 1024 - 1)) / (8 * 1024))
#define IXGBE_B2BT(BT)          (BT * 8)
#define FC_HIGH_WATER(MTU) ((((PAUSE_RTT + PAUSE_MTU(MTU)) * 144) + 99) / 100 +\
                                PAUSE_MTU(MTU))
#define FC_LOW_WATER(MTU)  (2 * (2 * PAUSE_MTU(MTU) + PAUSE_RTT))

/* Calculate Delay to respond to PFC */
#define IXGBE_PFC_D     672

/* Calculate Cable Delay */
#define IXGBE_CABLE_DC  5556 /* Delay Copper */
#define IXGBE_CABLE_DO  5000 /* Delay Optical */

/* Calculate Interface Delay X540 */
#define IXGBE_PHY_DC    25600 /* Delay 10G BASET */
#define IXGBE_MAC_DC    8192  /* Delay Copper XAUI interface */
#define IXGBE_XAUI_DC   (2 * 2048) /* Delay Copper Phy */

#define IXGBE_ID_X540   (IXGBE_MAC_DC + IXGBE_XAUI_DC + IXGBE_PHY_DC)

/* Calculate Interface Delay 82598, 82599 */
#define IXGBE_PHY_D     12800
#define IXGBE_MAC_D     4096
#define IXGBE_XAUI_D    (2 * 1024)

#define IXGBE_ID        (IXGBE_MAC_D + IXGBE_XAUI_D + IXGBE_PHY_D)

/* Calculate Delay incurred from higher layer */
#define IXGBE_HD        6144

/* Calculate PCI Bus delay for low thresholds */
#define IXGBE_PCI_DELAY 10000

/* Calculate X540 delay value in bit times */
#define IXGBE_DV_X540(_max_frame_link, _max_frame_tc) \
                        ((36 * \
                          (IXGBE_B2BT(_max_frame_link) + \
                           IXGBE_PFC_D + \
                           (2 * IXGBE_CABLE_DC) + \
                           (2 * IXGBE_ID_X540) + \
                           IXGBE_HD) / 25 + 1) + \
                         2 * IXGBE_B2BT(_max_frame_tc))

/* Calculate 82599, 82598 delay value in bit times */
#define IXGBE_DV(_max_frame_link, _max_frame_tc) \
                        ((36 * \
                          (IXGBE_B2BT(_max_frame_link) + \
                           IXGBE_PFC_D + \
                           (2 * IXGBE_CABLE_DC) + \
                           (2 * IXGBE_ID) + \
                           IXGBE_HD) / 25 + 1) + \
                         2 * IXGBE_B2BT(_max_frame_tc))

/* Calculate low threshold delay values */
#define IXGBE_LOW_DV_X540(_max_frame_tc) \
                        (2 * IXGBE_B2BT(_max_frame_tc) + \
                        (36 * IXGBE_PCI_DELAY / 25) + 1)
#define IXGBE_LOW_DV(_max_frame_tc) \
                        (2 * IXGBE_LOW_DV_X540(_max_frame_tc))

/* Software ATR hash keys */
#define IXGBE_ATR_BUCKET_HASH_KEY       0x3DAD14E2
```

```
#define IXGBE_ATR_SIGNATURE_HASH_KEY    0x174D3614

/* Software ATR input stream values and masks */
#define IXGBE_ATR_HASH_MASK             0x7fff
#define IXGBE_ATR_L4TYPE_MASK           0x3
#define IXGBE_ATR_L4TYPE_UDP            0x1
#define IXGBE_ATR_L4TYPE_TCP            0x2
#define IXGBE_ATR_L4TYPE_SCTP           0x3
#define IXGBE_ATR_L4TYPE_IPV6_MASK      0x4
enum ixgbe_atr_flow_type {
        IXGBE_ATR_FLOW_TYPE_IPV4        = 0x0,
        IXGBE_ATR_FLOW_TYPE_UDPV4       = 0x1,
        IXGBE_ATR_FLOW_TYPE_TCPV4       = 0x2,
        IXGBE_ATR_FLOW_TYPE_SCTPV4      = 0x3,
        IXGBE_ATR_FLOW_TYPE_IPV6        = 0x4,
        IXGBE_ATR_FLOW_TYPE_UDPV6       = 0x5,
        IXGBE_ATR_FLOW_TYPE_TCPV6       = 0x6,
        IXGBE_ATR_FLOW_TYPE_SCTPV6      = 0x7,
};

/* Flow Director ATR input struct. */
union ixgbe_atr_input {
        /*
         * Byte layout in order, all values with MSB first:
         *
         * vm_pool      - 1 byte
         * flow_type    - 1 byte
         * vlan_id      - 2 bytes
         * src_ip       - 16 bytes
         * dst_ip       - 16 bytes
         * src_port     - 2 bytes
         * dst_port     - 2 bytes
         * flex_bytes   - 2 bytes
         * bkt_hash     - 2 bytes
         * rsvd0      - 2 bytes - space reserved must be 0.
         */
        struct {
                u8 vm_pool;
                u8 flow_type;
                __be16 vlan_id;
                __be32 dst_ip[4];
                __be32 src_ip[4];
                __be16 src_port;
                __be16 dst_port;
                __be16 flex_bytes;
                __be16 bkt_hash;
                __be16 rsvd0;
        } formatted;
        __be32 dword_stream[11];
};
```

***unchanged_portion_omitted_***

```
struct ixgbe_atr_input_masks {
        __be16 rsvd0;
        __be16 vlan_id_mask;
        __be32 dst_ip_mask[4];
        __be32 src_ip_mask[4];
        __be16 src_port_mask;
        __be16 dst_port_mask;
        __be16 flex_mask;
};
```

```
/*
 * Unavailable: The FCoE Boot Option ROM is not present in the flash.
 * Disabled: Present; boot order is not set for any targets on the port.
```



```
 * Enabled: Present; boot order is set for at least one target on the port.
 */
enum ixgbe_fcoe_boot_status {
        ixgbe_fcoe_bootstatus_disabled = 0,
        ixgbe_fcoe_bootstatus_enabled = 1,
        ixgbe_fcoe_bootstatus_unavailable = 0xFFFF
};
```

***unchanged_portion_omitted_***

```
enum ixgbe_mac_type {
        ixgbe_mac_unknown = 0,
        ixgbe_mac_82598EB,
        ixgbe_mac_82599EB,
        ixgbe_mac_82599_vf,
        ixgbe_mac_X540,
        ixgbe_mac_X540_vf,
        ixgbe_num_macs
};
```

***unchanged_portion_omitted_***

```
/*
 * SFP+ module type IDs:
 *
 * ID   Module Type
 * =============
 * 0    SFP_DA_CU
 * 1    SFP_SR
 * 2    SFP_LR
 * 3    SFP_DA_CU_CORE0 - 82599-specific
 * 4    SFP_DA_CU_CORE1 - 82599-specific
 * 5    SFP_SR/LR_CORE0 - 82599-specific
 * 6    SFP_SR/LR_CORE1 - 82599-specific
 */
enum ixgbe_sfp_type {
        ixgbe_sfp_type_da_cu = 0,
        ixgbe_sfp_type_sr = 1,
        ixgbe_sfp_type_lr = 2,
        ixgbe_sfp_type_da_cu_core0 = 3,
        ixgbe_sfp_type_da_cu_core1 = 4,
        ixgbe_sfp_type_srlr_core0 = 5,
        ixgbe_sfp_type_srlr_core1 = 6,
        ixgbe_sfp_type_da_act_lmt_core0 = 7,
        ixgbe_sfp_type_da_act_lmt_core1 = 8,
        ixgbe_sfp_type_1g_cu_core0 = 9,
        ixgbe_sfp_type_1g_cu_core1 = 10,
        ixgbe_sfp_type_1g_sx_core0 = 11,
        ixgbe_sfp_type_1g_sx_core1 = 12,
        ixgbe_sfp_type_not_present = 0xFFFE,
        ixgbe_sfp_type_unknown = 0xFFFF
};
```

***unchanged_portion_omitted_***

```
/* PCI bus speeds */
enum ixgbe_bus_speed {
        ixgbe_bus_speed_unknown = 0,
        ixgbe_bus_speed_33      = 33,
        ixgbe_bus_speed_66      = 66,
        ixgbe_bus_speed_100     = 100,
        ixgbe_bus_speed_120     = 120,
        ixgbe_bus_speed_133     = 133,
        ixgbe_bus_speed_2500    = 2500,
        ixgbe_bus_speed_5000    = 5000,
        ixgbe_bus_speed_8000    = 8000,
        ixgbe_bus_speed_reserved
};
```

***unchanged_portion_omitted_***

```
2836 /* Flow control parameters */
2837 struct ixgbe_fc_info {
2838         u32 high_water[IXGBE_DCB_MAX_TRAFFIC_CLASS]; /* Flow Ctrl High-water */
2839         u32 low_water[IXGBE_DCB_MAX_TRAFFIC_CLASS]; /* Flow Ctrl Low-water */
2525         u32 high_water; /* Flow Control High-water */
2526         u32 low_water; /* Flow Control Low-water */
2840         u16 pause_time; /* Flow Control Pause timer */
2841         bool send_xon; /* Flow control send XON */
2842         bool strict_ieee; /* Strict IEEE mode */
2843         bool disable_fc_autoneg; /* Do not autonegotiate FC */
2844         bool fc_was_autonegged; /* Is current_mode the result of autonegging? */
2845         enum ixgbe_fc_mode current_mode; /* FC mode in effect */
2846         enum ixgbe_fc_mode requested_mode; /* FC mode requested by caller */
2847 };

2849 /* Statistics counters collected by the MAC */
2850 struct ixgbe_hw_stats {
2851         u64 crcerrs;
2852         u64 illerrc;
2853         u64 errbc;
2854         u64 mspdc;
2855         u64 mpctotal;
2856         u64 mpc[8];
2857         u64 mlfc;
2858         u64 mrfc;
2859         u64 rlec;
2860         u64 lxontxc;
2861         u64 lxonrxc;
2862         u64 lxofftxc;
2863         u64 lxoffrxc;
2864         u64 pxontxc[8];
2865         u64 pxonrxc[8];
2866         u64 pxofftxc[8];
2867         u64 pxoffrxc[8];
2868         u64 prc64;
2869         u64 prc127;
2870         u64 prc255;
2871         u64 prc511;
2872         u64 prc1023;
2873         u64 prc1522;
2874         u64 gprc;
2875         u64 bprc;
2876         u64 mprc;
2877         u64 gptc;
2878         u64 gorc;
2879         u64 gotc;
2880         u64 rnbc[8];
2881         u64 ruc;
2882         u64 rfc;
2883         u64 roc;
2884         u64 rjc;
2885         u64 mngprc;
2886         u64 mngpdc;
2887         u64 mngptc;
2888         u64 tor;
2889         u64 tpr;
2890         u64 tpt;
2891         u64 ptc64;
2892         u64 ptc127;
2893         u64 ptc255;
2894         u64 ptc511;
2895         u64 ptc1023;
2896         u64 ptc1522;
2897         u64 mptc;
2898         u64 bptc;
```



```
2899         u64 xec;
2900         u64 qprc[16];
2901         u64 qptc[16];
2902         u64 qbrc[16];
2903         u64 qbtc[16];
2904         u64 qprdc[16];
2905         u64 pxon2offc[8];
2906         u64 fdirustat_add;
2907         u64 fdirustat_remove;
2908         u64 fdirfstat_fadd;
2909         u64 fdirfstat_fremove;
2910         u64 fdirmatch;
2911         u64 fdirmiss;
2912         u64 fccrc;
2913         u64 fclast;
2914         u64 fcoerpdc;
2915         u64 fcoeprc;
2916         u64 fcoeptc;
2917         u64 fcoedwrc;
2918         u64 fcoedwtc;
2919         u64 fcoe_noddp;
2920         u64 fcoe_noddp_ext_buff;
2921         u64 ldpcec;
2922         u64 pcrc8ec;
2923         u64 b2ospc;
2924         u64 b2ogprc;
2925         u64 o2bgptc;
2926         u64 o2bspc;
2927 };

2929 /* forward declaration */
2930 struct ixgbe_hw;

2932 /* iterator type for walking multicast address lists */
2933 typedef u8* (*ixgbe_mc_addr_itr) (struct ixgbe_hw *hw, u8 **mc_addr_ptr,
2934                                   u32 *vmdq);

2936 /* Function pointer table */
2937 struct ixgbe_eeprom_operations {
2938         s32 (*init_params)(struct ixgbe_hw *);
2939         s32 (*read)(struct ixgbe_hw *, u16, u16 *);
2940         s32 (*read_buffer)(struct ixgbe_hw *, u16, u16, u16 *);
2941         s32 (*write)(struct ixgbe_hw *, u16, u16);
2942         s32 (*write_buffer)(struct ixgbe_hw *, u16, u16, u16 *);
2943         s32 (*validate_checksum)(struct ixgbe_hw *, u16 *);
2944         s32 (*update_checksum)(struct ixgbe_hw *);
2945         u16 (*calc_checksum)(struct ixgbe_hw *);
2946 };

2948 struct ixgbe_mac_operations {
2949         s32 (*init_hw)(struct ixgbe_hw *);
2950         s32 (*reset_hw)(struct ixgbe_hw *);
2951         s32 (*start_hw)(struct ixgbe_hw *);
2952         s32 (*clear_hw_cntrs)(struct ixgbe_hw *);
2953         void (*enable_relaxed_ordering)(struct ixgbe_hw *);
2954         enum ixgbe_media_type (*get_media_type)(struct ixgbe_hw *);
2955         u32 (*get_supported_physical_layer)(struct ixgbe_hw *);
2956         s32 (*get_mac_addr)(struct ixgbe_hw *, u8 *);
2957         s32 (*get_san_mac_addr)(struct ixgbe_hw *, u8 *);
2958         s32 (*set_san_mac_addr)(struct ixgbe_hw *, u8 *);
2959         s32 (*get_device_caps)(struct ixgbe_hw *, u16 *);
2960         s32 (*get_wwn_prefix)(struct ixgbe_hw *, u16 *, u16 *);
2961         s32 (*get_fcoe_boot_status)(struct ixgbe_hw *, u16 *);
2962         s32 (*stop_adapter)(struct ixgbe_hw *);
2963         s32 (*get_bus_info)(struct ixgbe_hw *);
2964         void (*set_lan_id)(struct ixgbe_hw *);
```

```
2965         s32 (*read_analog_reg8)(struct ixgbe_hw*, u32, u8*);
2966         s32 (*write_analog_reg8)(struct ixgbe_hw*, u32, u8);
2967         s32 (*setup_sfp)(struct ixgbe_hw *);
2968         s32 (*enable_rx_dma)(struct ixgbe_hw *, u32);
2969         s32 (*disable_sec_rx_path)(struct ixgbe_hw *);
2970         s32 (*enable_sec_rx_path)(struct ixgbe_hw *);
2971         s32 (*acquire_swfw_sync)(struct ixgbe_hw *, u16);
2972         void (*release_swfw_sync)(struct ixgbe_hw *, u16);

2974         /* Link */
2975         void (*disable_tx_laser)(struct ixgbe_hw *);
2976         void (*enable_tx_laser)(struct ixgbe_hw *);
2977         void (*flap_tx_laser)(struct ixgbe_hw *);
2978         s32 (*setup_link)(struct ixgbe_hw *, ixgbe_link_speed, bool, bool);
2979         s32 (*check_link)(struct ixgbe_hw *, ixgbe_link_speed *, bool *, bool);
2980         s32 (*get_link_capabilities)(struct ixgbe_hw *, ixgbe_link_speed *,
2981                                      bool *);

2983         /* Packet Buffer manipulation */
2984         void (*setup_rxpba)(struct ixgbe_hw *, int, u32, int);

2986         /* LED */
2987         s32 (*led_on)(struct ixgbe_hw *, u32);
2988         s32 (*led_off)(struct ixgbe_hw *, u32);
2989         s32 (*blink_led_start)(struct ixgbe_hw *, u32);
2990         s32 (*blink_led_stop)(struct ixgbe_hw *, u32);

2992         /* RAR, Multicast, VLAN */
2993         s32 (*set_rar)(struct ixgbe_hw *, u32, u8 *, u32, u32);
2994         s32 (*set_uc_addr)(struct ixgbe_hw *, u32, u8 *);
2995         s32 (*clear_rar)(struct ixgbe_hw *, u32);
2996         s32 (*insert_mac_addr)(struct ixgbe_hw *, u8 *, u32);
2997         s32 (*set_vmdq)(struct ixgbe_hw *, u32, u32);
2998         s32 (*set_vmdq_san_mac)(struct ixgbe_hw *, u32);
2999         s32 (*clear_vmdq)(struct ixgbe_hw *, u32, u32);
3000         s32 (*init_rx_addrs)(struct ixgbe_hw *);
3001         s32 (*update_uc_addr_list)(struct ixgbe_hw *, u8 *, u32,
3002                                    ixgbe_mc_addr_itr);
3003         s32 (*update_mc_addr_list)(struct ixgbe_hw *, u8 *, u32,
3004                                    ixgbe_mc_addr_itr, bool clear);
2674                                    ixgbe_mc_addr_itr);
3005         s32 (*enable_mc)(struct ixgbe_hw *);
3006         s32 (*disable_mc)(struct ixgbe_hw *);
3007         s32 (*clear_vfta)(struct ixgbe_hw *);
3008         s32 (*set_vfta)(struct ixgbe_hw *, u32, u32, bool);
3009         s32 (*set_vlvf)(struct ixgbe_hw *, u32, u32, bool, bool *);
3010         s32 (*init_uta_tables)(struct ixgbe_hw *);
3011         void (*set_mac_anti_spoofing)(struct ixgbe_hw *, bool, int);
3012         void (*set_vlan_anti_spoofing)(struct ixgbe_hw *, bool, int);

3014         /* Flow Control */
3015         s32 (*fc_enable)(struct ixgbe_hw *);

3017         /* Manageability interface */
3018         s32 (*set_fw_drv_ver)(struct ixgbe_hw *, u8, u8, u8, u8);
2684         s32 (*fc_enable)(struct ixgbe_hw *, s32);
3019 };
```

*unchanged_portion_omitted_*

```
3041 struct ixgbe_eeprom_info {
3042         struct ixgbe_eeprom_operations ops;
3043         enum ixgbe_eeprom_type type;
3044         u32 semaphore_delay;
3045         u16 word_size;
3046         u16 address_bits;
3047         u16 word_page_size;
```



```
3048 };

3050 #define IXGBE_FLAGS_DOUBLE_RESET_REQUIRED       0x01
3051 struct ixgbe_mac_info {
3052         struct ixgbe_mac_operations ops;
3053         enum ixgbe_mac_type type;
3054         u8 addr[IXGBE_ETH_LENGTH_OF_ADDRESS];
3055         u8 perm_addr[IXGBE_ETH_LENGTH_OF_ADDRESS];
3056         u8 san_addr[IXGBE_ETH_LENGTH_OF_ADDRESS];
3057         /* prefix for World Wide Node Name (WWNN) */
3058         u16 wwnn_prefix;
3059         /* prefix for World Wide Port Name (WWPN) */
3060         u16 wwpn_prefix;
3061 #define IXGBE_MAX_MTA                   128
3062         u32 mta_shadow[IXGBE_MAX_MTA];
3063         s32 mc_filter_type;
3064         u32 mcft_size;
3065         u32 vft_size;
3066         u32 num_rar_entries;
3067         u32 rar_highwater;
3068         u32 rx_pb_size;
3069         u32 max_tx_queues;
3070         u32 max_rx_queues;
2736         u32                             max_msix_vectors;
2737         bool                            msix_vectors_from_pcie;
3071         u32 orig_autoc;
3072         u8  san_mac_rar_index;
3073         u32 orig_autoc2;
3074         u16 max_msix_vectors;
3075         bool arc_subsystem_valid;
3076         bool orig_link_settings_stored;
3077         bool autotry_restart;
3078         u8 flags;
3079 };
```

*unchanged_portion_omitted_*

```
3098 #include "ixgbe_mbx.h"

3100 struct ixgbe_mbx_operations {
3101         void (*init_params)(struct ixgbe_hw *hw);
3102         s32  (*read)(struct ixgbe_hw *, u32 *, u16,  u16);
3103         s32  (*write)(struct ixgbe_hw *, u32 *, u16, u16);
3104         s32  (*read_posted)(struct ixgbe_hw *, u32 *, u16,  u16);
3105         s32  (*write_posted)(struct ixgbe_hw *, u32 *, u16, u16);
3106         s32  (*check_for_msg)(struct ixgbe_hw *, u16);
3107         s32  (*check_for_ack)(struct ixgbe_hw *, u16);
3108         s32  (*check_for_rst)(struct ixgbe_hw *, u16);
3109 };

3111 struct ixgbe_mbx_stats {
3112         u32 msgs_tx;
3113         u32 msgs_rx;

3115         u32 acks;
3116         u32 reqs;
3117         u32 rsts;
3118 };

3120 struct ixgbe_mbx_info {
3121         struct ixgbe_mbx_operations ops;
3122         struct ixgbe_mbx_stats stats;
3123         u32 timeout;
3124         u32 usec_delay;
3125         u32 v2p_mailbox;
3126         u16 size;
3127 };
```

```
3129 struct ixgbe_hw {
3130         u8 *hw_addr;
3131         void *back;
3132         struct ixgbe_mac_info mac;
3133         struct ixgbe_addr_filter_info addr_ctrl;
3134         struct ixgbe_fc_info fc;
3135         struct ixgbe_phy_info phy;
3136         struct ixgbe_eeprom_info eeprom;
3137         struct ixgbe_bus_info bus;
3138         struct ixgbe_mbx_info mbx;
3139         u16 device_id;
3140         u16 vendor_id;
3141         u16 subsystem_device_id;
3142         u16 subsystem_vendor_id;
3143         u8 revision_id;
3144         bool adapter_stopped;
3145         bool force_full_reset;
3146         bool allow_unsupported_sfp;
3147 };

3149 #define ixgbe_call_func(hw, func, params, error) \
3150                 (func != NULL) ? func params : error


3153 /* Error Codes */
3154 #define IXGBE_SUCCESS                           0
3155 #define IXGBE_ERR_EEPROM                        -1
3156 #define IXGBE_ERR_EEPROM_CHECKSUM               -2
3157 #define IXGBE_ERR_PHY                           -3
3158 #define IXGBE_ERR_CONFIG                        -4
3159 #define IXGBE_ERR_PARAM                         -5
3160 #define IXGBE_ERR_MAC_TYPE                      -6
3161 #define IXGBE_ERR_UNKNOWN_PHY                   -7
3162 #define IXGBE_ERR_LINK_SETUP                    -8
3163 #define IXGBE_ERR_ADAPTER_STOPPED               -9
3164 #define IXGBE_ERR_INVALID_MAC_ADDR              -10
3165 #define IXGBE_ERR_DEVICE_NOT_SUPPORTED          -11
3166 #define IXGBE_ERR_MASTER_REQUESTS_PENDING       -12
3167 #define IXGBE_ERR_INVALID_LINK_SETTINGS         -13
3168 #define IXGBE_ERR_AUTONEG_NOT_COMPLETE          -14
3169 #define IXGBE_ERR_RESET_FAILED                  -15
3170 #define IXGBE_ERR_SWFW_SYNC                     -16
3171 #define IXGBE_ERR_PHY_ADDR_INVALID              -17
3172 #define IXGBE_ERR_I2C                           -18
3173 #define IXGBE_ERR_SFP_NOT_SUPPORTED             -19
3174 #define IXGBE_ERR_SFP_NOT_PRESENT               -20
3175 #define IXGBE_ERR_SFP_NO_INIT_SEQ_PRESENT       -21
3176 #define IXGBE_ERR_NO_SAN_ADDR_PTR               -22
3177 #define IXGBE_ERR_FDIR_REINIT_FAILED            -23
3178 #define IXGBE_ERR_EEPROM_VERSION                -24
3179 #define IXGBE_ERR_NO_SPACE                      -25
3180 #define IXGBE_ERR_OVERTEMP                      -26
3181 #define IXGBE_ERR_FC_NOT_NEGOTIATED             -27
3182 #define IXGBE_ERR_FC_NOT_SUPPORTED              -28
2813 #define IXGBE_ERR_FLOW_CONTROL                  -29
3183 #define IXGBE_ERR_SFP_SETUP_NOT_COMPLETE        -30
3184 #define IXGBE_ERR_PBA_SECTION                   -31
3185 #define IXGBE_ERR_INVALID_ARGUMENT              -32
3186 #define IXGBE_ERR_HOST_INTERFACE_COMMAND        -33
3187 #define IXGBE_ERR_OUT_OF_MEM                    -34

3189 #define IXGBE_NOT_IMPLEMENTED                   0x7FFFFFFF


3192 #endif /* _IXGBE_TYPE_H_ */
```

```
**********************************************************
   27909 Thu Jul 12 12:22:41 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_x540.c
XXXX Intel X540 support
**********************************************************
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_x540.c,v 1.2 2012/07/05 20:51:44 jfv Exp $*/

#include "ixgbe_x540.h"
#include "ixgbe_type.h"
#include "ixgbe_api.h"
#include "ixgbe_common.h"
#include "ixgbe_phy.h"

static s32 ixgbe_update_flash_X540(struct ixgbe_hw *hw);
static s32 ixgbe_poll_flash_update_done_X540(struct ixgbe_hw *hw);
static s32 ixgbe_get_swfw_sync_semaphore(struct ixgbe_hw *hw);
static void ixgbe_release_swfw_sync_semaphore(struct ixgbe_hw *hw);

/**
 *  ixgbe_init_ops_X540 - Inits func ptrs and MAC type
 *  @hw: pointer to hardware structure
 *
 *  Initialize the function pointers and assign the MAC type for X540.
 *  Does not touch the hardware.
 **/
s32 ixgbe_init_ops_X540(struct ixgbe_hw *hw)
{
        struct ixgbe_mac_info *mac = &hw->mac;
        struct ixgbe_phy_info *phy = &hw->phy;
        struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
        s32 ret_val;

        DEBUGFUNC("ixgbe_init_ops_X540");
```



```
        ret_val = ixgbe_init_phy_ops_generic(hw);
        ret_val = ixgbe_init_ops_generic(hw);


        /* EEPROM */
        eeprom->ops.init_params = &ixgbe_init_eeprom_params_X540;
        eeprom->ops.read = &ixgbe_read_eerd_X540;
        eeprom->ops.read_buffer = &ixgbe_read_eerd_buffer_X540;
        eeprom->ops.write = &ixgbe_write_eewr_X540;
        eeprom->ops.write_buffer = &ixgbe_write_eewr_buffer_X540;
        eeprom->ops.update_checksum = &ixgbe_update_eeprom_checksum_X540;
        eeprom->ops.validate_checksum = &ixgbe_validate_eeprom_checksum_X540;
        eeprom->ops.calc_checksum = &ixgbe_calc_eeprom_checksum_X540;

        /* PHY */
        phy->ops.init = &ixgbe_init_phy_ops_generic;
        phy->ops.reset = NULL;

        /* MAC */
        mac->ops.reset_hw = &ixgbe_reset_hw_X540;
        mac->ops.enable_relaxed_ordering = &ixgbe_enable_relaxed_ordering_gen2;
        mac->ops.get_media_type = &ixgbe_get_media_type_X540;
        mac->ops.get_supported_physical_layer =
                                &ixgbe_get_supported_physical_layer_X540;
        mac->ops.read_analog_reg8 = NULL;
        mac->ops.write_analog_reg8 = NULL;
        mac->ops.start_hw = &ixgbe_start_hw_X540;
        mac->ops.get_san_mac_addr = &ixgbe_get_san_mac_addr_generic;
        mac->ops.set_san_mac_addr = &ixgbe_set_san_mac_addr_generic;
        mac->ops.get_device_caps = &ixgbe_get_device_caps_generic;
        mac->ops.get_wwn_prefix = &ixgbe_get_wwn_prefix_generic;
        mac->ops.get_fcoe_boot_status = &ixgbe_get_fcoe_boot_status_generic;
        mac->ops.acquire_swfw_sync = &ixgbe_acquire_swfw_sync_X540;
        mac->ops.release_swfw_sync = &ixgbe_release_swfw_sync_X540;
        mac->ops.disable_sec_rx_path = &ixgbe_disable_sec_rx_path_generic;
        mac->ops.enable_sec_rx_path = &ixgbe_enable_sec_rx_path_generic;

        /* RAR, Multicast, VLAN */
        mac->ops.set_vmdq = &ixgbe_set_vmdq_generic;
        mac->ops.set_vmdq_san_mac = &ixgbe_set_vmdq_san_mac_generic;
        mac->ops.clear_vmdq = &ixgbe_clear_vmdq_generic;
        mac->ops.insert_mac_addr = &ixgbe_insert_mac_addr_generic;
        mac->rar_highwater = 1;
        mac->ops.set_vfta = &ixgbe_set_vfta_generic;
        mac->ops.set_vlvf = &ixgbe_set_vlvf_generic;
        mac->ops.clear_vfta = &ixgbe_clear_vfta_generic;
        mac->ops.init_uta_tables = &ixgbe_init_uta_tables_generic;
        mac->ops.set_mac_anti_spoofing = &ixgbe_set_mac_anti_spoofing;
        mac->ops.set_vlan_anti_spoofing = &ixgbe_set_vlan_anti_spoofing;

        /* Link */
        mac->ops.get_link_capabilities =
                                &ixgbe_get_copper_link_capabilities_generic;
        mac->ops.setup_link = &ixgbe_setup_mac_link_X540;
        mac->ops.setup_rxpba = &ixgbe_set_rxpba_generic;
        mac->ops.check_link = &ixgbe_check_mac_link_generic;

        mac->mcft_size          = 128;
        mac->vft_size           = 128;
        mac->num_rar_entries    = 128;
        mac->rx_pb_size         = 384;
        mac->max_tx_queues      = 128;
        mac->max_rx_queues      = 128;
        mac->max_msix_vectors   = ixgbe_get_pcie_msix_count_generic(hw);

        /*
```

```
128          * FWSM register
129          * ARC supported; valid only if manageability features are
130          * enabled.
131          */
132         mac->arc_subsystem_valid = (IXGBE_READ_REG(hw, IXGBE_FWSM) &
133                                     IXGBE_FWSM_MODE_MASK) ? TRUE : FALSE;

135         hw->mbx.ops.init_params = ixgbe_init_mbx_params_pf;

137         /* LEDs */
138         mac->ops.blink_led_start = ixgbe_blink_led_start_X540;
139         mac->ops.blink_led_stop = ixgbe_blink_led_stop_X540;

141         /* Manageability interface */
142         mac->ops.set_fw_drv_ver = &ixgbe_set_fw_drv_ver_generic;

144         return ret_val;
145 }

147 /**
148  *  ixgbe_get_link_capabilities_X540 - Determines link capabilities
149  *  @hw: pointer to hardware structure
150  *  @speed: pointer to link speed
151  *  @autoneg: TRUE when autoneg or autotry is enabled
152  *
153  *  Determines the link capabilities by reading the AUTOC register.
154  **/
155 s32 ixgbe_get_link_capabilities_X540(struct ixgbe_hw *hw,
156                                      ixgbe_link_speed *speed,
157                                      bool *autoneg)
158 {
159         ixgbe_get_copper_link_capabilities_generic(hw, speed, autoneg);

161         return IXGBE_SUCCESS;
162 }

164 /**
165  *  ixgbe_get_media_type_X540 - Get media type
166  *  @hw: pointer to hardware structure
167  *
168  *  Returns the media type (fiber, copper, backplane)
169  **/
170 enum ixgbe_media_type ixgbe_get_media_type_X540(struct ixgbe_hw *hw)
171 {
172         UNREFERENCED_1PARAMETER(hw);
173         return ixgbe_media_type_copper;
174 }

176 /**
177  *  ixgbe_setup_mac_link_X540 - Sets the auto advertised capabilities
178  *  @hw: pointer to hardware structure
179  *  @speed: new link speed
180  *  @autoneg: TRUE if autonegotiation enabled
181  *  @autoneg_wait_to_complete: TRUE when waiting for completion is needed
182  **/
183 s32 ixgbe_setup_mac_link_X540(struct ixgbe_hw *hw,
184                               ixgbe_link_speed speed, bool autoneg,
185                               bool autoneg_wait_to_complete)
186 {
187         DEBUGFUNC("ixgbe_setup_mac_link_X540");
188         return hw->phy.ops.setup_link_speed(hw, speed, autoneg,
189                                             autoneg_wait_to_complete);
190 }

192 /**
193  *  ixgbe_reset_hw_X540 - Perform hardware reset
```



```
194  *  @hw: pointer to hardware structure
195  *
196  *  Resets the hardware by resetting the transmit and receive units, masks
197  *  and clears all interrupts, and perform a reset.
198  **/
199 s32 ixgbe_reset_hw_X540(struct ixgbe_hw *hw)
200 {
201         s32 status;
202         u32 ctrl, i;

204         DEBUGFUNC("ixgbe_reset_hw_X540");

206         /* Call adapter stop to disable tx/rx and clear interrupts */
207         status = hw->mac.ops.stop_adapter(hw);
208         if (status != IXGBE_SUCCESS)
209                 goto reset_hw_out;

211         /* flush pending Tx transactions */
212         ixgbe_clear_tx_pending(hw);

214 mac_reset_top:
215         ctrl = IXGBE_CTRL_RST;
216         ctrl |= IXGBE_READ_REG(hw, IXGBE_CTRL);
217         IXGBE_WRITE_REG(hw, IXGBE_CTRL, ctrl);
218         IXGBE_WRITE_FLUSH(hw);

220         /* Poll for reset bit to self-clear indicating reset is complete */
221         for (i = 0; i < 10; i++) {
222                 usec_delay(1);
223                 ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
224                 if (!(ctrl & IXGBE_CTRL_RST_MASK))
225                         break;
226         }

228         if (ctrl & IXGBE_CTRL_RST_MASK) {
229                 status = IXGBE_ERR_RESET_FAILED;
230                 DEBUGOUT("Reset polling failed to complete.\n");
231         }
232         msec_delay(100);

234         /*
235          * Double resets are required for recovery from certain error
236          * conditions.  Between resets, it is necessary to stall to allow time
237          * for any pending HW events to complete.
238          */
239         if (hw->mac.flags & IXGBE_FLAGS_DOUBLE_RESET_REQUIRED) {
240                 hw->mac.flags &= ~IXGBE_FLAGS_DOUBLE_RESET_REQUIRED;
241                 goto mac_reset_top;
242         }

244         /* Set the Rx packet buffer size. */
245         IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(0), 384 << IXGBE_RXPBSIZE_SHIFT);

247         /* Store the permanent mac address */
248         hw->mac.ops.get_mac_addr(hw, hw->mac.perm_addr);

250         /*
251          * Store MAC address from RAR0, clear receive address registers, and
252          * clear the multicast table.  Also reset num_rar_entries to 128,
253          * since we modify this value when programming the SAN MAC address.
254          */
255         hw->mac.num_rar_entries = 128;
256         hw->mac.ops.init_rx_addrs(hw);

258         /* Store the permanent SAN mac address */
259         hw->mac.ops.get_san_mac_addr(hw, hw->mac.san_addr);
```

```
261         /* Add the SAN MAC address to the RAR only if it's a valid address */
262         if (ixgbe_validate_mac_addr(hw->mac.san_addr) == 0) {
263                 hw->mac.ops.set_rar(hw, hw->mac.num_rar_entries - 1,
264                                     hw->mac.san_addr, 0, IXGBE_RAH_AV);

266                 /* Save the SAN MAC RAR index */
267                 hw->mac.san_mac_rar_index = hw->mac.num_rar_entries - 1;

269                 /* Reserve the last RAR for the SAN MAC address */
270                 hw->mac.num_rar_entries--;
271         }

273         /* Store the alternative WWNN/WWPN prefix */
274         hw->mac.ops.get_wwn_prefix(hw, &hw->mac.wwnn_prefix,
275                                    &hw->mac.wwpn_prefix);

277 reset_hw_out:
278         return status;
279 }

281 /**
282  *  ixgbe_start_hw_X540 - Prepare hardware for Tx/Rx
283  *  @hw: pointer to hardware structure
284  *
285  *  Starts the hardware using the generic start_hw function
286  *  and the generation start_hw function.
287  *  Then performs revision-specific operations, if any.
288  **/
289 s32 ixgbe_start_hw_X540(struct ixgbe_hw *hw)
290 {
291         s32 ret_val = IXGBE_SUCCESS;

293         DEBUGFUNC("ixgbe_start_hw_X540");

295         ret_val = ixgbe_start_hw_generic(hw);
296         if (ret_val != IXGBE_SUCCESS)
297                 goto out;

299         ret_val = ixgbe_start_hw_gen2(hw);

301 out:
302         return ret_val;
303 }

305 /**
306  *  ixgbe_get_supported_physical_layer_X540 - Returns physical layer type
307  *  @hw: pointer to hardware structure
308  *
309  *  Determines physical layer capabilities of the current configuration.
310  **/
311 u32 ixgbe_get_supported_physical_layer_X540(struct ixgbe_hw *hw)
312 {
313         u32 physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
314         u16 ext_ability = 0;

316         DEBUGFUNC("ixgbe_get_supported_physical_layer_X540");

318         hw->phy.ops.read_reg(hw, IXGBE_MDIO_PHY_EXT_ABILITY,
319         IXGBE_MDIO_PMA_PMD_DEV_TYPE, &ext_ability);
320         if (ext_ability & IXGBE_MDIO_PHY_10GBASET_ABILITY)
321                 physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_T;
322         if (ext_ability & IXGBE_MDIO_PHY_1000BASET_ABILITY)
323                 physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_T;
324         if (ext_ability & IXGBE_MDIO_PHY_100BASETX_ABILITY)
325                 physical_layer |= IXGBE_PHYSICAL_LAYER_100BASE_TX;
```



```
327         return physical_layer;
328 }

330 /**
331  *  ixgbe_init_eeprom_params_X540 - Initialize EEPROM params
332  *  @hw: pointer to hardware structure
333  *
334  *  Initializes the EEPROM parameters ixgbe_eeprom_info within the
335  *  ixgbe_hw struct in order to set up EEPROM access.
336  **/
337 s32 ixgbe_init_eeprom_params_X540(struct ixgbe_hw *hw)
338 {
339         struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
340         u32 eec;
341         u16 eeprom_size;

343         DEBUGFUNC("ixgbe_init_eeprom_params_X540");

345         if (eeprom->type == ixgbe_eeprom_uninitialized) {
346                 eeprom->semaphore_delay = 10;
347                 eeprom->type = ixgbe_flash;

349                 eec = IXGBE_READ_REG(hw, IXGBE_EEC);
350                 eeprom_size = (u16)((eec & IXGBE_EEC_SIZE) >>
351                                     IXGBE_EEC_SIZE_SHIFT);
352                 eeprom->word_size = 1 << (eeprom_size +
353                                           IXGBE_EEPROM_WORD_SIZE_SHIFT);

355                 DEBUGOUT2("Eeprom params: type = %d, size = %d\n",
356                           eeprom->type, eeprom->word_size);
357         }

359         return IXGBE_SUCCESS;
360 }

362 /**
363  *  ixgbe_read_eerd_X540- Read EEPROM word using EERD
364  *  @hw: pointer to hardware structure
365  *  @offset: offset of  word in the EEPROM to read
366  *  @data: word read from the EEPROM
367  *
368  *  Reads a 16 bit word from the EEPROM using the EERD register.
369  **/
370 s32 ixgbe_read_eerd_X540(struct ixgbe_hw *hw, u16 offset, u16 *data)
371 {
372         s32 status = IXGBE_SUCCESS;

374         DEBUGFUNC("ixgbe_read_eerd_X540");
375         if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
376             IXGBE_SUCCESS)
377                 status = ixgbe_read_eerd_generic(hw, offset, data);
378         else
379                 status = IXGBE_ERR_SWFW_SYNC;

381         hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
382         return status;
383 }

385 /**
386  *  ixgbe_read_eerd_buffer_X540- Read EEPROM word(s) using EERD
387  *  @hw: pointer to hardware structure
388  *  @offset: offset of  word in the EEPROM to read
389  *  @words: number of words
390  *  @data: word(s) read from the EEPROM
391  *
```

```
 *  Reads a 16 bit word(s) from the EEPROM using the EERD register.
 **/
s32 ixgbe_read_eerd_buffer_X540(struct ixgbe_hw *hw,
                                u16 offset, u16 words, u16 *data)
{
        s32 status = IXGBE_SUCCESS;

        DEBUGFUNC("ixgbe_read_eerd_buffer_X540");
        if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
            IXGBE_SUCCESS)
                status = ixgbe_read_eerd_buffer_generic(hw, offset,
                                                        words, data);
        else
                status = IXGBE_ERR_SWFW_SYNC;

        hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
        return status;
}

/**
 *  ixgbe_write_eewr_X540 - Write EEPROM word using EEWR
 *  @hw: pointer to hardware structure
 *  @offset: offset of  word in the EEPROM to write
 *  @data: word write to the EEPROM
 *
 *  Write a 16 bit word to the EEPROM using the EEWR register.
 **/
s32 ixgbe_write_eewr_X540(struct ixgbe_hw *hw, u16 offset, u16 data)
{
        s32 status = IXGBE_SUCCESS;

        DEBUGFUNC("ixgbe_write_eewr_X540");
        if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
            IXGBE_SUCCESS)
                status = ixgbe_write_eewr_generic(hw, offset, data);
        else
                status = IXGBE_ERR_SWFW_SYNC;

        hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
        return status;
}

/**
 *  ixgbe_write_eewr_buffer_X540 - Write EEPROM word(s) using EEWR
 *  @hw: pointer to hardware structure
 *  @offset: offset of  word in the EEPROM to write
 *  @words: number of words
 *  @data: word(s) write to the EEPROM
 *
 *  Write a 16 bit word(s) to the EEPROM using the EEWR register.
 **/
s32 ixgbe_write_eewr_buffer_X540(struct ixgbe_hw *hw,
                                 u16 offset, u16 words, u16 *data)
{
        s32 status = IXGBE_SUCCESS;

        DEBUGFUNC("ixgbe_write_eewr_buffer_X540");
        if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
            IXGBE_SUCCESS)
                status = ixgbe_write_eewr_buffer_generic(hw, offset,
                                                         words, data);
        else
                status = IXGBE_ERR_SWFW_SYNC;

        hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
        return status;
```



```
}

/**
 *  ixgbe_calc_eeprom_checksum_X540 - Calculates and returns the checksum
 *
 *  This function does not use synchronization for EERD and EEWR. It can
 *  be used internally by function which utilize ixgbe_acquire_swfw_sync_X540.
 *
 *  @hw: pointer to hardware structure
 **/
u16 ixgbe_calc_eeprom_checksum_X540(struct ixgbe_hw *hw)
{
        u16 i;
        u16 j;
        u16 checksum = 0;
        u16 length = 0;
        u16 pointer = 0;
        u16 word = 0;

        /*
         * Do not use hw->eeprom.ops.read because we do not want to take
         * the synchronization semaphores here. Instead use
         * ixgbe_read_eerd_generic
         */

        DEBUGFUNC("ixgbe_calc_eeprom_checksum_X540");

        /* Include 0x0-0x3F in the checksum */
        for (i = 0; i < IXGBE_EEPROM_CHECKSUM; i++) {
                if (ixgbe_read_eerd_generic(hw, i, &word) != IXGBE_SUCCESS) {
                        DEBUGOUT("EEPROM read failed\n");
                        break;
                }
                checksum += word;
        }

        /*
         * Include all data from pointers 0x3, 0x6-0xE.  This excludes the
         * FW, PHY module, and PCIe Expansion/Option ROM pointers.
         */
        for (i = IXGBE_PCIE_ANALOG_PTR; i < IXGBE_FW_PTR; i++) {
                if (i == IXGBE_PHY_PTR || i == IXGBE_OPTION_ROM_PTR)
                        continue;

                if (ixgbe_read_eerd_generic(hw, i, &pointer) != IXGBE_SUCCESS) {
                        DEBUGOUT("EEPROM read failed\n");
                        break;
                }

                /* Skip pointer section if the pointer is invalid. */
                if (pointer == 0xFFFF || pointer == 0 ||
                    pointer >= hw->eeprom.word_size)
                        continue;

                if (ixgbe_read_eerd_generic(hw, pointer, &length) !=
                    IXGBE_SUCCESS) {
                        DEBUGOUT("EEPROM read failed\n");
                        break;
                }

                /* Skip pointer section if length is invalid. */
                if (length == 0xFFFF || length == 0 ||
                    (pointer + length) >= hw->eeprom.word_size)
                        continue;

                for (j = pointer+1; j <= pointer+length; j++) {
```

```
524                         if (ixgbe_read_eerd_generic(hw, j, &word) !=
525                             IXGBE_SUCCESS) {
526                                 DEBUGOUT("EEPROM read failed\n");
527                                 break;
528                         }
529                         checksum += word;
530                 }
531         }

533         checksum = (u16)IXGBE_EEPROM_SUM - checksum;

535         return checksum;
536 }

538 /**
539  *  ixgbe_validate_eeprom_checksum_X540 - Validate EEPROM checksum
540  *  @hw: pointer to hardware structure
541  *  @checksum_val: calculated checksum
542  *
543  *  Performs checksum calculation and validates the EEPROM checksum.  If the
544  *  caller does not need checksum_val, the value can be NULL.
545  **/
546 s32 ixgbe_validate_eeprom_checksum_X540(struct ixgbe_hw *hw,
547                                         u16 *checksum_val)
548 {
549         s32 status;
550         u16 checksum;
551         u16 read_checksum = 0;

553         DEBUGFUNC("ixgbe_validate_eeprom_checksum_X540");

555         /*
556          * Read the first word from the EEPROM. If this times out or fails, do
557          * not continue or we could be in for a very long wait while every
558          * EEPROM read fails
559          */
560         status = hw->eeprom.ops.read(hw, 0, &checksum);

562         if (status != IXGBE_SUCCESS) {
563                 DEBUGOUT("EEPROM read failed\n");
564                 goto out;
565         }

567         if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
568             IXGBE_SUCCESS) {
569                 checksum = hw->eeprom.ops.calc_checksum(hw);

571                 /*
572                  * Do not use hw->eeprom.ops.read because we do not want to take
573                  * the synchronization semaphores twice here.
574                  */
575                 ixgbe_read_eerd_generic(hw, IXGBE_EEPROM_CHECKSUM,
576                                         &read_checksum);

578                 /*
579                  * Verify read checksum from EEPROM is the same as
580                  * calculated checksum
581                  */
582                 if (read_checksum != checksum)
583                         status = IXGBE_ERR_EEPROM_CHECKSUM;

585                 /* If the user cares, return the calculated checksum */
586                 if (checksum_val)
587                         *checksum_val = checksum;
588         } else {
589                 status = IXGBE_ERR_SWFW_SYNC;
```



```
590         }

592         hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
593 out:
594         return status;
595 }

597 /**
598  * ixgbe_update_eeprom_checksum_X540 - Updates the EEPROM checksum and flash
599  * @hw: pointer to hardware structure
600  *
601  * After writing EEPROM to shadow RAM using EEWR register, software calculates
602  * checksum and updates the EEPROM and instructs the hardware to update
603  * the flash.
604  **/
605 s32 ixgbe_update_eeprom_checksum_X540(struct ixgbe_hw *hw)
606 {
607         s32 status;
608         u16 checksum;

610         DEBUGFUNC("ixgbe_update_eeprom_checksum_X540");

612         /*
613          * Read the first word from the EEPROM. If this times out or fails, do
614          * not continue or we could be in for a very long wait while every
615          * EEPROM read fails
616          */
617         status = hw->eeprom.ops.read(hw, 0, &checksum);

619         if (status != IXGBE_SUCCESS)
620                 DEBUGOUT("EEPROM read failed\n");

622         if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
623             IXGBE_SUCCESS) {
624                 checksum = hw->eeprom.ops.calc_checksum(hw);

626                 /*
627                  * Do not use hw->eeprom.ops.write because we do not want to
628                  * take the synchronization semaphores twice here.
629                  */
630                 status = ixgbe_write_eewr_generic(hw, IXGBE_EEPROM_CHECKSUM,
631                                                   checksum);

633         if (status == IXGBE_SUCCESS)
634                 status = ixgbe_update_flash_X540(hw);
635         else
636                 status = IXGBE_ERR_SWFW_SYNC;
637         }

639         hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);

641         return status;
642 }

644 /**
645  *  ixgbe_update_flash_X540 - Instruct HW to copy EEPROM to Flash device
646  *  @hw: pointer to hardware structure
647  *
648  *  Set FLUP (bit 23) of the EEC register to instruct Hardware to copy
649  *  EEPROM from shadow RAM to the flash device.
650  **/
651 static s32 ixgbe_update_flash_X540(struct ixgbe_hw *hw)
652 {
653         u32 flup;
654         s32 status = IXGBE_ERR_EEPROM;
```

```
656         DEBUGFUNC("ixgbe_update_flash_X540");

658         status = ixgbe_poll_flash_update_done_X540(hw);
659         if (status == IXGBE_ERR_EEPROM) {
660                 DEBUGOUT("Flash update time out\n");
661                 goto out;
662         }

664         flup = IXGBE_READ_REG(hw, IXGBE_EEC) | IXGBE_EEC_FLUP;
665         IXGBE_WRITE_REG(hw, IXGBE_EEC, flup);

667         status = ixgbe_poll_flash_update_done_X540(hw);
668         if (status == IXGBE_SUCCESS)
669                 DEBUGOUT("Flash update complete\n");
670         else
671                 DEBUGOUT("Flash update time out\n");

673         if (hw->revision_id == 0) {
674                 flup = IXGBE_READ_REG(hw, IXGBE_EEC);

676                 if (flup & IXGBE_EEC_SEC1VAL) {
677                         flup |= IXGBE_EEC_FLUP;
678                         IXGBE_WRITE_REG(hw, IXGBE_EEC, flup);
679                 }

681                 status = ixgbe_poll_flash_update_done_X540(hw);
682                 if (status == IXGBE_SUCCESS)
683                         DEBUGOUT("Flash update complete\n");
684                 else
685                         DEBUGOUT("Flash update time out\n");
686         }
687 out:
688         return status;
689 }

691 /**
692  *  ixgbe_poll_flash_update_done_X540 - Poll flash update status
693  *  @hw: pointer to hardware structure
694  *
695  *  Polls the FLUDONE (bit 26) of the EEC Register to determine when the
696  *  flash update is done.
697  **/
698 static s32 ixgbe_poll_flash_update_done_X540(struct ixgbe_hw *hw)
699 {
700         u32 i;
701         u32 reg;
702         s32 status = IXGBE_ERR_EEPROM;

704         DEBUGFUNC("ixgbe_poll_flash_update_done_X540");

706         for (i = 0; i < IXGBE_FLUDONE_ATTEMPTS; i++) {
707                 reg = IXGBE_READ_REG(hw, IXGBE_EEC);
708                 if (reg & IXGBE_EEC_FLUDONE) {
709                         status = IXGBE_SUCCESS;
710                         break;
711                 }
712                 usec_delay(5);
713         }
714         return status;
715 }

717 /**
718  *  ixgbe_acquire_swfw_sync_X540 - Acquire SWFW semaphore
719  *  @hw: pointer to hardware structure
720  *  @mask: Mask to specify which semaphore to acquire
721  *
```



```
722  *  Acquires the SWFW semaphore thought the SW_FW_SYNC register for
723  *  the specified function (CSR, PHY0, PHY1, NVM, Flash)
724  **/
725 s32 ixgbe_acquire_swfw_sync_X540(struct ixgbe_hw *hw, u16 mask)
726 {
727         u32 swfw_sync;
728         u32 swmask = mask;
729         u32 fwmask = mask << 5;
730         u32 hwmask = 0;
731         u32 timeout = 200;
732         u32 i;
733         s32 ret_val = IXGBE_SUCCESS;

735         DEBUGFUNC("ixgbe_acquire_swfw_sync_X540");

737         if (swmask == IXGBE_GSSR_EEP_SM)
738                 hwmask = IXGBE_GSSR_FLASH_SM;

740         /* SW only mask doesn't have FW bit pair */
741         if (swmask == IXGBE_GSSR_SW_MNG_SM)
742                 fwmask = 0;

744         for (i = 0; i < timeout; i++) {
745                 /*
746                  * SW NVM semaphore bit is used for access to all
747                  * SW_FW_SYNC bits (not just NVM)
748                  */
749                 if (ixgbe_get_swfw_sync_semaphore(hw)) {
750                         ret_val = IXGBE_ERR_SWFW_SYNC;
751                         goto out;
752                 }

754                 swfw_sync = IXGBE_READ_REG(hw, IXGBE_SWFW_SYNC);
755                 if (!(swfw_sync & (fwmask | swmask | hwmask))) {
756                         swfw_sync |= swmask;
757                         IXGBE_WRITE_REG(hw, IXGBE_SWFW_SYNC, swfw_sync);
758                         ixgbe_release_swfw_sync_semaphore(hw);
759                         msec_delay(5);
760                         goto out;
761                 } else {
762                         /*
763                          * Firmware currently using resource (fwmask), hardware
764                          * currently using resource (hwmask), or other software
765                          * thread currently using resource (swmask)
766                          */
767                         ixgbe_release_swfw_sync_semaphore(hw);
768                         msec_delay(5);
769                 }
770         }

772         /* Failed to get SW only semaphore */
773         if (swmask == IXGBE_GSSR_SW_MNG_SM) {
774                 ret_val = IXGBE_ERR_SWFW_SYNC;
775                 goto out;
776         }

778         /* If the resource is not released by the FW/HW the SW can assume that
779          * the FW/HW malfunctions. In that case the SW should sets the SW bit(s)
780          * of the requested resource(s) while ignoring the corresponding FW/HW
781          * bits in the SW_FW_SYNC register.
782          */
783         swfw_sync = IXGBE_READ_REG(hw, IXGBE_SWFW_SYNC);
784         if (swfw_sync & (fwmask | hwmask)) {
785                 if (ixgbe_get_swfw_sync_semaphore(hw)) {
786                         ret_val = IXGBE_ERR_SWFW_SYNC;
787                         goto out;
```

```
788                 }
790                 swfw_sync |= swmask;
791                 IXGBE_WRITE_REG(hw, IXGBE_SWFW_SYNC, swfw_sync);
792                 ixgbe_release_swfw_sync_semaphore(hw);
793                 msec_delay(5);
794         }
796 out:
797         return ret_val;
798 }
800 /**
801  *  ixgbe_release_swfw_sync_X540 - Release SWFW semaphore
802  *  @hw: pointer to hardware structure
803  *  @mask: Mask to specify which semaphore to release
804  *
805  *  Releases the SWFW semaphore through the SW_FW_SYNC register
806  *  for the specified function (CSR, PHY0, PHY1, EVM, Flash)
807  **/
808 void ixgbe_release_swfw_sync_X540(struct ixgbe_hw *hw, u16 mask)
809 {
810         u32 swfw_sync;
811         u32 swmask = mask;
813         DEBUGFUNC("ixgbe_release_swfw_sync_X540");
815         ixgbe_get_swfw_sync_semaphore(hw);
817         swfw_sync = IXGBE_READ_REG(hw, IXGBE_SWFW_SYNC);
818         swfw_sync &= ~swmask;
819         IXGBE_WRITE_REG(hw, IXGBE_SWFW_SYNC, swfw_sync);
821         ixgbe_release_swfw_sync_semaphore(hw);
822         msec_delay(5);
823 }
825 /**
826  *  ixgbe_get_nvm_semaphore - Get hardware semaphore
827  *  @hw: pointer to hardware structure
828  *
829  *  Sets the hardware semaphores so SW/FW can gain control of shared resources
830  **/
831 static s32 ixgbe_get_swfw_sync_semaphore(struct ixgbe_hw *hw)
832 {
833         s32 status = IXGBE_ERR_EEPROM;
834         u32 timeout = 2000;
835         u32 i;
836         u32 swsm;
838         DEBUGFUNC("ixgbe_get_swfw_sync_semaphore");
840         /* Get SMBI software semaphore between device drivers first */
841         for (i = 0; i < timeout; i++) {
842                 /*
843                  * If the SMBI bit is 0 when we read it, then the bit will be
844                  * set and we have the semaphore
845                  */
846                 swsm = IXGBE_READ_REG(hw, IXGBE_SWSM);
847                 if (!(swsm & IXGBE_SWSM_SMBI)) {
848                         status = IXGBE_SUCCESS;
849                         break;
850                 }
851                 usec_delay(50);
852         }
```



```
854         /* Now get the semaphore between SW/FW through the REGSMP bit */
855         if (status == IXGBE_SUCCESS) {
856                 for (i = 0; i < timeout; i++) {
857                         swsm = IXGBE_READ_REG(hw, IXGBE_SWFW_SYNC);
858                         if (!(swsm & IXGBE_SWFW_REGSMP))
859                                 break;
861                         usec_delay(50);
862                 }
864                 /*
865                  * Release semaphores and return error if SW NVM semaphore
866                  * was not granted because we don't have access to the EEPROM
867                  */
868                 if (i >= timeout) {
869                         DEBUGOUT("REGSMP Software NVM semaphore not "
870                                  "granted.\n");
871                         ixgbe_release_swfw_sync_semaphore(hw);
872                         status = IXGBE_ERR_EEPROM;
873                 }
874         } else {
875                 DEBUGOUT("Software semaphore SMBI between device drivers "
876                          "not granted.\n");
877         }
879         return status;
880 }
882 /**
883  *  ixgbe_release_nvm_semaphore - Release hardware semaphore
884  *  @hw: pointer to hardware structure
885  *
886  *  This function clears hardware semaphore bits.
887  **/
888 static void ixgbe_release_swfw_sync_semaphore(struct ixgbe_hw *hw)
889 {
890         u32 swsm;
892         DEBUGFUNC("ixgbe_release_swfw_sync_semaphore");
894         /* Release both semaphores by writing 0 to the bits REGSMP and SMBI */
896         swsm = IXGBE_READ_REG(hw, IXGBE_SWSM);
897         swsm &= ~IXGBE_SWSM_SMBI;
898         IXGBE_WRITE_REG(hw, IXGBE_SWSM, swsm);
900         swsm = IXGBE_READ_REG(hw, IXGBE_SWFW_SYNC);
901         swsm &= ~IXGBE_SWFW_REGSMP;
902         IXGBE_WRITE_REG(hw, IXGBE_SWFW_SYNC, swsm);
904         IXGBE_WRITE_FLUSH(hw);
905 }
907 /**
908  * ixgbe_blink_led_start_X540 - Blink LED based on index.
909  * @hw: pointer to hardware structure
910  * @index: led number to blink
911  *
912  * Devices that implement the version 2 interface:
913  *   X540
914  **/
915 s32 ixgbe_blink_led_start_X540(struct ixgbe_hw *hw, u32 index)
916 {
917         u32 macc_reg;
918         u32 ledctl_reg;
919         ixgbe_link_speed speed;
```

```
920         bool link_up;
921
922         DEBUGFUNC("ixgbe_blink_led_start_X540");
923
924         /*
925          * Link should be up in order for the blink bit in the LED control
926          * register to work. Force link and speed in the MAC if link is down.
927          * This will be reversed when we stop the blinking.
928          */
929         hw->mac.ops.check_link(hw, &speed, &link_up, FALSE);
930         if (link_up == FALSE) {
931                 macc_reg = IXGBE_READ_REG(hw, IXGBE_MACC);
932                 macc_reg |= IXGBE_MACC_FLU | IXGBE_MACC_FSV_10G | IXGBE_MACC_FS;
933                 IXGBE_WRITE_REG(hw, IXGBE_MACC, macc_reg);
934         }
935         /* Set the LED to LINK_UP + BLINK. */
936         ledctl_reg = IXGBE_READ_REG(hw, IXGBE_LEDCTL);
937         ledctl_reg &= ~IXGBE_LED_MODE_MASK(index);
938         ledctl_reg |= IXGBE_LED_BLINK(index);
939         IXGBE_WRITE_REG(hw, IXGBE_LEDCTL, ledctl_reg);
940         IXGBE_WRITE_FLUSH(hw);
941
942         return IXGBE_SUCCESS;
943 }
944
945 /**
946  * ixgbe_blink_led_stop_X540 - Stop blinking LED based on index.
947  * @hw: pointer to hardware structure
948  * @index: led number to stop blinking
949  *
950  * Devices that implement the version 2 interface:
951  *   X540
952  **/
953 s32 ixgbe_blink_led_stop_X540(struct ixgbe_hw *hw, u32 index)
954 {
955         u32 macc_reg;
956         u32 ledctl_reg;
957
958         DEBUGFUNC("ixgbe_blink_led_stop_X540");
959
960         /* Restore the LED to its default value. */
961         ledctl_reg = IXGBE_READ_REG(hw, IXGBE_LEDCTL);
962         ledctl_reg &= ~IXGBE_LED_MODE_MASK(index);
963         ledctl_reg |= IXGBE_LED_LINK_ACTIVE << IXGBE_LED_MODE_SHIFT(index);
964         ledctl_reg &= ~IXGBE_LED_BLINK(index);
965         IXGBE_WRITE_REG(hw, IXGBE_LEDCTL, ledctl_reg);
966
967         /* Unforce link and speed in the MAC. */
968         macc_reg = IXGBE_READ_REG(hw, IXGBE_MACC);
969         macc_reg &= ~(IXGBE_MACC_FLU | IXGBE_MACC_FSV_10G | IXGBE_MACC_FS);
970         IXGBE_WRITE_REG(hw, IXGBE_MACC, macc_reg);
971         IXGBE_WRITE_FLUSH(hw);
972
973         return IXGBE_SUCCESS;
974 }
```

```
**********************************************************
    3172 Thu Jul 12 12:22:42 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_x540.h
XXXX Intel X540 support
**********************************************************
```

```
/******************************************************************************

  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

   3. Neither the name of the Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

******************************************************************************/
/*$FreeBSD$*/

#ifndef _IXGBE_X540_H_
#define _IXGBE_X540_H_

#include "ixgbe_type.h"

s32 ixgbe_get_link_capabilities_X540(struct ixgbe_hw *hw,
                                     ixgbe_link_speed *speed, bool *autoneg);
enum ixgbe_media_type ixgbe_get_media_type_X540(struct ixgbe_hw *hw);
s32 ixgbe_setup_mac_link_X540(struct ixgbe_hw *hw, ixgbe_link_speed speed,
                              bool autoneg, bool link_up_wait_to_complete);
s32 ixgbe_reset_hw_X540(struct ixgbe_hw *hw);
s32 ixgbe_start_hw_X540(struct ixgbe_hw *hw);
u32 ixgbe_get_supported_physical_layer_X540(struct ixgbe_hw *hw);

s32 ixgbe_init_eeprom_params_X540(struct ixgbe_hw *hw);
s32 ixgbe_read_eerd_X540(struct ixgbe_hw *hw, u16 offset, u16 *data);
s32 ixgbe_read_eerd_buffer_X540(struct ixgbe_hw *hw, u16 offset, u16 words,
                                u16 *data);
s32 ixgbe_write_eewr_X540(struct ixgbe_hw *hw, u16 offset, u16 data);
s32 ixgbe_write_eewr_buffer_X540(struct ixgbe_hw *hw, u16 offset, u16 words,
                                 u16 *data);
s32 ixgbe_update_eeprom_checksum_X540(struct ixgbe_hw *hw);
s32 ixgbe_validate_eeprom_checksum_X540(struct ixgbe_hw *hw, u16 *checksum_val);
u16 ixgbe_calc_eeprom_checksum_X540(struct ixgbe_hw *hw);

s32 ixgbe_acquire_swfw_sync_X540(struct ixgbe_hw *hw, u16 mask);
void ixgbe_release_swfw_sync_X540(struct ixgbe_hw *hw, u16 mask);
```



```
s32 ixgbe_blink_led_start_X540(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_blink_led_stop_X540(struct ixgbe_hw *hw, u32 index);
#endif /* _IXGBE_X540_H_ */
```