

new/exception_lists/cstyle

1

42294 Fri Jul 20 10:59:28 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/nsrtok.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
```

new/exception_lists/cstyle

2

```
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/afstring2key.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/encrypt_compare.c
83 usr/src/lib/gss_mechs/mech_krb5/crypto/encrypt_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_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_encrypt.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_encrypt.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
```

```

128 usr/src/lib/gss_mechs/mech_krb5/include/osconf.h
129 usr/src/lib/gss_mechs/mech_krb5/include/port-sockets.h
130 usr/src/lib/gss_mechs/mech_krb5/include/preauth_plugin.h
131 usr/src/lib/gss_mechs/mech_krb5/include/socket-utils.h
132 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.c
133 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.h
134 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.c
135 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.h
136 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.c
137 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.h
138 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_decode.c
139 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_decode.h
140 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_encode.c
141 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_encode.h
142 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.c
143 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.h
144 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.c
145 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.h
146 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1buf.h
147 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krb5_decode.c
148 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krb5_encode.c
149 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krbasn1.h
150 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/ldap_key_seq.c
151 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_file.c
152 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_memory.c
153 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_retr.c
154 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_int.h
155 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccbase.c
156 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cccopy.c
157 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccdefault.c
158 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccdefops.c
159 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccfns.c
160 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/fcc.h
161 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/scc.h
162 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ser_cc.c
163 usr/src/lib/gss_mechs/mech_krb5/krb5/error_tables/adm_err.h
164 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/file/ktfile.h
165 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt_file.c
166 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt_srvtab.c
167 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt_int.h
168 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktadd.c
169 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktbase.c
170 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktdefault.c
171 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktfns.c
172 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktfr_entry.c
173 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktremove.c
174 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/read_servi.c
175 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_comp.c
176 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_order.c
177 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_srch.c
178 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/appdefault.c
179 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/bld_pr_ext.c
180 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/bld_princ.c
181 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/chk_trans.c
182 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/cleanup.h
183 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/conv_princ.c
184 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_addrs.c
185 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_creds.c
186 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_data.c
187 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_tick.c
188 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/cp_key_cnt.c
189 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/decode_kdc.c
190 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/decrypt_tk.c
191 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/deltat.c
192 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/enc_helper.c
193 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/encode_kdc.c

```

```

194 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/encrypt_tk.c
195 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/free_rtree.c
196 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/fwd_tgt.c
197 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gc_frm_kdc.c
198 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gc_via_tkt.c
199 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gen_seqnum.c
200 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gen_subkey.c
201 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/get_creds.c
202 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/get_in_tkt.c
203 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_keytab.c
204 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_opt.c
205 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_pwd.c
206 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/init_keyblock.c
207 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/int-proto.h
208 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/kdc_rep_dc.c
209 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/kerfs.c
210 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_error.c
211 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_priv.c
212 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_rep.c
213 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_req_ext.c
214 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_req.c
215 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_safe.c
216 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/pac.c
217 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/pr_to_salt.c
218 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/preauth.c
219 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/preauth2.c
220 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/princ_comp.c
221 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_cred.c
222 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_error.c
223 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_priv.c
224 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_rep.c
225 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_req_dec.c
226 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_req.c
227 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_safe.c
228 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/recvauth.c
229 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/send_tgs.c
230 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/sendauth.c
231 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/set_realm.c
232 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/srv_rcache.c
233 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/str_conv.c
234 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/tgtname.c
235 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/valid_times.c
236 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/vic_opt.c
237 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/walk_rtree.c
238 usr/src/lib/gss_mechs/mech_krb5/krb5/os/accessor.c
239 usr/src/lib/gss_mechs/mech_krb5/krb5/os/an_to_in.c
240 usr/src/lib/gss_mechs/mech_krb5/krb5/os/ccdefine.c
241 usr/src/lib/gss_mechs/mech_krb5/krb5/os/changepw.c
242 usr/src/lib/gss_mechs/mech_krb5/krb5/os/def_realm.c
243 usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnsglue.c
244 usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnsglue.h
245 usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnssrv.c
246 usr/src/lib/gss_mechs/mech_krb5/krb5/os/foreachaddr.c
247 usr/src/lib/gss_mechs/mech_krb5/krb5/os/free_hstrl.c
248 usr/src/lib/gss_mechs/mech_krb5/krb5/os/free_krbhs.c
249 usr/src/lib/gss_mechs/mech_krb5/krb5/os/full_ipadr.c
250 usr/src/lib/gss_mechs/mech_krb5/krb5/os/gen_port.c
251 usr/src/lib/gss_mechs/mech_krb5/krb5/os/gen_rname.c
252 usr/src/lib/gss_mechs/mech_krb5/krb5/os/genaddrs.c
253 usr/src/lib/gss_mechs/mech_krb5/krb5/os/get_krbhst.c
254 usr/src/lib/gss_mechs/mech_krb5/krb5/os/gmt_mktime.c
255 usr/src/lib/gss_mechs/mech_krb5/krb5/os/hostaddr.c
256 usr/src/lib/gss_mechs/mech_krb5/krb5/os/hst_realm.c
257 usr/src/lib/gss_mechs/mech_krb5/krb5/os/ktdefine.c
258 usr/src/lib/gss_mechs/mech_krb5/krb5/os/kuserok.c
259 usr/src/lib/gss_mechs/mech_krb5/krb5/os/localaddr.c

```

new/exception_lists/cstyle

5

```
260 usr/src/lib/gss_mechs/mech_krb5/krb5/os/locate_kdc.c
261 usr/src/lib/gss_mechs/mech_krb5/krb5/os/lock_file.c
262 usr/src/lib/gss_mechs/mech_krb5/krb5/os/mk_faddr.c
263 usr/src/lib/gss_mechs/mech_krb5/krb5/os/net_read.c
264 usr/src/lib/gss_mechs/mech_krb5/krb5/os/net_write.c
265 usr/src/lib/gss_mechs/mech_krb5/krb5/os/os-proto.h
266 usr/src/lib/gss_mechs/mech_krb5/krb5/os/osconfig.c
267 usr/src/lib/gss_mechs/mech_krb5/krb5/os/port2ip.c
268 usr/src/lib/gss_mechs/mech_krb5/krb5/os/prompter.c
269 usr/src/lib/gss_mechs/mech_krb5/krb5/os/promptusr.c
270 usr/src/lib/gss_mechs/mech_krb5/krb5/os/read_msg.c
271 usr/src/lib/gss_mechs/mech_krb5/krb5/os/read_pwd.c
272 usr/src/lib/gss_mechs/mech_krb5/krb5/os/realmdom.c
273 usr/src/lib/gss_mechs/mech_krb5/krb5/os/realmd_iter.c
274 usr/src/lib/gss_mechs/mech_krb5/krb5/os/sendto_kdc.c
275 usr/src/lib/gss_mechs/mech_krb5/krb5/os/sn2princ.c
276 usr/src/lib/gss_mechs/mech_krb5/krb5/os/thread_safe.c
277 usr/src/lib/gss_mechs/mech_krb5/krb5/os/unlck_file.c
278 usr/src/lib/gss_mechs/mech_krb5/krb5/os/ustime.c
279 usr/src/lib/gss_mechs/mech_krb5/krb5/os/write_msg.c
280 usr/src/lib/gss_mechs/mech_krb5/krb5/posix/daemon.c
281 usr/src/lib/gss_mechs/mech_krb5/krb5/posix/setenv.c
282 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_base.h
283 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_conv.c
284 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_io.h
285 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_none.c
286 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc-int.h
287 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rcdef.c
288 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rcfns.c
289 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/ser_rc.c
290 usr/src/lib/gss_mechs/mech_krb5/mech/accept_sec_context.c
291 usr/src/lib/gss_mechs/mech_krb5/mech/acquire_cred_with_pw.c
292 usr/src/lib/gss_mechs/mech_krb5/mech/acquire_cred.c
293 usr/src/lib/gss_mechs/mech_krb5/mech/add_cred.c
294 usr/src/lib/gss_mechs/mech_krb5/mech/compare_name.c
295 usr/src/lib/gss_mechs/mech_krb5/mech/context_time.c
296 usr/src/lib/gss_mechs/mech_krb5/mech/copy_ccache.c
297 usr/src/lib/gss_mechs/mech_krb5/mech/disp_com_err_status.c
298 usr/src/lib/gss_mechs/mech_krb5/mech/disp_major_status.c
299 usr/src/lib/gss_mechs/mech_krb5/mech/disp_name.c
300 usr/src/lib/gss_mechs/mech_krb5/mech/disp_status.c
301 usr/src/lib/gss_mechs/mech_krb5/mech/export_name.c
302 usr/src/lib/gss_mechs/mech_krb5/mech/export_sec_context.c
303 usr/src/lib/gss_mechs/mech_krb5/mech/get_tkt_flags.c
304 usr/src/lib/gss_mechs/mech_krb5/mech/gss_libinit.h
305 usr/src/lib/gss_mechs/mech_krb5/mech/import_name.c
306 usr/src/lib/gss_mechs/mech_krb5/mech/indicate_mechs.c
307 usr/src/lib/gss_mechs/mech_krb5/mech/init_sec_context.c
308 usr/src/lib/gss_mechs/mech_krb5/mech/inq_context.c
309 usr/src/lib/gss_mechs/mech_krb5/mech/inq_cred.c
310 usr/src/lib/gss_mechs/mech_krb5/mech/inq_names.c
311 usr/src/lib/gss_mechs/mech_krb5/mech/krb5_gss_glue.c
312 usr/src/lib/gss_mechs/mech_krb5/mech/lucid_context.c
313 usr/src/lib/gss_mechs/mech_krb5/mech/oid_ops.c
314 usr/src/lib/gss_mechs/mech_krb5/mech/process_context_token.c
315 usr/src/lib/gss_mechs/mech_krb5/mech/rel_buffer.c
316 usr/src/lib/gss_mechs/mech_krb5/mech/rel_cred.c
317 usr/src/lib/gss_mechs/mech_krb5/mech/rel_name.c
318 usr/src/lib/gss_mechs/mech_krb5/mech/rel_oid_set.c
319 usr/src/lib/gss_mechs/mech_krb5/mech/rel_oid.c
320 usr/src/lib/gss_mechs/mech_krb5/mech/set_allowable_etypes.c
321 usr/src/lib/gss_mechs/mech_krb5/mech/set_ccache.c
322 usr/src/lib/gss_mechs/mech_krb5/mech/util_buffer_set.c
323 usr/src/lib/gss_mechs/mech_krb5/mech/util_buffer.c
324 usr/src/lib/gss_mechs/mech_krb5/mech/util_cksum.c
325 usr/src/lib/gss_mechs/mech_krb5/mech/util_ctxsetup.c
```

new/exception_lists/cstyle

6

```
326 usr/src/lib/gss_mechs/mech_krb5/mech/util_dup.c
327 usr/src/lib/gss_mechs/mech_krb5/mech/util_localhost.c
328 usr/src/lib/gss_mechs/mech_krb5/mech/util_nohash_validate.c
329 usr/src/lib/gss_mechs/mech_krb5/profile/prof_err.h
330 usr/src/lib/gss_mechs/mech_krb5/profile/prof_get.c
331 usr/src/lib/gss_mechs/mech_krb5/profile/prof_set.c
332 usr/src/lib/gss_mechs/mech_krb5/support/errors.c
333 usr/src/lib/gss_mechs/mech_krb5/support/fake-addrinfo.c
334 usr/src/lib/gss_mechs/mech_krb5/support/init-addrinfo.c
335 usr/src/lib/gss_mechs/mech_krb5/support/plugins.c
336 usr/src/lib/gss_mechs/mech_krb5/support/supp-int.h
337 usr/src/lib/gss_mechs/mech_krb5/support/threads.c
338 usr/src/lib/gss_mechs/mech_krb5/support/utf8_conv.c
339 usr/src/lib/gss_mechs/mech_krb5/support/utf8.c
340 usr/src/lib/krb5/dyn/dyn_append.c
341 usr/src/lib/krb5/dyn/dyn_create.c
342 usr/src/lib/krb5/dyn/dyn_debug.c
343 usr/src/lib/krb5/dyn/dyn_delete.c
344 usr/src/lib/krb5/dyn/dyn_initzero.c
345 usr/src/lib/krb5/dyn/dyn_insert.c
346 usr/src/lib/krb5/dyn/dyn_paranoid.c
347 usr/src/lib/krb5/dyn/dyn_put.c
348 usr/src/lib/krb5/dyn/dyn_realloc.c
349 usr/src/lib/krb5/dyn/dyn_size.c
350 usr/src/lib/krb5/kadm5/admin_internal.h
351 usr/src/lib/krb5/kadm5/admin_xdr.h
352 usr/src/lib/krb5/kadm5/admin.h
353 usr/src/lib/krb5/kadm5/alt_prof.c
354 usr/src/lib/krb5/kadm5/chpass_util_strings.h
355 usr/src/lib/krb5/kadm5/chpass_util.c
356 usr/src/lib/krb5/kadm5/clnt/changepw.c
357 usr/src/lib/krb5/kadm5/clnt/client_handle.c
358 usr/src/lib/krb5/kadm5/clnt/client_init.c
359 usr/src/lib/krb5/kadm5/clnt/client_internal.h
360 usr/src/lib/krb5/kadm5/clnt/client_principal.c
361 usr/src/lib/krb5/kadm5/clnt/client_rpc.c
362 usr/src/lib/krb5/kadm5/clnt/clnt_chpass_util.c
363 usr/src/lib/krb5/kadm5/clnt/clnt_policy.c
364 usr/src/lib/krb5/kadm5/clnt/clnt_privs.c
365 usr/src/lib/krb5/kadm5/clnt/logger.c
366 usr/src/lib/krb5/kadm5/kadm_err.h
367 usr/src/lib/krb5/kadm5/kadm_rpc_xdr.c
368 usr/src/lib/krb5/kadm5/kadm_rpc.h
369 usr/src/lib/krb5/kadm5/misc_free.c
370 usr/src/lib/krb5/kadm5/server_internal.h
371 usr/src/lib/krb5/kadm5/srv/adb_xdr.c
372 usr/src/lib/krb5/kadm5/srv/chgpwd.c
373 usr/src/lib/krb5/kadm5/srv/logger.c
374 usr/src/lib/krb5/kadm5/srv/server_acl.c
375 usr/src/lib/krb5/kadm5/srv/server_acl.h
376 usr/src/lib/krb5/kadm5/srv/server_dict.c
377 usr/src/lib/krb5/kadm5/srv/server_handle.c
378 usr/src/lib/krb5/kadm5/srv/server_init.c
379 usr/src/lib/krb5/kadm5/srv/server_kdb.c
380 usr/src/lib/krb5/kadm5/srv/server_misc.c
381 usr/src/lib/krb5/kadm5/srv/svr_chpass_util.c
382 usr/src/lib/krb5/kadm5/srv/svr_iters.c
383 usr/src/lib/krb5/kadm5/srv/svr_misc_free.c
384 usr/src/lib/krb5/kadm5/srv/svr_policy.c
385 usr/src/lib/krb5/kadm5/srv/svr_principal.c
386 usr/src/lib/krb5/kadm5/srv/xdr_alloc.c
387 usr/src/lib/krb5/kadm5/str_conv.c
388 usr/src/lib/krb5/kdb/adb_err.h
389 usr/src/lib/krb5/kdb/decrypt_key.c
390 usr/src/lib/krb5/kdb/encrypt_key.c
391 usr/src/lib/krb5/kdb/kdb_cpw.c
```

new/exception_lists/cstyle

7

```
392 usr/src/lib/krb5/kdb/kdb_default.c
393 usr/src/lib/krb5/kdb/kdb5.c
394 usr/src/lib/krb5/kdb/kdb5.h
395 usr/src/lib/krb5/kdb/keytab.c
396 usr/src/lib/krb5/plugins/kdb/db2/adb_openclose.c
397 usr/src/lib/krb5/plugins/kdb/db2/adb_policy.c
398 usr/src/lib/krb5/plugins/kdb/db2/db2_exp.c
399 usr/src/lib/krb5/plugins/kdb/db2/kdb_compat.h
400 usr/src/lib/krb5/plugins/kdb/db2/kdb_db2.c
401 usr/src/lib/krb5/plugins/kdb/db2/kdb_db2.h
402 usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.c
403 usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.h
404 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_close.c
405 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_conv.c
406 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_debug.c
407 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_delete.c
408 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_get.c
409 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_open.c
410 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_overflow.c
411 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_page.c
412 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_put.c
413 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_search.c
414 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_seq.c
415 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_split.c
416 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_utils.c
417 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/btree.h
418 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/extern.h
419 usr/src/lib/krb5/plugins/kdb/db2/libdb2/db/db.c
420 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/dbm.c
421 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/extern.h
422 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_bigkey.c
423 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_func.c
424 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_log2.c
425 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_page.c
426 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash.c
427 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash.h
428 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hsearch.c
429 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/page.h
430 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/search.h
431 usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-int.h
432 usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-ndbm.h
433 usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-queue.h
434 usr/src/lib/krb5/plugins/kdb/db2/libdb2/mpool/mpool.c
435 usr/src/lib/krb5/plugins/kdb/db2/libdb2/mpool/mpool.h
436 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/extern.h
437 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_close.c
438 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_delete.c
439 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_get.c
440 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_open.c
441 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_put.c
442 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_search.c
443 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_seq.c
444 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_utils.c
445 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/recno.h
446 usr/src/lib/krb5/plugins/kdb/db2/pol_xdr.c
447 usr/src/lib/krb5/plugins/kdb/db2/policy_db.h
448 usr/src/lib/krb5/plugins/kdb/ldap/ldap_exp.c
449 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap_conn.c
450 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap.c
451 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap.h
452 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.c
453 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.h
454 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_create.c
455 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.c
456 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.h
457 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_fetch_mkey.c
```

new/exception_lists/cstyle

8

```
458 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.c
459 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.h
460 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.c
461 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.h
462 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_main.h
463 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_misc.c
464 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_misc.h
465 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal.c
466 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal.h
467 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal2.c
468 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_pwd_policy.c
469 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_pwd_policy.h
470 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_realm.c
471 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_realm.h
472 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_rights.c
473 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_stash.c
474 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_stash.h
475 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.c
476 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.h
477 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.c
478 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.h
479 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.c
480 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.h
481 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_clnt.c
482 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto_openssl.c
483 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto_openssl.h
484 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto.h
485 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_identity.c
486 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_lib.c
487 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_matching.c
488 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_profile.c
489 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_srv.c
490 usr/src/lib/krb5/plugins/preauth/pkinit/pkinit.h
491 usr/src/lib/krb5/ss/copyright.h
492 usr/src/lib/krb5/ss/data.c
493 usr/src/lib/krb5/ss/error.c
494 usr/src/lib/krb5/ss/execute_cmd.c
495 usr/src/lib/krb5/ss/help.c
496 usr/src/lib/krb5/ss/invocation.c
497 usr/src/lib/krb5/ss/list_rqs.c
498 usr/src/lib/krb5/ss/listen.c
499 usr/src/lib/krb5/ss/mit-sipb-copyright.h
500 usr/src/lib/krb5/ss/mk_cmds.c
501 usr/src/lib/krb5/ss/options.c
502 usr/src/lib/krb5/ss/pager.c
503 usr/src/lib/krb5/ss/parse.c
504 usr/src/lib/krb5/ss/prompt.c
505 usr/src/lib/krb5/ss/request_tbl.c
506 usr/src/lib/krb5/ss/requests.c
507 usr/src/lib/krb5/ss/ss_internal.h
508 usr/src/lib/krb5/ss/ss.h
509 usr/src/lib/krb5/ss/std_rqs.c
510 usr/src/lib/krb5/ss/utls.c
511 usr/src/lib/libgss/g_glue.c
512 usr/src/lib/libkrb5/common/base.h
513 usr/src/lib/libkrb5/common/choose.h
514 usr/src/lib/libkrb5/common/edge.c
515 usr/src/lib/libkrb5/common/edge.h
516 usr/src/lib/libkrb5/common/migrate.c
517 usr/src/lib/libkrb5/common/migrate.h
518 usr/src/lib/libkrb5/common/p2p.c
519 usr/src/lib/libkrb5/common/p2p.h
520 usr/src/lib/libkrb5/common/pcost.c
521 usr/src/lib/libkrb5/common/pcost.h
522 usr/src/lib/libkrb5/common/port.c
523 usr/src/lib/libkrb5/common/port.h
```

```

524 usr/src/lib/librstp/common/portinfo.c
525 usr/src/lib/librstp/common/portinfo.h
526 usr/src/lib/librstp/common/rolesel.c
527 usr/src/lib/librstp/common/rolesel.h
528 usr/src/lib/librstp/common/roletxns.c
529 usr/src/lib/librstp/common/roletxns.h
530 usr/src/lib/librstp/common/statmch.c
531 usr/src/lib/librstp/common/statmch.h
532 usr/src/lib/librstp/common/stp_bpdu.h
533 usr/src/lib/librstp/common/stp_in.c
534 usr/src/lib/librstp/common/stp_in.h
535 usr/src/lib/librstp/common/stp_to.h
536 usr/src/lib/librstp/common/stp_vectors.h
537 usr/src/lib/librstp/common/stpm.c
538 usr/src/lib/librstp/common/stpm.h
539 usr/src/lib/librstp/common/stpmgmt.c
540 usr/src/lib/librstp/common/sttrans.c
541 usr/src/lib/librstp/common/sttrans.h
542 usr/src/lib/librstp/common/times.c
543 usr/src/lib/librstp/common/times.h
544 usr/src/lib/librstp/common/topoch.c
545 usr/src/lib/librstp/common/topoch.h
546 usr/src/lib/librstp/common/transmit.c
547 usr/src/lib/librstp/common/transmit.h
548 usr/src/lib/librstp/common/uid_stp.h
549 usr/src/lib/librstp/common/vector.c
550 usr/src/lib/librstp/common/vector.h
551 usr/src/uts/common/gssapi/gssapi.h
552 usr/src/uts/common/gssapi/mechs/krb5/crypto/block_size.c
553 usr/src/uts/common/gssapi/mechs/krb5/crypto/checksum_length.c
554 usr/src/uts/common/gssapi/mechs/krb5/crypto/cksumtypes.c
555 usr/src/uts/common/gssapi/mechs/krb5/crypto/combine_keys.c
556 usr/src/uts/common/gssapi/mechs/krb5/crypto/crc32/crc32.c
557 usr/src/uts/common/gssapi/mechs/krb5/crypto/decrypt.c
558 usr/src/uts/common/gssapi/mechs/krb5/crypto/default_state.c
559 usr/src/uts/common/gssapi/mechs/krb5/crypto/des/d3_cbc.c
560 usr/src/uts/common/gssapi/mechs/krb5/crypto/des/f_cbc.c
561 usr/src/uts/common/gssapi/mechs/krb5/crypto/des/f_parity.c
562 usr/src/uts/common/gssapi/mechs/krb5/crypto/des/weak_key.c
563 usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/checksum.c
564 usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/derive.c
565 usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/dk_decrypt.c
566 usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/dk_encrypt.c
567 usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/arcfour_provider.c
568 usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/des.c
569 usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/des3.c
570 usr/src/uts/common/gssapi/mechs/krb5/crypto/encrypt_length.c
571 usr/src/uts/common/gssapi/mechs/krb5/crypto/encrypt.c
572 usr/src/uts/common/gssapi/mechs/krb5/crypto/etypes.c
573 usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_crc32.c
574 usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_kmd5.c
575 usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_ksha1.c
576 usr/src/uts/common/gssapi/mechs/krb5/crypto/hmac.c
577 usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/descbc.c
578 usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/k_hmac_md5.c
579 usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/k5_kmd5des.c
580 usr/src/uts/common/gssapi/mechs/krb5/crypto/make_checksum.c
581 usr/src/uts/common/gssapi/mechs/krb5/crypto/mandatory_sumtype.c
582 usr/src/uts/common/gssapi/mechs/krb5/crypto/nfold.c
583 usr/src/uts/common/gssapi/mechs/krb5/crypto/old/old_decrypt.c
584 usr/src/uts/common/gssapi/mechs/krb5/crypto/old/old_encrypt.c
585 usr/src/uts/common/gssapi/mechs/krb5/crypto/prng.c
586 usr/src/uts/common/gssapi/mechs/krb5/crypto/raw/raw_decrypt.c
587 usr/src/uts/common/gssapi/mechs/krb5/crypto/raw/raw_encrypt.c
588 usr/src/uts/common/gssapi/mechs/krb5/crypto/verify_checksum.c
589 usr/src/uts/common/gssapi/mechs/krb5/include/aes_s2k.h

```

```

590 usr/src/uts/common/gssapi/mechs/krb5/include/auth_con.h
591 usr/src/uts/common/gssapi/mechs/krb5/include/cksumtypes.h
592 usr/src/uts/common/gssapi/mechs/krb5/include/crc-32.h
593 usr/src/uts/common/gssapi/mechs/krb5/include/des_int.h
594 usr/src/uts/common/gssapi/mechs/krb5/include/dk.h
595 usr/src/uts/common/gssapi/mechs/krb5/include/enc_provider.h
596 usr/src/uts/common/gssapi/mechs/krb5/include/etypes.h
597 usr/src/uts/common/gssapi/mechs/krb5/include/gssapi_generic.h
598 usr/src/uts/common/gssapi/mechs/krb5/include/gssapi_krb5.h
599 usr/src/uts/common/gssapi/mechs/krb5/include/gssapiP_generic.h
600 usr/src/uts/common/gssapi/mechs/krb5/include/gssapiP_krb5.h
601 usr/src/uts/common/gssapi/mechs/krb5/include/hash_provider.h
602 usr/src/uts/common/gssapi/mechs/krb5/include/k5-int.h
603 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_16.h
604 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_32.h
605 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_64.h
606 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_16.h
607 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_32.h
608 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_64.h
609 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform.h
610 usr/src/uts/common/gssapi/mechs/krb5/include/k5-thread.h
611 usr/src/uts/common/gssapi/mechs/krb5/include/keyhash_provider.h
612 usr/src/uts/common/gssapi/mechs/krb5/include/krb5.h
613 usr/src/uts/common/gssapi/mechs/krb5/include/old.h
614 usr/src/uts/common/gssapi/mechs/krb5/include/raw.h
615 usr/src/uts/common/gssapi/mechs/krb5/include/rsa-md4.h
616 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/copy_athctr.c
617 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/copy_auth.c
618 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/copy_cksum.c
619 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/copy_key.c
620 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/copy_princ.c
621 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/init_ctx.c
622 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/kfFree.c
623 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/parse.c
624 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/ser_actx.c
625 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/ser_adata.c
626 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/ser_addr.c
627 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/ser_auth.c
628 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/ser_cksum.c
629 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/ser_ctx.c
630 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/ser_key.c
631 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/ser_princ.c
632 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/serialize.c
633 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb5/unparse.c
634 usr/src/uts/common/gssapi/mechs/krb5/krb5/os/c_ustime.c
635 usr/src/uts/common/gssapi/mechs/krb5/krb5/os/init_os_ctx.c
636 usr/src/uts/common/gssapi/mechs/krb5/krb5/os/timeofday.c
637 usr/src/uts/common/gssapi/mechs/krb5/krb5/os/toffset.c
638 usr/src/uts/common/gssapi/mechs/krb5/mech/delete_sec_context.c
639 usr/src/uts/common/gssapi/mechs/krb5/mech/gssapi_krb5.c
640 usr/src/uts/common/gssapi/mechs/krb5/mech/import_sec_context.c
641 usr/src/uts/common/gssapi/mechs/krb5/mech/k5seal.c
642 usr/src/uts/common/gssapi/mechs/krb5/mech/k5sealv3.c
643 usr/src/uts/common/gssapi/mechs/krb5/mech/k5unseal.c
644 usr/src/uts/common/gssapi/mechs/krb5/mech/seal.c
645 usr/src/uts/common/gssapi/mechs/krb5/mech/ser_sctx.c
646 usr/src/uts/common/gssapi/mechs/krb5/mech/sign.c
647 usr/src/uts/common/gssapi/mechs/krb5/mech/unseal.c
648 usr/src/uts/common/gssapi/mechs/krb5/mech/util_crypt.c
649 usr/src/uts/common/gssapi/mechs/krb5/mech/util_ordering.c
650 usr/src/uts/common/gssapi/mechs/krb5/mech/util_seed.c
651 usr/src/uts/common/gssapi/mechs/krb5/mech/util_seqnum.c
652 usr/src/uts/common/gssapi/mechs/krb5/mech/util_set.c
653 usr/src/uts/common/gssapi/mechs/krb5/mech/util_token.c
654 usr/src/uts/common/gssapi/mechs/krb5/mech/util_validate.c
655 usr/src/uts/common/gssapi/mechs/krb5/mech/val_cred.c

```

```

656 usr/src/uts/common/gssapi/mechs/krb5/mech/verify.c
657 usr/src/uts/common/gssapi/mechs/krb5/mech/wrap_size_limit.c
658 usr/src/uts/common/io/ixgbe/ixgbe_82598.c
659 usr/src/uts/common/io/ixgbe/ixgbe_82598.h
660 usr/src/uts/common/io/ixgbe/ixgbe_82599.c
661 usr/src/uts/common/io/ixgbe/ixgbe_api.c
662 usr/src/uts/common/io/ixgbe/ixgbe_api.h
663 usr/src/uts/common/io/ixgbe/ixgbe_common.c
664 usr/src/uts/common/io/ixgbe/ixgbe_common.h
665 usr/src/uts/common/io/ixgbe/ixgbe_mbx.c
666 usr/src/uts/common/io/ixgbe/ixgbe_mbx.h
667 usr/src/uts/common/io/ixgbe/ixgbe_osdep.h
668 usr/src/uts/common/io/ixgbe/ixgbe_phy.c
669 usr/src/uts/common/io/ixgbe/ixgbe_phy.h
670 usr/src/uts/common/io/ixgbe/ixgbe_type.h
671 usr/src/uts/common/io/ixgbe/ixgbe_x540.c
672 usr/src/uts/common/io/ixgbe/ixgbe_x540.h
673 usr/src/uts/intel/io/acpica/debugger/dbcmds.c
674 usr/src/uts/intel/io/acpica/debugger/dbdisply.c
675 usr/src/uts/intel/io/acpica/debugger/dbexec.c
676 usr/src/uts/intel/io/acpica/debugger/dbfileio.c
677 usr/src/uts/intel/io/acpica/debugger/dbhistry.c
678 usr/src/uts/intel/io/acpica/debugger/dbinput.c
679 usr/src/uts/intel/io/acpica/debugger/dbmethod.c
680 usr/src/uts/intel/io/acpica/debugger/dbnames.c
681 usr/src/uts/intel/io/acpica/debugger/dbstats.c
682 usr/src/uts/intel/io/acpica/debugger/dbutils.c
683 usr/src/uts/intel/io/acpica/debugger/dbxface.c
684 usr/src/uts/intel/io/acpica/disassembler/dmbuffer.c
685 usr/src/uts/intel/io/acpica/disassembler/dmnames.c
686 usr/src/uts/intel/io/acpica/disassembler/dmobject.c
687 usr/src/uts/intel/io/acpica/disassembler/dmopcode.c
688 usr/src/uts/intel/io/acpica/disassembler/dmresrc.c
689 usr/src/uts/intel/io/acpica/disassembler/dmresrc1.c
690 usr/src/uts/intel/io/acpica/disassembler/dmresrc2.c
691 usr/src/uts/intel/io/acpica/disassembler/dmutils.c
692 usr/src/uts/intel/io/acpica/disassembler/dmwalk.c
693 usr/src/uts/intel/io/acpica/dispatcher/dsargs.c
694 usr/src/uts/intel/io/acpica/dispatcher/dscontrol.c
695 usr/src/uts/intel/io/acpica/dispatcher/dsfield.c
696 usr/src/uts/intel/io/acpica/dispatcher/dsinit.c
697 usr/src/uts/intel/io/acpica/dispatcher/dsmethod.c
698 usr/src/uts/intel/io/acpica/dispatcher/dsmthdat.c
699 usr/src/uts/intel/io/acpica/dispatcher/dsobject.c
700 usr/src/uts/intel/io/acpica/dispatcher/dsopcode.c
701 usr/src/uts/intel/io/acpica/dispatcher/dsutils.c
702 usr/src/uts/intel/io/acpica/dispatcher/dswexec.c
703 usr/src/uts/intel/io/acpica/dispatcher/dswload.c
704 usr/src/uts/intel/io/acpica/dispatcher/dswload2.c
705 usr/src/uts/intel/io/acpica/dispatcher/dswscope.c
706 usr/src/uts/intel/io/acpica/dispatcher/dswstate.c
707 usr/src/uts/intel/io/acpica/events/evevent.c
708 usr/src/uts/intel/io/acpica/events/evglock.c
709 usr/src/uts/intel/io/acpica/events/evgpe.c
710 usr/src/uts/intel/io/acpica/events/evgpeblk.c
711 usr/src/uts/intel/io/acpica/events/evgpeinit.c
712 usr/src/uts/intel/io/acpica/events/evgpeutil.c
713 usr/src/uts/intel/io/acpica/events/evmisc.c
714 usr/src/uts/intel/io/acpica/events/evregion.c
715 usr/src/uts/intel/io/acpica/events/evrgnini.c
716 usr/src/uts/intel/io/acpica/events/evsci.c
717 usr/src/uts/intel/io/acpica/events/evxface.c
718 usr/src/uts/intel/io/acpica/events/evxfevent.c
719 usr/src/uts/intel/io/acpica/events/evxfpe.c
720 usr/src/uts/intel/io/acpica/events/evxfregn.c
721 usr/src/uts/intel/io/acpica/executor/exconfig.c

```

```

722 usr/src/uts/intel/io/acpica/executor/exconvrt.c
723 usr/src/uts/intel/io/acpica/executor/excreate.c
724 usr/src/uts/intel/io/acpica/executor/exdebug.c
725 usr/src/uts/intel/io/acpica/executor/exdump.c
726 usr/src/uts/intel/io/acpica/executor/exfield.c
727 usr/src/uts/intel/io/acpica/executor/exfldio.c
728 usr/src/uts/intel/io/acpica/executor/exmisc.c
729 usr/src/uts/intel/io/acpica/executor/exmutex.c
730 usr/src/uts/intel/io/acpica/executor/exnames.c
731 usr/src/uts/intel/io/acpica/executor/exoparg1.c
732 usr/src/uts/intel/io/acpica/executor/exoparg2.c
733 usr/src/uts/intel/io/acpica/executor/exoparg3.c
734 usr/src/uts/intel/io/acpica/executor/exoparg6.c
735 usr/src/uts/intel/io/acpica/executor/exprep.c
736 usr/src/uts/intel/io/acpica/executor/exregion.c
737 usr/src/uts/intel/io/acpica/executor/exresnte.c
738 usr/src/uts/intel/io/acpica/executor/exresolv.c
739 usr/src/uts/intel/io/acpica/executor/exresop.c
740 usr/src/uts/intel/io/acpica/executor/exstore.c
741 usr/src/uts/intel/io/acpica/executor/exstoren.c
742 usr/src/uts/intel/io/acpica/executor/exstorob.c
743 usr/src/uts/intel/io/acpica/executor/exsystem.c
744 usr/src/uts/intel/io/acpica/executor/exutils.c
745 usr/src/uts/intel/io/acpica/hardware/hwacpi.c
746 usr/src/uts/intel/io/acpica/hardware/hwgpe.c
747 usr/src/uts/intel/io/acpica/hardware/hwpci.c
748 usr/src/uts/intel/io/acpica/hardware/hwregs.c
749 usr/src/uts/intel/io/acpica/hardware/hwsleep.c
750 usr/src/uts/intel/io/acpica/hardware/hwtimer.c
751 usr/src/uts/intel/io/acpica/hardware/hwvalid.c
752 usr/src/uts/intel/io/acpica/hardware/hwxface.c
753 usr/src/uts/intel/io/acpica/namespace/nsaccess.c
754 usr/src/uts/intel/io/acpica/namespace/nsalloc.c
755 usr/src/uts/intel/io/acpica/namespace/nsdump.c
756 usr/src/uts/intel/io/acpica/namespace/nsdumpdv.c
757 usr/src/uts/intel/io/acpica/namespace/nseval.c
758 usr/src/uts/intel/io/acpica/namespace/nsinit.c
759 usr/src/uts/intel/io/acpica/namespace/nsload.c
760 usr/src/uts/intel/io/acpica/namespace/nsnames.c
761 usr/src/uts/intel/io/acpica/namespace/nsobject.c
762 usr/src/uts/intel/io/acpica/namespace/nsparse.c
763 usr/src/uts/intel/io/acpica/namespace/nspredef.c
764 usr/src/uts/intel/io/acpica/namespace/nsrepair.c
765 usr/src/uts/intel/io/acpica/namespace/nsrepair2.c
766 usr/src/uts/intel/io/acpica/namespace/nssearch.c
767 usr/src/uts/intel/io/acpica/namespace/nsutils.c
768 usr/src/uts/intel/io/acpica/namespace/nswalk.c
769 usr/src/uts/intel/io/acpica/namespace/nsxfeval.c
770 usr/src/uts/intel/io/acpica/namespace/nsxfname.c
771 usr/src/uts/intel/io/acpica/namespace/nsxfobj.c
772 usr/src/uts/intel/io/acpica/parser/psargs.c
773 usr/src/uts/intel/io/acpica/parser/psloop.c
774 usr/src/uts/intel/io/acpica/parser/psopcode.c
775 usr/src/uts/intel/io/acpica/parser/psparse.c
776 usr/src/uts/intel/io/acpica/parser/psscope.c
777 usr/src/uts/intel/io/acpica/parser/psree.c
778 usr/src/uts/intel/io/acpica/parser/psutils.c
779 usr/src/uts/intel/io/acpica/parser/pswalk.c
780 usr/src/uts/intel/io/acpica/parser/psxface.c
781 usr/src/uts/intel/io/acpica/resources/rsaddr.c
782 usr/src/uts/intel/io/acpica/resources/rscale.c
783 usr/src/uts/intel/io/acpica/resources/rscreate.c
784 usr/src/uts/intel/io/acpica/resources/rsdump.c
785 usr/src/uts/intel/io/acpica/resources/rsinfo.c
786 usr/src/uts/intel/io/acpica/resources/rsio.c
787 usr/src/uts/intel/io/acpica/resources/rsirq.c

```

```

788 usr/src/uts/intel/io/acpica/resources/rslist.c
789 usr/src/uts/intel/io/acpica/resources/rsmemory.c
790 usr/src/uts/intel/io/acpica/resources/rsmisc.c
791 usr/src/uts/intel/io/acpica/resources/rsutils.c
792 usr/src/uts/intel/io/acpica/resources/rsxface.c
793 usr/src/uts/intel/io/acpica/tables/tbfadt.c
794 usr/src/uts/intel/io/acpica/tables/tbfind.c
795 usr/src/uts/intel/io/acpica/tables/tbinstal.c
796 usr/src/uts/intel/io/acpica/tables/tbutils.c
797 usr/src/uts/intel/io/acpica/tables/tbxface.c
798 usr/src/uts/intel/io/acpica/tables/tbxfrout.c
799 usr/src/uts/intel/io/acpica/utilities/utalloc.c
800 usr/src/uts/intel/io/acpica/utilities/utcache.c
801 usr/src/uts/intel/io/acpica/utilities/utclib.c
802 usr/src/uts/intel/io/acpica/utilities/utcopy.c
803 usr/src/uts/intel/io/acpica/utilities/utdebug.c
804 usr/src/uts/intel/io/acpica/utilities/utdecode.c
805 usr/src/uts/intel/io/acpica/utilities/utdelete.c
806 usr/src/uts/intel/io/acpica/utilities/uteval.c
807 usr/src/uts/intel/io/acpica/utilities/utglobal.c
808 usr/src/uts/intel/io/acpica/utilities/utids.c
809 usr/src/uts/intel/io/acpica/utilities/utinit.c
810 usr/src/uts/intel/io/acpica/utilities/utlock.c
811 usr/src/uts/intel/io/acpica/utilities/utmath.c
812 usr/src/uts/intel/io/acpica/utilities/utmisc.c
813 usr/src/uts/intel/io/acpica/utilities/utmutex.c
814 usr/src/uts/intel/io/acpica/utilities/utobject.c
815 usr/src/uts/intel/io/acpica/utilities/utosi.c
816 usr/src/uts/intel/io/acpica/utilities/utresrc.c
817 usr/src/uts/intel/io/acpica/utilities/utstate.c
818 usr/src/uts/intel/io/acpica/utilities/uttrack.c
819 usr/src/uts/intel/io/acpica/utilities/utxface.c
820 usr/src/uts/intel/io/acpica/utilities/utxferror.c
821 usr/src/uts/intel/sys/acpi/acapps.h
822 usr/src/uts/intel/sys/acpi/accommon.h
823 usr/src/uts/intel/sys/acpi/acconfig.h
824 usr/src/uts/intel/sys/acpi/acdebug.h
825 usr/src/uts/intel/sys/acpi/acdisasm.h
826 usr/src/uts/intel/sys/acpi/acdispat.h
827 usr/src/uts/intel/sys/acpi/acevents.h
828 usr/src/uts/intel/sys/acpi/acexcep.h
829 usr/src/uts/intel/sys/acpi/acglobal.h
830 usr/src/uts/intel/sys/acpi/achware.h
831 usr/src/uts/intel/sys/acpi/acinterp.h
832 usr/src/uts/intel/sys/acpi/aclocal.h
833 usr/src/uts/intel/sys/acpi/acmacros.h
834 usr/src/uts/intel/sys/acpi/acnames.h
835 usr/src/uts/intel/sys/acpi/acnamesp.h
836 usr/src/uts/intel/sys/acpi/acobject.h
837 usr/src/uts/intel/sys/acpi/acopcode.h
838 usr/src/uts/intel/sys/acpi/acoutput.h
839 usr/src/uts/intel/sys/acpi/acparser.h
840 usr/src/uts/intel/sys/acpi/acpi.h
841 usr/src/uts/intel/sys/acpi/acpiosxf.h
842 usr/src/uts/intel/sys/acpi/acpixf.h
843 usr/src/uts/intel/sys/acpi/acpredef.h
844 usr/src/uts/intel/sys/acpi/acresrc.h
845 usr/src/uts/intel/sys/acpi/acrestyp.h
846 usr/src/uts/intel/sys/acpi/acstruct.h
847 usr/src/uts/intel/sys/acpi/actables.h
848 usr/src/uts/intel/sys/acpi/actbl.h
849 usr/src/uts/intel/sys/acpi/actbl1.h
850 usr/src/uts/intel/sys/acpi/actbl2.h
851 usr/src/uts/intel/sys/acpi/actypes.h
852 usr/src/uts/intel/sys/acpi/acutils.h
853 usr/src/uts/intel/sys/acpi/amlcode.h

```

```

854 usr/src/uts/intel/sys/acpi/amlresrc.h
855 usr/src/uts/intel/sys/acpi/platform/acpygwin.h
856 usr/src/uts/intel/sys/acpi/platform/acefi.h
857 usr/src/uts/intel/sys/acpi/platform/acenv.h
858 usr/src/uts/intel/sys/acpi/platform/acfreebsd.h
859 usr/src/uts/intel/sys/acpi/platform/acgcc.h
860 usr/src/uts/intel/sys/acpi/platform/acintel.h
861 usr/src/uts/intel/sys/acpi/platform/aclinux.h
862 usr/src/uts/intel/sys/acpi/platform/acmsvc.h
863 usr/src/uts/intel/sys/acpi/platform/acnetbsd.h
864 usr/src/uts/intel/sys/acpi/platform/acos2.h
865 usr/src/uts/intel/sys/acpi/platform/acsolaris.h
866 usr/src/uts/intel/sys/acpi/platform/acwin.h
867 usr/src/uts/intel/sys/acpi/platform/acwin64.h

```

new/usr/src/uts/common/Makefile.files

1

42852 Fri Jul 20 10:59:29 2012

new/usr/src/uts/common/Makefile.files

XXX 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 #
21 #
22 #
23 # Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 #
27 #
28 #
29 # This Makefile defines all file modules for the directory uts/common
30 # and its children. These are the source files which may be considered
31 # common to all SunOS systems.
32 #
33 i386_CORE_OBJS += \
34     atomic.o      \
35     avintr.o      \
36     pic.o
37 #
38 sparc_CORE_OBJS +=
39 #
40 COMMON_CORE_OBJS += \
41     beep.o        \
42     bitset.o      \
43     bp_map.o      \
44     brand.o       \
45     cpucaps.o     \
46     cmt.o         \
47     cmt_policy.o  \
48     cpu.o         \
49     cpu_event.o   \
50     cpu_intr.o    \
51     cpu_pm.o      \
52     cpupart.o     \
53     cap_util.o    \
54     disp.o        \
55     group.o       \
56     kstat_fr.o    \
57     iscsiboot_prop.o \
58     lgrp.o        \
59     lgrp_topo.o   \
60     mmapobj.o     \
61     mutex.o
```

new/usr/src/uts/common/Makefile.files

2

```
62     page_lock.o   \
63     page_retire.o \
64     panic.o       \
65     param.o       \
66     pg.o          \
67     pghw.o        \
68     putnext.o     \
69     rctl_proc.o   \
70     rwlock.o      \
71     seg_kmem.o    \
72     softint.o     \
73     string.o      \
74     strtol.o      \
75     strtoul.o     \
76     strtoll.o     \
77     strtoull.o    \
78     thread_intr.o \
79     vm_page.o     \
80     vm_pagelist.o \
81     zlib_obj.o    \
82     clock_tick.o
83 #
84 CORE_OBJS += $(COMMON_CORE_OBJS) $(MACH)_CORE_OBJS
85 #
86 ZLIB_OBJS = zutil.o zmod.o zmod_subr.o \
87     adler32.o crc32.o deflate.o inffast.o \
88     inflate.o inftrees.o trees.o
89 #
90 GENUNIX_OBJS += \
91     access.o      \
92     acl.o         \
93     acl_common.o  \
94     adjtime.o     \
95     alarm.o       \
96     aio_subr.o    \
97     auditsys.o    \
98     audit_core.o  \
99     audit_zone.o  \
100     audit_memory.o \
101     autoconf.o    \
102     avl.o         \
103     bdev_dsort.o  \
104     bio.o         \
105     bitmap.o      \
106     blabel.o      \
107     brandsys.o    \
108     bz2blocksort.o \
109     bz2compress.o \
110     bz2decompress.o \
111     bz2randtable.o \
112     bz2bzlib.o    \
113     bz2crctable.o \
114     bz2huffman.o  \
115     callb.o       \
116     callout.o     \
117     chdir.o       \
118     chmod.o       \
119     chown.o       \
120     cladm.o       \
121     class.o       \
122     clock.o       \
123     clock_highres.o \
124     clock_realtime.o \
125     close.o       \
126     compress.o    \
127     condvar.o
```

new/usr/src/uts/common/Makefile.files

```

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      rusesys.o   \
192      gid.o       \
193      groups.o    \

```

3

new/usr/src/uts/common/Makefile.files

```

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      kpcp.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       \

```

4

new/usr/src/uts/common/Makefile.files

```

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      \

```

5

new/usr/src/uts/common/Makefile.files

```

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      sundpi.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      tn timer.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          \

```

6

new/usr/src/uts/common/Makefile.files

7

```
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
```

new/usr/src/uts/common/Makefile.files

8

```
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 += audiotls.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 OLDPPTY_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_queue.o \
551 queue.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 socksd.o socksdpsubr.o
595 SCTP SOCK_MOD_OBJS += sockmod_sctp.o sockscctp.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 rds_v3_ddi.o bind.o loop.o threads.o connection.o \
607 transport.o cong.o sysctl.o message.o rds_rcv.o send.o \
608 stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
609 ib_rcv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
610 rds_v3_sc.o rds_v3_debug.o rds_v3_impl.o rdma.o rds_v3_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 += genty.o
621 TCO_OBJS += ticots.o
623 TCO0_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_6T04_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
```

```
657 MAC_IB_OBJS +=      mac_ib.o
659 IPTUN_OBJS +=      iptun_dev.o iptun_ctl.o iptun.o
661 AGGR_OBJS +=      aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
662                  aggr_send.o aggr_recv.o aggr_lacp.o
664 SOFTMAC_OBJS +=      softmac_main.o softmac_ctl.o softmac_capab.o \
665                  softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o
667 NET80211_OBJS +=      net80211.o net80211_proto.o net80211_input.o \
668                  net80211_output.o net80211_node.o net80211_crypto.o \
669                  net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
670                  net80211_crypto_tkip.o net80211_crypto_ccmp.o \
671                  net80211_ht.o
673 VNIC_OBJS +=      vnic_ctl.o vnic_dev.o
675 SIMNET_OBJS +=      simnet.o
677 IB_OBJS +=      ibnex.o ibnex_ioctl.o ibnex_hca.o
679 IBCM_OBJS +=      ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
680                  ibcm_arp.o ibcm_arp_link.o
682 IBDM_OBJS +=      ibdm.o
684 IBDMA_OBJS +=      ibdma.o
686 IBMF_OBJS +=      ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.
687                  ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
688                  ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
689                  ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o
691 IBTL_OBJS +=      ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
692                  ibtl_cq.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
693                  ibtl_mcg.o ibtl_ibnex.o ibtl_srq.o ibtl_part.o
695 TAVOR_OBJS +=      tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
696                  tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
697                  tavor_mr.o tavor_qp.o tavor_qpm.o tavor_rsrc.o \
698                  tavor_srq.o tavor_stats.o tavor_umap.o tavor_wr.o
700 HERMON_OBJS +=      hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
701                  hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
702                  hermon_mr.o hermon_qp.o hermon_qpm.o hermon_rsrc.o \
703                  hermon_srq.o hermon_stats.o hermon_umap.o hermon_wr.o \
704                  hermon_fcoib.o hermon_fm.o
706 DAPLT_OBJS +=      daplt.o
708 SOL_OFS_OBJS +=      sol_cma.o sol_ib_cma.o sol_uobj.o \
709                  sol_ofs_debug_util.o sol_ofs_gen_util.o \
710                  sol_kverbs.o
712 SOL_UCMA_OBJS +=      sol_ucma.o
714 SOL_UVERBS_OBJS +=      sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
715                  sol_uverbs_hca.o sol_uverbs_qp.o
717 SOL_UMAD_OBJS +=      sol_umad.o
719 KSTAT_OBJS +=      kstat.o
721 KSYMS_OBJS +=      ksyms.o
```

```
723 INSTANCE_OBJS +=      inst_sync.o
725 IWSCN_OBJS +=      iwscons.o
727 LOFI_OBJS +=      lofi.o LzmaDec.o
729 FSSNAP_OBJS +=      fssnap.o
731 FSSNAPIF_OBJS +=      fssnap_if.o
733 MM_OBJS +=      mem.o
735 PHYSMEM_OBJS +=      physmem.o
737 OPTIONS_OBJS +=      options.o
739 WINLOCK_OBJS +=      winlockio.o
741 PM_OBJS +=      pm.o
742 SRN_OBJS +=      srn.o
744 PSEUDO_OBJS +=      pseudonex.o
746 RAMDISK_OBJS +=      ramdisk.o
748 LLC1_OBJS +=      llc1.o
750 USBKBM_OBJS +=      usbkbm.o
752 USBWCM_OBJS +=      usbwcm.o
754 BOFI_OBJS +=      bofi.o
756 HID_OBJS +=      hid.o
758 HWA_RC_OBJS +=      hwarc.o
760 USBSKEL_OBJS +=      usbskel.o
762 USBVC_OBJS +=      usbvc.o usbvc_v412.o
764 HIDPARSER_OBJS +=      hidparser.o
766 USB_AC_OBJS +=      usb_ac.o
768 USB_AS_OBJS +=      usb_as.o
770 USB_AH_OBJS +=      usb_ah.o
772 USBMS_OBJS +=      usbms.o
774 USBPRN_OBJS +=      usbprn.o
776 UGEN_OBJS +=      ugen.o
778 USBSER_OBJS +=      usbser.o usbser_rseq.o
780 USBSACM_OBJS +=      usb sacm.o
782 USBSER_KEYSPAN_OBJS +=      usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o
784 USBS49_FW_OBJS +=      keyspan_49fw.o
786 USBSPRL_OBJS +=      usbser_pl2303.o pl2303_dsd.o
```

new/usr/src/uts/common/Makefile.files

13

```
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_uugen.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_uugen.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_async.o \
837                 s1394_bus_reset.o s1394_cmp.o s1394_csr.o s1394_dev_disco.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
```

new/usr/src/uts/common/Makefile.files

14

```
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
```

```
921 NSKERN_OBJS += nsc_ddi.o \
922                nsc_proc.o \
923                nsc_raw.o \
924                nsc_thread.o \
925                nskernd.o

927 SV_OBJS += sv.o

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

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

935 #
936 #      Build up defines and paths.

938 ST_OBJS += st.o st_conf.o

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

947 EMLXS_FW_OBJS += emlxs_fw.o

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

953 FCT_OBJS += discovery.o fct.o

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

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

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

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

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

965 ISCSIT_SHARED_OBJS += \
966                    iscsit_common.o

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

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

975 STMF_OBJS += lun_map.o stmf.o

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

979 SYSMSG_OBJS += sysmsg.o

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

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

```
986 LOGINDMUX_OBJS += logindmux.o

988 DEVINFO_OBJS += devinfo.o

990 DEVPOLL_OBJS += devpoll.o

992 DEVP00L_OBJS += devpool.o

994 I8042_OBJS += i8042.o

996 KB8042_OBJS += \
997             at_keyprocess.o \
998             kb8042.o \
999             kb8042_keytables.o

1001 MOUSE8042_OBJS += mouse8042.o

1003 FDC_OBJS += fdc.o

1005 ASY_OBJS += asy.o

1007 ECPP_OBJS += ecpp.o

1009 VUIDM3P_OBJS += vuidmice.o vuidm3p.o

1011 VUIDM4P_OBJS += vuidmice.o vuidm4p.o

1013 VUIDM5P_OBJS += vuidmice.o vuidm5p.o

1015 VUIDPS2_OBJS += vuidmice.o vuidps2.o

1017 HPCSV_C_OBJS += hpcsvc.o

1019 PCIE_MISC_OBJS += pcie.o pcie_fault.o pcie_hp.o pciehpc.o pcishpc.o pcie_pwr.o p

1021 PCIHPNEXUS_OBJS += pcihp.o

1023 OPENEEPR_OBJS += openprom.o

1025 RANDOM_OBJS += random.o

1027 PSHOT_OBJS += pshot.o

1029 GEN_DRV_OBJS += gen_drv.o

1031 TCLIENT_OBJS += tclient.o

1033 TPHCI_OBJS += tphci.o

1035 TVHCI_OBJS += tvhci.o

1037 EMUL64_OBJS += emul64.o emul64_bsd.o

1039 FCP_OBJS += fcp.o

1041 FCIP_OBJS += fcip.o

1043 FCSM_OBJS += fcsn.o

1045 FCTL_OBJS += fctl.o

1047 FP_OBJS += fp.o

1049 QLC_OBJS += ql_api.o ql_debug.o ql_hba_fru.o ql_init.o ql_iocb.o ql_ioctl.o \
1050            ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o
```

new/usr/src/uts/common/Makefile.files**17**

```
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 \
```

new/usr/src/uts/common/Makefile.files**18**

```
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 \
```

new/usr/src/uts/common/Makefile.files

```

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_odor.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

```

19

new/usr/src/uts/common/Makefile.files

```

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 priocntl.o prsubr.o prusr.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 += specsusr.o specvfsops.o specvnops.o

1267 SOCK_OBJS += socksusr.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 nl7http.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_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 #
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

```

20

new/usr/src/uts/common/Makefile.files

21

```

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 \

```

new/usr/src/uts/common/Makefile.files

22

```

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     zlock.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 PKCT_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     xdrdma_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 +=        rlmmod.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 old.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 ksslioctl.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 \

```

new/usr/src/uts/common/Makefile.files

25

```

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= des 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=

```

new/usr/src/uts/common/Makefile.files

26

```

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 DLB00T_OBJS += bootparam_xdr.o nfs_dlnet.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 KAI0_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 E0IB_OBJS += eib_adm.o eib_chan.o eib_cmnm.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 DLPSTUB_OBJS += dlpistub.o
1753 SDP_OBJS += sdppdi.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_rcv2.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_rcv.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
```

new/usr/src/uts/common/Makefile.files

29

```

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 assembler
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 csapi.o espi.o ix1010.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_nvmm.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_nvmm.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

```

new/usr/src/uts/common/Makefile.files

30

```

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 \

```

new/usr/src/uts/common/Makefile.files

31

```
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
```

new/usr/src/uts/common/io/ixgbe/ixgbe_82598.c

1

```
*****
39876 Fri Jul 20 10:59:29 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_82598.c
XXX Intel X540 support
*****
1 /*****
3 Copyright (c) 2001-2012, Intel Corporation
3 Copyright (c) 2001-2010, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_82598.c,v 1.13 2012/07/05 20:51:44 jfv Exp $
34 /*$FreeBSD$*/
35 #include "ixgbe_type.h"
36 #include "ixgbe_82598.h"
37 #include "ixgbe_api.h"
38 #include "ixgbe_common.h"
39 #include "ixgbe_phy.h"
40 u32 ixgbe_get_pcie_msix_count_82598(struct ixgbe_hw *hw);
41 s32 ixgbe_init_ops_82598(struct ixgbe_hw *hw);
42 static s32 ixgbe_get_link_capabilities_82598(struct ixgbe_hw *hw,
43 ixgbe_link_speed *speed,
44 bool *autoneg);
45 static enum ixgbe_media_type ixgbe_get_media_type_82598(struct ixgbe_hw *hw);
46 s32 ixgbe_fc_enable_82598(struct ixgbe_hw *hw, s32 packetbuf_num);
47 static s32 ixgbe_start_mac_link_82598(struct ixgbe_hw *hw,
48 bool autoneg_wait_to_complete);
49 static s32 ixgbe_check_mac_link_82598(struct ixgbe_hw *hw,
50 ixgbe_link_speed *speed, bool *link_up,
51 bool link_up_wait_to_complete);
52 static s32 ixgbe_setup_mac_link_82598(struct ixgbe_hw *hw,
53 ixgbe_link_speed speed,
54 bool autoneg,
55 bool autoneg_wait_to_complete);
56 static s32 ixgbe_setup_copper_link_82598(struct ixgbe_hw *hw,
57 ixgbe_link_speed speed,
58 bool autoneg,
```

new/usr/src/uts/common/io/ixgbe/ixgbe_82598.c

2

```
57 bool autoneg_wait_to_complete);
58 static s32 ixgbe_reset_hw_82598(struct ixgbe_hw *hw);
59 s32 ixgbe_start_hw_82598(struct ixgbe_hw *hw);
60 void ixgbe_enable_relaxed_ordering_82598(struct ixgbe_hw *hw);
61 s32 ixgbe_set_vmdq_82598(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
62 static s32 ixgbe_clear_vmdq_82598(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
63 s32 ixgbe_set_vfta_82598(struct ixgbe_hw *hw, u32 vlan,
64 u32 vmd, bool vlan_on);
65 static s32 ixgbe_clear_vfta_82598(struct ixgbe_hw *hw);
66 static void ixgbe_set_rxpba_82598(struct ixgbe_hw *hw, int num_pb,
67 u32 headroom, int strategy);
68 s32 ixgbe_read_analog_reg8_82598(struct ixgbe_hw *hw, u32 reg, u8 *val);
69 s32 ixgbe_write_analog_reg8_82598(struct ixgbe_hw *hw, u32 reg, u8 val);
70 s32 ixgbe_read_i2c_eeprom_82598(struct ixgbe_hw *hw, u8 byte_offset,
71 u8 *eeprom_data);
72 u32 ixgbe_get_supported_physical_layer_82598(struct ixgbe_hw *hw);
73 s32 ixgbe_init_phy_ops_82598(struct ixgbe_hw *hw);
74 void ixgbe_set_lan_id_multi_port_pcie_82598(struct ixgbe_hw *hw);
75 void ixgbe_set_pcie_completion_timeout(struct ixgbe_hw *hw);
76
77 /**
78 * ixgbe_set_pcie_completion_timeout - set pci-e completion timeout
79 * @hw: pointer to the HW structure
80 *
81 * The defaults for 82598 should be in the range of 50us to 50ms,
82 * however the hardware default for these parts is 500us to 1ms which is less
83 * than the 10ms recommended by the pci-e spec. To address this we need to
84 * increase the value to either 10ms to 250ms for capability version 1 config,
85 * or 16ms to 55ms for version 2.
86 */
87 void ixgbe_set_pcie_completion_timeout(struct ixgbe_hw *hw)
88 {
89     u32 gcr = IXGBE_READ_REG(hw, IXGBE_GCR);
90     u16 pcie_devctl2;
91
92     /* only take action if timeout value is defaulted to 0 */
93     if (gcr & IXGBE_GCR_CMPL_TMOUT_MASK)
94         goto out;
95
96     /*
97      * if capabilities version is type 1 we can write the
98      * timeout of 10ms to 250ms through the GCR register
99      */
100     if (!(gcr & IXGBE_GCR_CAP_VER2)) {
101         gcr |= IXGBE_GCR_CMPL_TMOUT_10ms;
102         goto out;
103     }
104
105     /* for version 2 capabilities we need to write the config space
106      * directly in order to set the completion timeout value for
107      * 16ms to 55ms
108      */
109     pcie_devctl2 = IXGBE_READ_PCIE_WORD(hw, IXGBE_PCI_DEVICE_CONTROL2);
110     pcie_devctl2 |= IXGBE_PCI_DEVICE_CONTROL2_16ms;
111     IXGBE_WRITE_PCIE_WORD(hw, IXGBE_PCI_DEVICE_CONTROL2, pcie_devctl2);
112 out:
113     /* disable completion timeout resend */
114     gcr &= ~IXGBE_GCR_CMPL_TMOUT_RESEND;
115     IXGBE_WRITE_REG(hw, IXGBE_GCR, gcr);
116 }
117
118 /**
119 * ixgbe_get_pcie_msix_count_82598 - Gets MSI-X vector count
120 * @hw: pointer to hardware structure
121 */
```

```

123  * Read PCIe configuration space, and get the MSI-X vector count from
124  * the capabilities table.
125  **/
126  u32 ixgbe_get_pcie_msix_count_82598(struct ixgbe_hw *hw)
127  {
128      u32 msix_count = 18;

130      DEBUGFUNC("ixgbe_get_pcie_msix_count_82598");

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

137          /* MSI-X count is zero-based in HW, so increment to give
138             * proper value */
139          msix_count++;
140      }
141      return msix_count;
142  }

144  /**
145   * ixgbe_init_ops_82598 - Inits func ptrs and MAC type
146   * @hw: pointer to hardware structure
147   *
148   * Initialize the function pointers and assign the MAC type for 82598.
149   * Does not touch the hardware.
150   **/
151  s32 ixgbe_init_ops_82598(struct ixgbe_hw *hw)
152  {
153      struct ixgbe_mac_info *mac = &hw->mac;
154      struct ixgbe_phy_info *phy = &hw->phy;
155      s32 ret_val;

157      DEBUGFUNC("ixgbe_init_ops_82598");

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

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

165      /* MAC */
166      mac->ops.start_hw = &ixgbe_start_hw_82598;
167      mac->ops.enable_relaxed_ordering = &ixgbe_enable_relaxed_ordering_82598;
168      mac->ops.reset_hw = &ixgbe_reset_hw_82598;
169      mac->ops.get_media_type = &ixgbe_get_media_type_82598;
170      mac->ops.get_supported_physical_layer =
171          &ixgbe_get_supported_physical_layer_82598;
172      mac->ops.read_analog_reg8 = &ixgbe_read_analog_reg8_82598;
173      mac->ops.write_analog_reg8 = &ixgbe_write_analog_reg8_82598;
174      mac->ops.set_lan_id = &ixgbe_set_lan_id_multi_port_pcie_82598;

176      /* RAR, Multicast, VLAN */
177      mac->ops.set_vmdq = &ixgbe_set_vmdq_82598;
178      mac->ops.clear_vmdq = &ixgbe_clear_vmdq_82598;
179      mac->ops.set_vfta = &ixgbe_set_vfta_82598;
180      mac->ops.set_vlvy = NULL;
181      mac->ops.clear_vfta = &ixgbe_clear_vfta_82598;

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

186      mac->mcft_size = 128;
187      mac->vft_size = 128;
188      mac->num_rar_entries = 16;

```

```

151      mac->rx_pb_size = 512;
152      mac->max_tx_queues = 32;
153      mac->max_rx_queues = 64;
154      mac->max_msix_vectors = ixgbe_get_pcie_msix_count_generic(hw);
155      mac->max_msix_vectors = ixgbe_get_pcie_msix_count_82598(hw);

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

160      /* Link */
161      mac->ops.check_link = &ixgbe_check_mac_link_82598;
162      mac->ops.setup_link = &ixgbe_setup_mac_link_82598;
163      mac->ops.flap_tx_laser = NULL;
164      mac->ops.get_link_capabilities = &ixgbe_get_link_capabilities_82598;
165      mac->ops.setup_rxpba = &ixgbe_set_rxpba_82598;
166      mac->ops.get_link_capabilities =
167          &ixgbe_get_link_capabilities_82598;

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

172      return ret_val;
173  }

175  /**
176   * ixgbe_init_phy_ops_82598 - PHY/SFP specific init
177   * @hw: pointer to hardware structure
178   *
179   * Initialize any function pointers that were not able to be
180   * set during init_shared_code because the PHY/SFP type was
181   * not known. Perform the SFP init if necessary.
182   **/
183  s32 ixgbe_init_phy_ops_82598(struct ixgbe_hw *hw)
184  {
185      struct ixgbe_mac_info *mac = &hw->mac;
186      struct ixgbe_phy_info *phy = &hw->phy;
187      s32 ret_val = IXGBE_SUCCESS;
188      u16 list_offset, data_offset;

190      DEBUGFUNC("ixgbe_init_phy_ops_82598");

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

195      /* Overwrite the link function pointers if copper PHY */
196      if (mac->ops.get_media_type(hw) == ixgbe_media_type_copper) {
197          mac->ops.setup_link = &ixgbe_setup_copper_link_82598;
198          mac->ops.get_link_capabilities =
199              &ixgbe_get_copper_link_capabilities_generic;
200      }

202      switch (hw->phy.type) {
203      case ixgbe_phy_tn:
204          phy->ops.setup_link = &ixgbe_setup_phy_link_tnx;
205          phy->ops.check_link = &ixgbe_check_phy_link_tnx;
206          phy->ops.get_firmware_version =
207              &ixgbe_get_phy_firmware_version_tnx;
208          break;
209      case ixgbe_phy_aq:
210          phy->ops.get_firmware_version =
211              &ixgbe_get_phy_firmware_version_generic;
212          break;
213      case ixgbe_phy_nl:
214          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;
259         regval &= ~IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
260         IXGBE_WRITE_REG(hw, IXGBE_DCA_TXCTRL(i), regval);
261     }

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);
267         regval &= ~(IXGBE_DCA_RXCTRL_DESC_WRO_EN |
268                    IXGBE_DCA_RXCTRL_DESC_HSR0_EN);
269         IXGBE_WRITE_REG(hw, IXGBE_DCA_RXCTRL(i), regval);
270     }

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:
354     case ixgbe_phy_aq:
355         media_type = ixgbe_media_type_copper;
356         goto out;
357     default:
358         break;
359     }

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
394  * @packetbuf_num: packet buffer number (0-7)
395  *
396  * Enable flow control according to the current settings.
397  */

```

```

397 s32 ixgbe_fc_enable_82598(struct ixgbe_hw *hw)
437 s32 ixgbe_fc_enable_82598(struct ixgbe_hw *hw, s32 packetbuf_num)
398 {
399     s32 ret_val = IXGBE_SUCCESS;
400     u32 fctrl_reg;
401     u32 rmcs_reg;
402     u32 reg;
403     u32 fctrl, fcrth;
443     u32 rx_pba_size;
404     u32 link_speed = 0;
405     int i;
406     bool link_up;

408     DEBUGFUNC("ixgbe_fc_enable_82598");

410     /* Validate the water mark configuration */
411     if (!hw->fc.pause_time) {
412         ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
413         goto out;
414     }

416     /* Low water mark of zero causes XOFF floods */
417     for (i = 0; i < IXGBE_DCB_MAX_TRAFFIC_CLASS; i++) {
418         if ((hw->fc.current_mode & ixgbe_fc_tx_pause) &&
419             hw->fc.high_water[i]) {
420             if (!hw->fc.low_water[i] ||
421                 hw->fc.low_water[i] >= hw->fc.high_water[i]) {
422                 DEBUGOUT("Invalid water mark configuration\n");
423                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
424                 goto out;
425             }
426         }
427     }

429     /*
430     * On 82598 having Rx FC on causes resets while doing 1G
431     * so if it's on turn it off once we know link_speed. For
432     * more details see 82598 Specification update.
433     */
434     hw->mac.ops.check_link(hw, &link_speed, &link_up, FALSE);
435     if (link_up && link_speed == IXGBE_LINK_SPEED_1GB_FULL) {
436         switch (hw->fc.requested_mode) {
437             case ixgbe_fc_full:
438                 hw->fc.requested_mode = ixgbe_fc_tx_pause;
439                 break;
440             case ixgbe_fc_rx_pause:
441                 hw->fc.requested_mode = ixgbe_fc_none;
442                 break;
443             default:
444                 /* no change */
445                 break;
446         }
447     }

449     /* Negotiate the fc mode to use */
450     ixgbe_fc_autoneg(hw);
470     ret_val = ixgbe_fc_autoneg(hw);
471     if (ret_val == IXGBE_ERR_FLOW_CONTROL)
472         goto out;

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

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

```

```

459     /*
460     * The possible values of fc.current_mode are:
461     * 0: Flow control is completely disabled
462     * 1: Rx flow control is enabled (we can receive pause frames,
463     *   but not send pause frames).
464     * 2: Tx flow control is enabled (we can send pause frames but
465     *   we do not support receiving pause frames).
466     * 3: Both Rx and Tx flow control (symmetric) are enabled.
467     * other: Invalid.
468     */
469     switch (hw->fc.current_mode) {
470     case ixgbe_fc_none:
471         /*
472         * Flow control is disabled by software override or autoneg.
473         * The code below will actually disable it in the HW.
474         */
475         break;
476     case ixgbe_fc_rx_pause:
477         /*
478         * Rx Flow control is enabled and Tx Flow control is
479         * disabled by software override. Since there really
480         * isn't a way to advertise that we are capable of Rx
481         * Pause ONLY, we will advertise that we support both
482         * symmetric and asymmetric Rx PAUSE. Later, we will
483         * disable the adapter's ability to send PAUSE frames.
484         */
485         fctrl_reg |= IXGBE_FCTRL_RFCE;
486         break;
487     case ixgbe_fc_tx_pause:
488         /*
489         * Tx Flow control is enabled, and Rx Flow control is
490         * disabled by software override.
491         */
492         rmcs_reg |= IXGBE_RMCS_TFCE_802_3X;
493         break;
494     case ixgbe_fc_full:
495         /* Flow control (both Rx and Tx) is enabled by SW override. */
496         fctrl_reg |= IXGBE_FCTRL_RFCE;
497         rmcs_reg |= IXGBE_RMCS_TFCE_802_3X;
498         break;
499     default:
500         DEBUGOUT("Flow control param set incorrectly\n");
501         ret_val = IXGBE_ERR_CONFIG;
502         goto out;
503     }

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

510     /* Set up and enable Rx high/low water mark thresholds, enable XON. */
511     for (i = 0; i < IXGBE_DCB_MAX_TRAFFIC_CLASS; i++) {
512         if ((hw->fc.current_mode & ixgbe_fc_tx_pause) &&
513             hw->fc.high_water[i]) {
514             fctrl = (hw->fc.low_water[i] << 10) | IXGBE_FCRTL_XONE;
515             fcrth = (hw->fc.high_water[i] << 10) | IXGBE_FCRTH_FCEN;
516             IXGBE_WRITE_REG(hw, IXGBE_FCRTL(i), fctrl);
517             IXGBE_WRITE_REG(hw, IXGBE_FCRTH(i), fcrth);
518         } else {
519             IXGBE_WRITE_REG(hw, IXGBE_FCRTL(i), 0);
520             IXGBE_WRITE_REG(hw, IXGBE_FCRTH(i), 0);
521         }
522     }
523     if (hw->fc.current_mode & ixgbe_fc_tx_pause) {
524         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_FCRTL_FCEN;

546         IXGBE_WRITE_REG(hw, IXGBE_FCRTL(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_FCRTL, hw->fc.pause_time / 2);
557     IXGBE_WRITE_REG(hw, IXGBE_FCRTL, (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. 0xC00C 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     }
737 }
738 out:
739     return IXGBE_SUCCESS;
740 }

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;
728
729     DEBUGFUNC("ixgbe_setup_mac_link_82598");
730
731     /* Check to see if speed passed in is supported. */
732     ixgbe_get_link_capabilities(hw, &link_capabilities, &autoneg);
733     (void) ixgbe_get_link_capabilities(hw, &link_capabilities, &autoneg);
734     speed &= link_capabilities;
735
736     if (speed == IXGBE_LINK_SPEED_UNKNOWN)
737         status = IXGBE_ERR_LINK_SETUP;
738
739     /* Set KX4/KX support according to speed requested */
740     else if (link_mode == IXGBE_AUTOC_LMS_KX4_AN ||
741             link_mode == IXGBE_AUTOC_LMS_KX4_AN_1G_AN) {
742         autoc &= ~IXGBE_AUTOC_KX4_KX_SUPP_MASK;
743         if (speed & IXGBE_LINK_SPEED_10GB_FULL)
744             autoc |= IXGBE_AUTOC_KX4_SUPP;
745         if (speed & IXGBE_LINK_SPEED_1GB_FULL)
746             autoc |= IXGBE_AUTOC_KX_SUPP;
747         if (autoc != curr_autoc)
748             IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc);
749     }
750
751     if (status == IXGBE_SUCCESS) {
752         /*
753          * Setup and restart the link based on the new values in
754          * ixgbe_hw This will write the AUTOC register based on the new
755          * stored values
756          */
757         status = ixgbe_start_mac_link_82598(hw,
758                                             autoneg_wait_to_complete);
759     }
760
761     return status;
762 }
763
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;
779
780     DEBUGFUNC("ixgbe_setup_copper_link_82598");
781
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
786     /* Set up MAC */
787     ixgbe_start_mac_link_82598(hw, autoneg_wait_to_complete);
788     (void) ixgbe_start_mac_link_82598(hw, autoneg_wait_to_complete);
789
790     return status;
791 }
792
793 /**
794 * ixgbe_reset_hw_82598 - Performs hardware reset
795 * @hw: pointer to hardware structure
796 *
797 * Resets the hardware by resetting the transmit and receive units, masks and
798 * clears all interrupts, performing a PHY reset, and performing a link (MAC)
799 * reset.
800 **/
801 static s32 ixgbe_reset_hw_82598(struct ixgbe_hw *hw)
802 {
803     s32 status = IXGBE_SUCCESS;
804     s32 phy_status = IXGBE_SUCCESS;
805     u32 ctrl;
806     u32 gheccr;
807     u32 i;
808     u32 autoc;
809     u8 analog_val;
810
811     DEBUGFUNC("ixgbe_reset_hw_82598");
812
813     /* Call adapter stop to disable tx/rx and clear interrupts */
814     status = hw->mac.ops.stop_adapter(hw);
815     if (status != IXGBE_SUCCESS)
816         goto reset_hw_out;
817     hw->mac.ops.stop_adapter(hw);
818
819     /*
820      * Power up the Atlas Tx lanes if they are currently powered down.
821      * Atlas Tx lanes are powered down for MAC loopback tests, but
822      * they are not automatically restored on reset.
823      */
824     hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_LPBK, &analog_val);
825     if (analog_val & IXGBE_ATLAS_PDN_TX_REG_EN) {
826         /* Enable Tx Atlas so packets can be transmitted again */
827         hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_LPBK,
828                                     &analog_val);
829         analog_val &= ~IXGBE_ATLAS_PDN_TX_REG_EN;
830         hw->mac.ops.write_analog_reg8(hw, IXGBE_ATLAS_PDN_LPBK,
831                                     analog_val);
832     }
833
834     hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_10G,
835                                 &analog_val);
836     analog_val &= ~IXGBE_ATLAS_PDN_TX_10G_QL_ALL;
837     hw->mac.ops.write_analog_reg8(hw, IXGBE_ATLAS_PDN_10G,
838                                 analog_val);
839
840     hw->mac.ops.read_analog_reg8(hw, IXGBE_ATLAS_PDN_1G,
841                                 &analog_val);
842     analog_val &= ~IXGBE_ATLAS_PDN_TX_1G_QL_ALL;
843     hw->mac.ops.write_analog_reg8(hw, IXGBE_ATLAS_PDN_1G,
844                                 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;
859         else if (phy_status == IXGBE_ERR_SFP_NOT_PRESENT)
860             goto no_phy_reset;
861     }

862     no_phy_reset:
863     /*
864     * Prevent the PCI-E bus from hanging by disabling PCI-E master
865     * access and verify no pending requests before reset
866     */
867     (void) ixgbe_disable_pcie_master(hw);

868     mac_reset_top:
869     /*
870     * Issue global reset to the MAC. This needs to be a SW reset.
871     * If link reset is used, it might reset the MAC when mng is using it
872     */
873     ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL) | IXGBE_CTRL_RST;
874     IXGBE_WRITE_REG(hw, IXGBE_CTRL, ctrl);
875     ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
876     IXGBE_WRITE_REG(hw, IXGBE_CTRL, (ctrl | IXGBE_CTRL_RST));
877     IXGBE_WRITE_FLUSH(hw);

878     /* Poll for reset bit to self-clear indicating reset is complete */
879     for (i = 0; i < 10; i++) {
880         usec_delay(1);
881         ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
882         if (!(ctrl & IXGBE_CTRL_RST))
883             break;
884     }
885     if (ctrl & IXGBE_CTRL_RST) {
886         status = IXGBE_ERR_RESET_FAILED;
887         DEBUGOUT("Reset polling failed to complete.\n");
888     }

889     msec_delay(50);

890     /*
891     * Double resets are required for recovery from certain error
892     * conditions. Between resets, it is necessary to stall to allow time
893     * for any pending HW events to complete. We use 1usec since that is
894     * what is needed for ixgbe_disable_pcie_master(). The second reset
895     * then clears out any effects of those events.
896     */
897     if (hw->mac.flags & IXGBE_FLAGS_DOUBLE_RESET_REQUIRED) {
898         hw->mac.flags &= ~IXGBE_FLAGS_DOUBLE_RESET_REQUIRED;
899         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 defaults.
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 }

928     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);
967         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:
1189     case ixgbe_phy_aq:
1190     case ixgbe_phy_cu_unknown:
1191         hw->phy.ops.read_reg(hw, IXGBE_MDIO_PHY_EXT_ABILITY,
1192             IXGBE_MDIO_PMA_PMD_DEV_TYPE, &ext_ability);
1193         if (ext_ability & IXGBE_MDIO_PHY_10GBASET_ABILITY)
1194             physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_T;
1195         if (ext_ability & IXGBE_MDIO_PHY_1000BASET_ABILITY)
1196             physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_T;
1197         if (ext_ability & IXGBE_MDIO_PHY_100BASETX_ABILITY)
1198             physical_layer |= IXGBE_PHYSICAL_LAYER_100BASE_TX;
1199         goto out;
1200     default:
1201         break;
1202     }

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;
1320         regval |= IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
1321         IXGBE_WRITE_REG(hw, IXGBE_DCA_TXCTRL(i), regval);
1322     }

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;
1328         regval |= (IXGBE_DCA_RXCTRL_DESC_WRO_EN |
1329                   IXGBE_DCA_RXCTRL_DESC_HSR0_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

new/usr/src/uts/common/io/ixgbe/ixgbe_82598.h

1

```
*****
2755 Fri Jul 20 10:59:30 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_82598.h
XXX Intel X540 support
*****
1 /*****
3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_82598.h,v 1.2 2012/07/05 20:51:44 jfv Exp $*/
34
35 #ifndef _IXGBE_82598_H_
36 #define _IXGBE_82598_H_
37
38 u32 ixgbe_get_pcie_msix_count_82598(struct ixgbe_hw *hw);
39 s32 ixgbe_fc_enable_82598(struct ixgbe_hw *hw);
40 s32 ixgbe_start_hw_82598(struct ixgbe_hw *hw);
41 void ixgbe_enable_relaxed_ordering_82598(struct ixgbe_hw *hw);
42 s32 ixgbe_set_vmdq_82598(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
43 s32 ixgbe_set_vfta_82598(struct ixgbe_hw *hw, u32 vlan, u32 vind, bool vlan_on);
44 s32 ixgbe_read_analog_reg8_82598(struct ixgbe_hw *hw, u32 reg, u8 *val);
45 s32 ixgbe_write_analog_reg8_82598(struct ixgbe_hw *hw, u32 reg, u8 val);
46 s32 ixgbe_read_i2c_eeprom_82598(struct ixgbe_hw *hw, u8 byte_offset,
47 u8 *eeprom_data);
48 u32 ixgbe_get_supported_physical_layer_82598(struct ixgbe_hw *hw);
49 s32 ixgbe_init_phy_ops_82598(struct ixgbe_hw *hw);
50 void ixgbe_set_lan_id_multi_port_pcie_82598(struct ixgbe_hw *hw);
51 void ixgbe_set_pcie_completion_timeout(struct ixgbe_hw *hw);
52 #endif /* _IXGBE_82598_H_ */
```

new/usr/src/uts/common/io/ixgbe/ixgbe_82599.c

1

```
*****
69464 Fri Jul 20 10:59:31 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_82599.c
XXX Intel X540 support
*****
1 /*****
3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_82599.c,v 1.8 2012/07/05 20:51:44 jfv Exp $*/
34 /*$FreeBSD$*/
35 #include "ixgbe_type.h"
36 #include "ixgbe_82599.h"
37 #include "ixgbe_api.h"
38 #include "ixgbe_common.h"
39 #include "ixgbe_phy.h"
40 s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw);
41 s32 ixgbe_get_link_capabilities_82599(struct ixgbe_hw *hw,
42 ixgbe_link_speed *speed,
43 bool *autoneg);
44 enum ixgbe_media_type ixgbe_get_media_type_82599(struct ixgbe_hw *hw);
45 void ixgbe_disable_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
46 void ixgbe_enable_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
47 void ixgbe_flap_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
48 s32 ixgbe_setup_mac_link_multispeed_fiber(struct ixgbe_hw *hw,
49 ixgbe_link_speed speed, bool autoneg,
50 bool autoneg_wait_to_complete);
51 s32 ixgbe_setup_mac_link_smartspeed(struct ixgbe_hw *hw,
52 ixgbe_link_speed speed, bool autoneg,
53 bool autoneg_wait_to_complete);
54 s32 ixgbe_start_mac_link_82599(struct ixgbe_hw *hw,
55 bool autoneg_wait_to_complete);
56 s32 ixgbe_setup_mac_link_82599(struct ixgbe_hw *hw,
57 ixgbe_link_speed speed,
58 bool autoneg,
59 bool autoneg_wait_to_complete);
```

new/usr/src/uts/common/io/ixgbe/ixgbe_82599.c

2

```
41 static s32 ixgbe_setup_copper_link_82599(struct ixgbe_hw *hw,
42 ixgbe_link_speed speed,
43 bool autoneg,
44 bool autoneg_wait_to_complete);
45
46 s32 ixgbe_setup_sfp_modules_82599(struct ixgbe_hw *hw);
47 void ixgbe_init_mac_link_ops_82599(struct ixgbe_hw *hw);
48 s32 ixgbe_reset_hw_82599(struct ixgbe_hw *hw);
49 s32 ixgbe_read_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 *val);
50 s32 ixgbe_write_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 val);
51 s32 ixgbe_start_hw_rev_1_82599(struct ixgbe_hw *hw);
52 s32 ixgbe_identify_phy_82599(struct ixgbe_hw *hw);
53 s32 ixgbe_init_phy_ops_82599(struct ixgbe_hw *hw);
54 u32 ixgbe_get_supported_physical_layer_82599(struct ixgbe_hw *hw);
55 s32 ixgbe_enable_rx_dma_82599(struct ixgbe_hw *hw, u32 regval);
56 static s32 ixgbe_verify_fw_version_82599(struct ixgbe_hw *hw);
57 static s32 ixgbe_read_eeprom_82599(struct ixgbe_hw *hw,
58 u16 offset, u16 *data);
59 static s32 ixgbe_read_eeprom_buffer_82599(struct ixgbe_hw *hw, u16 offset,
60 u16 words, u16 *data);
61 bool ixgbe_verify_lesm_fw_enabled_82599(struct ixgbe_hw *hw);
62
63 void ixgbe_init_mac_link_ops_82599(struct ixgbe_hw *hw)
64 {
65 struct ixgbe_mac_info *mac = &hw->mac;
66
67 DEBUGFUNC("ixgbe_init_mac_link_ops_82599");
68
69 /* enable the laser control functions for SFP+ fiber */
70 if (mac->ops.get_media_type(hw) == ixgbe_media_type_fiber) {
71 mac->ops.disable_tx_laser =
72 &ixgbe_disable_tx_laser_multispeed_fiber;
73 mac->ops.enable_tx_laser =
74 &ixgbe_enable_tx_laser_multispeed_fiber;
75 mac->ops.flap_tx_laser = &ixgbe_flap_tx_laser_multispeed_fiber;
76
77 } else {
78 mac->ops.disable_tx_laser = NULL;
79 mac->ops.enable_tx_laser = NULL;
80 mac->ops.flap_tx_laser = NULL;
81
82 }
83
84 if (hw->phy.multispeed_fiber) {
85 /* Set up dual speed SFP+ support */
86 mac->ops.setup_link = &ixgbe_setup_mac_link_multispeed_fiber;
87 } else {
88 if ((ixgbe_get_media_type(hw) == ixgbe_media_type_backplane) &&
89 (hw->phy.smart_speed == ixgbe_smart_speed_auto ||
90 hw->phy.smart_speed == ixgbe_smart_speed_on) &&
91 !ixgbe_verify_lesm_fw_enabled_82599(hw)) {
92 mac->ops.setup_link = &ixgbe_setup_mac_link_smartspeed;
93 } else {
94 mac->ops.setup_link = &ixgbe_setup_mac_link_82599;
95 }
96 }
97
98 /**
99 * ixgbe_init_phy_ops_82599 - PHY/SFP specific init
100 * @hw: pointer to hardware structure
101 *
102 * Initialize any function pointers that were not able to be
103 * set during init_shared_code because the PHY/SFP type was
104 * not known. Perform the SFP init if necessary.
105 */
```

```

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;
128     case ixgbe_phy_aq:
129         phy->ops.get_firmware_version =
130             &ixgbe_get_phy_firmware_version_generic;
131         break;
132     default:
133         break;
134     }

135     init_phy_ops_out:
136     return ret_val;
137 }

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

144     DEBUGFUNC("ixgbe_setup_sfp_modules_82599");

145     if (hw->phy.sfp_type != ixgbe_sfp_type_unknown) {
146         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);
187     if (ret_val != IXGBE_SUCCESS) {
188         ret_val = IXGBE_ERR_SWFW_SYNC;
189         goto setup_sfp_out;
190     }

192     hw->eeprom.ops.read(hw, ++data_offset, &data_value);
193     while (data_value != 0xffff) {
194         IXGBE_WRITE_REG(hw, IXGBE_CORECTL, data_value);
195         IXGBE_WRITE_FLUSH(hw);
196         hw->eeprom.ops.read(hw, ++data_offset, &data_value);
197     }

199     /* Release the semaphore */
200     hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_MAC_CSR_SM);
201     ixgbe_release_swfw_sync(hw, IXGBE_GSSR_MAC_CSR_SM);
202     /* Delay obtaining semaphore again to allow FW access */
203     msec_delay(hw->eeprom.semaphore_delay);

205     /* Now restart DSP by setting Restart_AN and clearing LMS */
206     IXGBE_WRITE_REG(hw, IXGBE_AUTOC, ((IXGBE_READ_REG(hw,
207                                                         IXGBE_AUTOC) & ~IXGBE_AUTOC_LMS_MASK) |
208                                         IXGBE_AUTOC_AN_RESTART));

210     /* Wait for AN to leave state 0 */
211     for (i = 0; i < 10; i++) {
212         msec_delay(4);
213         reg_anlp1 = IXGBE_READ_REG(hw, IXGBE_ANLP1);
214         if (reg_anlp1 & IXGBE_ANLP1_AN_STATE_MASK)
215             break;
216     }
217     if (!(reg_anlp1 & IXGBE_ANLP1_AN_STATE_MASK)) {
218         DEBUGOUT("sfp module setup not complete\n");
219         ret_val = IXGBE_ERR_SFP_SETUP_NOT_COMPLETE;
220         goto setup_sfp_out;
221     }

223     /* Restart DSP by setting Restart_AN and return to SFI mode */
224     IXGBE_WRITE_REG(hw, IXGBE_AUTOC, (IXGBE_READ_REG(hw,
225                                                         IXGBE_AUTOC) | IXGBE_AUTOC_LMS_10G_SERIAL |
226                                         IXGBE_AUTOC_AN_RESTART));
227 }

229 setup_sfp_out:
230     return ret_val;
231 }

233 /**
234  * ixgbe_init_ops_82599 - Inits func ptrs and MAC type
235  * @hw: pointer to hardware structure
236  *
237  * Initialize the function pointers and assign the MAC type for 82599.
238  * Does not touch the hardware.
239  */

241 s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw)
242 {
243     struct ixgbe_mac_info *mac = &hw->mac;
244     struct ixgbe_phy_info *phy = &hw->phy;
245     struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
246     s32 ret_val;

248     DEBUGFUNC("ixgbe_init_ops_82599");

250     ret_val = ixgbe_init_phy_ops_generic(hw);
251     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;

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;
238 mac->ops.start_hw = &ixgbe_start_hw_rev_1_82599;
239 mac->ops.get_san_mac_addr = &ixgbe_get_san_mac_addr_generic;
240 mac->ops.set_san_mac_addr = &ixgbe_set_san_mac_addr_generic;
241 mac->ops.get_device_caps = &ixgbe_get_device_caps_generic;
242 mac->ops.get_wwn_prefix = &ixgbe_get_wwn_prefix_generic;
243 mac->ops.get_fcoe_boot_status = &ixgbe_get_fcoe_boot_status_generic;

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;

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);

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);

272 mac->arc_subsystem_valid = (IXGBE_READ_REG(hw, IXGBE_FWSM) &
273     IXGBE_FWSM_MODE_MASK) ? TRUE : FALSE;

275 hw->mbx.ops.init_params = ixgbe_init_mbx_params_pf;

277 /* EEPROM */
278 eeprom->ops.read = &ixgbe_read_eeprom_82599;
279 eeprom->ops.read_buffer = &ixgbe_read_eeprom_buffer_82599;

281 /* Manageability interface */
282 mac->ops.set_fw_drv_ver = &ixgbe_set_fw_drv_ver_generic;

285 return ret_val;

```

```

286 }

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;

303     DEBUGFUNC("ixgbe_get_link_capabilities_82599");

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) {
311         *speed = IXGBE_LINK_SPEED_1GB_FULL;
312         *negotiation = TRUE;
313         goto out;
314     }

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);

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;

332     case IXGBE_AUTOC_LMS_10G_LINK_NO_AN:
333         *speed = IXGBE_LINK_SPEED_10GB_FULL;
334         *negotiation = FALSE;
335         break;

337     case IXGBE_AUTOC_LMS_1G_AN:
338         *speed = IXGBE_LINK_SPEED_1GB_FULL;
339         *negotiation = TRUE;
340         break;

342     case IXGBE_AUTOC_LMS_10G_SERIAL:
343         *speed = IXGBE_LINK_SPEED_10GB_FULL;
344         *negotiation = FALSE;
345         break;

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:
406     case ixgbe_phy_aq:
407         media_type = ixgbe_media_type_copper;
408         goto out;
409     default:
410         break;
411     }

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);
957     (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 {

```

```

971     ixgbe_link_speed link_speed;
972     s32 status;
973     u32 ctrl, i, autoc, autoc2;
974     bool link_up = FALSE;
981     s32 status = IXGBE_SUCCESS;
982     u32 ctrl;
983     u32 i;
984     u32 autoc;
985     u32 autoc2;

976     DEBUGFUNC("ixgbe_reset_hw_82599");

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

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

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

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

991     if (status == IXGBE_ERR_SFP_NOT_SUPPORTED)
992         goto reset_hw_out;

994     /* Setup SFP module if there is one present. */
995     if (hw->phy.sfp_setup_needed) {
996         status = hw->mac.ops.setup_sfp(hw);
997         hw->phy.sfp_setup_needed = FALSE;
998     }

1000     if (status == IXGBE_ERR_SFP_NOT_SUPPORTED)
1001         goto reset_hw_out;

1003     /* Reset PHY */
1004     if (hw->phy.reset_disable == FALSE && hw->phy.ops.reset != NULL)
1005         hw->phy.ops.reset(hw);

1007 mac_reset_top:
1008     /*
1009      * Issue global reset to the MAC. Needs to be SW reset if link is up.
1010      * If link reset is used when link is up, it might reset the PHY when
1011      * mng is using it. If link is down or the flag to force full link
1012      * reset is set, then perform link reset.
1013      * Prevent the PCI-E bus from hanging by disabling PCI-E master
1014      * access and verify no pending requests before reset
1015      */
1016     ctrl = IXGBE_CTRL_LNK_RST;
1017     if (!hw->force_full_reset) {
1018         hw->mac.ops.check_link(hw, &link_speed, &link_up, FALSE);
1019         if (link_up)
1020             ctrl = IXGBE_CTRL_RST;
1021     }
1022     (void) ixgbe_disable_pcie_master(hw);

1021     ctrl |= IXGBE_READ_REG(hw, IXGBE_CTRL);
1022     IXGBE_WRITE_REG(hw, IXGBE_CTRL, ctrl);
1019 mac_reset_top:
1020     /*
1021      * Issue global reset to the MAC. This needs to be a SW reset.
1022      * If link reset is used, it might reset the MAC when mng is using it
1023      */

```

```

1024     ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
1025     IXGBE_WRITE_REG(hw, IXGBE_CTRL, (ctrl | IXGBE_CTRL_RST));
1023     IXGBE_WRITE_FLUSH(hw);

1025     /* Poll for reset bit to self-clear indicating reset is complete */
1026     for (i = 0; i < 10; i++) {
1027         usec_delay(1);
1028         ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
1029         if (!(ctrl & IXGBE_CTRL_RST_MASK))
1030             if (!(ctrl & IXGBE_CTRL_RST))
1031                 break;

1033     if (ctrl & IXGBE_CTRL_RST_MASK) {
1034         if (ctrl & IXGBE_CTRL_RST) {
1035             status = IXGBE_ERR_RESET_FAILED;
1036             DEBUGOUT("Reset polling failed to complete.\n");
1037         }
1038     }

1038     msec_delay(50);

1040     /*
1041      * Double resets are required for recovery from certain error
1042      * conditions. Between resets, it is necessary to stall to allow time
1043      * for any pending HW events to complete. We use 1usec since that is
1044      * what is needed for ixgbe_disable_pcie_master(). The second reset
1045      * then clears out any effects of those events.
1046      */
1047     if (hw->mac.flags & IXGBE_FLAGS_DOUBLE_RESET_REQUIRED) {
1048         hw->mac.flags &= ~IXGBE_FLAGS_DOUBLE_RESET_REQUIRED;
1049         usec_delay(1);
1050         goto mac_reset_top;
1051     }

1053     msec_delay(50);

1050     /*
1051      * Store the original AUTOC/AUTOC2 values if they have not been
1052      * stored off yet. Otherwise restore the stored original
1053      * values since the reset operation sets back to defaults.
1054      */
1055     autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
1056     autoc2 = IXGBE_READ_REG(hw, IXGBE_AUTOC2);
1057     if (hw->mac.orig_link_settings_stored == FALSE) {
1058         hw->mac.orig_autoc = autoc;
1059         hw->mac.orig_autoc2 = autoc2;
1060         hw->mac.orig_link_settings_stored = TRUE;
1061     } else {
1062         if (autoc != hw->mac.orig_autoc)
1063             IXGBE_WRITE_REG(hw, IXGBE_AUTOC, (hw->mac.orig_autoc |
1064                 IXGBE_AUTOC_AN_RESTART));

1066         if ((autoc2 & IXGBE_AUTOC2_UPPER_MASK) !=
1067             (hw->mac.orig_autoc2 & IXGBE_AUTOC2_UPPER_MASK)) {
1068             autoc2 &= ~IXGBE_AUTOC2_UPPER_MASK;
1069             autoc2 |= (hw->mac.orig_autoc2 &
1070                 IXGBE_AUTOC2_UPPER_MASK);
1071             IXGBE_WRITE_REG(hw, IXGBE_AUTOC2, autoc2);
1072         }
1073     }

1075     /* Store the permanent mac address */
1076     hw->mac.ops.get_mac_addr(hw, hw->mac.perm_addr);

1078     /*

```

```

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");
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);
1182      (void) IXGBE_READ_REG(hw, IXGBE_FDIRUSTAT);
1183      (void) IXGBE_READ_REG(hw, IXGBE_FDIRFSTAT);
1184      (void) IXGBE_READ_REG(hw, IXGBE_FDIRMATCH);
1185      (void) IXGBE_READ_REG(hw, IXGBE_FDIRMISS);
1186      (void) IXGBE_READ_REG(hw, IXGBE_FDIRLEN);

1188      return IXGBE_SUCCESS;
1189 }

1190 /**
1191  * ixgbe_fdir_enable_82599 - Initialize Flow Director control registers
1192  * ixgbe_init_fdir_signature_82599 - Initialize Flow Director signature filters
1193  * @hw: pointer to hardware structure
1194  * @fdirctrl: value to write to flow director control register
1195  * @pballocc: which mode to allocate filters with
1196  */
1197 static void ixgbe_fdir_enable_82599(struct ixgbe_hw *hw, u32 fdirctrl)
1198 {
1199     u32 fdirctrl = 0;
1200     u32 pbsize;
1201     int i;

1202     /*
1203      * 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_PBALLLOC_SIZE_SHIFT + pballloc));
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       * initialized 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 (pballloc) {
1226      case IXGBE_FDIR_PBALLLOC_64K:
1227        /* 8k - 1 signature filters */
1228        fdirctrl |= IXGBE_FDIRCTRL_PBALLLOC_64K;
1229        break;
1230      case IXGBE_FDIR_PBALLLOC_128K:
1231        /* 16k - 1 signature filters */
1232        fdirctrl |= IXGBE_FDIRCTRL_PBALLLOC_128K;
1233        break;
1234      case IXGBE_FDIR_PBALLLOC_256K:
1235        /* 32k - 1 signature filters */
1236        fdirctrl |= IXGBE_FDIRCTRL_PBALLLOC_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: PBALLOCC = 11b, timing is 60us
1204       * 1G: PBALLOCC = 11b, timing is 600us
1205       * 100M: PBALLOCC = 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");
1225        DEBUGOUT("Flow Director Signature poll time exceeded!\n");

1227        return IXGBE_SUCCESS;
1228      }

1229      /*
1230       * ixgbe_init_fdir_signature_82599 - Initialize Flow Director signature filters
1231       * ixgbe_init_fdir_perfect_82599 - Initialize Flow Director perfect filters
1232       * @hw: pointer to hardware structure
1233       * @fdirctrl: value to write to flow director control register, initially
1234       * contains just the value of the Rx packet buffer allocation
1235       * @pballloc: which mode to allocate filters with
1236       */
1237      s32 ixgbe_init_fdir_signature_82599(struct ixgbe_hw *hw, u32 fdirctrl)
1238      s32 ixgbe_init_fdir_perfect_82599(struct ixgbe_hw *hw, u32 pballloc)
1239      {
1240        DEBUGFUNC("ixgbe_init_fdir_signature_82599");
1241        u32 fdirctrl = 0;
1242        u32 pbsize;
1243        int i;

1245        DEBUGFUNC("ixgbe_init_fdir_perfect_82599");

1247        /*
1248         * Continue setup of fdirctrl register bits:
1249         * Move the flexible bytes to use the ethertype - shift 6 words
1250         * Set the maximum length per hash bucket to 0xA filters
1251         * Send interrupt when 64 filters are left
1252         * Before enabling Flow Director, the Rx Packet Buffer size
1253         * must be reduced. The new value is the current size minus
1254         * flow director memory usage size.
1255         */
1256        fdirctrl |= (0x6 << IXGBE_FDIRCTRL_FLEX_SHIFT) |
1257          (0xA << IXGBE_FDIRCTRL_MAX_LENGTH_SHIFT) |
1258          (4 << IXGBE_FDIRCTRL_FULL_THRESH_SHIFT);
1259        pbsize = (1 << (IXGBE_FDIR_PBALLLOC_SIZE_SHIFT + pballloc));
1260        IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(0),
1261          (IXGBE_READ_REG(hw, IXGBE_RXPBSIZE(0)) - pbsize));

1263        /* write hashes and fdirctrl register, poll for completion */
1264        ixgbe_fdir_enable_82599(hw, fdirctrl);

1266        /*
1267         * The defaults in the HW for RX PB 1-7 are not zero and so should be
1268         * initialized to zero for non DCB mode otherwise actual total RX PB
1269         * would be bigger than programmed and filter space would run into
1270         * the PB 0 region.
1271         */
1272        for (i = 1; i < 8; i++)
1273          IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), 0);

1275        /* Send interrupt when 64 filters are left */
1276        fdirctrl |= 4 << IXGBE_FDIRCTRL_FULL_THRESH_SHIFT;

1278        /* Initialize the drop queue to Rx queue 127 */
1279        fdirctrl |= (127 << IXGBE_FDIRCTRL_DROP_Q_SHIFT);

1281        switch (pballloc) {
1282        case IXGBE_FDIR_PBALLLOC_64K:
1283          /* 2k - 1 perfect filters */
1284          fdirctrl |= IXGBE_FDIRCTRL_PBALLLOC_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");

1372     return IXGBE_SUCCESS;
1373 }

1375 /**
1376  * ixgbe_init_fdir_perfect_82599 - Initialize Flow Director perfect filters
1377  * @hw: pointer to hardware structure
1378  * @fdirctrl: value to write to flow director control register, initially
1379  *   contains just the value of the Rx packet buffer allocation
1380  * @stream: input bitstream to compute the hash on
1381  * @key: 32-bit hash key
1382  */
1383 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)
1382 {
1383     DEBUGFUNC("ixgbe_init_fdir_perfect_82599");
1384     /*
1385      * The algorithm is as follows:
1386      *   Hash[15:0] = Sum { S[n] x K[n+16] }, n = 0...350
1387      *   where Sum {A[n]}, n = 0...n is bitwise XOR of A[0], A[1]...A[n]
1388      *   and A[n] x B[n] is bitwise AND between same length strings
1389      *
1390      * K[n] is 16 bits, defined as:
1391      *   for n modulo 32 >= 15, K[n] = K[n % 32 : (n % 32) - 15]
1392      *   for n modulo 32 < 15, K[n] =
1393      *       K[(n % 32:0) | (31:31 - (14 - (n % 32)))]
1394      *
1395      * S[n] is 16 bits, defined as:
1396      *   for n >= 15, S[n] = S[n:n - 15]
1397      *   for n < 15, S[n] = S[(n:0) | (350:350 - (14 - n))]
1398      *
1399      * To simplify for programming, the algorithm is implemented
1400      * in software this way:
1401      *
1402      * key[31:0], hi_hash_dword[31:0], lo_hash_dword[31:0], hash[15:0]
1403      *
1404      * for (i = 0; i < 352; i+=32)
1405      *     hi_hash_dword[31:0] ^= Stream[(i+31):i];
1406      *
1407      * lo_hash_dword[15:0] ^= Stream[15:0];
1408      * lo_hash_dword[15:0] ^= hi_hash_dword[31:16];
1409      * lo_hash_dword[31:16] ^= hi_hash_dword[15:0];
1410      *
1411      * hi_hash_dword[31:0] ^= Stream[351:320];
1412      *
1413      * if(key[0])
1414      *     hash[15:0] ^= Stream[15:0];
1415      *
1416      * for (i = 0; i < 16; i++) {
1417      *     if (key[i])
1418      *         hash[15:0] ^= lo_hash_dword[(i+15):i];
1419      *         if (key[i + 16])
1420      *             hash[15:0] ^= hi_hash_dword[(i+15):i];
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 optimizations 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

```

```

1378 * @input: unique input dword
1379 * @common: compressed common input dword
1380 * @stream: input bitstream
1381 * @queue: queue index to direct traffic to
1382 */
1383 s32 ixgbe_fdir_add_signature_filter_82599(struct ixgbe_hw *hw,
1384                                         union ixgbe_atr_hash_dword input,
1385                                         union ixgbe_atr_hash_dword common,
1386                                         u8 queue)
1387 {
1388     u64 fdirhashcmd;
1389     u32 fdircmd;
1390
1391     DEBUGFUNC("ixgbe_fdir_add_signature_filter_82599");
1392
1393     /*
1394      * Get the flow_type in order to program FDIRCMD properly
1395      * lowest 2 bits are FDIRCMD.L4TYPE, third lowest bit is FDIRCMD.IPV6
1396      */
1397     switch (input.formatted.flow_type) {
1398     case IXGBE_ATR_FLOW_TYPE_TCPV4:
1399     case IXGBE_ATR_FLOW_TYPE_UDPV4:
1400     case IXGBE_ATR_FLOW_TYPE_SCTPV4:
1401     case IXGBE_ATR_FLOW_TYPE_TCPV6:
1402     case IXGBE_ATR_FLOW_TYPE_UDPV6:
1403     case IXGBE_ATR_FLOW_TYPE_SCTPV6:
1404         break;
1405     default:
1406         DEBUGOUT(" Error on flow type input\n");
1407         return IXGBE_ERR_CONFIG;
1408     }
1409
1410     /* configure FDIRCMD register */
1411     fdircmd = IXGBE_FDIRCMD_CMD_ADD_FLOW | IXGBE_FDIRCMD_FILTER_UPDATE |
1412             IXGBE_FDIRCMD_LAST | IXGBE_FDIRCMD_QUEUE_EN;
1413     fdircmd |= input.formatted.flow_type << IXGBE_FDIRCMD_FLOW_TYPE_SHIFT;
1414     fdircmd |= (u32)queue << IXGBE_FDIRCMD_RX_QUEUE_SHIFT;
1415
1416     /*
1417      * The lower 32-bits of fdirhashcmd is for FDIRHASH, the upper 32-bits
1418      * is for FDIRCMD. Then do a 64-bit register write from FDIRHASH.
1419      */
1420     fdirhashcmd = (u64)fdircmd << 32;
1421     fdirhashcmd |= ixgbe_atr_compute_sig_hash_82599(input, common);
1422     IXGBE_WRITE_REG64(hw, IXGBE_FDIRHASH, fdirhashcmd);
1423
1424     DEBUGOUT2("Tx Queue=%x hash=%x\n", queue, (u32)fdirhashcmd);
1425
1426     return IXGBE_SUCCESS;
1427 }
1428
1429 #define IXGBE_COMPUTE_BKT_HASH_ITERATION(_n) \
1430 do { \
1431     u32 n = (_n); \
1432     if (IXGBE_ATR_BUCKET_HASH_KEY & (0x01 << n)) \
1433         bucket_hash ^= lo_hash_dword >> n; \
1434     if (IXGBE_ATR_BUCKET_HASH_KEY & (0x01 << (n + 16))) \
1435         bucket_hash ^= hi_hash_dword >> n; \
1436 } while (0);
1437
1438 /**
1439 * ixgbe_atr_compute_perfect_hash_82599 - Compute the perfect filter hash
1440 * @atr_input: input bitstream to compute the hash on
1441 * @input_mask: mask for the input bitstream
1442 * This function serves two main purposes. First it applies the input_mask

```

```

1443 * to the atr_input resulting in a cleaned up atr_input data stream.
1444 * Secondly it computes the hash and stores it in the bkt_hash field at
1445 * the end of the input byte stream. This way it will be available for
1446 * future use without needing to recompute the hash.
1447 */
1448 void ixgbe_atr_compute_perfect_hash_82599(union ixgbe_atr_input *input,
1449                                           union ixgbe_atr_input *input_mask)
1450 {
1451     u32 hi_hash_dword, lo_hash_dword, flow_vm_vlan;
1452     u32 bucket_hash = 0;
1453
1454     /* Apply masks to input data */
1455     input->dword_stream[0] &= input_mask->dword_stream[0];
1456     input->dword_stream[1] &= input_mask->dword_stream[1];
1457     input->dword_stream[2] &= input_mask->dword_stream[2];
1458     input->dword_stream[3] &= input_mask->dword_stream[3];
1459     input->dword_stream[4] &= input_mask->dword_stream[4];
1460     input->dword_stream[5] &= input_mask->dword_stream[5];
1461     input->dword_stream[6] &= input_mask->dword_stream[6];
1462     input->dword_stream[7] &= input_mask->dword_stream[7];
1463     input->dword_stream[8] &= input_mask->dword_stream[8];
1464     input->dword_stream[9] &= input_mask->dword_stream[9];
1465     input->dword_stream[10] &= input_mask->dword_stream[10];
1466
1467     /* record the flow_vm_vlan bits as they are a key part to the hash */
1468     flow_vm_vlan = IXGBE_NTOHL(input->dword_stream[0]);
1469
1470     /* generate common hash dword */
1471     hi_hash_dword = IXGBE_NTOHL(input->dword_stream[1] ^
1472                                 input->dword_stream[2] ^
1473                                 input->dword_stream[3] ^
1474                                 input->dword_stream[4] ^
1475                                 input->dword_stream[5] ^
1476                                 input->dword_stream[6] ^
1477                                 input->dword_stream[7] ^
1478                                 input->dword_stream[8] ^
1479                                 input->dword_stream[9] ^
1480                                 input->dword_stream[10]);
1481
1482     /* low dword is word swapped version of common */
1483     lo_hash_dword = (hi_hash_dword >> 16) | (hi_hash_dword << 16);
1484
1485     /* apply flow ID/VM pool/VLAN ID bits to hash words */
1486     hi_hash_dword ^= flow_vm_vlan ^ (flow_vm_vlan >> 16);
1487
1488     /* Process bits 0 and 16 */
1489     IXGBE_COMPUTE_BKT_HASH_ITERATION(0);
1490
1491     /*
1492      * apply flow ID/VM pool/VLAN ID bits to lo hash dword, we had to
1493      * delay this because bit 0 of the stream should not be processed
1494      * so we do not add the vlan until after bit 0 was processed
1495      */
1496     lo_hash_dword ^= flow_vm_vlan ^ (flow_vm_vlan << 16);
1497
1498     /* Process remaining 30 bit of the key */
1499     IXGBE_COMPUTE_BKT_HASH_ITERATION(1);
1500     IXGBE_COMPUTE_BKT_HASH_ITERATION(2);
1501     IXGBE_COMPUTE_BKT_HASH_ITERATION(3);
1502     IXGBE_COMPUTE_BKT_HASH_ITERATION(4);
1503     IXGBE_COMPUTE_BKT_HASH_ITERATION(5);
1504     IXGBE_COMPUTE_BKT_HASH_ITERATION(6);
1505     IXGBE_COMPUTE_BKT_HASH_ITERATION(7);
1506     IXGBE_COMPUTE_BKT_HASH_ITERATION(8);
1507     IXGBE_COMPUTE_BKT_HASH_ITERATION(9);
1508

```

```

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)
1533 static u32 ixgbe_get_fdirtcpm_82599(struct ixgbe_atr_input_masks *input_masks)
1534 {
1535     u32 mask = IXGBE_NTOHS(input_mask->formatted.dst_port);
1536     u32 mask = IXGBE_NTOHS(input_masks->dst_port_mask);
1537     mask <= IXGBE_FDIRTCPM_DPORTM_SHIFT;
1538     mask |= IXGBE_NTOHS(input_mask->formatted.src_port);
1539     mask |= IXGBE_NTOHS(input_masks->src_port_mask);
1540     mask = ((mask & 0x55555555) << 1) | ((mask & 0xAAAAAAAA) >> 1);
1541     mask = ((mask & 0x33333333) << 2) | ((mask & 0xCCCCCCCC) >> 2);
1542     mask = ((mask & 0x0F0F0F0F) << 4) | ((mask & 0xFF0F0F0F) >> 4);
1543     return ((mask & 0x00FF00FF) << 8) | ((mask & 0xFF00FF00) >> 8);
1544 }

1546 /*
1547  * These two macros are meant to address the fact that we have registers
1548  * that are either all or in part big-endian. As a result on big-endian
1549  * systems we will end up byte swapping the value to little-endian before
1550  * it is byte swapped again and written to the hardware in the original
1551  * big-endian format.
1552  */
1553 #define IXGBE_STORE_AS_BE32(_value) \
1554     (((u32)(_value) >> 24) | (((u32)(_value) & 0x00FF0000) >> 8) | \
1555      (((u32)(_value) & 0x0000FF00) << 8) | ((u32)(_value) << 24))

1557 #define IXGBE_WRITE_REG_BE32(a, reg, value) \
1558     IXGBE_WRITE_REG((a), (reg), IXGBE_STORE_AS_BE32(IXGBE_NTOHL(value)))

1560 #define IXGBE_STORE_AS_BE16(_value) \
1561     IXGBE_NTOHS(((u16)(_value) >> 8) | ((u16)(_value) << 8))
1562     (((u16)(_value) >> 8) | ((u16)(_value) << 8))

1564 s32 ixgbe_fdir_set_input_mask_82599(struct ixgbe_hw *hw,
1565     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)
1660 {
1661     /* mask IPv6 since it is currently not supported */
1662     u32 fdir = IXGBE_FDIRM_DIPv6;
1663     u32 fdirtcpm;
1664     u32 fdirhash;
1665     u32 fdircmd;
1666     u32 fdirport, fdirtcpm;
1667     u32 fdirvlan;
1668     /* start with VLAN, flex bytes, VM pool, and IPv6 destination masked */
1669     u32 fdir = IXGBE_FDIRM_VLANID | IXGBE_FDIRM_VLANP | IXGBE_FDIRM_FLEX |
1670         IXGBE_FDIRM_POOL | IXGBE_FDIRM_DIPv6;

1672     DEBUGFUNC("ixgbe_fdir_set_atr_input_mask_82599");
1673     DEBUGFUNC("ixgbe_fdir_add_perfect_filter_82599");

1675     /*
1676      * Check flow_type formatting, and bail out before we touch the hardware
1677      * if there's a configuration issue
1678      */
1679     switch (input->formatted.flow_type) {
1680     case IXGBE_ATR_FLOW_TYPE_IPV4:
1681         /* use the L4 protocol mask for raw IPv4/IPv6 traffic */
1682         fdir |= IXGBE_FDIRM_L4P;
1683         /* FALLTHRU */
1684     case IXGBE_ATR_FLOW_TYPE_SCTPV4:
1685         if (input_masks->dst_port_mask || input_masks->src_port_mask) {
1686             DEBUGOUT(" Error on src/dst port mask\n");
1687             return IXGBE_ERR_CONFIG;
1688         }
1689         break;
1690     case IXGBE_ATR_FLOW_TYPE_TCPV4:
1691         break;
1692     case IXGBE_ATR_FLOW_TYPE_UDPV4:
1693         break;
1694     default:
1695         DEBUGOUT(" Error on flow type input\n");
1696         return IXGBE_ERR_CONFIG;
1697     }

1699     /*
1700      * Program the relevant mask registers. If src/dst_port or src/dst_addr
1701      * are zero, then assume a full mask for that field. Also assume that
1702      * a VLAN of 0 is unspecified, so mask that out as well. L4type
1703      * cannot be masked out in this implementation.
1704      *
1705      * This also assumes IPv4 only. IPv6 masking isn't supported at this
1706      * point in time.
1707      */

1709     /* verify bucket hash is cleared on hash generation */
1710     if (input_mask->formatted.bkt_hash)
1711         DEBUGOUT(" bucket hash should always be 0 in mask\n");

1713     /* Program FDIRM and verify partial masks */
1714     switch (input_mask->formatted.vm_pool & 0x7F) {
1715     case 0x0:
1716         fdir |= IXGBE_FDIRM_POOL;
1717     case 0x7F:
1718         /* 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;
1714         break;
1715     default:
1716         DEBUGOUT(" Error on vm pool mask\n");
1717         return IXGBE_ERR_CONFIG;
1718     }
1719
1720     switch (input_mask->formatted.flow_type & IXGBE_ATR_L4TYPE_MASK) {
1721     case 0x0:
1722         fdirm |= IXGBE_FDIRM_L4P;
1723         if (input_mask->formatted.dst_port ||
1724             input_mask->formatted.src_port) {
1725             DEBUGOUT(" Error on src/dst port mask\n");
1726             return IXGBE_ERR_CONFIG;
1727         }
1728     case IXGBE_ATR_L4TYPE_MASK:
1729     case 0xFFFF:
1730         /* Unmask VLAN ID - bit 0 */
1731         fdirm &= ~IXGBE_FDIRM_VLANID;
1732         break;
1733     default:
1734         DEBUGOUT(" Error on flow type mask\n");
1735         return IXGBE_ERR_CONFIG;
1736     }
1737
1738     switch (IXGBE_NTOHS(input_mask->formatted.vlan_id) & 0xEFFF) {
1739     case 0x0000:
1740         /* mask VLAN ID, fall through to mask VLAN priority */
1741         fdirm |= IXGBE_FDIRM_VLANID;
1742     case 0x0FFF:
1743         /* mask VLAN priority */
1744         fdirm |= IXGBE_FDIRM_VLANP;
1745         /* do nothing, vlans already masked */
1746         break;
1747     case 0xE000:
1748         /* mask VLAN ID only, fall through */
1749         fdirm |= IXGBE_FDIRM_VLANID;
1750     case 0xEFFF:
1751         /* no VLAN fields masked */
1752         break;
1753     default:
1754         DEBUGOUT(" Error on VLAN mask\n");
1755         return IXGBE_ERR_CONFIG;
1756     }
1757
1758     switch (input_mask->formatted.flex_bytes & 0xFFFF) {
1759     case 0x0000:
1760         /* Mask Flex Bytes, fall through */
1761         fdirm |= IXGBE_FDIRM_FLEX;
1762     case 0xFFFF:
1763         break;
1764     default:
1765         if (input_masks->flex_mask & 0xFFFF) {
1766             if ((input_masks->flex_mask & 0xFFFF) != 0xFFFF) {
1767                 DEBUGOUT(" Error on flexible byte mask\n");
1768                 return IXGBE_ERR_CONFIG;
1769             }
1770         }
1771         /* Unmask Flex Bytes - bit 4 */
1772         fdirm &= ~IXGBE_FDIRM_FLEX;

```

```

1734     }
1735
1736     /* Now mask VM pool and destination IPv6 - bits 5 and 2 */
1737     IXGBE_WRITE_REG(hw, IXGBE_FDIRM, fdirm);
1738
1739     /* store the TCP/UDP port masks, bit reversed from port layout */
1740     fdirtcpm = ixgbe_get_fdirtcpm_82599(input_mask);
1741     fdirtcpm = ixgbe_get_fdirtcpm_82599(input_masks);
1742
1743     /* write both the same so that UDP and TCP use the same mask */
1744     IXGBE_WRITE_REG(hw, IXGBE_FDIRTCPM, ~fdirtcpm);
1745     IXGBE_WRITE_REG(hw, IXGBE_FDIRUDPM, ~fdirtcpm);
1746
1747     /* store source and destination IP masks (big-endian) */
1748     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIP4M,
1749         ~input_mask->formatted.src_ip[0]);
1750     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRDIP4M,
1751         ~input_mask->formatted.dst_ip[0]);
1752     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIP4M,
1753         ~input_masks->src_ip_mask[0]);
1754     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRDIP4M,
1755         ~input_masks->dst_ip_mask[0]);
1756
1757     return IXGBE_SUCCESS;
1758 }
1759
1760 /* Apply masks to input data */
1761 input->formatted.vlan_id &= input_masks->vlan_id_mask;
1762 input->formatted.flex_bytes &= input_masks->flex_mask;
1763 input->formatted.src_port &= input_masks->src_port_mask;
1764 input->formatted.dst_port &= input_masks->dst_port_mask;
1765 input->formatted.src_ip[0] &= input_masks->src_ip_mask[0];
1766 input->formatted.dst_ip[0] &= input_masks->dst_ip_mask[0];
1767
1768 s32 ixgbe_fdir_write_perfect_filter_82599(struct ixgbe_hw *hw,
1769     union ixgbe_atr_input *input,
1770     u16 soft_id, u8 queue)
1771 {
1772     u32 fdirport, fdirvlan, fdirhash, fdircmd;
1773     /* record vlan (little-endian) and flex_bytes(big-endian) */
1774     fdirvlan =
1775         IXGBE_STORE_AS_BE16(IXGBE_NTOHS(input->formatted.flex_bytes));
1776     fdirvlan <= IXGBE_FDIRVLAN_FLEX_SHIFT;
1777     fdirvlan |= IXGBE_NTOHS(input->formatted.vlan_id);
1778     IXGBE_WRITE_REG(hw, IXGBE_FDIRVLAN, fdirvlan);
1779
1780     DEBUGFUNC("ixgbe_fdir_write_perfect_filter_82599");
1781
1782     /* currently IPv6 is not supported, must be programmed with 0 */
1783     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIPV6(0),
1784         input->formatted.src_ip[0]);
1785     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIPV6(1),
1786         input->formatted.src_ip[1]);
1787     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRSIPV6(2),
1788         input->formatted.src_ip[2]);
1789
1790     /* record the source address (big-endian) */
1791     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRIPSA, input->formatted.src_ip[0]);
1792
1793     /* record the first 32 bits of the destination address (big-endian) */
1794     IXGBE_WRITE_REG_BE32(hw, IXGBE_FDIRIPDA, input->formatted.dst_ip[0]);
1795
1796     /* record source and destination port (little-endian) */
1797     fdirport = IXGBE_NTOHS(input->formatted.dst_port);
1798     fdirport <= IXGBE_FDIRPORT_DESTINATION_SHIFT;
1799     fdirport |= IXGBE_NTOHS(input->formatted.src_port);
1800     IXGBE_WRITE_REG(hw, IXGBE_FDIRPORT, fdirport);
1801
1802     /* 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;
1728     /* we only want the bucket hash so drop the upper 16 bits */
1729     fdirhash = ixgbe_atr_compute_hash_82599(input,
1730                                             IXGBE_ATR_BUCKET_HASH_KEY);
1731     fdirhash |= soft_id << IXGBE_FDIRHASH_SIG_SW_INDEX_SHIFT;
1732     IXGBE_WRITE_REG(hw, IXGBE_FDIRHASH, fdirhash);

1734     /* flush hash to HW */
1735     IXGBE_WRITE_FLUSH(hw);

1737     /* Query if filter is present */
1738     IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD, IXGBE_FDIRCMD_CMD_QUERY_REM_FILT);

1739     for (retry_count = 10; retry_count; retry_count--) {
1740         /* allow 10us for query to process */
1741         usec_delay(10);
1742         /* verify query completed successfully */
1743         fdircmd = IXGBE_READ_REG(hw, IXGBE_FDIRCMD);
1744         if (!(fdircmd & IXGBE_FDIRCMD_CMD_MASK))
1745             break;
1746     }

```

```

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
1868  * ixgbe_start_hw_rev_1_82599 - Prepare hardware for Tx/Rx
1869  * @hw: pointer to hardware structure
1870  *
1871  * Starts the hardware using the generic start_hw function
1872  * and the generation start_hw function.
1873  * Then performs revision-specific operations, if any.
1874  */
1874 s32 ixgbe_start_hw_82599(struct ixgbe_hw *hw)
1875 s32 ixgbe_start_hw_rev_1_82599(struct ixgbe_hw *hw)
1876 {
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 {
2062     #define IXGBE_MAX_SECRX_POLL 30
2063     int i;
2064     int secrxreg;

2065     DEBUGFUNC("ixgbe_enable_rx_dma_82599");

2066     /*
2067      * Workaround for 82599 silicon errata when enabling the Rx datapath.
2068      * If traffic is incoming before we enable the Rx unit, it could hang
2069      * the Rx DMA unit. Therefore, make sure the security engine is
2070      * completely disabled prior to enabling the Rx unit.
2071      */
2072     secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXCTRL);
2073     secrxreg |= IXGBE_SECRXCTRL_RX_DIS;
2074     IXGBE_WRITE_REG(hw, IXGBE_SECRXCTRL, secrxreg);
2075     for (i = 0; i < IXGBE_MAX_SECRX_POLL; i++) {
2076         secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXSTAT);
2077         if (secrxreg & IXGBE_SECRXSTAT_SECRX_RDY)
2078             break;
2079         else
2080             /* Use interrupt-safe sleep just in case */
2081             usec_delay(10);
2082     }

2083     hw->mac.ops.disable_sec_rx_path(hw);
2084     /* For informational purposes only */
2085     if (i >= IXGBE_MAX_SECRX_POLL)
2086         DEBUGOUT("Rx unit being enabled before security ")

```

```

2064         "path fully disabled. Continuing with init.\n");

2074     IXGBE_WRITE_REG(hw, IXGBE_RXCTRL, regval);
2067     secrxreg = IXGBE_READ_REG(hw, IXGBE_SECRXCTRL);
2068     secrxreg &= ~IXGBE_SECRXCTRL_RX_DIS;
2069     IXGBE_WRITE_REG(hw, IXGBE_SECRXCTRL, secrxreg);
2070     IXGBE_WRITE_FLUSH(hw);

2076     hw->mac.ops.enable_sec_rx_path(hw);

2078     return IXGBE_SUCCESS;
2079 }

2081 /**
2082  * ixgbe_verify_fw_version_82599 - verify fw version for 82599
2083  * @hw: pointer to hardware structure
2084  *
2085  * Verifies that installed the firmware version is 0.6 or higher
2086  * for SFI devices. All 82599 SFI devices should have version 0.6 or higher.
2087  *
2088  * Returns IXGBE_ERR_EEPROM_VERSION if the FW is not present or
2089  * if the FW version is not supported.
2090  */
2091 static s32 ixgbe_verify_fw_version_82599(struct ixgbe_hw *hw)
2092 {
2093     s32 status = IXGBE_ERR_EEPROM_VERSION;
2094     u16 fw_offset, fw_ptp_cfg_offset;
2095     u16 fw_version = 0;

2097     DEBUGFUNC("ixgbe_verify_fw_version_82599");

2099     /* firmware check is only necessary for SFI devices */
2100     if (hw->phy.media_type != ixgbe_media_type_fiber) {
2101         status = IXGBE_SUCCESS;
2102         goto fw_version_out;
2103     }

2105     /* get the offset to the Firmware Module block */
2106     hw->eeprom.ops.read(hw, IXGBE_FW_PTR, &fw_offset);

2108     if ((fw_offset == 0) || (fw_offset == 0xFFFF))
2109         goto fw_version_out;

2111     /* get the offset to the Pass Through Patch Configuration block */
2112     hw->eeprom.ops.read(hw, (fw_offset +
2113         IXGBE_FW_PASSTHROUGH_PATCH_CONFIG_PTR),
2114         &fw_ptp_cfg_offset);

2116     if ((fw_ptp_cfg_offset == 0) || (fw_ptp_cfg_offset == 0xFFFF))
2117         goto fw_version_out;

2119     /* get the firmware version */
2120     hw->eeprom.ops.read(hw, (fw_ptp_cfg_offset +
2121         IXGBE_FW_PATCH_VERSION_4), &fw_version);
2115     IXGBE_FW_PATCH_VERSION_4,
2116     &fw_version);

2123     if (fw_version > 0x5)
2124         status = IXGBE_SUCCESS;

2126     fw_version_out:
2127     return status;
2128 }
2129
2130 unchanged portion omitted

2174 /**

```

```

2175  * ixgbe_read_eeprom_buffer_82599 - Read EEPROM word(s) using
2176  * fastest available method
2177  *
2178  * @hw: pointer to hardware structure
2179  * @offset: offset of word in EEPROM to read
2180  * @words: number of words
2181  * @data: word(s) read from the EEPROM
2182  *
2183  * Retrieves 16 bit word(s) read from EEPROM
2184  */
2185 static s32 ixgbe_read_eeprom_buffer_82599(struct ixgbe_hw *hw, u16 offset,
2186     u16 words, u16 *data)
2187 {
2188     struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
2189     s32 ret_val = IXGBE_ERR_CONFIG;

2191     DEBUGFUNC("ixgbe_read_eeprom_buffer_82599");

2193     /*
2194      * If EEPROM is detected and can be addressed using 14 bits,
2195      * use EERD otherwise use bit bang
2196      */
2197     if ((eeprom->type == ixgbe_eeprom_spi) &&
2198         (offset + (words - 1) <= IXGBE_EERD_MAX_ADDR))
2199         ret_val = ixgbe_read_eerd_buffer_generic(hw, offset, words,
2200             data);
2201     else
2202         ret_val = ixgbe_read_eeprom_buffer_bit_bang_generic(hw, offset,
2203             words, data);

2206     return ret_val;
2207 }

2209 /**
2210  * ixgbe_read_eeprom_82599 - Read EEPROM word using
2211  * fastest available method
2212  *
2213  * @hw: pointer to hardware structure
2214  * @offset: offset of word in the EEPROM to read
2215  * @data: word read from the EEPROM
2216  *
2217  * Reads a 16 bit word from the EEPROM
2218  */
2219 static s32 ixgbe_read_eeprom_82599(struct ixgbe_hw *hw,
2220     u16 offset, u16 *data)
2221 {
2222     struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
2223     s32 ret_val = IXGBE_ERR_CONFIG;

2225     DEBUGFUNC("ixgbe_read_eeprom_82599");

2227     /*
2228      * If EEPROM is detected and can be addressed using 14 bits,
2229      * use EERD otherwise use bit bang
2230      */
2231     if ((eeprom->type == ixgbe_eeprom_spi) &&
2232         (offset <= IXGBE_EERD_MAX_ADDR))
2233         ret_val = ixgbe_read_eerd_generic(hw, offset, data);
2234     else
2235         ret_val = ixgbe_read_eeprom_bit_bang_generic(hw, offset, data);

2237     return ret_val;
2238 }

```

new/usr/src/uts/common/io/ixgbe/ixgbe_82599.h

1

```
*****
3452 Fri Jul 20 10:59:32 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_82599.h
XXX Intel X540 support
*****
1 /*****
3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_82599.h,v 1.1 2012/01/30 16:42:02 jfv Exp $*
34
35 #ifndef _IXGBE_82599_H_
36 #define _IXGBE_82599_H_
37
38 s32 ixgbe_get_link_capabilities_82599(struct ixgbe_hw *hw,
39 ixgbe_link_speed *speed, bool *autoneg);
40 enum ixgbe_media_type ixgbe_get_media_type_82599(struct ixgbe_hw *hw);
41 void ixgbe_disable_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
42 void ixgbe_enable_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
43 void ixgbe_flap_tx_laser_multispeed_fiber(struct ixgbe_hw *hw);
44 s32 ixgbe_setup_mac_link_multispeed_fiber(struct ixgbe_hw *hw,
45 ixgbe_link_speed speed, bool autoneg,
46 bool autoneg_wait_to_complete);
47 s32 ixgbe_setup_mac_link_smartspeed(struct ixgbe_hw *hw,
48 ixgbe_link_speed speed, bool autoneg,
49 bool autoneg_wait_to_complete);
50 s32 ixgbe_start_mac_link_82599(struct ixgbe_hw *hw,
51 bool autoneg_wait_to_complete);
52 s32 ixgbe_setup_mac_link_82599(struct ixgbe_hw *hw, ixgbe_link_speed speed,
53 bool autoneg, bool autoneg_wait_to_complete);
54 s32 ixgbe_setup_sfp_modules_82599(struct ixgbe_hw *hw);
55 void ixgbe_init_mac_link_ops_82599(struct ixgbe_hw *hw);
56 s32 ixgbe_reset_hw_82599(struct ixgbe_hw *hw);
57 s32 ixgbe_read_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 *val);
58 s32 ixgbe_write_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 val);
59 s32 ixgbe_start_hw_82599(struct ixgbe_hw *hw);
60 s32 ixgbe_identify_phy_82599(struct ixgbe_hw *hw);
61 s32 ixgbe_init_phy_ops_82599(struct ixgbe_hw *hw);
```

new/usr/src/uts/common/io/ixgbe/ixgbe_82599.h

2

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

new/usr/src/uts/common/io/ixgbe/ixgbe_api.c

1

```
*****
35253 Fri Jul 20 10:59:32 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_api.c
XXX Intel X540 support
*****
1 /*****
3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_api.c,v 1.13 2012/07/05 20:51:44 jfv Exp $*/
34 /*$FreeBSD$*/
35 #include "ixgbe_api.h"
36 #include "ixgbe_common.h"
37
38 extern s32 ixgbe_init_ops_82598(struct ixgbe_hw *hw);
39 extern s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw);
40
41 /**
42 * ixgbe_init_shared_code - Initialize the shared code
43 * @hw: pointer to hardware structure
44 *
45 * This will assign function pointers and assign the MAC type and PHY code.
46 * Does not touch the hardware. This function must be called prior to any
47 * other function in the shared code. The ixgbe_hw structure should be
48 * memset to 0 prior to calling this function. The following fields in
49 * hw structure should be filled in prior to calling this function:
50 * hw_addr, back, device_id, vendor_id, subsystem_device_id,
51 * subsystem_vendor_id, and revision_id
52 */
53 s32 ixgbe_init_shared_code(struct ixgbe_hw *hw)
54 {
55     s32 status;
56
57     DEBUGFUNC("ixgbe_init_shared_code");
58
59     /*
60      * Set the mac type
61      */
62 }
```

new/usr/src/uts/common/io/ixgbe/ixgbe_api.c

2

```
58 */
59 ixgbe_set_mac_type(hw);
60 (void) ixgbe_set_mac_type(hw);
61
62 switch (hw->mac.type) {
63 case ixgbe_mac_82598EB:
64     status = ixgbe_init_ops_82598(hw);
65     break;
66 case ixgbe_mac_82599EB:
67     status = ixgbe_init_ops_82599(hw);
68     break;
69 case ixgbe_mac_X540:
70     status = ixgbe_init_ops_X540(hw);
71     break;
72 default:
73     status = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
74     break;
75 }
76
77 return status;
78 }
79
80 /**
81 * ixgbe_set_mac_type - Sets MAC type
82 * @hw: pointer to the HW structure
83 *
84 * This function sets the mac type of the adapter based on the
85 * vendor ID and device ID stored in the hw structure.
86 */
87 s32 ixgbe_set_mac_type(struct ixgbe_hw *hw)
88 {
89     s32 ret_val = IXGBE_SUCCESS;
90
91     DEBUGFUNC("ixgbe_set_mac_type\n");
92
93     if (hw->vendor_id == IXGBE_INTEL_VENDOR_ID) {
94         switch (hw->device_id) {
95             case IXGBE_DEV_ID_82598:
96             case IXGBE_DEV_ID_82598_BX:
97             case IXGBE_DEV_ID_82598AF_SINGLE_PORT:
98             case IXGBE_DEV_ID_82598AF_DUAL_PORT:
99             case IXGBE_DEV_ID_82598AT:
100             case IXGBE_DEV_ID_82598AT2:
101             case IXGBE_DEV_ID_82598EB_CX4:
102             case IXGBE_DEV_ID_82598_CX4_DUAL_PORT:
103             case IXGBE_DEV_ID_82598_DA_DUAL_PORT:
104             case IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM:
105             case IXGBE_DEV_ID_82598EB_XF_LR:
106             case IXGBE_DEV_ID_82598EB_SFP_LOM:
107                 hw->mac.type = ixgbe_mac_82598EB;
108                 break;
109             case IXGBE_DEV_ID_82599_KX4:
110             case IXGBE_DEV_ID_82599_KX4_MEZZ:
111             case IXGBE_DEV_ID_82599_XAUI_LOM:
112             case IXGBE_DEV_ID_82599_COMBO_BACKPLANE:
113             case IXGBE_DEV_ID_82599_KR:
114             case IXGBE_DEV_ID_82599_SFP:
115             case IXGBE_DEV_ID_82599_BACKPLANE_FCOE:
116             case IXGBE_DEV_ID_82599_SFP_FCOE:
117             case IXGBE_DEV_ID_82599_SFP_EM:
118             case IXGBE_DEV_ID_82599_SFP_SF2:
119             case IXGBE_DEV_ID_82599EN_SFP:
120             case IXGBE_DEV_ID_82599_CX4:
121             case IXGBE_DEV_ID_82599_T3_LOM:
122                 hw->mac.type = ixgbe_mac_82599EB;
123                 break;
124 }
```

```

123         case IXGBE_DEV_ID_82599_VF:
124             hw->mac.type = ixgbe_mac_82599_vf;
125             break;
126         case IXGBE_DEV_ID_X540_VF:
127             hw->mac.type = ixgbe_mac_X540_vf;
128             break;
129         case IXGBE_DEV_ID_X540T:
130         case IXGBE_DEV_ID_X540T1:
131             hw->mac.type = ixgbe_mac_X540;
132             break;
133         default:
134             ret_val = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
135             break;
136     }
137 } else {
138     ret_val = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
139 }
141
142     DEBUGOUT2("ixgbe_set_mac_type found mac: %d, returns: %d\n",
143             hw->mac.type, ret_val);
144     return ret_val;
145 }

```

unchanged_portion_omitted

```

375 /**
376  * ixgbe_read_pba_length - Reads part number string length from EEPROM
377  * @hw: pointer to hardware structure
378  * @pba_num_size: part number string buffer length
379  *
380  * Reads the part number length from the EEPROM.
381  * Returns expected buffer size in pba_num_size.
382  */
383 s32 ixgbe_read_pba_length(struct ixgbe_hw *hw, u32 *pba_num_size)
384 {
385     return ixgbe_read_pba_length_generic(hw, pba_num_size);
386 }

```

```

378 /**
379  * ixgbe_read_pba_num - Reads part number from EEPROM
380  * @hw: pointer to hardware structure
381  * @pba_num: stores the part number from the EEPROM
382  *
383  * Reads the part number from the EEPROM.
384  */
385 s32 ixgbe_read_pba_num(struct ixgbe_hw *hw, u32 *pba_num)
386 {
387     return ixgbe_read_pba_num_generic(hw, pba_num);
388 }

```

unchanged_portion_omitted

```

440 /**
441  * ixgbe_read_phy_reg - Read PHY register
442  * @hw: pointer to hardware structure
443  * @reg_addr: 32 bit address of PHY register to read
444  * @phy_data: Pointer to read data from PHY register
445  *
446  * Reads a value from a specified PHY register
447  */
448 s32 ixgbe_read_phy_reg(struct ixgbe_hw *hw, u32 reg_addr, u32 device_type,
449                        u16 *phy_data)
450 {
451     if (hw->phy.id == 0)
452         ixgbe_identify_phy(hw);
453     (void) ixgbe_identify_phy(hw);
454
455     return ixgbe_call_func(hw, hw->phy.ops.read_reg, (hw, reg_addr,

```

```

455         device_type, phy_data), IXGBE_NOT_IMPLEMENTED);
456 }
457
458 /**
459  * ixgbe_write_phy_reg - Write PHY register
460  * @hw: pointer to hardware structure
461  * @reg_addr: 32 bit PHY register to write
462  * @phy_data: Data to write to the PHY register
463  *
464  * Writes a value to specified PHY register
465  */
466 s32 ixgbe_write_phy_reg(struct ixgbe_hw *hw, u32 reg_addr, u32 device_type,
467                        u16 phy_data)
468 {
469     if (hw->phy.id == 0)
470         ixgbe_identify_phy(hw);
471     (void) ixgbe_identify_phy(hw);
472
473     return ixgbe_call_func(hw, hw->phy.ops.write_reg, (hw, reg_addr,
474         device_type, phy_data), IXGBE_NOT_IMPLEMENTED);
475 }

```

unchanged_portion_omitted

```

683 /**
684  * ixgbe_write_eeprom_buffer - Write word(s) to EEPROM
685  * @hw: pointer to hardware structure
686  * @offset: offset within the EEPROM to be written to
687  * @data: 16 bit word(s) to be written to the EEPROM
688  * @words: number of words
689  *
690  * Writes 16 bit word(s) to EEPROM. If ixgbe_eeprom_update_checksum is not
691  * called after this function, the EEPROM will most likely contain an
692  * invalid checksum.
693  */
694 s32 ixgbe_write_eeprom_buffer(struct ixgbe_hw *hw, u16 offset, u16 words,
695                              u16 *data)
696 {
697     return ixgbe_call_func(hw, hw->eeprom.ops.write_buffer,
698         (hw, offset, words, data),
699         IXGBE_NOT_IMPLEMENTED);
700 }

```

```

702 /**
703  * ixgbe_read_eeprom - Read word from EEPROM
704  * @hw: pointer to hardware structure
705  * @offset: offset within the EEPROM to be read
706  * @data: read 16 bit value from EEPROM
707  *
708  * Reads 16 bit value from EEPROM
709  */
710 s32 ixgbe_read_eeprom(struct ixgbe_hw *hw, u16 offset, u16 *data)
711 {
712     return ixgbe_call_func(hw, hw->eeprom.ops.read, (hw, offset, data),
713         IXGBE_NOT_IMPLEMENTED);
714 }

```

```

716 /**
717  * ixgbe_read_eeprom_buffer - Read word(s) from EEPROM
718  * @hw: pointer to hardware structure
719  * @offset: offset within the EEPROM to be read
720  * @data: read 16 bit word(s) from EEPROM
721  * @words: number of words
722  *
723  * Reads 16 bit word(s) from EEPROM
724  */
725 s32 ixgbe_read_eeprom_buffer(struct ixgbe_hw *hw, u16 offset,

```

```

726         u16 words, u16 *data)
727 {
728     return ixgbe_call_func(hw, hw->eeprom.ops.read_buffer,
729         (hw, offset, words, data),
730         IXGBE_NOT_IMPLEMENTED);
731 }

733 /**
734  * ixgbe_validate_eeprom_checksum - Validate EEPROM checksum
735  * @hw: pointer to hardware structure
736  * @checksum_val: calculated checksum
737  *
738  * Performs checksum calculation and validates the EEPROM checksum
739  */
740 s32 ixgbe_validate_eeprom_checksum(struct ixgbe_hw *hw, u16 *checksum_val)
741 {
742     return ixgbe_call_func(hw, hw->eeprom.ops.validate_checksum,
743         (hw, checksum_val), IXGBE_NOT_IMPLEMENTED);
744 }
    unchanged_portion_omitted

802 /**
803  * ixgbe_set_vmdq - Associate a VMDq index with a receive address
804  * @hw: pointer to hardware structure
805  * @rar: receive address register index to associate with VMDq index
806  * @vmdq: VMDq set or pool index
807  */
808 s32 ixgbe_set_vmdq(struct ixgbe_hw *hw, u32 rar, u32 vmdq)
809 {
810     return ixgbe_call_func(hw, hw->mac.ops.set_vmdq, (hw, rar, vmdq),
811         IXGBE_NOT_IMPLEMENTED);
812 }

815 /**
816  * ixgbe_set_vmdq_san_mac - Associate VMDq index 127 with a receive address
817  * @hw: pointer to hardware structure
818  * @vmdq: VMDq default pool index
819  */
820 s32 ixgbe_set_vmdq_san_mac(struct ixgbe_hw *hw, u32 vmdq)
821 {
822     return ixgbe_call_func(hw, hw->mac.ops.set_vmdq_san_mac,
823         (hw, vmdq), IXGBE_NOT_IMPLEMENTED);
824 }

826 /**
827  * ixgbe_clear_vmdq - Disassociate a VMDq index from a receive address
828  * @hw: pointer to hardware structure
829  * @rar: receive address register index to disassociate with VMDq index
830  * @vmdq: VMDq set or pool index
831  */
832 s32 ixgbe_clear_vmdq(struct ixgbe_hw *hw, u32 rar, u32 vmdq)
833 {
834     return ixgbe_call_func(hw, hw->mac.ops.clear_vmdq, (hw, rar, vmdq),
835         IXGBE_NOT_IMPLEMENTED);
836 }
    unchanged_portion_omitted

880 /**
881  * ixgbe_update_mc_addr_list - Updates the MAC's list of multicast addresses
882  * @hw: pointer to hardware structure
883  * @mc_addr_list: the list of new multicast addresses
884  * @mc_addr_count: number of addresses
885  * @func: iterator function to walk the multicast address list
886  *
887  * The given list replaces any existing list. Clears the MC addrs from receive

```

```

888  * address registers and the multicast table. Uses unused receive address
889  * registers for the first multicast addresses, and hashes the rest into the
890  * multicast table.
891  */
892 s32 ixgbe_update_mc_addr_list(struct ixgbe_hw *hw, u8 *mc_addr_list,
893     u32 mc_addr_count, ixgbe_mc_addr_itr func,
894     bool clear)
895 {
896     return ixgbe_call_func(hw, hw->mac.ops.update_mc_addr_list, (hw,
897         mc_addr_list, mc_addr_count, func, clear),
898         mc_addr_list, mc_addr_count, func),
899         IXGBE_NOT_IMPLEMENTED);
    unchanged_portion_omitted

952 /**
953  * ixgbe_set_vlvf - Set VLAN Pool Filter
954  * @hw: pointer to hardware structure
955  * @vlan: VLAN id to write to VLAN filter
956  * @vind: VMDq output index that maps queue to VLAN id in VFVF
957  * @vlan_on: boolean flag to turn on/off VLAN in VFVF
958  * @vfta_changed: pointer to boolean flag which indicates whether VFTA
959  *                should be changed
960  *
961  * Turn on/off specified bit in VLVF table.
962  */
963 s32 ixgbe_set_vlvf(struct ixgbe_hw *hw, u32 vlan, u32 vind, bool vlan_on,
964     bool *vfta_changed)
965 {
966     return ixgbe_call_func(hw, hw->mac.ops.set_vlvf, (hw, vlan, vind,
967         vlan_on, vfta_changed), IXGBE_NOT_IMPLEMENTED);
968 }

970 /**
971  * ixgbe_fc_enable - Enable flow control
972  * @hw: pointer to hardware structure
973  * @packetbuf_num: packet buffer number (0-7)
974  *
975  * Configures the flow control settings based on SW configuration.
976  */
977 s32 ixgbe_fc_enable(struct ixgbe_hw *hw)
978 {
979     return ixgbe_call_func(hw, hw->mac.ops.fc_enable, (hw),
980         return ixgbe_call_func(hw, hw->mac.ops.fc_enable, (hw, packetbuf_num),
981         IXGBE_NOT_IMPLEMENTED);
982 }

982 /**
983  * ixgbe_set_fw_drv_ver - Try to send the driver version number FW
984  * @hw: pointer to hardware structure
985  * @maj: driver major number to be sent to firmware
986  * @min: driver minor number to be sent to firmware
987  * @build: driver build number to be sent to firmware
988  * @ver: driver version number to be sent to firmware
989  */
990 s32 ixgbe_set_fw_drv_ver(struct ixgbe_hw *hw, u8 maj, u8 min, u8 build,
991     u8 ver)
992 {
993     return ixgbe_call_func(hw, hw->mac.ops.set_fw_drv_ver, (hw, maj, min,
994         build, ver), IXGBE_NOT_IMPLEMENTED);
995 }

998 /**

```

```
999  * ixgbe_read_analog_reg8 - Reads 8 bit analog register
1000  * @hw: pointer to hardware structure
1001  * @reg: analog register to read
1002  * @val: read value
1003  *
1004  * Performs write operation to analog register specified.
1005  **/
1006 s32 ixgbe_read_analog_reg8(struct ixgbe_hw *hw, u32 reg, u8 *val)
1007 {
1008     return ixgbe_call_func(hw, hw->mac.ops.read_analog_reg8, (hw, reg,
1009     val), IXGBE_NOT_IMPLEMENTED);
1010 }
    unchanged_portion_omitted

1113 /**
1114  * 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
1115  * @hw: pointer to hardware structure
1116  * @regval: bitfield to write to the Rx DMA register
1117  *
1118  * Enables the Rx DMA unit of the device.
1119  **/
1120 s32 ixgbe_enable_rx_dma(struct ixgbe_hw *hw, u32 regval)
1121 {
1122     return ixgbe_call_func(hw, hw->mac.ops.enable_rx_dma,
1123     (hw, regval), IXGBE_NOT_IMPLEMENTED);
1124 }

1126 /**
1127  * ixgbe_disable_sec_rx_path - Stops the receive data path
1128  * @hw: pointer to hardware structure
1129  *
1130  * Stops the receive data path.
1131  **/
1132 s32 ixgbe_disable_sec_rx_path(struct ixgbe_hw *hw)
1133 {
1134     return ixgbe_call_func(hw, hw->mac.ops.disable_sec_rx_path,
1135     (hw), IXGBE_NOT_IMPLEMENTED);
1136 }

1138 /**
1139  * ixgbe_enable_sec_rx_path - Enables the receive data path
1140  * @hw: pointer to hardware structure
1141  *
1142  * Enables the receive data path.
1143  **/
1144 s32 ixgbe_enable_sec_rx_path(struct ixgbe_hw *hw)
1145 {
1146     return ixgbe_call_func(hw, hw->mac.ops.enable_sec_rx_path,
1147     (hw), IXGBE_NOT_IMPLEMENTED);
1148 }

1150 /**
1151  * ixgbe_acquire_swfw_semaphore - Acquire SWFW semaphore
1152  * @hw: pointer to hardware structure
1153  * @mask: Mask to specify which semaphore to acquire
1154  *
1155  * Acquires the SWFW semaphore through SW_FW_SYNC register for the specified
1156  * function (CSR, PHY0, PHY1, EEPROM, Flash)
1157  **/
1158 s32 ixgbe_acquire_swfw_semaphore(struct ixgbe_hw *hw, u16 mask)
1159 {
1160     return ixgbe_call_func(hw, hw->mac.ops.acquire_swfw_sync,
1161     (hw, mask), IXGBE_NOT_IMPLEMENTED);
1162 }
    unchanged_portion_omitted
```

```

*****
8429 Fri Jul 20 10:59:33 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_api.h
XXX Intel X540 support
*****
1 /*****

3 Copyright (c) 2001-2012, Intel Corporation
3 Copyright (c) 2001-2010, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.

32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_api.h,v 1.14 2012/07/05 20:51:44 jfv Exp $*/

35 #ifndef _IXGBE_API_H_
36 #define _IXGBE_API_H_

38 #include "ixgbe_type.h"

40 s32 ixgbe_init_shared_code(struct ixgbe_hw *hw);

42 extern s32 ixgbe_init_ops_82598(struct ixgbe_hw *hw);
43 extern s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw);
44 extern s32 ixgbe_init_ops_X540(struct ixgbe_hw *hw);
45 extern s32 ixgbe_init_ops_vf(struct ixgbe_hw *hw);

47 s32 ixgbe_set_mac_type(struct ixgbe_hw *hw);
48 s32 ixgbe_init_hw(struct ixgbe_hw *hw);
49 s32 ixgbe_reset_hw(struct ixgbe_hw *hw);
50 s32 ixgbe_start_hw(struct ixgbe_hw *hw);
51 void ixgbe_enable_relaxed_ordering(struct ixgbe_hw *hw);
52 s32 ixgbe_clear_hw_cntrs(struct ixgbe_hw *hw);
53 enum ixgbe_media_type ixgbe_get_media_type(struct ixgbe_hw *hw);
54 s32 ixgbe_get_mac_addr(struct ixgbe_hw *hw, u8 *mac_addr);
55 s32 ixgbe_get_bus_info(struct ixgbe_hw *hw);
56 u32 ixgbe_get_num_of_tx_queues(struct ixgbe_hw *hw);
57 u32 ixgbe_get_num_of_rx_queues(struct ixgbe_hw *hw);
58 s32 ixgbe_stop_adapter(struct ixgbe_hw *hw);
59 s32 ixgbe_read_pba_num(struct ixgbe_hw *hw, u32 *pba_num);
60 s32 ixgbe_read_pba_string(struct ixgbe_hw *hw, u8 *pba_num, u32 pba_num_size);

```

```

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

62 s32 ixgbe_identify_phy(struct ixgbe_hw *hw);
63 s32 ixgbe_reset_phy(struct ixgbe_hw *hw);
64 s32 ixgbe_read_phy_reg(struct ixgbe_hw *hw, u32 reg_addr, u32 device_type,
65 u16 *phy_data);
66 s32 ixgbe_write_phy_reg(struct ixgbe_hw *hw, u32 reg_addr, u32 device_type,
67 u16 phy_data);

69 s32 ixgbe_setup_phy_link(struct ixgbe_hw *hw);
70 s32 ixgbe_check_phy_link(struct ixgbe_hw *hw,
71 ixgbe_link_speed *speed,
72 bool *link_up);
73 s32 ixgbe_setup_phy_link_speed(struct ixgbe_hw *hw,
74 ixgbe_link_speed speed,
75 bool autoneg,
76 bool autoneg_wait_to_complete);
77 void ixgbe_disable_tx_laser(struct ixgbe_hw *hw);
78 void ixgbe_enable_tx_laser(struct ixgbe_hw *hw);
79 void ixgbe_flap_tx_laser(struct ixgbe_hw *hw);
80 s32 ixgbe_setup_link(struct ixgbe_hw *hw, ixgbe_link_speed speed,
81 bool autoneg, bool autoneg_wait_to_complete);
82 s32 ixgbe_check_link(struct ixgbe_hw *hw, ixgbe_link_speed *speed,
83 bool *link_up, bool link_up_wait_to_complete);
84 s32 ixgbe_get_link_capabilities(struct ixgbe_hw *hw, ixgbe_link_speed *speed,
85 bool *autoneg);
86 s32 ixgbe_led_on(struct ixgbe_hw *hw, u32 index);
87 s32 ixgbe_led_off(struct ixgbe_hw *hw, u32 index);
88 s32 ixgbe_blink_led_start(struct ixgbe_hw *hw, u32 index);
89 s32 ixgbe_blink_led_stop(struct ixgbe_hw *hw, u32 index);

91 s32 ixgbe_init_eeprom_params(struct ixgbe_hw *hw);
92 s32 ixgbe_write_eeprom(struct ixgbe_hw *hw, u16 offset, u16 data);
93 s32 ixgbe_write_eeprom_buffer(struct ixgbe_hw *hw, u16 offset,
94 u16 words, u16 *data);
95 s32 ixgbe_read_eeprom(struct ixgbe_hw *hw, u16 offset, u16 *data);
96 s32 ixgbe_read_eeprom_buffer(struct ixgbe_hw *hw, u16 offset,
97 u16 words, u16 *data);

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

102 s32 ixgbe_insert_mac_addr(struct ixgbe_hw *hw, u8 *addr, u32 vmdq);
103 s32 ixgbe_set_rar(struct ixgbe_hw *hw, u32 index, u8 *addr, u32 vmdq,
104 u32 enable_addr);
105 s32 ixgbe_clear_rar(struct ixgbe_hw *hw, u32 index);
106 s32 ixgbe_set_vmdq(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
107 s32 ixgbe_set_vmdq_san_mac(struct ixgbe_hw *hw, u32 vmdq);
108 s32 ixgbe_clear_vmdq(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
109 s32 ixgbe_init_rx_addrs(struct ixgbe_hw *hw);
110 u32 ixgbe_get_num_rx_addrs(struct ixgbe_hw *hw);
111 s32 ixgbe_update_uc_addr_list(struct ixgbe_hw *hw, u8 *addr_list,
112 u32 addr_count, ixgbe_mc_addr_itr func);
113 s32 ixgbe_update_mc_addr_list(struct ixgbe_hw *hw, u8 *mc_addr_list,
114 u32 mc_addr_count, ixgbe_mc_addr_itr func,
115 bool clear);
116 void ixgbe_add_uc_addr(struct ixgbe_hw *hw, u8 *addr_list, u32 vmdq);
117 s32 ixgbe_enable_mc(struct ixgbe_hw *hw);
118 s32 ixgbe_disable_mc(struct ixgbe_hw *hw);
119 s32 ixgbe_clear_vfta(struct ixgbe_hw *hw);
120 s32 ixgbe_set_vfta(struct ixgbe_hw *hw, u32 vlan,
121 u32 vind, bool vlan_on);
122 s32 ixgbe_set_vlvf(struct ixgbe_hw *hw, u32 vlan, u32 vind,
123 bool vlan_on, bool *vfta_changed);
124 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 pballloc);
124 s32 ixgbe_init_fdir_perfect_82599(struct ixgbe_hw *hw, u32 pballloc);
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_wnn_prefix(struct ixgbe_hw *hw, u16 *wnn_prefix,
173                          u16 *wppn_prefix);
174 s32 ixgbe_get_fcoe_boot_status(struct ixgbe_hw *hw, u16 *bs);

176 #endif /* _IXGBE_API_H_ */
```

new/usr/src/uts/common/io/ixgbe/ixgbe_common.c

1

```
*****
116733 Fri Jul 20 10:59:34 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_common.c
XXX Intel X540 support
*****
1 /*****

3 Copyright (c) 2001-2012, Intel Corporation
3 Copyright (c) 2001-2010, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.

32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_common.c,v 1.14 2012/07/05 20:51:44 jfv Exp
33 /*$FreeBSD$*/

35 #include "ixgbe_common.h"
36 #include "ixgbe_phy.h"
37 #include "ixgbe_api.h"

39 static s32 ixgbe_acquire_eeprom(struct ixgbe_hw *hw);
40 static s32 ixgbe_get_eeprom_semaphore(struct ixgbe_hw *hw);
41 static void ixgbe_release_eeprom_semaphore(struct ixgbe_hw *hw);
42 static s32 ixgbe_ready_eeprom(struct ixgbe_hw *hw);
43 static void ixgbe_standby_eeprom(struct ixgbe_hw *hw);
44 static void ixgbe_shift_out_eeprom_bits(struct ixgbe_hw *hw, u16 data,
45 u16 count);
46 static u16 ixgbe_shift_in_eeprom_bits(struct ixgbe_hw *hw, u16 count);
47 static void ixgbe_raise_eeprom_clk(struct ixgbe_hw *hw, u32 *eec);
48 static void ixgbe_lower_eeprom_clk(struct ixgbe_hw *hw, u32 *eec);
49 static void ixgbe_release_eeprom(struct ixgbe_hw *hw);

51 static s32 ixgbe_mta_vector(struct ixgbe_hw *hw, u8 *mc_addr);
52 static s32 ixgbe_get_san_mac_addr_offset(struct ixgbe_hw *hw,
53 u16 *san_mac_offset);
54 static s32 ixgbe_read_eeprom_buffer_bit_bang(struct ixgbe_hw *hw, u16 offset,
55 u16 words, u16 *data);
56 static s32 ixgbe_write_eeprom_buffer_bit_bang(struct ixgbe_hw *hw, u16 offset,
57 u16 words, u16 *data);
58 static s32 ixgbe_detect_eeprom_page_size_generic(struct ixgbe_hw *hw,
59 u16 offset);
```

new/usr/src/uts/common/io/ixgbe/ixgbe_common.c

2

```
54 static s32 ixgbe_fc_autoneg_fiber(struct ixgbe_hw *hw);
55 static s32 ixgbe_fc_autoneg_backplane(struct ixgbe_hw *hw);
56 static s32 ixgbe_fc_autoneg_copper(struct ixgbe_hw *hw);
57 static s32 ixgbe_device_supports_autoneg_fc(struct ixgbe_hw *hw);
58 static s32 ixgbe_negotiate_fc(struct ixgbe_hw *hw, u32 adv_reg, u32 lp_reg,
59 u32 adv_sym, u32 adv_asm, u32 lp_sym, u32 lp_asm);

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

62 /**
62 * ixgbe_init_ops_generic - Inits function ptrs
63 * @hw: pointer to the hardware structure
64 *
65 * Initialize the function pointers.
66 */
67 s32 ixgbe_init_ops_generic(struct ixgbe_hw *hw)
68 {
69 struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
70 struct ixgbe_mac_info *mac = &hw->mac;
71 u32 eec = IXGBE_READ_REG(hw, IXGBE_EEC);

73 DEBUGFUNC("ixgbe_init_ops_generic");

75 /* EEPROM */
76 eeprom->ops.init_params = &ixgbe_init_eeprom_params_generic;
77 /* If EEPROM is valid (bit 8 = 1), use EERD otherwise use bit bang */
78 if (eec & IXGBE_EEC_PRES) {
79 if (eec & (1 << 8))
80 eeprom->ops.read = &ixgbe_read_eerd_generic;
81 eeprom->ops.read_buffer = &ixgbe_read_eerd_buffer_generic;
82 } else {
83 eeprom->ops.read = &ixgbe_read_eeprom_bit_bang_generic;
84 eeprom->ops.read_buffer =
85 &ixgbe_read_eeprom_buffer_bit_bang_generic;
86 }
87 eeprom->ops.write = &ixgbe_write_eeprom_generic;
88 eeprom->ops.write_buffer = &ixgbe_write_eeprom_buffer_bit_bang_generic;
89 eeprom->ops.validate_checksum =
90 &ixgbe_validate_eeprom_checksum_generic;
91 eeprom->ops.update_checksum = &ixgbe_update_eeprom_checksum_generic;
92 eeprom->ops.calc_checksum = &ixgbe_calc_eeprom_checksum_generic;

93 /* MAC */
94 mac->ops.init_hw = &ixgbe_init_hw_generic;
95 mac->ops.reset_hw = NULL;
96 mac->ops.start_hw = &ixgbe_start_hw_generic;
97 mac->ops.clear_hw_cntrs = &ixgbe_clear_hw_cntrs_generic;
98 mac->ops.get_media_type = NULL;
99 mac->ops.get_supported_physical_layer = NULL;
100 mac->ops.enable_rx_dma = &ixgbe_enable_rx_dma_generic;
101 mac->ops.get_mac_addr = &ixgbe_get_mac_addr_generic;
102 mac->ops.stop_adapter = &ixgbe_stop_adapter_generic;
103 mac->ops.get_bus_info = &ixgbe_get_bus_info_generic;
104 mac->ops.set_lan_id = &ixgbe_set_lan_id_multi_port_pcie;
105 mac->ops.acquire_swfw_sync = &ixgbe_acquire_swfw_sync;
106 mac->ops.release_swfw_sync = &ixgbe_release_swfw_sync;

108 /* LEDs */
109 mac->ops.led_on = &ixgbe_led_on_generic;
110 mac->ops.led_off = &ixgbe_led_off_generic;
111 mac->ops.blink_led_start = &ixgbe_blink_led_start_generic;
112 mac->ops.blink_led_stop = &ixgbe_blink_led_stop_generic;

114 /* RAR, Multicast, VLAN */
115 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 }
275
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);
283
284     /* Disable AN timeout */
285     if (hw->fc.strict_ieee)
286         reg &= ~IXGBE_PCS1GLCTL_AN_1G_TIMEOUT_EN;
287
288     IXGBE_WRITE_REG(hw, IXGBE_PCS1GLCTL, reg);
289     DEBUGOUT1("Set up FC; PCS1GLCTL = 0x%08X\n", reg);
290 }
291
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  * }
305
306     DEBUGOUT1("Set up FC; IXGBE_AUTOC = 0x%08X\n", reg);
307 out:
308     return ret_val;
309 }
310
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;
324
325     DEBUGFUNC("ixgbe_start_hw_generic");
326
327     /* Set the media type */
328     hw->phy.media_type = hw->mac.ops.get_media_type(hw);
329
330     /* PHY ops initialization must be done in reset_hw() */
331
332     /* Clear the VLAN filter table */
333     hw->mac.ops.clear_vfta(hw);
334
335     /* Clear statistics registers */
336     hw->mac.ops.clear_hw_cntrs(hw);
337
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);
343
344     /* Setup flow control */
345     ret_val = ixgbe_setup_fc(hw);
346     if (ret_val != IXGBE_SUCCESS)
347         goto out;
348     (void) ixgbe_setup_fc(hw, 0);
349
350     /* Clear adapter stopped flag */
351     hw->adapter_stopped = FALSE;
352
353 out:
354     return ret_val;
355     return IXGBE_SUCCESS;
356 }
357
358 /**
359  * ixgbe_start_hw_gen2 - Init sequence for common device family
360  * @hw: pointer to hw structure
361  *
362  * Performs the init sequence common to the second generation
363  * of 10 GbE devices.
364  * Devices in the second generation:
365  * 82599
366  * X540
367  */
368 s32 ixgbe_start_hw_gen2(struct ixgbe_hw *hw)
369 {
370     u32 i;
371     u32 regval;
372
373     /* Clear the rate limiters */
374     for (i = 0; i < hw->mac.max_tx_queues; i++) {
375         IXGBE_WRITE_REG(hw, IXGBE_RTTDQSEL, i);
376         IXGBE_WRITE_REG(hw, IXGBE_RTTBCNRC, 0);
377     }
378     IXGBE_WRITE_FLUSH(hw);

```

```

378 /* Disable relaxed ordering */
379 for (i = 0; i < hw->mac.max_tx_queues; i++) {
380     regval = IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL_82599(i));
381     regval &= ~IXGBE_DCA_TXCTRL_DESC_WRO_EN;
203     regval &= ~IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
382     IXGBE_WRITE_REG(hw, IXGBE_DCA_TXCTRL_82599(i), regval);
383 }

385 for (i = 0; i < hw->mac.max_rx_queues; i++) {
386     regval = IXGBE_READ_REG(hw, IXGBE_DCA_RXCTRL(i));
387     regval &= ~(IXGBE_DCA_RXCTRL_DATA_WRO_EN |
388                IXGBE_DCA_RXCTRL_HEAD_WRO_EN);
209     regval &= ~(IXGBE_DCA_RXCTRL_DESC_WRO_EN |
210                IXGBE_DCA_RXCTRL_DESC_HSRO_EN);
389     IXGBE_WRITE_REG(hw, IXGBE_DCA_RXCTRL(i), regval);
390 }

392 return IXGBE_SUCCESS;
393 }
unchanged_portion_omitted

422 /**
423  * ixgbe_clear_hw_cntrs_generic - Generic clear hardware counters
424  * @hw: pointer to hardware structure
425  *
426  * Clears all hardware statistics counters by reading them from the hardware
427  * Statistics counters are clear on read.
428  */
429 static ixgbe_clear_hw_cntrs_generic(struct ixgbe_hw *hw)
430 {
431     u16 i = 0;

433     DEBUGFUNC("ixgbe_clear_hw_cntrs_generic");

435     IXGBE_READ_REG(hw, IXGBE_CRCERRS);
436     IXGBE_READ_REG(hw, IXGBE_ILLERRC);
437     IXGBE_READ_REG(hw, IXGBE_ERRBC);
438     IXGBE_READ_REG(hw, IXGBE_MSPDC);
257     (void) IXGBE_READ_REG(hw, IXGBE_CRCERRS);
258     (void) IXGBE_READ_REG(hw, IXGBE_ILLERRC);
259     (void) IXGBE_READ_REG(hw, IXGBE_ERRBC);
260     (void) IXGBE_READ_REG(hw, IXGBE_MSPDC);
439     for (i = 0; i < 8; i++)
440         IXGBE_READ_REG(hw, IXGBE_MPC(i));
262     (void) IXGBE_READ_REG(hw, IXGBE_MPC(i));

442     IXGBE_READ_REG(hw, IXGBE_MLFC);
443     IXGBE_READ_REG(hw, IXGBE_MRFC);
444     IXGBE_READ_REG(hw, IXGBE_RLEC);
445     IXGBE_READ_REG(hw, IXGBE_LXONTXC);
446     IXGBE_READ_REG(hw, IXGBE_LXOFFTXC);
264     (void) IXGBE_READ_REG(hw, IXGBE_MLFC);
265     (void) IXGBE_READ_REG(hw, IXGBE_MRFC);
266     (void) IXGBE_READ_REG(hw, IXGBE_RLEC);
267     (void) IXGBE_READ_REG(hw, IXGBE_LXONTXC);
268     (void) IXGBE_READ_REG(hw, IXGBE_LXOFFTXC);
447     if (hw->mac.type >= ixgbe_mac_82599EB) {
448         IXGBE_READ_REG(hw, IXGBE_LXONRXCNT);
449         IXGBE_READ_REG(hw, IXGBE_LXOFFRXCNT);
270         (void) IXGBE_READ_REG(hw, IXGBE_LXONRXCNT);
271         (void) IXGBE_READ_REG(hw, IXGBE_LXOFFRXCNT);
450     } else {
451         IXGBE_READ_REG(hw, IXGBE_LXONRXC);
452         IXGBE_READ_REG(hw, IXGBE_LXOFFRXC);
273         (void) IXGBE_READ_REG(hw, IXGBE_LXONRXC);
274         (void) IXGBE_READ_REG(hw, IXGBE_LXOFFRXC);

```

```

453     }

455     for (i = 0; i < 8; i++) {
456         IXGBE_READ_REG(hw, IXGBE_PXONTXC(i));
457         IXGBE_READ_REG(hw, IXGBE_PXOFFTXC(i));
278         (void) IXGBE_READ_REG(hw, IXGBE_PXONTXC(i));
279         (void) IXGBE_READ_REG(hw, IXGBE_PXOFFTXC(i));
458         if (hw->mac.type >= ixgbe_mac_82599EB) {
459             IXGBE_READ_REG(hw, IXGBE_PXONRXCNT(i));
460             IXGBE_READ_REG(hw, IXGBE_PXOFFRXCNT(i));
281             (void) IXGBE_READ_REG(hw, IXGBE_PXONRXCNT(i));
282             (void) IXGBE_READ_REG(hw, IXGBE_PXOFFRXCNT(i));
461         } else {
462             IXGBE_READ_REG(hw, IXGBE_PXONRXC(i));
463             IXGBE_READ_REG(hw, IXGBE_PXOFFRXC(i));
284             (void) IXGBE_READ_REG(hw, IXGBE_PXONRXC(i));
285             (void) IXGBE_READ_REG(hw, IXGBE_PXOFFRXC(i));
464         }
465     }
466     if (hw->mac.type >= ixgbe_mac_82599EB)
467         for (i = 0; i < 8; i++)
468             IXGBE_READ_REG(hw, IXGBE_PXON2OFFCNT(i));
469     IXGBE_READ_REG(hw, IXGBE_PRC64);
470     IXGBE_READ_REG(hw, IXGBE_PRC127);
471     IXGBE_READ_REG(hw, IXGBE_PRC255);
472     IXGBE_READ_REG(hw, IXGBE_PRC511);
473     IXGBE_READ_REG(hw, IXGBE_PRC1023);
474     IXGBE_READ_REG(hw, IXGBE_PRC1522);
475     IXGBE_READ_REG(hw, IXGBE_GPRC);
476     IXGBE_READ_REG(hw, IXGBE_BPRC);
477     IXGBE_READ_REG(hw, IXGBE_MPRC);
478     IXGBE_READ_REG(hw, IXGBE_GPTC);
479     IXGBE_READ_REG(hw, IXGBE_GORCL);
480     IXGBE_READ_REG(hw, IXGBE_GORCH);
481     IXGBE_READ_REG(hw, IXGBE_GOTCL);
482     IXGBE_READ_REG(hw, IXGBE_GOTCH);
483     if (hw->mac.type == ixgbe_mac_82598EB)
290         (void) IXGBE_READ_REG(hw, IXGBE_PXON2OFFCNT(i));
291     (void) IXGBE_READ_REG(hw, IXGBE_PRC64);
292     (void) IXGBE_READ_REG(hw, IXGBE_PRC127);
293     (void) IXGBE_READ_REG(hw, IXGBE_PRC255);
294     (void) IXGBE_READ_REG(hw, IXGBE_PRC511);
295     (void) IXGBE_READ_REG(hw, IXGBE_PRC1023);
296     (void) IXGBE_READ_REG(hw, IXGBE_PRC1522);
297     (void) IXGBE_READ_REG(hw, IXGBE_GPRC);
298     (void) IXGBE_READ_REG(hw, IXGBE_BPRC);
299     (void) IXGBE_READ_REG(hw, IXGBE_MPRC);
300     (void) IXGBE_READ_REG(hw, IXGBE_GPTC);
301     (void) IXGBE_READ_REG(hw, IXGBE_GORCL);
302     (void) IXGBE_READ_REG(hw, IXGBE_GORCH);
303     (void) IXGBE_READ_REG(hw, IXGBE_GOTCL);
304     (void) IXGBE_READ_REG(hw, IXGBE_GOTCH);
484     for (i = 0; i < 8; i++)
485         IXGBE_READ_REG(hw, IXGBE_RNBC(i));
486     IXGBE_READ_REG(hw, IXGBE_RUC);
487     IXGBE_READ_REG(hw, IXGBE_RFC);
488     IXGBE_READ_REG(hw, IXGBE_ROC);
489     IXGBE_READ_REG(hw, IXGBE_RJC);
490     IXGBE_READ_REG(hw, IXGBE_MNGPRC);
491     IXGBE_READ_REG(hw, IXGBE_MNGPDC);
492     IXGBE_READ_REG(hw, IXGBE_MNGPTC);
493     IXGBE_READ_REG(hw, IXGBE_TORL);
494     IXGBE_READ_REG(hw, IXGBE_TORH);
495     IXGBE_READ_REG(hw, IXGBE_TPR);
496     IXGBE_READ_REG(hw, IXGBE_TPT);
497     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);
505 (void) IXGBE_READ_REG(hw, IXGBE_RNBC(i));
506 (void) IXGBE_READ_REG(hw, IXGBE_RUC);
507 (void) IXGBE_READ_REG(hw, IXGBE_RFC);
508 (void) IXGBE_READ_REG(hw, IXGBE_ROC);
509 (void) IXGBE_READ_REG(hw, IXGBE_RJC);
510 (void) IXGBE_READ_REG(hw, IXGBE_MNGPRC);
511 (void) IXGBE_READ_REG(hw, IXGBE_MNGPDC);
512 (void) IXGBE_READ_REG(hw, IXGBE_MNGPTC);
513 (void) IXGBE_READ_REG(hw, IXGBE_TORL);
514 (void) IXGBE_READ_REG(hw, IXGBE_TORH);
515 (void) IXGBE_READ_REG(hw, IXGBE_TPR);
516 (void) IXGBE_READ_REG(hw, IXGBE_TPT);
517 (void) IXGBE_READ_REG(hw, IXGBE_PTC64);
518 (void) IXGBE_READ_REG(hw, IXGBE_PTC127);
519 (void) IXGBE_READ_REG(hw, IXGBE_PTC255);
520 (void) IXGBE_READ_REG(hw, IXGBE_PTC511);
521 (void) IXGBE_READ_REG(hw, IXGBE_PTC1023);
522 (void) IXGBE_READ_REG(hw, IXGBE_PTC1522);
523 (void) IXGBE_READ_REG(hw, IXGBE_MPTC);
524 (void) IXGBE_READ_REG(hw, IXGBE_BPTC);
525 for (i = 0; i < 16; i++) {
526     IXGBE_READ_REG(hw, IXGBE_QPRC(i));
527     IXGBE_READ_REG(hw, IXGBE_QPTC(i));
528     (void) IXGBE_READ_REG(hw, IXGBE_QPRC(i));
529     (void) IXGBE_READ_REG(hw, IXGBE_QPTC(i));
530     if (hw->mac.type >= ixgbe_mac_82599EB) {
531         IXGBE_READ_REG(hw, IXGBE_QBRC_L(i));
532         IXGBE_READ_REG(hw, IXGBE_QBRC_H(i));
533         IXGBE_READ_REG(hw, IXGBE_QBTC_L(i));
534         IXGBE_READ_REG(hw, IXGBE_QBTC_H(i));
535         IXGBE_READ_REG(hw, IXGBE_QPRDC(i));
536         (void) IXGBE_READ_REG(hw, IXGBE_QBRC_L(i));
537         (void) IXGBE_READ_REG(hw, IXGBE_QBRC_H(i));
538         (void) IXGBE_READ_REG(hw, IXGBE_QBTC_L(i));
539         (void) IXGBE_READ_REG(hw, IXGBE_QBTC_H(i));
540         (void) IXGBE_READ_REG(hw, IXGBE_QPRDC(i));
541     } else {
542         IXGBE_READ_REG(hw, IXGBE_QBRC(i));
543         IXGBE_READ_REG(hw, IXGBE_QBTC(i));
544         (void) IXGBE_READ_REG(hw, IXGBE_QBRC(i));
545         (void) IXGBE_READ_REG(hw, IXGBE_QBTC(i));
546     }
547 }
548
549 if (hw->mac.type == ixgbe_mac_X540) {
550     if (hw->phy.id == 0)
551         ixgbe_identify_phy(hw);
552     hw->phy.ops.read_reg(hw, IXGBE_PCR8ECL,
553                         IXGBE_MDIO_PCS_DEV_TYPE, &i);
554     hw->phy.ops.read_reg(hw, IXGBE_PCR8ECH,
555                         IXGBE_MDIO_PCS_DEV_TYPE, &i);
556     hw->phy.ops.read_reg(hw, IXGBE_LDPCECL,
557                         IXGBE_MDIO_PCS_DEV_TYPE, &i);
558     hw->phy.ops.read_reg(hw, IXGBE_LDPCECH,
559                         IXGBE_MDIO_PCS_DEV_TYPE, &i);
560 }
561
562 return IXGBE_SUCCESS;
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

647 /**
648 * ixgbe_read_pba_length_generic - Reads part number length from EEPROM
649 * @hw: pointer to hardware structure
650 * @pba_num_size: part number string buffer length
651 *
652 * Reads the part number length from the EEPROM.
653 * Returns expected buffer size in pba_num_size
654 */
655 s32 ixgbe_read_pba_length_generic(struct ixgbe_hw *hw, u32 *pba_num_size)
656 {
657     s32 ret_val;
658     u16 data;
659     u16 pba_ptr;
660     u16 length;
661
662     DEBUGFUNC("ixgbe_read_pba_length_generic");
663
664     if (pba_num_size == NULL) {
665         DEBUGOUT("PBA buffer size was null\n");
666         return IXGBE_ERR_INVALID_ARGUMENT;
667     }
668
669     ret_val = hw->eeprom.ops.read(hw, IXGBE_PBANUM0_PTR, &data);
670     if (ret_val) {
671         DEBUGOUT("NVM Read Error\n");
672         return ret_val;
673     }
674
675     ret_val = hw->eeprom.ops.read(hw, IXGBE_PBANUM1_PTR, &pba_ptr);
676     if (ret_val) {
677         DEBUGOUT("NVM Read Error\n");
678         return ret_val;
679     }
680
681     /* if data is not ptr guard the PBA must be in legacy format */
682     if (data != IXGBE_PBANUM_PTR_GUARD) {
683         *pba_num_size = 11;
684         return IXGBE_SUCCESS;
685     }
686
687     ret_val = hw->eeprom.ops.read(hw, pba_ptr, &length);
688     if (ret_val) {
689         DEBUGOUT("NVM Read Error\n");
690         return ret_val;
691     }
692
693     if (length == 0xFFFF || length == 0) {
694         DEBUGOUT("NVM PBA number section invalid length\n");
695         return IXGBE_ERR_PBA_SECTION;
696     }
697
698     /*
699     * Convert from length in u16 values to u8 chars, add 1 for NULL,
700     * and subtract 2 because length field is included in length.
701     */
702     *pba_num_size = ((u32)length * 2) - 1;
703
704     return IXGBE_SUCCESS;
705 }
706
707 /**
708 * ixgbe_read_pba_num_generic - Reads part number from EEPROM
709 * @hw: pointer to hardware structure
710 * @pba_num: stores the part number from the EEPROM
711 */

```

```

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;
658
659     DEBUGFUNC("ixgbe_read_pba_num_generic");
660
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);
670
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;
677
678     return IXGBE_SUCCESS;
679 }
680
681 unchanged portion omitted
682
683 /**
684 * ixgbe_get_bus_info_generic - Generic set PCI bus info
685 * @hw: pointer to hardware structure
686 *
687 * Sets the PCI bus info (speed, width, type) within the ixgbe_hw structure
688 */
689 s32 ixgbe_get_bus_info_generic(struct ixgbe_hw *hw)
690 {
691     struct ixgbe_mac_info *mac = &hw->mac;
692     u16 link_status;
693
694     DEBUGFUNC("ixgbe_get_bus_info_generic");
695
696     hw->bus.type = ixgbe_bus_type_pci_express;
697
698     /* Get the negotiated link width and speed from PCI config space */
699     link_status = IXGBE_READ_PCIE_WORD(hw, IXGBE_PCI_LINK_STATUS);
700
701     switch (link_status & IXGBE_PCI_LINK_WIDTH) {
702     case IXGBE_PCI_LINK_WIDTH_1:
703         hw->bus.width = ixgbe_bus_width_pcie_x1;
704         break;
705     case IXGBE_PCI_LINK_WIDTH_2:
706         hw->bus.width = ixgbe_bus_width_pcie_x2;
707         break;
708     case IXGBE_PCI_LINK_WIDTH_4:
709         hw->bus.width = ixgbe_bus_width_pcie_x4;
710         break;
711     case IXGBE_PCI_LINK_WIDTH_8:
712         hw->bus.width = ixgbe_bus_width_pcie_x8;
713         break;
714     default:
715         hw->bus.width = ixgbe_bus_width_unknown;
716         break;
717     }
718 }

```

```

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 }
760
761 mac->ops.set_lan_id(hw);
762
763 return IXGBE_SUCCESS;
764 }
765
766 unchanged portion omitted
767
768 /**
769 * ixgbe_stop_adapter_generic - Generic stop Tx/Rx units
770 * @hw: pointer to hardware structure
771 *
772 * Sets the adapter_stopped flag within ixgbe_hw struct. Clears interrupts,
773 * disables transmit and receive units. The adapter_stopped flag is used by
774 * the shared code and drivers to determine if the adapter is in a stopped
775 * state and should not touch the hardware.
776 */
777 s32 ixgbe_stop_adapter_generic(struct ixgbe_hw *hw)
778 {
779     u32 number_of_queues;
780     u32 reg_val;
781     u16 i;
782
783     DEBUGFUNC("ixgbe_stop_adapter_generic");
784
785     /*
786      * Set the adapter_stopped flag so other driver functions stop touching
787      * the hardware
788      */
789     hw->adapter_stopped = TRUE;
790
791     /* Disable the receive unit */
792     IXGBE_WRITE_REG(hw, IXGBE_RXCTRL, 0);
793     reg_val = IXGBE_READ_REG(hw, IXGBE_RXCTRL);
794     reg_val &= ~(IXGBE_RXCTRL_RXEN);
795     IXGBE_WRITE_REG(hw, IXGBE_RXCTRL, reg_val);
796     IXGBE_WRITE_FLUSH(hw);
797     msec_delay(2);
798
799     /* Clear interrupt mask to stop interrupts from being generated */
800     /* Clear interrupt mask to stop from interrupts being generated */
801     IXGBE_WRITE_REG(hw, IXGBE_EIMC, IXGBE_IRQ_CLEAR_MASK);
802
803     /* Clear any pending interrupts, flush previous writes */
804     IXGBE_READ_REG(hw, IXGBE_EICR);
805     /* Clear any pending interrupts */
806     (void) IXGBE_READ_REG(hw, IXGBE_EICR);
807
808     /* Disable the transmit unit. Each queue must be disabled. */
809     for (i = 0; i < hw->mac.max_tx_queues; i++)
810         IXGBE_WRITE_REG(hw, IXGBE_TXDCTL(i), IXGBE_TXDCTL_SWFLSH);
811
812     /* Disable the receive unit by stopping each queue */
813 }

```

```

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);
831     }
832     number_of_queues = hw->mac.max_tx_queues;
833     for (i = 0; i < number_of_queues; i++) {
834         reg_val = IXGBE_READ_REG(hw, IXGBE_TXDCTL(i));
835         if (reg_val & IXGBE_TXDCTL_ENABLE) {
836             reg_val &= ~IXGBE_TXDCTL_ENABLE;
837             IXGBE_WRITE_REG(hw, IXGBE_TXDCTL(i), reg_val);
838         }
839     }
840     /* flush all queues disables */
841     IXGBE_WRITE_FLUSH(hw);
842     msec_delay(2);
843
844     /*
845      * Prevent the PCI-E bus from from hanging by disabling PCI-E master
846      * access and verify no pending requests
847      */
848     return ixgbe_disable_pcie_master(hw);
849     (void) ixgbe_disable_pcie_master(hw);
850
851     return IXGBE_SUCCESS;
852 }
853
854 /*
855  * ixgbe_init_eeeprom_params_generic - Initialize EEPROM params
856  * @hw: pointer to hardware structure
857  *
858  * Initializes the EEPROM parameters ixgbe_eeeprom_info within the
859  * ixgbe_hw struct in order to set up EEPROM access.
860  */
861 s32 ixgbe_init_eeeprom_params_generic(struct ixgbe_hw *hw)
862 {
863     struct ixgbe_eeeprom_info *eeeprom = &hw->eeeprom;
864     u32 eec;
865     u16 eeeprom_size;
866
867     DEBUGFUNC("ixgbe_init_eeeprom_params_generic");
868
869     if (eeeprom->type == ixgbe_eeeprom_uninitialized) {
870         eeeprom->type = ixgbe_eeeprom_none;
871         /* Set default semaphore delay to 10ms which is a well
872          * tested value */
873         eeeprom->semaphore_delay = 10;
874         /* Clear EEPROM page size, it will be initialized as needed */
875         eeeprom->word_page_size = 0;
876
877         /*
878          * Check for EEPROM present first.
879          * If not present leave as none
880          */
881         eec = IXGBE_READ_REG(hw, IXGBE_EEC);
882         if (eec & IXGBE_EEC_PRESENT) {
883             eeeprom->type = ixgbe_eeeprom_spi;
884
885             /*
886              * SPI EEPROM is assumed here. This code would need to
887              * change if a future EEPROM is not SPI.
888              */
889             eeeprom_size = (u16)((eec & IXGBE_EEC_SIZE) >>
890                                 IXGBE_EEC_SIZE_SHIFT);

```

```

921         eeeprom->word_size = 1 << (eeeprom_size +
922                                     IXGBE_EEPROM_WORD_SIZE_SHIFT);
923         IXGBE_EEPROM_WORD_SIZE_BASE_SHIFT);
924     }
925
926     if (eec & IXGBE_EEC_ADDR_SIZE)
927         eeeprom->address_bits = 16;
928     else
929         eeeprom->address_bits = 8;
930     DEBUGOUT3("Eeprom params: type = %d, size = %d, address bits: "
931              "%d\n", eeeprom->type, eeeprom->word_size,
932              eeeprom->address_bits);
933 }
934
935     return IXGBE_SUCCESS;
936 }
937
938 /**
939  * ixgbe_write_eeeprom_buffer_bit_bang_generic - Write EEPROM using bit-bang
940  * ixgbe_write_eeeprom_generic - Writes 16 bit value to EEPROM
941  * @hw: pointer to hardware structure
942  * @offset: offset within the EEPROM to write
943  * @words: number of word(s)
944  * @data: 16 bit word(s) to write to EEPROM
945  * @offset: offset within the EEPROM to be written to
946  * @data: 16 bit word to be written to the EEPROM
947  */
948
949 * Reads 16 bit word(s) from EEPROM through bit-bang method
950 * If ixgbe_eeeprom_update_checksum is not called after this function, the
951 * EEPROM will most likely contain an invalid checksum.
952 */
953 s32 ixgbe_write_eeeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
954                                                u16 words, u16 *data)
955 {
956     s32 ixgbe_write_eeeprom_generic(struct ixgbe_hw *hw, u16 offset, u16 data)
957     {
958         s32 status = IXGBE_SUCCESS;
959         u16 i, count;
960         s32 status;
961         u8 write_opcode = IXGBE_EEPROM_WRITE_OPCODE_SPI;
962
963         DEBUGFUNC("ixgbe_write_eeeprom_buffer_bit_bang_generic");
964         DEBUGFUNC("ixgbe_write_eeeprom_generic");
965
966         hw->eeeprom.ops.init_params(hw);
967
968         if (words == 0) {
969             status = IXGBE_ERR_INVALID_ARGUMENT;
970             goto out;
971         }
972
973         if (offset + words > hw->eeeprom.word_size) {
974             if (offset >= hw->eeeprom.word_size) {
975                 status = IXGBE_ERR_EEPROM;
976                 goto out;
977             }
978         }
979
980         /*
981          * The EEPROM page size cannot be queried from the chip. We do lazy
982          * initialization. It is worth to do that when we write large buffer.
983          */
984         if ((hw->eeeprom.word_page_size == 0) &&
985             (words > IXGBE_EEPROM_PAGE_SIZE_MAX))
986             ixgbe_detect_eeeprom_page_size_generic(hw, offset);
987
988         /*
989          * 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
1188              * Some SPI eeproms use the 8th address bit embedded in the
1189              * opcode
1190              */
1189             if ((hw->eeprom.address_bits == 8) &&
1190                 ((offset + i) >= 128))
1191                 if ((hw->eeprom.address_bits == 8) && (offset >= 128))
1192                     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),
1197                                         ixgbe_shift_out_eeprom_bits(hw, (u16)(offset*2),
1198                                                                     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         }
1203         *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

```

```

1215 * @data: read 16 bit value from EEPROM
1216 *
1217 * Reads 16 bit value from EEPROM through bit-bang method
1218 **/
1219 s32 ixgbe_read_eeprom_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
1220                                     u16 *data)
1221 {
1222     s32 status;
1223
1224     DEBUGFUNC("ixgbe_read_eeprom_bit_bang_generic");
1225
1226     hw->eeprom.ops.init_params(hw);
1227
1228     if (offset >= hw->eeprom.word_size) {
1229         status = IXGBE_ERR_EEPROM;
1230         goto out;
1231     }
1232
1233     status = ixgbe_read_eeprom_buffer_bit_bang(hw, offset, 1, data);
1234
1235 out:
1236     return status;
1237 }
1238
1239 /**
1240 * ixgbe_read_eerd_buffer_generic - Read EEPROM word(s) using EERD
1241 * ixgbe_read_eerd_generic - Read EEPROM word using EERD
1242 * @hw: pointer to hardware structure
1243 * @offset: offset of word in the EEPROM to read
1244 * @words: number of word(s)
1245 * @data: 16 bit word(s) from the EEPROM
1246 * @data: word read from the EEPROM
1247 **/
1248 s32 ixgbe_read_eerd_buffer_generic(struct ixgbe_hw *hw, u16 offset,
1249                                   u16 words, u16 *data)
1250 {
1251     u32 eerd;
1252     s32 status = IXGBE_SUCCESS;
1253     u32 i;
1254     s32 status;
1255
1256     DEBUGFUNC("ixgbe_read_eerd_buffer_generic");
1257     DEBUGFUNC("ixgbe_read_eerd_generic");
1258
1259     hw->eeprom.ops.init_params(hw);
1260
1261     if (words == 0) {
1262         status = IXGBE_ERR_INVALID_ARGUMENT;
1263         goto out;
1264     }
1265
1266     if (offset >= hw->eeprom.word_size) {
1267         status = IXGBE_ERR_EEPROM;
1268         goto out;
1269     }
1270
1271     for (i = 0; i < words; i++) {
1272         eerd = ((offset + i) << IXGBE_EEPROM_RW_ADDR_SHIFT) +
1273             (offset << IXGBE_EEPROM_RW_ADDR_SHIFT) +
1274             IXGBE_EEPROM_RW_REG_START;
1275
1276         IXGBE_WRITE_REG(hw, IXGBE_EERD, eerd);

```

```

1274         status = ixgbe_poll_eerd_eewr_done(hw, IXGBE_NVM_POLL_READ);
1275
1276         if (status == IXGBE_SUCCESS) {
1277             data[i] = (IXGBE_READ_REG(hw, IXGBE_EERD) >>
1278                     959
1279                     if (status == IXGBE_SUCCESS)
1280                         *data = (IXGBE_READ_REG(hw, IXGBE_EERD) >>
1281                             IXGBE_EEPROM_RW_REG_DATA);
1282             } else {
1283                 DEBUGOUT("Eeprom read timed out\n");
1284                 goto out;
1285             }
1286         }
1287     }
1288     return status;
1289 }
1290
1291 /**
1292 * ixgbe_detect_eeprom_page_size_generic - Detect EEPROM page size
1293 * @hw: pointer to hardware structure
1294 * @offset: offset within the EEPROM to be used as a scratch pad
1295 * Discover EEPROM page size by writing marching data at given offset.
1296 * This function is called only when we are writing a new large buffer
1297 * at given offset so the data would be overwritten anyway.
1298 **/
1299 static s32 ixgbe_detect_eeprom_page_size_generic(struct ixgbe_hw *hw,
1300                                                  u16 offset)
1301 {
1302     u16 data[IXGBE_EEPROM_PAGE_SIZE_MAX];
1303     s32 status = IXGBE_SUCCESS;
1304     u16 i;
1305
1306     DEBUGFUNC("ixgbe_detect_eeprom_page_size_generic");
1307
1308     for (i = 0; i < IXGBE_EEPROM_PAGE_SIZE_MAX; i++)
1309         data[i] = i;
1310
1311     hw->eeprom.word_page_size = IXGBE_EEPROM_PAGE_SIZE_MAX;
1312     status = ixgbe_write_eeprom_buffer_bit_bang(hw, offset,
1313                                                  IXGBE_EEPROM_PAGE_SIZE_MAX, data);
1314
1315     hw->eeprom.word_page_size = 0;
1316     if (status != IXGBE_SUCCESS)
1317         goto out;
1318
1319     status = ixgbe_read_eeprom_buffer_bit_bang(hw, offset, 1, data);
1320     if (status != IXGBE_SUCCESS)
1321         goto out;
1322
1323     /*
1324      * When writing in burst more than the actual page size
1325      * EEPROM address wraps around current page.
1326      */
1327     hw->eeprom.word_page_size = IXGBE_EEPROM_PAGE_SIZE_MAX - data[0];
1328
1329     DEBUGOUT1("Detected EEPROM page size = %d words.",
1330              hw->eeprom.word_page_size);
1331 out:
1332     return status;
1333 }
1334
1335 /**
1336 * ixgbe_read_eerd_generic - Read EEPROM word using EERD
1337 * ixgbe_write_eewr_generic - Write EEPROM word using EEW
1338 * @hw: pointer to hardware structure
1339 * @offset: offset of word in the EEPROM to read

```

```

1336 * @data: word read from the EEPROM
1337 *
1338 * Reads a 16 bit word from the EEPROM using the EERD register.
1339 **/
1340 s32 ixgbe_read_eerd_generic(struct ixgbe_hw *hw, u16 offset, u16 *data)
1341 {
1342     return ixgbe_read_eerd_buffer_generic(hw, offset, 1, data);
1343 }
1344
1345 /**
1346 * ixgbe_write_eewr_buffer_generic - Write EEPROM word(s) using EEWR
1347 * @hw: pointer to hardware structure
1348 * @offset: offset of word in the EEPROM to write
1349 * @words: number of word(s)
1350 * @data: word(s) write to the EEPROM
1351 * @data: word write to the EEPROM
1352 **/
1353 s32 ixgbe_write_eewr_buffer_generic(struct ixgbe_hw *hw, u16 offset,
1354                                     u16 words, u16 *data)
1355 {
1356     s32 ixgbe_write_eewr_generic(struct ixgbe_hw *hw, u16 offset, u16 data)
1357 {
1358     u32 eewr;
1359     s32 status = IXGBE_SUCCESS;
1360     u16 i;
1361     s32 status;
1362
1363     DEBUGFUNC("ixgbe_write_eewr_generic");
1364
1365     hw->eeprom.ops.init_params(hw);
1366
1367     if (words == 0) {
1368         status = IXGBE_ERR_INVALID_ARGUMENT;
1369         goto out;
1370     }
1371
1372     if (offset >= hw->eeprom.word_size) {
1373         status = IXGBE_ERR_EEPROM;
1374         goto out;
1375     }
1376
1377     for (i = 0; i < words; i++) {
1378         eewr = ((offset + i) << IXGBE_EEPROM_RW_ADDR_SHIFT) |
1379               (data[i] << IXGBE_EEPROM_RW_REG_DATA) |
1380               IXGBE_EEPROM_RW_REG_START;
1381         eewr = (offset << IXGBE_EEPROM_RW_ADDR_SHIFT) |
1382               (data << IXGBE_EEPROM_RW_REG_DATA) | IXGBE_EEPROM_RW_REG_START;
1383
1384         status = ixgbe_poll_eerd_eewr_done(hw, IXGBE_NVM_POLL_WRITE);
1385         if (status != IXGBE_SUCCESS) {
1386             DEBUGOUT("Eeprom write EEWR timed out\n");
1387             goto out;
1388         }
1389
1390         IXGBE_WRITE_REG(hw, IXGBE_EEWR, eewr);
1391
1392         status = ixgbe_poll_eerd_eewr_done(hw, IXGBE_NVM_POLL_WRITE);
1393         if (status != IXGBE_SUCCESS) {
1394             DEBUGOUT("Eeprom write EEWR timed out\n");
1395             goto out;
1396         }
1397     }
1398
1399     out:

```

```

1396     return status;
1397 }
1398
1399 /**
1400 * ixgbe_write_eewr_generic - Write EEPROM word using EEWR
1401 * @hw: pointer to hardware structure
1402 * @offset: offset of word in the EEPROM to write
1403 * @data: word write to the EEPROM
1404 *
1405 * Write a 16 bit word to the EEPROM using the EEWR register.
1406 **/
1407 s32 ixgbe_write_eewr_generic(struct ixgbe_hw *hw, u16 offset, u16 data)
1408 {
1409     return ixgbe_write_eewr_buffer_generic(hw, offset, 1, &data);
1410 }
1411
1412 /**
1413 * ixgbe_poll_eerd_eewr_done - Poll EERD read or EEWR write status
1414 * @hw: pointer to hardware structure
1415 * @ee_reg: EEPROM flag for polling
1416 *
1417 * Polls the status bit (bit 1) of the EERD or EEWR to determine when the
1418 * read or write is done respectively.
1419 **/
1420 s32 ixgbe_poll_eerd_eewr_done(struct ixgbe_hw *hw, u32 ee_reg)
1421 {
1422     u32 i;
1423     u32 reg;
1424     s32 status = IXGBE_ERR_EEPROM;
1425
1426     DEBUGFUNC("ixgbe_poll_eerd_eewr_done");
1427
1428     for (i = 0; i < IXGBE_EERD_EEWR_ATTEMPTS; i++) {
1429         if (ee_reg == IXGBE_NVM_POLL_READ)
1430             reg = IXGBE_READ_REG(hw, IXGBE_EERD);
1431         else
1432             reg = IXGBE_READ_REG(hw, IXGBE_EEWR);
1433
1434         if (reg & IXGBE_EEPROM_RW_REG_DONE) {
1435             status = IXGBE_SUCCESS;
1436             break;
1437         }
1438         usec_delay(5);
1439     }
1440     return status;
1441 }
1442
1443 /**
1444 * ixgbe_acquire_eeprom - Acquire EEPROM using bit-bang
1445 * @hw: pointer to hardware structure
1446 *
1447 * Prepares EEPROM for access using bit-bang method. This function should
1448 * be called before issuing a command to the EEPROM.
1449 **/
1450 static s32 ixgbe_acquire_eeprom(struct ixgbe_hw *hw)
1451 {
1452     s32 status = IXGBE_SUCCESS;
1453     u32 eec;
1454     u32 i;
1455
1456     DEBUGFUNC("ixgbe_acquire_eeprom");
1457
1458     if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM)
1459         != IXGBE_SUCCESS)
1460         if (ixgbe_acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) != IXGBE_SUCCESS)
1461             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);
1483             ixgbe_release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
1484             status = IXGBE_ERR_EEPROM;
1485         }

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_eeeprom_semaphore - Get hardware semaphore
1500  * @hw: pointer to hardware structure
1501  * Sets the hardware semaphores so EEPROM access can occur for bit-bang method
1502  */
1503 static s32 ixgbe_get_eeeprom_semaphore(struct ixgbe_hw *hw)
1504 {
1505     s32 status = IXGBE_ERR_EEPROM;
1506     u32 timeout = 2000;
1507     u32 i;
1508     u32 swsm;

1511     DEBUGFUNC("ixgbe_get_eeeprom_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_eeeprom_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_eeeprom_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_eeeprom - Release EEPROM, release semaphores
1808  * @hw: pointer to hardware structure

```

```

1809 **/
1810 static void ixgbe_release_eeprom(struct ixgbe_hw *hw)
1811 {
1812     u32 eec;
1813
1814     DEBUGFUNC("ixgbe_release_eeprom");
1815
1816     eec = IXGBE_READ_REG(hw, IXGBE_EEC);
1817
1818     eec |= IXGBE_EEC_CS; /* Pull CS high */
1819     eec &= ~IXGBE_EEC_SK; /* Lower SCK */
1820
1821     IXGBE_WRITE_REG(hw, IXGBE_EEC, eec);
1822     IXGBE_WRITE_FLUSH(hw);
1823
1824     usec_delay(1);
1825
1826     /* Stop requesting EEPROM access */
1827     eec &= ~IXGBE_EEC_REQ;
1828     IXGBE_WRITE_REG(hw, IXGBE_EEC, eec);
1829
1830     hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
1831     ixgbe_release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
1832
1833     /* Delay before attempt to obtain semaphore again to allow FW access */
1834     msec_delay(hw->eeprom.semaphore_delay);
1835 }
1836
1837 unchanged_portion_omitted
1838
1839 /**
1840  * ixgbe_init_rx_addrs_generic - Initializes receive address filters.
1841  * @hw: pointer to hardware structure
1842  *
1843  * Places the MAC address in receive address register 0 and clears the rest
1844  * of the receive address registers. Clears the multicast table. Assumes
1845  * the receiver is in reset when the routine is called.
1846  */
1847 s32 ixgbe_init_rx_addrs_generic(struct ixgbe_hw *hw)
1848 {
1849     u32 i;
1850     u32 rar_entries = hw->mac.num_rar_entries;
1851
1852     DEBUGFUNC("ixgbe_init_rx_addrs_generic");
1853
1854     /*
1855      * If the current mac address is valid, assume it is a software override
1856      * to the permanent address.
1857      * Otherwise, use the permanent address from the eeprom.
1858      */
1859     if (ixgbe_validate_mac_addr(hw->mac.addr) ==
1860         IXGBE_ERR_INVALID_MAC_ADDR) {
1861         /* Get the MAC address from the RAR0 for later reference */
1862         hw->mac.ops.get_mac_addr(hw, hw->mac.addr);
1863
1864         DEBUGOUT3(" Keeping Current RAR0 Addr =%.2X %.2X %.2X ",
1865             hw->mac.addr[0], hw->mac.addr[1],
1866             hw->mac.addr[2]);
1867         DEBUGOUT3("%.2X %.2X %.2X\n", hw->mac.addr[3],
1868             hw->mac.addr[4], hw->mac.addr[5]);
1869     } else {
1870         /* Setup the receive address. */
1871         DEBUGOUT("Overriding MAC Address in RAR[0]\n");
1872         DEBUGOUT3(" New MAC Addr =%.2X %.2X %.2X ",
1873             hw->mac.addr[0], hw->mac.addr[1],
1874             hw->mac.addr[2]);
1875         DEBUGOUT3("%.2X %.2X %.2X\n", hw->mac.addr[3],
1876             hw->mac.addr[4], hw->mac.addr[5]);
1877     }
1878 }

```

```

2113         hw->mac.addr[4], hw->mac.addr[5]);
2114
2115         hw->mac.ops.set_rar(hw, 0, hw->mac.addr, 0, IXGBE_RAH_AV);
2116
2117         /* clear VMDq pool/queue selection for RAR 0 */
2118         hw->mac.ops.clear_vmdq(hw, 0, IXGBE_CLEAR_VMDQ_ALL);
2119     }
2120     hw->addr_ctrl.overflow_promisc = 0;
2121
2122     hw->addr_ctrl.rar_used_count = 1;
2123
2124     /* Zero out the other receive addresses. */
2125     DEBUGOUT1("Clearing RAR[1-%d]\n", rar_entries - 1);
2126     for (i = 1; i < rar_entries; i++) {
2127         IXGBE_WRITE_REG(hw, IXGBE_RAL(i), 0);
2128         IXGBE_WRITE_REG(hw, IXGBE_RAH(i), 0);
2129     }
2130
2131     /* Clear the MTA */
2132     hw->addr_ctrl.mta_in_use = 0;
2133     IXGBE_WRITE_REG(hw, IXGBE_MCSTCTRL, hw->mac.mc_filter_type);
2134
2135     DEBUGOUT(" Clearing MTA\n");
2136     for (i = 0; i < hw->mac.mcft_size; i++)
2137         IXGBE_WRITE_REG(hw, IXGBE_MTA(i), 0);
2138
2139     ixgbe_init_uta_tables(hw);
2140     (void) ixgbe_init_uta_tables(hw);
2141
2142     return IXGBE_SUCCESS;
2143 }
2144
2145 unchanged_portion_omitted
2146
2147 /**
2148  * ixgbe_update_mc_addr_list_generic - Updates MAC list of multicast addresses
2149  * @hw: pointer to hardware structure
2150  * @mc_addr_list: the list of new multicast addresses
2151  * @mc_addr_count: number of addresses
2152  * @next: iterator function to walk the multicast address list
2153  * @clear: flag, when set clears the table beforehand
2154  *
2155  * When the clear flag is set, the given list replaces any existing list.
2156  * Hashes the given addresses into the multicast table.
2157  * The given list replaces any existing list. Clears the MC addrs from receive
2158  * address registers and the multicast table. Uses unused receive address
2159  * registers for the first multicast addresses, and hashes the rest into the
2160  * multicast table.
2161  */
2162 s32 ixgbe_update_mc_addr_list_generic(struct ixgbe_hw *hw, u8 *mc_addr_list,
2163     u32 mc_addr_count, ixgbe_mc_addr_itr next,
2164     bool clear)
2165 {
2166     u32 i;
2167     u32 vmdq;
2168
2169     DEBUGFUNC("ixgbe_update_mc_addr_list_generic");
2170
2171     /*
2172      * Set the new number of MC addresses that we are being requested to
2173      * use.
2174      */
2175     hw->addr_ctrl.num_mc_addrs = mc_addr_count;
2176     hw->addr_ctrl.mta_in_use = 0;
2177
2178     /* Clear mta_shadow */
2179 }

```

```

2351     if (clear) {
2352         DEBUGOUT(" Clearing MTA\n");
2353         memset(&hw->mac.mta_shadow, 0, sizeof(hw->mac.mta_shadow));
2354     }
2355     (void) memset(&hw->mac.mta_shadow, 0, sizeof(hw->mac.mta_shadow));
2356
2357     /* Update mta_shadow */
2358     for (i = 0; i < mc_addr_count; i++) {
2359         DEBUGOUT(" Adding the multicast addresses:\n");
2360         ixgbe_set_mta(hw, next(hw, &mc_addr_list, &vmdq));
2361     }
2362
2363     /* Enable mta */
2364     for (i = 0; i < hw->mac.mcft_size; i++)
2365         IXGBE_WRITE_REG_ARRAY(hw, IXGBE_MTA(0), i,
2366                               hw->mac.mta_shadow[i]);
2367
2368     if (hw->addr_ctrl.mta_in_use > 0)
2369         IXGBE_WRITE_REG(hw, IXGBE_MCSTCTRL,
2370                         IXGBE_MCSTCTRL_MFE | hw->mac.mc_filter_type);
2371
2372     DEBUGOUT("ixgbe_update_mc_addr_list_generic Complete\n");
2373     return IXGBE_SUCCESS;
2374 }
2375
2376 unchanged_portion_omitted
2377
2378 /**
2379  * ixgbe_fc_enable_generic - Enable flow control
2380  * @hw: pointer to hardware structure
2381  * @packetbuf_num: packet buffer number (0-7)
2382  */
2383 void ixgbe_fc_enable_generic(struct ixgbe_hw *hw,
2384                             s32 packetbuf_num)
2385 {
2386     s32 ret_val = IXGBE_SUCCESS;
2387     u32 mflcn_reg, fccfg_reg;
2388     u32 rx_pba_size;
2389     u32 fcrtl, fcrtlh;
2390     int i;
2391
2392     DEBUGFUNC("ixgbe_fc_enable_generic");
2393
2394     /* Validate the water mark configuration */
2395     if (!hw->fc.pause_time) {
2396         ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
2397         /* Negotiate the fc mode to use */
2398         ret_val = ixgbe_fc_autoneg(hw);
2399         if (ret_val == IXGBE_ERR_FLOW_CONTROL)
2400             goto out;
2401     }
2402
2403     /* Low water mark of zero causes XOFF floods */
2404     for (i = 0; i < IXGBE_DCB_MAX_TRAFFIC_CLASS; i++) {
2405         if ((hw->fc.current_mode & ixgbe_fc_tx_pause) &&
2406             hw->fc.high_water[i]) {
2407             if (!hw->fc.low_water[i] ||
2408                 hw->fc.low_water[i] >= hw->fc.high_water[i]) {
2409                 DEBUGOUT("Invalid water mark configuration\n");
2410                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
2411                 goto out;
2412             }
2413         }
2414     }
2415 }

```

```

2447     /* Negotiate the fc mode to use */
2448     ixgbe_fc_autoneg(hw);
2449
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);
2453     mflcn_reg &= ~(IXGBE_MFLCN_RFCE | IXGBE_MFLCN_RPFCE);
2454
2455     fccfg_reg = IXGBE_READ_REG(hw, IXGBE_FCCFG);
2456     fccfg_reg &= ~(IXGBE_FCCFG_TFCE_802_3X | IXGBE_FCCFG_TFCE_PRIORITY);
2457
2458     /*
2459      * The possible values of fc.current_mode are:
2460      * 0: Flow control is completely disabled
2461      * 1: Rx flow control is enabled (we can receive pause frames,
2462        but not send pause frames).
2463      * 2: Tx flow control is enabled (we can send pause frames but
2464        we do not support receiving pause frames).
2465      * 3: Both Rx and Tx flow control (symmetric) are enabled.
2466      * other: Invalid.
2467      */
2468     switch (hw->fc.current_mode) {
2469     case ixgbe_fc_none:
2470         /*
2471          * Flow control is disabled by software override or autoneg.
2472          * The code below will actually disable it in the HW.
2473          */
2474         break;
2475     case ixgbe_fc_rx_pause:
2476         /*
2477          * Rx Flow control is enabled and Tx Flow control is
2478          * disabled by software override. Since there really
2479          * isn't a way to advertise that we are capable of RX
2480          * Pause ONLY, we will advertise that we support both
2481          * symmetric and asymmetric Rx PAUSE. Later, we will
2482          * disable the adapter's ability to send PAUSE frames.
2483          */
2484         mflcn_reg |= IXGBE_MFLCN_RFCE;
2485         break;
2486     case ixgbe_fc_tx_pause:
2487         /*
2488          * Tx Flow control is enabled, and Rx Flow control is
2489          * disabled by software override.
2490          */
2491         fccfg_reg |= IXGBE_FCCFG_TFCE_802_3X;
2492         break;
2493     case ixgbe_fc_full:
2494         /* Flow control (both Rx and Tx) is enabled by SW override. */
2495         mflcn_reg |= IXGBE_MFLCN_RFCE;
2496         fccfg_reg |= IXGBE_FCCFG_TFCE_802_3X;
2497         break;
2498     default:
2499         DEBUGOUT("Flow control param set incorrectly\n");
2500         ret_val = IXGBE_ERR_CONFIG;
2501         goto out;
2502     }
2503
2504     /* Set 802.3x based flow control settings. */
2505     mflcn_reg |= IXGBE_MFLCN_DPF;
2506     IXGBE_WRITE_REG(hw, IXGBE_MFLCN, mflcn_reg);
2507     IXGBE_WRITE_REG(hw, IXGBE_FCCFG, fccfg_reg);
2508
2509     rx_pba_size = IXGBE_READ_REG(hw, IXGBE_RXPBSIZE(packetbuf_num));
2510     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     }
2526     fcrth = (rx_pba_size - hw->fc.high_water) << 10;
2527     fcrtl = (rx_pba_size - hw->fc.low_water) << 10;
2528 }
2529
2530 IXGBE_WRITE_REG(hw, IXGBE_FCRTH_82599(i), fcrth);
2531 if (hw->fc.current_mode & ixgbe_fc_tx_pause) {
2532     fcrth |= IXGBE_FCRTH_FCEN;
2533     if (hw->fc.send_xon)
2534         fcrtl |= IXGBE_FCRTL_XONE;
2535 }
2536
2537 IXGBE_WRITE_REG(hw, IXGBE_FCRTH_82599(packetbuf_num), fcrth);
2538 IXGBE_WRITE_REG(hw, IXGBE_FCRTL_82599(packetbuf_num), fcrtl);
2539
2540 /* Configure pause time (2 TCs per register) */
2541 reg = hw->fc.pause_time * 0x00010001;
2542 for (i = 0; i < (IXGBE_DCB_MAX_TRAFFIC_CLASS / 2); i++)
2543     IXGBE_WRITE_REG(hw, IXGBE_FCTTV(i), reg);
2544 reg = IXGBE_READ_REG(hw, IXGBE_FCTTV(packetbuf_num / 2));
2545 if ((packetbuf_num & 1) == 0)
2546     reg = (reg & 0xFFFF0000) | hw->fc.pause_time;
2547 else
2548     reg = (reg & 0x0000FFFF) | (hw->fc.pause_time << 16);
2549 IXGBE_WRITE_REG(hw, IXGBE_FCTTV(packetbuf_num / 2), reg);
2550
2551 /* Configure flow control refresh threshold value */
2552 IXGBE_WRITE_REG(hw, IXGBE_FCRTV, hw->fc.pause_time / 2);
2553 IXGBE_WRITE_REG(hw, IXGBE_FCRTV, (hw->fc.pause_time >> 1));
2554
2555 out:
2556 return ret_val;
2557 }
2558
2559 /**
2560 * ixgbe_negotiate_fc - Negotiate flow control
2561 * ixgbe_fc_autoneg - Configure flow control
2562 * @hw: pointer to hardware structure
2563 * @adv_reg: flow control advertised settings
2564 * @lp_reg: link partner's flow control settings
2565 * @adv_sym: symmetric pause bit in advertisement
2566 * @adv_asm: asymmetric pause bit in advertisement
2567 * @lp_sym: symmetric pause bit in link partner advertisement
2568 * @lp_asm: asymmetric pause bit in link partner advertisement
2569 *
2570 * Find the intersection between advertised settings and link partner's
2571 * advertised settings
2572 * Compares our advertised flow control capabilities to those advertised by
2573 * our link partner, and determines the proper flow control mode to use.
2574 */
2575 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)
2557 {
2558     if ((!(adv_reg)) || (!(lp_reg)))
2559         return IXGBE_ERR_FC_NOT_NEGOTIATED;
2560     s32 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2561     ixgbe_link_speed speed;
2562     bool link_up;
2563
2564     if ((adv_reg & adv_sym) && (lp_reg & lp_sym)) {
2565         DEBUGFUNC("ixgbe_fc_autoneg");
2566         if (hw->fc.disable_fc_autoneg)
2567             goto out;
2568
2569         /*
2570          * Now we need to check if the user selected Rx ONLY
2571          * of pause frames. In this case, we had to advertise
2572          * FULL flow control because we could not advertise RX
2573          * ONLY. Hence, we must now check to see if we need to
2574          * turn OFF the TRANSMISSION of PAUSE frames.
2575          * AN should have completed when the cable was plugged in.
2576          * Look for reasons to bail out. Bail out if:
2577          * - FC autoneg is disabled, or if
2578          * - link is not up.
2579          * Since we're being called from an LSC, link is already known to be up.
2580          * So use link_up_wait_to_complete=FALSE.
2581          */
2582         if (hw->fc.requested_mode == ixgbe_fc_full) {
2583             hw->fc.current_mode = ixgbe_fc_full;
2584             DEBUGOUT("Flow Control = FULL.\n");
2585         } else {
2586             hw->fc.current_mode = ixgbe_fc_rx_pause;
2587             DEBUGOUT("Flow Control=RX PAUSE frames only\n");
2588             hw->mac.ops.check_link(hw, &speed, &link_up, FALSE);
2589             if (!link_up) {
2590                 ret_val = IXGBE_ERR_FLOW_CONTROL;
2591                 goto out;
2592             }
2593         }
2594     } else if (!(adv_reg & adv_sym) && (adv_reg & adv_asm) &&
2595                 (lp_reg & lp_sym) && (lp_reg & lp_asm)) {
2596         hw->fc.current_mode = ixgbe_fc_tx_pause;
2597         DEBUGOUT("Flow Control = TX PAUSE frames only.\n");
2598     } else if ((adv_reg & adv_sym) && (adv_reg & adv_asm) &&
2599                 !(lp_reg & lp_sym) && (lp_reg & lp_asm)) {
2600         hw->fc.current_mode = ixgbe_fc_rx_pause;
2601         DEBUGOUT("Flow Control = RX PAUSE frames only.\n");
2602     }
2603
2604     switch (hw->phy.media_type) {
2605     /* Autoneg flow control on fiber adapters */
2606     case ixgbe_media_type_fiber:
2607         if (speed == IXGBE_LINK_SPEED_1GB_FULL)
2608             ret_val = ixgbe_fc_autoneg_fiber(hw);
2609         break;
2610
2611     /* Autoneg flow control on backplane adapters */
2612     case ixgbe_media_type_backplane:
2613         ret_val = ixgbe_fc_autoneg_backplane(hw);
2614         break;
2615
2616     /* Autoneg flow control on copper adapters */
2617     case ixgbe_media_type_copper:
2618         if (ixgbe_device_supports_autoneg_fc(hw) == IXGBE_SUCCESS)
2619             ret_val = ixgbe_fc_autoneg_copper(hw);
2620         break;
2621     }
2622 }

```

```

2147     default:
2148         break;
2149     }

2151 out:
2152     if (ret_val == IXGBE_SUCCESS) {
2153         hw->fc.fc_was_autonegged = TRUE;
2154     } else {
2155         hw->fc.current_mode = ixgbe_fc_none;
2156         DEBUGOUT("Flow Control = NONE.\n");
2157         hw->fc.fc_was_autonegged = FALSE;
2158         hw->fc.current_mode = hw->fc.requested_mode;
2159     }
2160     return IXGBE_SUCCESS;
2161     return ret_val;
2162 }

2163 /**
2164 * ixgbe_fc_autoneg_fiber - Enable flow control on 1 gig fiber
2165 * @hw: pointer to hardware structure
2166 * @speed:
2167 * @link_up
2168 * Enable flow control according on 1 gig fiber.
2169 */
2170 static s32 ixgbe_fc_autoneg_fiber(struct ixgbe_hw *hw)
2171 {
2172     u32 pcs_anadv_reg, pcs_lpab_reg, linkstat;
2173     s32 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2174     s32 ret_val;

2175     /*
2176      * On multispeed fiber at 1g, bail out if
2177      * - link is up but AN did not complete, or if
2178      * - link is up and AN completed but timed out
2179      */

2180     linkstat = IXGBE_READ_REG(hw, IXGBE_PCS1GLSTA);
2181     if (((linkstat & IXGBE_PCS1GLSTA_AN_COMPLETE) == 0) ||
2182         ((linkstat & IXGBE_PCS1GLSTA_AN_TIMED_OUT) == 1))
2183     if (((linkstat & IXGBE_PCS1GLSTA_AN_COMPLETE) == 0) ||
2184         ((linkstat & IXGBE_PCS1GLSTA_AN_TIMED_OUT) == 1)) {
2185         ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2186         goto out;
2187     }

2188     pcs_anadv_reg = IXGBE_READ_REG(hw, IXGBE_PCS1GANA);
2189     pcs_lpab_reg = IXGBE_READ_REG(hw, IXGBE_PCS1GANLP);

2190     ret_val = ixgbe_negotiate_fc(hw, pcs_anadv_reg,
2191                                pcs_lpab_reg, IXGBE_PCS1GANA_SYM_PAUSE,
2192                                IXGBE_PCS1GANA_ASM_PAUSE,
2193                                IXGBE_PCS1GANA_SYM_PAUSE,
2194                                IXGBE_PCS1GANA_ASM_PAUSE);

2195 out:
2196     return ret_val;
2197 }

2198 /**
2199 * ixgbe_fc_autoneg_backplane - Enable flow control IEEE clause 37
2200 * @hw: pointer to hardware structure
2201 * Enable flow control according to IEEE clause 37.
2202 */

```

```

2203 static s32 ixgbe_fc_autoneg_backplane(struct ixgbe_hw *hw)
2204 {
2205     u32 links2, anlp1_reg, autoc_reg, links;
2206     s32 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2207     s32 ret_val;

2208     /*
2209      * On backplane, bail out if
2210      * - backplane autoneg was not completed, or if
2211      * - we are 82599 and link partner is not AN enabled
2212      */
2213     links = IXGBE_READ_REG(hw, IXGBE_LINKS);
2214     if ((links & IXGBE_LINKS_KX_AN_COMP) == 0)
2215     if ((links & IXGBE_LINKS_KX_AN_COMP) == 0) {
2216         hw->fc.fc_was_autonegged = FALSE;
2217         hw->fc.current_mode = hw->fc.requested_mode;
2218         ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2219         goto out;
2220     }

2221     if (hw->mac.type == ixgbe_mac_82599EB) {
2222         links2 = IXGBE_READ_REG(hw, IXGBE_LINKS2);
2223         if ((links2 & IXGBE_LINKS2_AN_SUPPORTED) == 0)
2224         if ((links2 & IXGBE_LINKS2_AN_SUPPORTED) == 0) {
2225             hw->fc.fc_was_autonegged = FALSE;
2226             hw->fc.current_mode = hw->fc.requested_mode;
2227             ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2228             goto out;
2229         }
2230     }

2231     /*
2232      * Read the 10g AN autoc and LP ability registers and resolve
2233      * local flow control settings accordingly
2234      */
2235     autoc_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC);
2236     anlp1_reg = IXGBE_READ_REG(hw, IXGBE_ANLP1);

2237     ret_val = ixgbe_negotiate_fc(hw, autoc_reg,
2238                                anlp1_reg, IXGBE_AUTOC_SYM_PAUSE, IXGBE_AUTOC_ASM_PAUSE,
2239                                IXGBE_ANLP1_SYM_PAUSE, IXGBE_ANLP1_ASM_PAUSE);

2240 out:
2241     return ret_val;
2242 }
2243
2244 unchanged_portion_omitted

2245 /**
2246 * ixgbe_fc_autoneg - Configure flow control
2247 * ixgbe_negotiate_fc - Negotiate flow control
2248 * @hw: pointer to hardware structure
2249 * @adv_reg: flow control advertised settings
2250 * @lp_reg: link partner's flow control settings
2251 * @adv_sym: symmetric pause bit in advertisement
2252 * @adv_asm: asymmetric pause bit in advertisement
2253 * @lp_sym: symmetric pause bit in link partner advertisement
2254 * @lp_asm: asymmetric pause bit in link partner advertisement
2255 *
2256 * Compares our advertised flow control capabilities to those advertised by
2257 * our link partner, and determines the proper flow control mode to use.
2258 * Find the intersection between advertised settings and link partner's
2259 * advertised settings
2260 */
2261 void ixgbe_fc_autoneg(struct ixgbe_hw *hw)
2262 static s32 ixgbe_negotiate_fc(struct ixgbe_hw *hw, u32 adv_reg, u32 lp_reg,
2263                             u32 adv_sym, u32 adv_asm, u32 lp_sym, u32 lp_asm)
2264 {

```

```

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_ADV,
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);
2411          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);
2441          }
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      }
2457
2458      out:
2459      if (ret_val == IXGBE_SUCCESS) {
2460          hw->fc.fc_was_autonegged = TRUE;
2461      } else {
2462          hw->fc.fc_was_autonegged = FALSE;
2463          hw->fc.current_mode = hw->fc.requested_mode;
2464      }
2465      /*

```

```

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);
2464
2465      /* Disable AN timeout */
2466      if (hw->fc.strict_ieee)
2467          reg &= ~IXGBE_PCS1GLCTL_AN_1G_TIMEOUT_EN;
2468
2469      IXGBE_WRITE_REG(hw, IXGBE_PCS1GLCTL, reg);
2470      DEBUGOUT1("Set up FC; PCS1GLCTL = 0x%08X\n", reg);
2471
2472      /*
2473       * AUTOC restart handles negotiation of 1G and 10G on backplane
2474       * and copper. There is no need to set the PCS1GLCTL register.
2475       */
2476      if (hw->phy.media_type == ixgbe_media_type_backplane) {
2477          reg_bp |= IXGBE_AUTOC_AN_RESTART;
2478          IXGBE_WRITE_REG(hw, IXGBE_AUTOC, reg_bp);
2479      } else if ((hw->phy.media_type == ixgbe_media_type_copper) &&
2480                (ixgbe_device_supports_autoneg_fc(hw) == IXGBE_SUCCESS)) {
2481          hw->phy.ops.write_reg(hw, IXGBE_MDIO_AUTO_NEG_ADVT,
2482                               IXGBE_MDIO_AUTO_NEG_DEV_TYPE, reg_cu);
2483      }
2484
2485      DEBUGOUT1("Set up FC; IXGBE_AUTOC = 0x%08X\n", reg);
2486      out:
2487      return ret_val;
2488  }
2489
2490  /**
2491   * ixgbe_disable_pcie_master - Disable PCI-express master access
2492   * @hw: pointer to hardware structure
2493   *
2494   * Disables PCI-Express master access and verifies there are no pending
2495   * requests. IXGBE_ERR_MASTER_REQUESTS_PENDING is returned if master disable
2496   * bit hasn't caused the master requests to be disabled, else IXGBE_SUCCESS
2497   * is returned signifying master requests disabled.
2498   */
2499  s32 ixgbe_disable_pcie_master(struct ixgbe_hw *hw)
2500  {
2501      s32 status = IXGBE_SUCCESS;
2502      u32 i;
2503      u32 reg_val;
2504      u32 number_of_queues;
2505      s32 status = IXGBE_SUCCESS;
2506
2507      DEBUGFUNC("ixgbe_disable_pcie_master");
2508
2509      /* Always set this bit to ensure any future transactions are blocked */
2510      IXGBE_WRITE_REG(hw, IXGBE_CTRL, IXGBE_CTRL_GIO_DIS);
2511
2512      /* Exit if master requests are blocked */
2513      /* Just jump out if bus mastering is already disabled */
2514      if (!(IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_GIO))
2515          goto out;
2516
2517      /* Poll for master request bit to clear */
2518      /* Disable the receive unit by stopping each queue */
2519      number_of_queues = hw->mac.max_rx_queues;
2520      for (i = 0; i < number_of_queues; i++) {
2521          reg_val = IXGBE_READ_REG(hw, IXGBE_RXDCTL(i));
2522          if (reg_val & IXGBE_RXDCTL_ENABLE) {
2523              reg_val &= ~IXGBE_RXDCTL_ENABLE;
2524          }
2525      }
2526      status = IXGBE_SUCCESS;
2527      goto out;
2528  }

```

```

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;
2777     goto check_device_status;
2778     usec_delay(100);
2779 }

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;
2799     break;
2800     usec_delay(100);
2801 }

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
2814  * Acquires the SWFW semaphore through the GSSR register for the specified
2815  * function (CSR, PHY0, PHY1, EEPROM, Flash)
2816  */
2817 s32 ixgbe_acquire_swfw_sync(struct ixgbe_hw *hw, u16 mask)
2818 {
2819     u32 gssr;
2820     u32 swmask = mask;
2821     u32 fwmask = mask << 5;
2822     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
2864  * Releases the SWFW semaphore through the GSSR register for the specified
2865  * function (CSR, PHY0, PHY1, EEPROM, Flash)
2866  */
2867 void ixgbe_release_swfw_sync(struct ixgbe_hw *hw, u16 mask)
2868 {
2869     u32 gssr;
2870     u32 swmask = mask;

2871     DEBUGFUNC("ixgbe_release_swfw_sync");

2873     ixgbe_get_eeprom_semaphore(hw);
2874     (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_eeeprom_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);
3062     (void) ixgbe_get_san_mac_addr_offset(hw, &san_mac_offset);
3063
3064     if ((san_mac_offset == 0) || (san_mac_offset == 0xFFFF)) {
3065         /*
3066         * No addresses available in this EEPROM. It's not an
3067         * error though, so just wipe the local address and return.
3068         */
3069         for (i = 0; i < 6; i++)
3070             san_mac_addr[i] = 0xFF;
3071
3072         goto san_mac_addr_out;
3073     }
3074
3075     /* make sure we know which port we need to program */
3076     hw->mac.ops.set_lan_id(hw);
3077     /* apply the port offset to the address offset */
3078     (hw->bus.func) ? (san_mac_offset += IXGBE_SAN_MAC_ADDR_PORT1_OFFSET) :
3079     (san_mac_offset += IXGBE_SAN_MAC_ADDR_PORT0_OFFSET);
3080     for (i = 0; i < 3; i++) {
3081         hw->eeprom.ops.read(hw, san_mac_offset, &san_mac_data);
3082         san_mac_addr[i * 2] = (u8)(san_mac_data);
3083         san_mac_addr[i * 2 + 1] = (u8)(san_mac_data >> 8);
3084         san_mac_offset++;
3085     }
3086
3087     san_mac_addr_out:
3088     return IXGBE_SUCCESS;
3089 }
3090
3091 /**
3092 * ixgbe_set_san_mac_addr_generic - Write the SAN MAC address to the EEPROM
3093 * @hw: pointer to hardware structure
3094 * @san_mac_addr: SAN MAC address
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;
3102
3103     DEBUGFUNC("ixgbe_set_san_mac_addr_generic");
3104
3105     /* Look for SAN mac address pointer. If not defined, return */
3106     ixgbe_get_san_mac_addr_offset(hw, &san_mac_offset);
3107     (void) ixgbe_get_san_mac_addr_offset(hw, &san_mac_offset);
3108
3109     if ((san_mac_offset == 0) || (san_mac_offset == 0xFFFF)) {
3110         status = IXGBE_ERR_NO_SAN_ADDR_PTR;
3111         goto san_mac_addr_out;
3112     }
3113
3114     /* Make sure we know which port we need to write */
3115     hw->mac.ops.set_lan_id(hw);
3116     /* Apply the port offset to the address offset */
3117     (hw->bus.func) ? (san_mac_offset += IXGBE_SAN_MAC_ADDR_PORT1_OFFSET) :
3118     (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     }
3125
3126     san_mac_addr_out:
3127     return status;
3128 }
3129
3130 /**
3131 * ixgbe_get_pcie_msix_count_generic - Gets MSI-X vector count
3132 * @hw: pointer to hardware structure
3133 * Read PCIe configuration space, and get the MSI-X vector count from
3134 * the capabilities table.
3135 */
3136 u16 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw)
3137 {
3138     u32 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw)
3139     {
3140         u16 msix_count = 1;
3141         u16 max_msix_count;
3142         u16 pcie_offset;
3143         u32 msix_count = 64;
3144
3145         switch (hw->mac.type) {
3146             case ixgbe_mac_82598EB:
3147                 pcie_offset = IXGBE_PCIE_MSIX_82598_CAPS;
3148                 max_msix_count = IXGBE_MAX_MSIX_VECTORS_82598;
3149                 break;
3150             case ixgbe_mac_82599EB:
3151                 pcie_offset = IXGBE_PCIE_MSIX_82599_CAPS;
3152                 max_msix_count = IXGBE_MAX_MSIX_VECTORS_82599;
3153                 break;
3154             case ixgbe_mac_X540:
3155                 pcie_offset = IXGBE_PCIE_MSIX_X540_CAPS;
3156                 max_msix_count = IXGBE_MAX_MSIX_VECTORS_X540;
3157                 break;
3158             default:
3159                 return msix_count;
3160         }
3161
3162         DEBUGFUNC("ixgbe_get_pcie_msix_count_generic");
3163         msix_count = IXGBE_READ_PCIE_WORD(hw, pcie_offset);
3164         if (hw->mac.msix_vectors_from_pcie) {
3165             msix_count = IXGBE_READ_PCIE_WORD(hw,
3166                 IXGBE_PCIE_MSIX_82599_CAPS);
3167             msix_count &= IXGBE_PCIE_MSIX_TBL_SZ_MASK;
3168
3169             /* MSI-X count is zero-based in HW */
3170             /* MSI-X count is zero-based in HW, so increment to give
3171             * proper value */
3172             msix_count++;
3173         }
3174
3175         if (msix_count > max_msix_count)
3176             msix_count = max_msix_count;
3177
3178         return msix_count;
3179     }
3180 }
3181
3182 /**
3183 * ixgbe_insert_mac_addr_generic - Find a RAR for this mac address
3184 * @hw: pointer to hardware structure
3185 * @addr: Address to put into receive address register
3186 * @vmdq: VMDq pool to assign
3187 */

```

```

3176 * Puts an ethernet address into a receive address register, or
3177 * finds the rar that it is already 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);
3217         (void) ixgbe_set_vmdq(hw, rar, vmdq);
3218     } else if (first_empty_rar != NO_EMPTY_RAR_FOUND) {
3219         /* stick it into first empty RAR slot we found */
3220         rar = first_empty_rar;
3221         ixgbe_set_rar(hw, rar, addr, vmdq, IXGBE_RAH_AV);
3222         (void) ixgbe_set_rar(hw, rar, addr, vmdq, IXGBE_RAH_AV);
3223     } else if (rar == hw->mac.rar_highwater) {
3224         /* add it to the top of the list and inc the highwater mark */
3225         ixgbe_set_rar(hw, rar, addr, vmdq, IXGBE_RAH_AV);
3226         (void) ixgbe_set_rar(hw, rar, addr, vmdq, IXGBE_RAH_AV);
3227         hw->mac.rar_highwater++;
3228     } else if (rar >= hw->mac.num_rar_entries) {
3229         return IXGBE_ERR_INVALID_MAC_ADDR;
3230     }
3231     /*
3232      * If we found rar[0], make sure the default pool bit (we use pool 0)
3233      * remains cleared to be sure default pool packets will get delivered
3234      */
3235     if (rar == 0)
3236         ixgbe_clear_vmdq(hw, rar, 0);
3237     (void) ixgbe_clear_vmdq(hw, rar, 0);
3239     return rar;
3240 }

```

```

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 * ixgbe_set_vmdq_san_mac - Associate default VMDq pool index with a rx address
3325 * @hw: pointer to hardware struct
3326 * @vmdq: VMDq pool index
3327 **/
3328 s32 ixgbe_set_vmdq_san_mac_generic(struct ixgbe_hw *hw, u32 vmdq)
3329 {
3330     u32 rar = hw->mac.san_mac_rar_index;
3331     DEBUGFUNC("ixgbe_set_vmdq_san_mac");
3333     if (vmdq < 32) {
3334         IXGBE_WRITE_REG(hw, IXGBE_MPSAR_LO(rar), 1 << vmdq);
3335         IXGBE_WRITE_REG(hw, IXGBE_MPSAR_HI(rar), 0);
3336     } else {
3337         IXGBE_WRITE_REG(hw, IXGBE_MPSAR_LO(rar), 0);
3338         IXGBE_WRITE_REG(hw, IXGBE_MPSAR_HI(rar), 1 << (vmdq - 32));
3339     }
3340     return IXGBE_SUCCESS;
3341 }
3342 }
3343 }
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 VFVF
3415 * @vfta_on: boolean flag to turn on/off VLAN in VFVF
3416 * Turn on/off specified VLAN in the VLAN filter table.
3417 **/
3418 s32 ixgbe_set_vfta_generic(struct ixgbe_hw *hw, u32 vlan, u32 vind,
3419                             bool vfta_on)
3420 {
3421     s32 regindex;
3422     u32 bitindex;
3423     u32 vfta;
3424     u32 bits;
3425     u32 vt;
3426     u32 targetbit;
3427     s32 ret_val = IXGBE_SUCCESS;
3428     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 VFVFB
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
3499      * If VT Mode is set
3500      *   Either vlan_on
3501      *   make sure the vlan is in VLVF
3502      *   set the vind bit in the matching VLVFB
3503      *   Or !vlan_on
3504      *   clear the pool bit and possibly the vind
3505      */
3506     vt = IXGBE_READ_REG(hw, IXGBE_VT_CTL);
3507     if (vt & IXGBE_VT_CTL_VT_ENABLE) {
3508         s32 vlvf_index;
3509         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));
3519                 bits |= (1 << vind);
3520                 IXGBE_WRITE_REG(hw,
3521                                 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));
3527                 IXGBE_WRITE_REG(hw,
3528                                 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));
3536                 bits &= ~(1 << vind);
3537                 IXGBE_WRITE_REG(hw,
3538                                 IXGBE_VLVFB(vlvf_index * 2),
3539                                 bits);
3540             } else {
3541                 bits = IXGBE_READ_REG(hw,
3542                                     IXGBE_VLVFB((vlvf_index * 2) + 1));
3543                 bits &= ~(1 << (vind - 32));
3544                 IXGBE_WRITE_REG(hw,
3545                                 IXGBE_VLVFB((vlvf_index * 2) + 1),
3546                                 bits);
3547             }
3548         }
3549     }
3550 }

```

```

3548         bits);
3549         bits |= IXGBE_READ_REG(hw,
3550                     IXGBE_VLVFB(vlvf_index * 2));
3551         IXGBE_VLVFB(vlvf_index*2));
3552     }
3553 }
3554
3555 /*
3556  * If there are still bits set in the VLVFB registers
3557  * for the VLAN ID indicated we need to see if the
3558  * caller is requesting that we clear the VFTA entry bit.
3559  * If the caller has requested that we clear the VFTA
3560  * entry bit but there are still pools/VFs using this VLAN
3561  * ID entry then ignore the request. We're not worried
3562  * about the case where we're turning the VFTA VLAN ID
3563  * entry bit on, only when requested to turn it off as
3564  * there may be multiple pools and/or VFs using the
3565  * VLAN ID entry. In that case we cannot clear the
3566  * VFTA bit until all pools/VFs using that VLAN ID have also
3567  * been cleared. This will be indicated by "bits" being
3568  * zero.
3569  */
3570 if (bits) {
3571     IXGBE_WRITE_REG(hw, IXGBE_VLVF(vlvf_index),
3572                     (IXGBE_VLVF_VIEN | vlan));
3573     if ((!vlan_on) && (vfta_changed != NULL)) {
3574         if (!vlan_on) {
3575             /* someone wants to clear the vfta entry
3576              * but some pools/VFs are still using it.
3577              * Ignore it. */
3578             *vfta_changed = FALSE;
3579             vfta_changed = FALSE;
3580         } else
3581             IXGBE_WRITE_REG(hw, IXGBE_VLVF(vlvf_index), 0);
3582     } else
3583         IXGBE_WRITE_REG(hw, IXGBE_VLVF(vlvf_index), 0);
3584 }
3585
3586 if (vfta_changed)
3587     IXGBE_WRITE_REG(hw, IXGBE_VFTA(regindex), vfta);
3588
3589 return IXGBE_SUCCESS;
3590 }
3591
3592 /**
3593  * ixgbe_clear_vfta_generic - Clear VLAN filter table
3594  * @hw: pointer to hardware structure
3595  * Clears the VLAN filter table, and the VMDq index associated with the filter
3596  */
3597 s32 ixgbe_clear_vfta_generic(struct ixgbe_hw *hw)
3598 {
3599     u32 offset;
3600
3601     DEBUGFUNC("ixgbe_clear_vfta_generic");
3602
3603     for (offset = 0; offset < hw->mac.vft_size; offset++)
3604         IXGBE_WRITE_REG(hw, IXGBE_VFTA(offset), 0);
3605
3606     for (offset = 0; offset < IXGBE_VLVF_ENTRIES; offset++) {
3607         IXGBE_WRITE_REG(hw, IXGBE_VLVF(offset), 0);
3608         IXGBE_WRITE_REG(hw, IXGBE_VLVFB(offset * 2), 0);
3609         IXGBE_WRITE_REG(hw, IXGBE_VLVFB((offset * 2) + 1), 0);
3610         IXGBE_WRITE_REG(hw, IXGBE_VLVFB((offset * 2) + 1), 0);
3611         IXGBE_WRITE_REG(hw, IXGBE_VLVFB((offset * 2) + 1), 0);
3612     }

```

```

3604     }
3605
3606     return IXGBE_SUCCESS;
3607 }
3608
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 s32 ixgbe_check_mac_link_generic(struct ixgbe_hw *hw, ixgbe_link_speed *speed,
3618                                 bool *link_up, bool link_up_wait_to_complete)
3619 {
3620     u32 links_reg, links_orig;
3621     u32 i;
3622
3623     DEBUGFUNC("ixgbe_check_mac_link_generic");
3624
3625     /* clear the old state */
3626     links_orig = IXGBE_READ_REG(hw, IXGBE_LINKS);
3627
3628     links_reg = IXGBE_READ_REG(hw, IXGBE_LINKS);
3629
3630     if (links_orig != links_reg) {
3631         DEBUGOUT2("LINKS changed from %08X to %08X\n",
3632                 links_orig, links_reg);
3633     }
3634
3635     if (link_up_wait_to_complete) {
3636         for (i = 0; i < IXGBE_LINK_UP_TIME; i++) {
3637             if (links_reg & IXGBE_LINKS_UP) {
3638                 *link_up = TRUE;
3639                 break;
3640             } else {
3641                 *link_up = FALSE;
3642                 msec_delay(100);
3643                 links_reg = IXGBE_READ_REG(hw, IXGBE_LINKS);
3644             }
3645         }
3646     } else {
3647         if (links_reg & IXGBE_LINKS_UP)
3648             *link_up = TRUE;
3649         else
3650             *link_up = FALSE;
3651     }
3652
3653     if ((links_reg & IXGBE_LINKS_SPEED_82599) ==
3654         IXGBE_LINKS_SPEED_10G_82599)
3655         *speed = IXGBE_LINK_SPEED_10GB_FULL;
3656     else if ((links_reg & IXGBE_LINKS_SPEED_82599) ==
3657             IXGBE_LINKS_SPEED_1G_82599)
3658         *speed = IXGBE_LINK_SPEED_1GB_FULL;
3659     else if ((links_reg & IXGBE_LINKS_SPEED_82599) ==
3660             IXGBE_LINKS_SPEED_100_82599)
3661         *speed = IXGBE_LINK_SPEED_100_FULL;
3662     else
3663         *speed = IXGBE_LINK_SPEED_UNKNOWN;
3664
3665     /* if link is down, zero out the current_mode */
3666     if (*link_up == FALSE) {
3667         hw->fc.current_mode = ixgbe_fc_none;
3668         hw->fc.fc_was_autonegged = FALSE;
3669     }

```

```

3300     }

3666     return IXGBE_SUCCESS;
3667 }
    unchanged_portion_omitted_

3765 /**
3766  * ixgbe_device_supports_autoneg_fc - Check if phy supports autoneg flow
3767  * control
3768  * @hw: pointer to hardware structure
3769  * There are several phys that do not support autoneg flow control. This
3770  * function check the device id to see if the associated phy supports
3771  * autoneg flow control.
3772  */
3773 static s32 ixgbe_device_supports_autoneg_fc(struct ixgbe_hw *hw)
3774 {
3775     DEBUGFUNC("ixgbe_device_supports_autoneg_fc");

3776     switch (hw->device_id) {
3777     case IXGBE_DEV_ID_82599_T3_LOM:
3778         return IXGBE_SUCCESS;
3779     default:
3780         return IXGBE_ERR_FC_NOT_SUPPORTED;
3781     }
3782 }

3783 /**
3784  * ixgbe_set_mac_anti_spoofing - Enable/Disable MAC anti-spoofing
3785  * @hw: pointer to hardware structure
3786  * @enable: enable or disable switch for anti-spoofing
3787  * @pf: Physical Function pool - do not enable anti-spoofing for the PF
3788  */
3789 void ixgbe_set_mac_anti_spoofing(struct ixgbe_hw *hw, bool enable, int pf)
3790 {
3791     int j;
3792     int pf_target_reg = pf >> 3;
3793     int pf_target_shift = pf % 8;
3794     u32 pfvfspooof = 0;

3795     if (hw->mac.type == ixgbe_mac_82598EB)
3796         return;

3797     if (enable)
3798         pfvfspooof = IXGBE_SPOOF_MACAS_MASK;

3799     /*
3800      * PFVFSPOOF register array is size 8 with 8 bits assigned to
3801      * MAC anti-spoof enables in each register array element.
3802      */
3803     for (j = 0; j < pf_target_reg; j++)
3804         for (j = 0; j < IXGBE_PVFSP00F_REG_COUNT; j++)
3805             IXGBE_WRITE_REG(hw, IXGBE_PVFSP00F(j), pfvfspooof);

3806     /* If not enabling anti-spoofing then done */
3807     if (!enable)
3808         return;

3809     /*
3810      * The PF should be allowed to spoof so that it can support
3811      * emulation mode NICs. Do not set the bits assigned to the PF
3812      * emulation mode NICs. Reset the bit assigned to the PF
3813      */
3814     pfvfspooof &= (1 << pf_target_shift) - 1;

```

```

3797     IXGBE_WRITE_REG(hw, IXGBE_PVFSP00F(j), pfvfspooof);

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_PVFSP00F_REG_COUNT; j++)
3804         IXGBE_WRITE_REG(hw, IXGBE_PVFSP00F(j), 0);
3805     pfvfspooof = IXGBE_READ_REG(hw, IXGBE_PVFSP00F(pf_target_reg));
3806     pfvfspooof ^= (1 << pf_target_shift);
3807     IXGBE_WRITE_REG(hw, IXGBE_PVFSP00F(pf_target_reg), pfvfspooof);
    unchanged_portion_omitted_

3848 /**
3849  * ixgbe_enable_relaxed_ordering_gen2 - Enable relaxed ordering
3850  * @hw: pointer to hardware structure
3851  */
3852 void ixgbe_enable_relaxed_ordering_gen2(struct ixgbe_hw *hw)
3853 {
3854     u32 regval;
3855     u32 i;

3856     DEBUGFUNC("ixgbe_enable_relaxed_ordering_gen2");

3857     /* Enable relaxed ordering */
3858     for (i = 0; i < hw->mac.max_tx_queues; i++) {
3859         regval = IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL_82599(i));
3860         regval |= IXGBE_DCA_TXCTRL_DESC_WRO_EN;
3861         regval |= IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
3862         IXGBE_WRITE_REG(hw, IXGBE_DCA_TXCTRL_82599(i), regval);
3863     }

3864     for (i = 0; i < hw->mac.max_rx_queues; i++) {
3865         regval = IXGBE_READ_REG(hw, IXGBE_DCA_RXCTRL(i));
3866         regval |= IXGBE_DCA_RXCTRL_DATA_WRO_EN |
3867                 IXGBE_DCA_RXCTRL_HEAD_WRO_EN;
3868         regval |= (IXGBE_DCA_RXCTRL_DESC_WRO_EN |
3869                 IXGBE_DCA_RXCTRL_DESC_HSR0_EN);
3870         IXGBE_WRITE_REG(hw, IXGBE_DCA_RXCTRL(i), regval);
3871     }
3872 }

3873 }

3874 /**
3875  * ixgbe_calculate_checksum - Calculate checksum for buffer
3876  * @buffer: pointer to EEPROM
3877  * @length: size of EEPROM to calculate a checksum for
3878  * Calculates the checksum for some buffer on a specified length. The
3879  * checksum calculated is returned.
3880  */
3881 static u8 ixgbe_calculate_checksum(u8 *buffer, u32 length)
3882 {
3883     u32 i;
3884     u8 sum = 0;

3885     DEBUGFUNC("ixgbe_calculate_checksum");

3886     if (!buffer)
3887         return 0;

3888     for (i = 0; i < length; i++)
3889         sum += buffer[i];

3890     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 hcr, 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         DEBUGGOUT("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     hcr = IXGBE_READ_REG(hw, IXGBE_HICR);
3929     if ((hcr & IXGBE_HICR_EN) == 0) {
3930         DEBUGGOUT("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, hcr | IXGBE_HICR_C);

3949     for (i = 0; i < IXGBE_HI_COMMAND_TIMEOUT; i++) {
3950         hcr = IXGBE_READ_REG(hw, IXGBE_HICR);
3951         if (!(hcr & 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         DEBUGGOUT("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         DEBUGGOUT("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 rxpktsz, txpktsz, 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         rxpktsz = (pbsize * 5) / (num_pb * 4);
4088         pbsize -= rxpktsz * (num_pb / 2);
4089         rxpktsz <= IXGBE_RXPBSIZE_SHIFT;
4090         for (; i < (num_pb / 2); i++)
4091             IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), rxpktsz);
4092         /* Fall through to configure remaining packet buffers */
4093     case PBA_STRATEGY_EQUAL:
4094         rxpktsz = (pbsize / (num_pb - i)) <= IXGBE_RXPBSIZE_SHIFT;

```

```

4095         for (; i < num_pb; i++)
4096             IXGBE_WRITE_REG(hw, IXGBE_RXPBSIZE(i), rxpktsz);
4097         break;
4098     default:
4099         break;
4100     }

4102     /* Only support an equally distributed Tx packet buffer strategy. */
4103     txpktsz = IXGBE_TXPBSIZE_MAX / num_pb;
4104     txpbthresh = (txpktsz / 1024) - IXGBE_TXPKT_SIZE_MAX;
4105     for (i = 0; i < num_pb; i++) {
4106         IXGBE_WRITE_REG(hw, IXGBE_TXPBSIZE(i), txpktsz);
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 }

```

new/usr/src/uts/common/io/ixgbe/ixgbe_common.h

1

```
*****
6867 Fri Jul 20 10:59:36 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_common.h
XXX Intel X540 support
*****
1 /*****

3 Copyright (c) 2001-2012, Intel Corporation
3 Copyright (c) 2001-2010, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.

32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_common.h,v 1.12 2012/07/05 20:51:44 jfv Exp
33 /*$FreeBSD$*/

35 #ifndef _IXGBE_COMMON_H_
36 #define _IXGBE_COMMON_H_

38 #include "ixgbe_type.h"
39 #define IXGBE_WRITE_REG(hw, reg, value) \
40     do { \
41         { \
42             IXGBE_WRITE_REG(hw, reg, (u32) value); \
43             IXGBE_WRITE_REG(hw, reg + 4, (u32) (value >> 32)); \
44         } while (0)
45     }

45 u16 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw);
45 u32 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw);

47 s32 ixgbe_init_ops_generic(struct ixgbe_hw *hw);
48 s32 ixgbe_init_hw_generic(struct ixgbe_hw *hw);
49 s32 ixgbe_start_hw_generic(struct ixgbe_hw *hw);
50 s32 ixgbe_start_hw_gen2(struct ixgbe_hw *hw);
51 s32 ixgbe_clear_hw_cntrs_generic(struct ixgbe_hw *hw);
52 s32 ixgbe_read_pba_num_generic(struct ixgbe_hw *hw, u32 *pba_num);
53 s32 ixgbe_read_pba_string_generic(struct ixgbe_hw *hw, u8 *pba_num,
54                                 u32 pba_num_size);
55 s32 ixgbe_read_pba_length_generic(struct ixgbe_hw *hw, u32 *pba_num_size);
55 s32 ixgbe_get_mac_addr_generic(struct ixgbe_hw *hw, u8 *mac_addr);
```

new/usr/src/uts/common/io/ixgbe/ixgbe_common.h

2

```
56 s32 ixgbe_get_bus_info_generic(struct ixgbe_hw *hw);
57 void ixgbe_set_lan_id_multi_port_pcie(struct ixgbe_hw *hw);
58 s32 ixgbe_stop_adapter_generic(struct ixgbe_hw *hw);

60 s32 ixgbe_led_on_generic(struct ixgbe_hw *hw, u32 index);
61 s32 ixgbe_led_off_generic(struct ixgbe_hw *hw, u32 index);

63 s32 ixgbe_init_eeeprom_params_generic(struct ixgbe_hw *hw);
64 s32 ixgbe_write_eeeprom_generic(struct ixgbe_hw *hw, u16 offset, u16 data);
65 s32 ixgbe_write_eeeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
66                                                u16 words, u16 *data);
67 s32 ixgbe_read_eerd_generic(struct ixgbe_hw *hw, u16 offset, u16 *data);
68 s32 ixgbe_read_eerd_buffer_generic(struct ixgbe_hw *hw, u16 offset,
69                                   u16 words, u16 *data);
70 s32 ixgbe_write_eewr_generic(struct ixgbe_hw *hw, u16 offset, u16 data);
71 s32 ixgbe_write_eewr_buffer_generic(struct ixgbe_hw *hw, u16 offset,
72                                   u16 words, u16 *data);
73 s32 ixgbe_read_eeeprom_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
74                                         u16 *data);
75 s32 ixgbe_read_eeeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
76                                                u16 words, u16 *data);
77 u16 ixgbe_calc_eeeprom_checksum_generic(struct ixgbe_hw *hw);
78 s32 ixgbe_validate_eeeprom_checksum_generic(struct ixgbe_hw *hw,
79                                             u16 *checksum_val);
80 s32 ixgbe_update_eeeprom_checksum_generic(struct ixgbe_hw *hw);
81 s32 ixgbe_poll_eerd_eewr_done(struct ixgbe_hw *hw, u32 ee_reg);

83 s32 ixgbe_set_rar_generic(struct ixgbe_hw *hw, u32 index, u8 *addr, u32 vmdq,
84                           u32 enable_addr);
85 s32 ixgbe_clear_rar_generic(struct ixgbe_hw *hw, u32 index);
86 s32 ixgbe_init_rx_addrs_generic(struct ixgbe_hw *hw);
87 s32 ixgbe_update_mc_addr_list_generic(struct ixgbe_hw *hw, u8 *mc_addr_list,
88                                       u32 mc_addr_count,
89                                       ixgbe_mc_addr_itr_func, bool clear);
89 ixgbe_mc_addr_itr_func);
90 s32 ixgbe_update_uc_addr_list_generic(struct ixgbe_hw *hw, u8 *addr_list,
91                                       u32 addr_count, ixgbe_mc_addr_itr_func);
92 s32 ixgbe_enable_mc_generic(struct ixgbe_hw *hw);
93 s32 ixgbe_disable_mc_generic(struct ixgbe_hw *hw);
94 s32 ixgbe_enable_rx_dma_generic(struct ixgbe_hw *hw, u32 regval);
95 s32 ixgbe_disable_sec_rx_path_generic(struct ixgbe_hw *hw);
96 s32 ixgbe_enable_sec_rx_path_generic(struct ixgbe_hw *hw);

98 s32 ixgbe_fc_enable_generic(struct ixgbe_hw *hw);
99 void ixgbe_fc_autoneg(struct ixgbe_hw *hw);
99 s32 ixgbe_setup_fc(struct ixgbe_hw *hw, s32 packetbuf_num);
99 s32 ixgbe_fc_enable_generic(struct ixgbe_hw *hw, s32 packetbuf_num);
99 s32 ixgbe_fc_autoneg(struct ixgbe_hw *hw);

101 s32 ixgbe_validate_mac_addr(u8 *mac_addr);
102 s32 ixgbe_acquire_swfw_sync(struct ixgbe_hw *hw, u16 mask);
103 void ixgbe_release_swfw_sync(struct ixgbe_hw *hw, u16 mask);
104 s32 ixgbe_disable_pcie_master(struct ixgbe_hw *hw);

106 s32 ixgbe_blink_led_start_generic(struct ixgbe_hw *hw, u32 index);
107 s32 ixgbe_blink_led_stop_generic(struct ixgbe_hw *hw, u32 index);

109 s32 ixgbe_get_san_mac_addr_generic(struct ixgbe_hw *hw, u8 *san_mac_addr);
110 s32 ixgbe_set_san_mac_addr_generic(struct ixgbe_hw *hw, u8 *san_mac_addr);

112 s32 ixgbe_set_vmdq_generic(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
113 s32 ixgbe_set_vmdq_san_mac_generic(struct ixgbe_hw *hw, u32 vmdq);
114 s32 ixgbe_clear_vmdq_generic(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
115 s32 ixgbe_insert_mac_addr_generic(struct ixgbe_hw *hw, u8 *addr, u32 vmdq);
116 s32 ixgbe_init_uta_tables_generic(struct ixgbe_hw *hw);
117 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 */
```

new/usr/src/uts/common/io/ixgbe/ixgbe_gld.c

1

18624 Fri Jul 20 10:59:36 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 ||
```

new/usr/src/uts/common/io/ixgbe/ixgbe_gld.c

2

```
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 interrupt throttling
714          * rate is a multiple of 8. This is enforced by the
715          * register definition.
716          * 82599 requires the interrupt throttling rate is
717          * a multiple of 8. This is enforced by the register
718          * definition.
719          */
720         if (hw->mac.type >= ixgbe_mac_82599EB)
721             if (hw->mac.type == ixgbe_mac_82599EB)
722                 ixgbe->intr_throttling[0] =
723                     ixgbe->intr_throttling[0] & 0xFF8;

725         for (i = 0; i < MAX_INTR_VECTOR; i++)
726             ixgbe->intr_throttling[i] =
727                 ixgbe->intr_throttling[0];

729         /* Set interrupt throttling rate */
```

new/usr/src/uts/common/io/ixgbe/ixgbe_gld.c

3

```
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_____
```

```

*****
142010 Fri Jul 20 10:59:37 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_main.c
XXX 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 prototypes
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_overtmp_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     (IXGBE_EICR_LSC
314      | IXGBE_EICR_GPI_SDP1
315      | IXGBE_EICR_GPI_SDP2), /* "other" interrupt types handled */
317     (IXGBE_SDP1_GPIEN
318      | IXGBE_SDP2_GPIEN), /* "other" interrupt types enable mask */
320     (IXGBE_FLAG_DCA_CAPABLE
321      | IXGBE_FLAG_RSS_CAPABLE
322      | IXGBE_FLAG_VMDQ_CAPABLE
323      | IXGBE_FLAG_RSC_CAPABLE) /* capability flags */
324 };

326 /*
327  * Module Initialization Functions.
328  */

330 int
331 _init(void)
332 {
333     int status;

335     mac_init_ops(&ixgbe_dev_ops, MODULE_NAME);

337     status = mod_install(&ixgbe_modlinkage);

339     if (status != DDI_SUCCESS) {
340         mac_fini_ops(&ixgbe_dev_ops);
341     }

343     return (status);
344 }
    unchanged_portion_omitted_

```

```

846 /*
847  * ixgbe_identify_hardware - Identify the type of the chipset.
848  */
849 static int
850 ixgbe_identify_hardware(ixgbe_t *ixgbe)
851 {
852     struct ixgbe_hw *hw = &ixgbe->hw;
853     struct ixgbe_osdep *osdep = &ixgbe->osdep;

855     /*
856      * Get the device id
857      */
858     hw->vendor_id =
859         pci_config_get16(osdep->cfg_handle, PCI_CONF_VENID);
860     hw->device_id =
861         pci_config_get16(osdep->cfg_handle, PCI_CONF_DEVID);
862     hw->revision_id =
863         pci_config_get8(osdep->cfg_handle, PCI_CONF_REVID);
864     hw->subsystem_device_id =
865         pci_config_get16(osdep->cfg_handle, PCI_CONF_SUBSYSID);
866     hw->subsystem_vendor_id =
867         pci_config_get16(osdep->cfg_handle, PCI_CONF_SUBVENID);

869     /*
870      * Set the mac type of the adapter based on the device id
871      */
872     if (ixgbe_set_mac_type(hw) != IXGBE_SUCCESS) {
873         return (IXGBE_FAILURE);
874     }

876     /*
877      * Install adapter capabilities
878      */
879     switch (hw->mac.type) {
880     case ixgbe_mac_82598EB:
881         IXGBE_DEBUGLOG_0(ixgbe, "identify 82598 adapter\n");
882         ixgbe->capab = &ixgbe_82598eb_cap;

884         if (ixgbe_get_media_type(hw) == ixgbe_media_type_copper) {
885             ixgbe->capab->flags |= IXGBE_FLAG_FAN_FAIL_CAPABLE;
886             ixgbe->capab->other_intr |= IXGBE_EICR_GPI_SDP1;
887             ixgbe->capab->other_gpie |= IXGBE_SDP1_GPIEN;
888         }
889         break;

891     case ixgbe_mac_82599EB:
892         IXGBE_DEBUGLOG_0(ixgbe, "identify 82599 adapter\n");
893         ixgbe->capab = &ixgbe_82599eb_cap;

895         if (hw->device_id == IXGBE_DEV_ID_82599_T3_LOM) {
896             ixgbe->capab->flags |= IXGBE_FLAG_TEMP_SENSOR_CAPABLE;
897             ixgbe->capab->other_intr |= IXGBE_EICR_GPI_SDP0;
898             ixgbe->capab->other_gpie |= IXGBE_SDP0_GPIEN;
899         }
900         break;

902     case ixgbe_mac_X540:
903         IXGBE_DEBUGLOG_0(ixgbe, "identify X540 adapter\n");
904         ixgbe->capab = &ixgbe_X540_cap;
905         /*
906          * For now, X540 is all set in its capab structure.
907          * As other X540 variants show up, things can change here.
908          */
909         break;

911     default:

```

```

912         IXGBE_DEBUGLOG_1(ixgbe,
913         "adapter not supported in ixgbe_identify_hardware(): %d\n",
914         hw->mac.type);
915         return (IXGBE_FAILURE);
916     }

918     return (IXGBE_SUCCESS);
919 }
    unchanged_portion_omitted

1193 /*
1194  * ixgbe_init - Initialize the device.
1195  */
1196 static int
1197 ixgbe_init(ixgbe_t *ixgbe)
1198 {
1199     struct ixgbe_hw *hw = &ixgbe->hw;

1201     mutex_enter(&ixgbe->gen_lock);

1203     /*
1204      * Reset chipset to put the hardware in a known state
1205      * before we try to do anything with the eeprom.
1206      */
1207     if (ixgbe_reset_hw(hw) != IXGBE_SUCCESS) {
1208         ixgbe_fm_ereport(ixgbe, DDI_FM_DEVICE_INVALID_STATE);
1209         goto init_fail;
1210     }

1212     /*
1213      * Need to init eeprom before validating the checksum.
1214      */
1215     if (ixgbe_init_eeprom_params(hw) < 0) {
1216         ixgbe_error(ixgbe,
1217         "Unable to initialize the eeprom interface.");
1218         ixgbe_fm_ereport(ixgbe, DDI_FM_DEVICE_INVALID_STATE);
1219         goto init_fail;
1220     }

1222     /*
1223      * NVM validation
1224      */
1225     if (ixgbe_validate_eeprom_checksum(hw, NULL) < 0) {
1226         /*
1227          * Some PCI-E parts fail the first check due to
1228          * the link being in sleep state. Call it again,
1229          * if it fails a second time it's a real issue.
1230          */
1231         if (ixgbe_validate_eeprom_checksum(hw, NULL) < 0) {
1232             ixgbe_error(ixgbe,
1233             "Invalid NVM checksum. Please contact "
1234             "the vendor to update the NVM.");
1235             ixgbe_fm_ereport(ixgbe, DDI_FM_DEVICE_INVALID_STATE);
1236             goto init_fail;
1237         }
1238     }

1240     /*
1241      * Setup default flow control thresholds - enable/disable
1242      * & flow control type is controlled by ixgbe.conf
1243      */
1244     hw->fc.high_water[0] = DEFAULT_FCRTL;
1245     hw->fc.low_water[0] = DEFAULT_FCRTL;
1246     hw->fc.high_water = DEFAULT_FCRTL;
1247     hw->fc.low_water = DEFAULT_FCRTL;
1248     hw->fc.pause_time = DEFAULT_FCPAUSE;

```

```

1247     hw->fc.send_xon = B_TRUE;

1249     /*
1250      * Initialize link settings
1251      */
1252     (void) ixgbe_driver_setup_link(ixgbe, B_FALSE);

1254     /*
1255      * Initialize the chipset hardware
1256      */
1257     if (ixgbe_chip_start(ixgbe) != IXGBE_SUCCESS) {
1258         ixgbe_fm_ereport(ixgbe, DDI_FM_DEVICE_INVALID_STATE);
1259         goto init_fail;
1260     }

1262     if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) != DDI_FM_OK) {
1263         goto init_fail;
1264     }

1266     mutex_exit(&ixgbe->gen_lock);
1267     return (IXGBE_SUCCESS);

1269 init_fail:
1270     /*
1271      * Reset PHY
1272      */
1273     (void) ixgbe_reset_phy(hw);

1275     mutex_exit(&ixgbe->gen_lock);
1276     ddi_fm_service_impact(ixgbe->dip, DDI_SERVICE_LOST);
1277     return (IXGBE_FAILURE);
1278 }
    unchanged_portion_omitted

2076 static void
2077 ixgbe_setup_rx_ring(ixgbe_rx_ring_t *rx_ring)
2078 {
2079     ixgbe_t *ixgbe = rx_ring->ixgbe;
2080     ixgbe_rx_data_t *rx_data = rx_ring->rx_data;
2081     struct ixgbe_hw *hw = &ixgbe->hw;
2082     rx_control_block_t *rcb;
2083     union ixgbe_adv_rx_desc *rbd;
2084     uint32_t size;
2085     uint32_t buf_low;
2086     uint32_t buf_high;
2087     uint32_t reg_val;
2088     int i;

2090     ASSERT(mutex_owned(&rx_ring->rx_lock));
2091     ASSERT(mutex_owned(&ixgbe->gen_lock));

2093     for (i = 0; i < ixgbe->rx_ring_size; i++) {
2094         rcb = rx_data->work_list[i];
2095         rbd = &rx_data->rbd_ring[i];

2097         rbd->read.pkt_addr = rcb->rx_buf.dma_address;
2098         rbd->read.hdr_addr = NULL;
2099     }

2101     /*
2102      * Initialize the length register
2103      */
2104     size = rx_data->ring_size * sizeof (union ixgbe_adv_rx_desc);
2105     IXGBE_WRITE_REG(hw, IXGBE_RDLEN(rx_ring->hw_index), size);

2107     /*

```

```

2108     * Initialize the base address registers
2109     */
2110     buf_low = (uint32_t)rx_data->rbd_area.dma_address;
2111     buf_high = (uint32_t)(rx_data->rbd_area.dma_address >> 32);
2112     IXGBE_WRITE_REG(hw, IXGBE_RDBAH(rx_ring->hw_index), buf_high);
2113     IXGBE_WRITE_REG(hw, IXGBE_RDBAL(rx_ring->hw_index), buf_low);

2115     /*
2116     * Setup head & tail pointers
2117     */
2118     IXGBE_WRITE_REG(hw, IXGBE_RDT(rx_ring->hw_index),
2119         rx_data->ring_size - 1);
2120     IXGBE_WRITE_REG(hw, IXGBE_RDH(rx_ring->hw_index), 0);

2122     rx_data->rbd_next = 0;
2123     rx_data->lro_first = 0;

2125     /*
2126     * Setup the Receive Descriptor Control Register (RXDCTL)
2127     * PTHRESH=32 descriptors (half the internal cache)
2128     * HTHRESH=0 descriptors (to minimize latency on fetch)
2129     * WTHRESH defaults to 1 (writeback each descriptor)
2130     */
2131     reg_val = IXGBE_READ_REG(hw, IXGBE_RXDCTL(rx_ring->hw_index));
2132     reg_val |= IXGBE_RXDCTL_ENABLE; /* enable queue */

2134     /* Not a valid value for 82599 */
2135     if (hw->mac.type < ixgbe_mac_82599EB) {
2136         reg_val |= 0x0020; /* pthresh */
2137     }
2138     IXGBE_WRITE_REG(hw, IXGBE_RXDCTL(rx_ring->hw_index), reg_val);

2140     if (hw->mac.type >= ixgbe_mac_82599EB) {
2099     if (hw->mac.type == ixgbe_mac_82599EB) {
2141         reg_val = IXGBE_READ_REG(hw, IXGBE_RDRXCTL);
2142         reg_val |= (IXGBE_RDRXCTL_CRCSTRIP | IXGBE_RDRXCTL_AGGDIS);
2143         IXGBE_WRITE_REG(hw, IXGBE_RDRXCTL, reg_val);
2144     }

2146     /*
2147     * Setup the Split and Replication Receive Control Register.
2148     * Set the rx buffer size and the advanced descriptor type.
2149     */
2150     reg_val = (ixgbe->rx_buf_size >> IXGBE_SRRCTL_BSIZEPKT_SHIFT) |
2151         IXGBE_SRRCTL_DESCTYPE_ADV_ONEBUF;
2152     reg_val |= IXGBE_SRRCTL_DROP_EN;
2153     IXGBE_WRITE_REG(hw, IXGBE_SRRCTL(rx_ring->hw_index), reg_val);
2154 }

    unchanged_portion_omitted

2317 static void
2318 ixgbe_setup_tx_ring(ixgbe_tx_ring_t *tx_ring)
2319 {
2320     ixgbe_t *ixgbe = tx_ring->ixgbe;
2321     struct ixgbe_hw *hw = &ixgbe->hw;
2322     uint32_t size;
2323     uint32_t buf_low;
2324     uint32_t buf_high;
2325     uint32_t reg_val;

2327     ASSERT(mutex_owned(&tx_ring->tx_lock));
2328     ASSERT(mutex_owned(&ixgbe->gen_lock));

2330     /*
2331     * Initialize the length register
2332     */

```

```

2333     size = tx_ring->ring_size * sizeof(union ixgbe_adv_tx_desc);
2334     IXGBE_WRITE_REG(hw, IXGBE_TDLN(tx_ring->index), size);

2336     /*
2337     * Initialize the base address registers
2338     */
2339     buf_low = (uint32_t)tx_ring->tbd_area.dma_address;
2340     buf_high = (uint32_t)(tx_ring->tbd_area.dma_address >> 32);
2341     IXGBE_WRITE_REG(hw, IXGBE_TDBAL(tx_ring->index), buf_low);
2342     IXGBE_WRITE_REG(hw, IXGBE_TDBAH(tx_ring->index), buf_high);

2344     /*
2345     * Setup head & tail pointers
2346     */
2347     IXGBE_WRITE_REG(hw, IXGBE_TDH(tx_ring->index), 0);
2348     IXGBE_WRITE_REG(hw, IXGBE_TDT(tx_ring->index), 0);

2350     /*
2351     * Setup head write-back
2352     */
2353     if (ixgbe->tx_head_wb_enable) {
2354         /*
2355         * The memory of the head write-back is allocated using
2356         * the extra tbd beyond the tail of the tbd ring.
2357         */
2358         tx_ring->tbd_head_wb = (uint32_t *)
2359             ((uintptr_t)tx_ring->tbd_area.address + size);
2360         *tx_ring->tbd_head_wb = 0;

2362         buf_low = (uint32_t)
2363             (tx_ring->tbd_area.dma_address + size);
2364         buf_high = (uint32_t)
2365             ((tx_ring->tbd_area.dma_address + size) >> 32);

2367         /* Set the head write-back enable bit */
2368         buf_low |= IXGBE_TDWBAL_HEAD_WB_ENABLE;

2370         IXGBE_WRITE_REG(hw, IXGBE_TDWBAL(tx_ring->index), buf_low);
2371         IXGBE_WRITE_REG(hw, IXGBE_TDBAH(tx_ring->index), buf_high);

2373         /*
2374         * Turn off relaxed ordering for head write back or it will
2375         * cause problems with the tx recycling
2376         */

2378         reg_val = (hw->mac.type == ixgbe_mac_82598EB) ?
2379             IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL(tx_ring->index)) :
2380             IXGBE_READ_REG(hw, IXGBE_DCA_TXCTRL_82599(tx_ring->index));
2381         reg_val &= ~IXGBE_DCA_TXCTRL_DESC_WRO_EN;
2382         if (hw->mac.type == ixgbe_mac_82598EB) {
2383             reg_val = IXGBE_READ_REG(hw,
2384                 IXGBE_DCA_TXCTRL(tx_ring->index));
2385             reg_val &= ~IXGBE_DCA_TXCTRL_TX_WB_RO_EN;
2386             IXGBE_WRITE_REG(hw,
2387                 IXGBE_DCA_TXCTRL(tx_ring->index), reg_val);
2388         } else {
2389             IXGBE_WRITE_REG(hw,
2390                 IXGBE_DCA_TXCTRL_82599(tx_ring->index), reg_val);
2391         }

2393         tx_ring->tbd_head_wb = NULL;
2394     }

2393     tx_ring->tbd_head = 0;
2394     tx_ring->tbd_tail = 0;
2395     tx_ring->tbd_free = tx_ring->ring_size;

```

```

2397     if (ixgbe->tx_ring_init == B_TRUE) {
2398         tx_ring->tcb_head = 0;
2399         tx_ring->tcb_tail = 0;
2400         tx_ring->tcb_free = tx_ring->free_list_size;
2401     }
2402
2403     /*
2404      * Initialize the s/w context structure
2405      */
2406     bzero(&tx_ring->tx_context, sizeof (ixgbe_tx_context_t));
2407 }
2408
2409 static void
2410 ixgbe_setup_tx(ixgbe_t *ixgbe)
2411 {
2412     struct ixgbe_hw *hw = &ixgbe->hw;
2413     ixgbe_tx_ring_t *tx_ring;
2414     uint32_t reg_val;
2415     uint32_t ring_mapping;
2416     int i;
2417
2418     for (i = 0; i < ixgbe->num_tx_rings; i++) {
2419         tx_ring = &ixgbe->tx_rings[i];
2420         ixgbe_setup_tx_ring(tx_ring);
2421     }
2422
2423     /*
2424      * Setup the per-ring statistics mapping.
2425      */
2426     ring_mapping = 0;
2427     for (i = 0; i < ixgbe->num_tx_rings; i++) {
2428         ring_mapping |= (i & 0xF) << (8 * (i & 0x3));
2429         if ((i & 0x3) == 0x3) {
2430             switch (hw->mac.type) {
2431                 case ixgbe_mac_82598EB:
2432                     IXGBE_WRITE_REG(hw, IXGBE_TQSMR(i >> 2),
2433                         ring_mapping);
2434                     break;
2435
2436                 case ixgbe_mac_82599EB:
2437                 case ixgbe_mac_X540:
2438                     IXGBE_WRITE_REG(hw, IXGBE_TQSM(i >> 2),
2439                         ring_mapping);
2440                     break;
2441
2442                 default:
2443                     break;
2444             }
2445
2446             ring_mapping = 0;
2447         }
2448     }
2449     if (i & 0x3) {
2450         switch (hw->mac.type) {
2451             case ixgbe_mac_82598EB:
2452                 IXGBE_WRITE_REG(hw, IXGBE_TQSMR(i >> 2), ring_mapping);
2453                 break;
2454
2455             case ixgbe_mac_82599EB:
2456             case ixgbe_mac_X540:
2457                 IXGBE_WRITE_REG(hw, IXGBE_TQSM(i >> 2), ring_mapping);
2458                 break;
2459
2460             default:
2461                 break;

```

```

2462     }
2463 }
2464
2465 /*
2466  * Enable CRC appending and TX padding (for short tx frames)
2467  */
2468 reg_val = IXGBE_READ_REG(hw, IXGBE_HLREG0);
2469 reg_val |= IXGBE_HLREG0_TXCRCEN | IXGBE_HLREG0_TXPADEN;
2470 IXGBE_WRITE_REG(hw, IXGBE_HLREG0, reg_val);
2471
2472 /*
2473  * enable DMA for 82599 and X540 parts
2474  * enable DMA for 82599 parts
2475  */
2476 if (hw->mac.type >= ixgbe_mac_82599EB) {
2477     if (hw->mac.type == ixgbe_mac_82599EB) {
2478         /* DMATXCTL.TE must be set after all Tx config is complete */
2479         reg_val = IXGBE_READ_REG(hw, IXGBE_DMATXCTL);
2480         reg_val |= IXGBE_DMATXCTL_TE;
2481         IXGBE_WRITE_REG(hw, IXGBE_DMATXCTL, reg_val);
2482
2483         /* Disable arbiter to set MTQC */
2484         reg_val = IXGBE_READ_REG(hw, IXGBE_RTTDCS);
2485         reg_val |= IXGBE_RTTDCS_ARBDIS;
2486         IXGBE_WRITE_REG(hw, IXGBE_RTTDCS, reg_val);
2487         IXGBE_WRITE_REG(hw, IXGBE_MTQC, IXGBE_MTQC_64Q_1PB);
2488         reg_val &= ~IXGBE_RTTDCS_ARBDIS;
2489         IXGBE_WRITE_REG(hw, IXGBE_RTTDCS, reg_val);
2490     }
2491
2492     /*
2493      * Enabling tx queues ..
2494      * For 82599 must be done after DMATXCTL.TE is set
2495      */
2496     for (i = 0; i < ixgbe->num_tx_rings; i++) {
2497         tx_ring = &ixgbe->tx_rings[i];
2498         reg_val = IXGBE_READ_REG(hw, IXGBE_TXDCTL(tx_ring->index));
2499         reg_val |= IXGBE_TXDCTL_ENABLE;
2500         IXGBE_WRITE_REG(hw, IXGBE_TXDCTL(tx_ring->index), reg_val);
2501     }
2502 }
2503
2504 unchanged portion omitted
2505
2506 /*
2507  * ixgbe_setup_vmdq - Setup MAC classification feature
2508  */
2509 static void
2510 ixgbe_setup_vmdq(ixgbe_t *ixgbe)
2511 {
2512     struct ixgbe_hw *hw = &ixgbe->hw;
2513     uint32_t vmdctl, i, vtctl;
2514
2515     /*
2516      * Setup the VMDq Control register, enable VMDq based on
2517      * packet destination MAC address:
2518      */
2519     switch (hw->mac.type) {
2520         case ixgbe_mac_82598EB:
2521             /*
2522              * VMDq Enable = 1;
2523              * VMDq Filter = 0; MAC filtering
2524              * Default VMDq output index = 0;
2525              */
2526             vmdctl = IXGBE_VMD_CTL_VMDQ_EN;
2527             IXGBE_WRITE_REG(hw, IXGBE_VMD_CTL, vmdctl);
2528             break;

```

```

2586     case ixgbe_mac_82599EB:
2587     case ixgbe_mac_X540:
2588         /*
2589          * Enable VMDq-only.
2590          */
2591         vmdctl = IXGBE_MRQC_VMDQEN;
2592         IXGBE_WRITE_REG(hw, IXGBE_MRQC, vmdctl);
2593
2594         for (i = 0; i < hw->mac.num_rar_entries; i++) {
2595             IXGBE_WRITE_REG(hw, IXGBE_MPSAR_L0(i), 0);
2596             IXGBE_WRITE_REG(hw, IXGBE_MPSAR_HI(i), 0);
2597         }
2598
2599         /*
2600          * Enable Virtualization and Replication.
2601          */
2602         vtctl = IXGBE_VT_CTL_VT_ENABLE | IXGBE_VT_CTL_REPLEN;
2603         IXGBE_WRITE_REG(hw, IXGBE_VT_CTL, vtctl);
2604
2605         /*
2606          * Enable receiving packets to all VFs
2607          */
2608         IXGBE_WRITE_REG(hw, IXGBE_VFRE(0), IXGBE_VFRE_ENABLE_ALL);
2609         IXGBE_WRITE_REG(hw, IXGBE_VFRE(1), IXGBE_VFRE_ENABLE_ALL);
2610         break;
2611
2612     default:
2613         break;
2614 }
2615 }
2616
2617 /*
2618  * ixgbe_setup_vmdq_rss - Setup both vmdq feature and rss feature.
2619  */
2620 static void
2621 ixgbe_setup_vmdq_rss(ixgbe_t *ixgbe)
2622 {
2623     struct ixgbe_hw *hw = &ixgbe->hw;
2624     uint32_t i, mrqc, rxcsum;
2625     uint32_t random;
2626     uint32_t reta;
2627     uint32_t ring_per_group;
2628     uint32_t vmdctl, vtctl;
2629
2630     /*
2631      * Fill out redirection table
2632      */
2633     reta = 0;
2634     ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;
2635     for (i = 0; i < 128; i++) {
2636         reta = (reta << 8) | (i % ring_per_group) |
2637             ((i % ring_per_group) << 4);
2638         if ((i & 3) == 3)
2639             IXGBE_WRITE_REG(hw, IXGBE_RETA(i >> 2), reta);
2640     }
2641
2642     /*
2643      * Fill out hash function seeds with a random constant
2644      */
2645     for (i = 0; i < 10; i++) {
2646         (void) random_get_pseudo_bytes((uint8_t *)&random,
2647             sizeof (uint32_t));
2648         IXGBE_WRITE_REG(hw, IXGBE_RSSRK(i), random);
2649     }

```

```

2651     /*
2652      * Enable and setup RSS and VMDq
2653      */
2654     switch (hw->mac.type) {
2655     case ixgbe_mac_82598EB:
2656         /*
2657          * Enable RSS & Setup RSS Hash functions
2658          */
2659         mrqc = IXGBE_MRQC_RSSEN |
2660             IXGBE_MRQC_RSS_FIELD_IPV4 |
2661             IXGBE_MRQC_RSS_FIELD_IPV4_TCP |
2662             IXGBE_MRQC_RSS_FIELD_IPV4_UDP |
2663             IXGBE_MRQC_RSS_FIELD_IPV6_EX_TCP |
2664             IXGBE_MRQC_RSS_FIELD_IPV6_EX |
2665             IXGBE_MRQC_RSS_FIELD_IPV6 |
2666             IXGBE_MRQC_RSS_FIELD_IPV6_TCP |
2667             IXGBE_MRQC_RSS_FIELD_IPV6_UDP |
2668             IXGBE_MRQC_RSS_FIELD_IPV6_EX_UDP;
2669         IXGBE_WRITE_REG(hw, IXGBE_MRQC, mrqc);
2670
2671         /*
2672          * Enable and Setup VMDq
2673          * VMDq Filter = 0; MAC filtering
2674          * Default VMDq output index = 0;
2675          */
2676         vmdctl = IXGBE_VMD_CTL_VMDQ_EN;
2677         IXGBE_WRITE_REG(hw, IXGBE_VMD_CTL, vmdctl);
2678         break;
2679
2680     case ixgbe_mac_82599EB:
2681     case ixgbe_mac_X540:
2682         /*
2683          * Enable RSS & Setup RSS Hash functions
2684          */
2685         mrqc = IXGBE_MRQC_RSS_FIELD_IPV4 |
2686             IXGBE_MRQC_RSS_FIELD_IPV4_TCP |
2687             IXGBE_MRQC_RSS_FIELD_IPV4_UDP |
2688             IXGBE_MRQC_RSS_FIELD_IPV6_EX_TCP |
2689             IXGBE_MRQC_RSS_FIELD_IPV6_EX |
2690             IXGBE_MRQC_RSS_FIELD_IPV6 |
2691             IXGBE_MRQC_RSS_FIELD_IPV6_TCP |
2692             IXGBE_MRQC_RSS_FIELD_IPV6_UDP |
2693             IXGBE_MRQC_RSS_FIELD_IPV6_EX_UDP;
2694
2695         /*
2696          * Enable VMDq+RSS.
2697          */
2698         if (ixgbe->num_rx_groups > 32) {
2699             mrqc = mrqc | IXGBE_MRQC_VMDQRSS64EN;
2700         } else {
2701             mrqc = mrqc | IXGBE_MRQC_VMDQRSS32EN;
2702         }
2703
2704         IXGBE_WRITE_REG(hw, IXGBE_MRQC, mrqc);
2705
2706         for (i = 0; i < hw->mac.num_rar_entries; i++) {
2707             IXGBE_WRITE_REG(hw, IXGBE_MPSAR_L0(i), 0);
2708             IXGBE_WRITE_REG(hw, IXGBE_MPSAR_HI(i), 0);
2709         }
2710         break;
2711
2712     default:
2713         break;
2714 }
2715 }

```

```

2717 /*
2718  * Disable Packet Checksum to enable RSS for multiple receive queues.
2719  * It is an adapter hardware limitation that Packet Checksum is
2720  * mutually exclusive with RSS.
2721  */
2722 rxcsum = IXGBE_READ_REG(hw, IXGBE_RXCSUM);
2723 rxcsum |= IXGBE_RXCSUM_PCSD;
2724 rxcsum &= ~IXGBE_RXCSUM_IPPCSE;
2725 IXGBE_WRITE_REG(hw, IXGBE_RXCSUM, rxcsum);

2727 if (hw->mac.type >= ixgbe_mac_82599EB) {
2667 if (hw->mac.type == ixgbe_mac_82599EB) {
2728 /*
2729  * Enable Virtualization and Replication.
2730  */
2731 vtctl = IXGBE_VT_CTL_VT_ENABLE | IXGBE_VT_CTL_REPLEN;
2732 IXGBE_WRITE_REG(hw, IXGBE_VT_CTL, vtctl);

2734 /*
2735  * Enable receiving packets to all VFs
2736  */
2737 IXGBE_WRITE_REG(hw, IXGBE_VFRE(0), IXGBE_VFRE_ENABLE_ALL);
2738 IXGBE_WRITE_REG(hw, IXGBE_VFRE(1), IXGBE_VFRE_ENABLE_ALL);
2739 }
2740 }
unchanged_portion_omitted

2878 /*
2879  * ixgbe_setup_multicast - Setup multicast data structures.
2880  *
2881  * This routine initializes all of the multicast related structures
2882  * and save them in the hardware registers.
2883  */
2884 static void
2885 ixgbe_setup_multicst(ixgbe_t *ixgbe)
2886 {
2887     uint8_t *mc_addr_list;
2888     uint32_t mc_addr_count;
2889     struct ixgbe_hw *hw = &ixgbe->hw;

2891     ASSERT(mutex_owned(&ixgbe->gen_lock));

2893     ASSERT(ixgbe->mcast_count <= MAX_NUM_MULTICAST_ADDRESSES);

2895     mc_addr_list = (uint8_t *)ixgbe->mcast_table;
2896     mc_addr_count = ixgbe->mcast_count;

2898     /*
2899     * Update the multicast addresses to the MTA registers
2900     */
2901     (void) ixgbe_update_mc_addr_list(hw, mc_addr_list, mc_addr_count,
2902     ixgbe_mc_table_itr, TRUE);
2842     ixgbe_mc_table_itr);
2903 }

2905 /*
2906  * ixgbe_setup_vmdq_rss_conf - Configure vmdq and rss (number and mode).
2907  *
2908  * Configure the rx classification mode (vmdq & rss) and vmdq & rss numbers.
2909  * Different chipsets may have different allowed configuration of vmdq and rss.
2910  */
2911 static void
2912 ixgbe_setup_vmdq_rss_conf(ixgbe_t *ixgbe)
2913 {
2914     struct ixgbe_hw *hw = &ixgbe->hw;
2915     uint32_t ring_per_group;

```

```

2917 switch (hw->mac.type) {
2918 case ixgbe_mac_82598EB:
2919     /*
2920     * 82598 supports the following combination:
2921     * vmdq no. x rss no.
2922     * [5..16] x 1
2923     * [1..4] x [1..16]
2924     * However 8 rss queue per pool (vmdq) is sufficient for
2925     * most cases.
2926     */
2927     ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;
2928     if (ixgbe->num_rx_groups > 4) {
2929         ixgbe->num_rx_rings = ixgbe->num_rx_groups;
2930     } else {
2931         ixgbe->num_rx_rings = ixgbe->num_rx_groups *
2932         min(8, ring_per_group);
2933     }

2935     break;

2937 case ixgbe_mac_82599EB:
2938 case ixgbe_mac_X540:
2939     /*
2940     * 82599 supports the following combination:
2941     * vmdq no. x rss no.
2942     * [33..64] x [1..2]
2943     * [2..32] x [1..4]
2944     * 1 x [1..16]
2945     * However 8 rss queue per pool (vmdq) is sufficient for
2946     * most cases.
2947     *
2948     * For now, treat X540 like the 82599.
2949     */
2950     ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;
2951     if (ixgbe->num_rx_groups == 1) {
2952         ixgbe->num_rx_rings = min(8, ring_per_group);
2953     } else if (ixgbe->num_rx_groups <= 32) {
2954         ixgbe->num_rx_rings = ixgbe->num_rx_groups *
2955         min(4, ring_per_group);
2956     } else if (ixgbe->num_rx_groups <= 64) {
2957         ixgbe->num_rx_rings = ixgbe->num_rx_groups *
2958         min(2, ring_per_group);
2959     }
2960     break;

2962 default:
2963     break;
2964 }

2966 ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;

2968 if (ixgbe->num_rx_groups == 1 && ring_per_group == 1) {
2969     ixgbe->classify_mode = IXGBE_CLASSIFY_NONE;
2970 } else if (ixgbe->num_rx_groups != 1 && ring_per_group == 1) {
2971     ixgbe->classify_mode = IXGBE_CLASSIFY_VMDQ;
2972 } else if (ixgbe->num_rx_groups != 1 && ring_per_group != 1) {
2973     ixgbe->classify_mode = IXGBE_CLASSIFY_VMDQ_RSS;
2974 } else {
2975     ixgbe->classify_mode = IXGBE_CLASSIFY_RSS;
2976 }

2978 IXGBE_DEBUGLOG_2(ixgbe, "rx group number:%d, rx ring number:%d",
2979 ixgbe->num_rx_groups, ixgbe->num_rx_rings);
2980 }

```

```

2982 /*
2983  * ixgbe_get_conf - Get driver configurations set in driver.conf.
2984  *
2985  * This routine gets user-configured values out of the configuration
2986  * file ixgbe.conf.
2987  *
2988  * For each configurable value, there is a minimum, a maximum, and a
2989  * default.
2990  * If user does not configure a value, use the default.
2991  * If user configures below the minimum, use the minimum.
2992  * If user configures above the maximum, use the maximum.
2993  */
2994 static void
2995 ixgbe_get_conf(ixgbe_t *ixgbe)
2996 {
2997     struct ixgbe_hw *hw = &ixgbe->hw;
2998     uint32_t flow_control;
2999
3000     /*
3001      * ixgbe driver supports the following user configurations:
3002      *
3003      * Jumbo frame configuration:
3004      *     default_mtu
3005      *
3006      * Ethernet flow control configuration:
3007      *     flow_control
3008      *
3009      * Multiple rings configurations:
3010      *     tx_queue_number
3011      *     tx_ring_size
3012      *     rx_queue_number
3013      *     rx_ring_size
3014      *
3015      * Call ixgbe_get_prop() to get the value for a specific
3016      * configuration parameter.
3017      */
3018
3019     /*
3020      * Jumbo frame configuration - max_frame_size controls host buffer
3021      * allocation, so includes MTU, ethernet header, vlan tag and
3022      * frame check sequence.
3023      */
3024     ixgbe->default_mtu = ixgbe_get_prop(ixgbe, PROP_DEFAULT_MTU,
3025     MIN_MTU, ixgbe->capab->max_mtu, DEFAULT_MTU);
3026
3027     ixgbe->max_frame_size = ixgbe->default_mtu +
3028     sizeof(struct ether_vlan_header) + ETHERFCSL;
3029
3030     /*
3031      * Ethernet flow control configuration
3032      */
3033     flow_control = ixgbe_get_prop(ixgbe, PROP_FLOW_CONTROL,
3034     ixgbe_fc_none, 3, ixgbe_fc_none);
3035     if (flow_control == 3)
3036         flow_control = ixgbe_fc_default;
3037
3038     /*
3039      * fc.requested mode is what the user requests. After autoneg,
3040      * fc.current_mode will be the flow_control mode that was negotiated.
3041      */
3042     hw->fc.requested_mode = flow_control;
3043
3044     /*
3045      * Multiple rings configurations
3046      */
3047     ixgbe->num_tx_rings = ixgbe_get_prop(ixgbe, PROP_TX_QUEUE_NUM,

```

```

3048     ixgbe->capab->min_tx_queue_num,
3049     ixgbe->capab->max_tx_queue_num,
3050     ixgbe->capab->def_tx_queue_num);
3051     ixgbe->tx_ring_size = ixgbe_get_prop(ixgbe, PROP_TX_RING_SIZE,
3052     MIN_TX_RING_SIZE, MAX_TX_RING_SIZE, DEFAULT_TX_RING_SIZE);
3053
3054     ixgbe->num_rx_rings = ixgbe_get_prop(ixgbe, PROP_RX_QUEUE_NUM,
3055     ixgbe->capab->min_rx_queue_num,
3056     ixgbe->capab->max_rx_queue_num,
3057     ixgbe->capab->def_rx_queue_num);
3058     ixgbe->rx_ring_size = ixgbe_get_prop(ixgbe, PROP_RX_RING_SIZE,
3059     MIN_RX_RING_SIZE, MAX_RX_RING_SIZE, DEFAULT_RX_RING_SIZE);
3060
3061     /*
3062      * Multiple groups configuration
3063      */
3064     ixgbe->num_rx_groups = ixgbe_get_prop(ixgbe, PROP_RX_GROUP_NUM,
3065     ixgbe->capab->min_rx_grp_num, ixgbe->capab->max_rx_grp_num,
3066     ixgbe->capab->def_rx_grp_num);
3067
3068     ixgbe->mr_enable = ixgbe_get_prop(ixgbe, PROP_MR_ENABLE,
3069     0, 1, DEFAULT_MR_ENABLE);
3070
3071     if (ixgbe->mr_enable == B_FALSE) {
3072         ixgbe->num_tx_rings = 1;
3073         ixgbe->num_rx_rings = 1;
3074         ixgbe->num_rx_groups = 1;
3075         ixgbe->classify_mode = IXGBE_CLASSIFY_NONE;
3076     } else {
3077         ixgbe->num_rx_rings = ixgbe->num_rx_groups *
3078         max(ixgbe->num_rx_rings / ixgbe->num_rx_groups, 1);
3079         /*
3080          * The combination of num_rx_rings and num_rx_groups
3081          * may be not supported by h/w. We need to adjust
3082          * them to appropriate values.
3083          */
3084         ixgbe_setup_vmdq_rss_conf(ixgbe);
3085     }
3086
3087     /*
3088      * Tunable used to force an interrupt type. The only use is
3089      * for testing of the lesser interrupt types.
3090      * 0 = don't force interrupt type
3091      * 1 = force interrupt type MSI-X
3092      * 2 = force interrupt type MSI
3093      * 3 = force interrupt type Legacy
3094      */
3095     ixgbe->intr_force = ixgbe_get_prop(ixgbe, PROP_INTR_FORCE,
3096     IXGBE_INTR_NONE, IXGBE_INTR_LEGACY, IXGBE_INTR_NONE);
3097
3098     ixgbe->tx_hcksum_enable = ixgbe_get_prop(ixgbe, PROP_TX_HCKSUM_ENABLE,
3099     0, 1, DEFAULT_TX_HCKSUM_ENABLE);
3100     ixgbe->rx_hcksum_enable = ixgbe_get_prop(ixgbe, PROP_RX_HCKSUM_ENABLE,
3101     0, 1, DEFAULT_RX_HCKSUM_ENABLE);
3102     ixgbe->lso_enable = ixgbe_get_prop(ixgbe, PROP_LSO_ENABLE,
3103     0, 1, DEFAULT_LSO_ENABLE);
3104     ixgbe->lro_enable = ixgbe_get_prop(ixgbe, PROP_LRO_ENABLE,
3105     0, 1, DEFAULT_LRO_ENABLE);
3106     ixgbe->tx_head_wb_enable = ixgbe_get_prop(ixgbe, PROP_TX_HEAD_WB_ENABLE,
3107     0, 1, DEFAULT_TX_HEAD_WB_ENABLE);
3108     ixgbe->relax_order_enable = ixgbe_get_prop(ixgbe,
3109     PROP_RELAX_ORDER_ENABLE, 0, 1, DEFAULT_RELAX_ORDER_ENABLE);
3110
3111     /* Head Write Back not recommended for 82599 and X540 */
3112     /* Head Write Back not recommended for 82599 */
3113     if (hw->mac.type >= ixgbe_mac_82599EB) {

```

```

3113         ixgbe->tx_head_wb_enable = B_FALSE;
3114     }

3116     /*
3117     * ixgbe LSO needs the tx h/w checksum support.
3118     * LSO will be disabled if tx h/w checksum is not
3119     * enabled.
3120     */
3121     if (ixgbe->tx_hcksum_enable == B_FALSE) {
3122         ixgbe->lso_enable = B_FALSE;
3123     }

3125     /*
3126     * ixgbe LRO needs the rx h/w checksum support.
3127     * LRO will be disabled if rx h/w checksum is not
3128     * enabled.
3129     */
3130     if (ixgbe->rx_hcksum_enable == B_FALSE) {
3131         ixgbe->lro_enable = B_FALSE;
3132     }

3134     /*
3135     * ixgbe LRO only been supported by 82599 and X540 now
3136     * ixgbe LRO only been supported by 82599 now
3137     */
3137     if (hw->mac.type < ixgbe_mac_82599EB) {
3138         if (hw->mac.type != ixgbe_mac_82599EB) {
3139             ixgbe->lro_enable = B_FALSE;
3140         }
3141         ixgbe->tx_copy_thresh = ixgbe_get_prop(ixgbe, PROP_TX_COPY_THRESHOLD,
3142             MIN_TX_COPY_THRESHOLD, MAX_TX_COPY_THRESHOLD,
3143             DEFAULT_TX_COPY_THRESHOLD);
3144         ixgbe->tx_recycle_thresh = ixgbe_get_prop(ixgbe,
3145             PROP_TX_RECYCLE_THRESHOLD, MIN_TX_RECYCLE_THRESHOLD,
3146             MAX_TX_RECYCLE_THRESHOLD, DEFAULT_TX_RECYCLE_THRESHOLD);
3147         ixgbe->tx_overload_thresh = ixgbe_get_prop(ixgbe,
3148             PROP_TX_OVERLOAD_THRESHOLD, MIN_TX_OVERLOAD_THRESHOLD,
3149             MAX_TX_OVERLOAD_THRESHOLD, DEFAULT_TX_OVERLOAD_THRESHOLD);
3150         ixgbe->tx_resched_thresh = ixgbe_get_prop(ixgbe,
3151             PROP_TX_RESCHED_THRESHOLD, MIN_TX_RESCHED_THRESHOLD,
3152             MAX_TX_RESCHED_THRESHOLD, DEFAULT_TX_RESCHED_THRESHOLD);

3153         ixgbe->rx_copy_thresh = ixgbe_get_prop(ixgbe, PROP_RX_COPY_THRESHOLD,
3154             MIN_RX_COPY_THRESHOLD, MAX_RX_COPY_THRESHOLD,
3155             DEFAULT_RX_COPY_THRESHOLD);
3156         ixgbe->rx_limit_per_intr = ixgbe_get_prop(ixgbe, PROP_RX_LIMIT_PER_INTR,
3157             MIN_RX_LIMIT_PER_INTR, MAX_RX_LIMIT_PER_INTR,
3158             DEFAULT_RX_LIMIT_PER_INTR);

3160         ixgbe->intr_throttling[0] = ixgbe_get_prop(ixgbe, PROP_INTR_THROTTLING,
3161             ixgbe->capab->min_intr_throttle,
3162             ixgbe->capab->max_intr_throttle,
3163             ixgbe->capab->def_intr_throttle);
3164     /*
3165     * 82599 and X540 require the interrupt throttling rate is
3166     * 82599 requires the interrupt throttling rate is
3167     * a multiple of 8. This is enforced by the register
3168     * definiton.
3169     */
3170     if (hw->mac.type >= ixgbe_mac_82599EB)
3171         if (hw->mac.type == ixgbe_mac_82599EB)
3172             ixgbe->intr_throttling[0] = ixgbe->intr_throttling[0] & 0xFF8;
3173     }

```

unchanged portion omitted

3275 /*

```

3276 * ixgbe_driver_link_check - Link status processing.
3277 *
3278 * This function can be called in both kernel context and interrupt context
3279 */
3280 static void
3281 ixgbe_driver_link_check(ixgbe_t *ixgbe)
3282 {
3283     struct ixgbe_hw *hw = &ixgbe->hw;
3284     ixgbe_link_speed speed = IXGBE_LINK_SPEED_UNKNOWN;
3285     boolean_t link_up = B_FALSE;
3286     boolean_t link_changed = B_FALSE;

3288     ASSERT(mutex_owned(&ixgbe->gen_lock));

3290     (void) ixgbe_check_link(hw, &speed, &link_up, false);
3291     if (link_up) {
3292         ixgbe->link_check_complete = B_TRUE;

3294         /* Link is up, enable flow control settings */
3295         (void) ixgbe_fc_enable(hw);
3296         (void) ixgbe_fc_enable(hw, 0);

3297     /*
3298     * The Link is up, check whether it was marked as down earlier
3299     */
3300     if (ixgbe->link_state != LINK_STATE_UP) {
3301         switch (speed) {
3302             case IXGBE_LINK_SPEED_10GB_FULL:
3303                 ixgbe->link_speed = SPEED_10GB;
3304                 break;
3305             case IXGBE_LINK_SPEED_1GB_FULL:
3306                 ixgbe->link_speed = SPEED_1GB;
3307                 break;
3308             case IXGBE_LINK_SPEED_100_FULL:
3309                 ixgbe->link_speed = SPEED_100;
3310         }
3311         ixgbe->link_duplex = LINK_DUPLEX_FULL;
3312         ixgbe->link_state = LINK_STATE_UP;
3313         link_changed = B_TRUE;
3314     }
3315     } else {
3316         if (ixgbe->link_check_complete == B_TRUE ||
3317             (ixgbe->link_check_complete == B_FALSE &&
3318              gethrtime() >= ixgbe->link_check_hrttime)) {
3319             /*
3320             * The link is really down
3321             */
3322             ixgbe->link_check_complete = B_TRUE;

3324             if (ixgbe->link_state != LINK_STATE_DOWN) {
3325                 ixgbe->link_speed = 0;
3326                 ixgbe->link_duplex = LINK_DUPLEX_UNKNOWN;
3327                 ixgbe->link_state = LINK_STATE_DOWN;
3328                 link_changed = B_TRUE;
3329             }
3330         }
3331     }

3333     /*
3334     * If we are in an interrupt context, need to re-enable the
3335     * interrupt, which was automasked
3336     */
3337     if (servicing_interrupt() != 0) {
3338         ixgbe->eims |= IXGBE_EICR_LSC;
3339         IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);
3340     }

```

```

3342         if (link_changed) {
3343             mac_link_update(ixgbe->mac_hdl, ixgbe->link_state);
3344         }
3345     }
    _____ unchanged_portion_omitted _____
3758 /*
3759  * ixgbe_enable_adapter_interrupts - Enable all hardware interrupts.
3760  */
3761 static void
3762 ixgbe_enable_adapter_interrupts(ixgbe_t *ixgbe)
3763 {
3764     struct ixgbe_hw *hw = &ixgbe->hw;
3765     uint32_t eiac, eiam;
3766     uint32_t gpie = IXGBE_READ_REG(hw, IXGBE_GPIE);

3768     /* interrupt types to enable */
3769     ixgbe->eims = IXGBE_EIMS_ENABLE_MASK; /* shared code default */
3770     ixgbe->eims &= ~IXGBE_EIMS_TCP_TIMER; /* minus tcp timer */
3771     ixgbe->eims |= ixgbe->capab->other_intr; /* "other" interrupt types */

3773     /* enable autmask on "other" causes that this adapter can generate */
3774     eiam = ixgbe->capab->other_intr;

3776     /*
3777      * msi-x mode
3778      */
3779     if (ixgbe->intr_type == DDI_INTR_TYPE_MSIX) {
3780         /* enable autoclear but not on bits 29:20 */
3781         eiac = (ixgbe->eims & ~IXGBE_OTHER_INTR);

3783         /* general purpose interrupt enable */
3784         gpie |= (IXGBE_GPIE_MSIX_MODE
3785                 | IXGBE_GPIE_PBA_SUPPORT
3786                 | IXGBE_GPIE_OCD
3787                 | IXGBE_GPIE_EIAME);
3788     /*
3789      * non-msi-x mode
3790      */
3791     } else {

3793         /* disable autoclear, leave gpie at default */
3794         eiac = 0;

3796         /*
3797          * General purpose interrupt enable.
3798          * For 82599, extended interrupt autmask enable
3799          * only in MSI or MSI-X mode
3800          */
3801         if ((hw->mac.type < ixgbe_mac_82599EB) ||
3802             (ixgbe->intr_type == DDI_INTR_TYPE_MSI)) {
3803             gpie |= IXGBE_GPIE_EIAME;
3804         }
3805     }

3807     /* Enable specific "other" interrupt types */
3808     switch (hw->mac.type) {
3809     case ixgbe_mac_82598EB:
3810         gpie |= ixgbe->capab->other_gpie;
3811         break;

3813     case ixgbe_mac_82599EB:
3814     case ixgbe_mac_X540:
3815         gpie |= ixgbe->capab->other_gpie;

```

```

3817         /* Enable RSC Delay Bus when LRO enabled */
3818         if (ixgbe->lro_enable) {
3819             gpie |= (1 << IXGBE_GPIE_RSC_DELAY_SHIFT);
3820         }
3821         break;

3823     default:
3824         break;
3825     }

3827     /* write to interrupt control registers */
3828     IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);
3829     IXGBE_WRITE_REG(hw, IXGBE_EIAC, eiac);
3830     IXGBE_WRITE_REG(hw, IXGBE_EIAM, eiam);
3831     IXGBE_WRITE_REG(hw, IXGBE_GPIE, gpie);
3832     IXGBE_WRITE_FLUSH(hw);
3833 }
    _____ unchanged_portion_omitted _____
3954 /*
3955  * ixgbe_set_internal_mac_loopback - Set the internal MAC loopback mode.
3956  */
3957 static void
3958 ixgbe_set_internal_mac_loopback(ixgbe_t *ixgbe)
3959 {
3960     struct ixgbe_hw *hw;
3961     uint32_t reg;
3962     uint8_t atlas;

3964     hw = &ixgbe->hw;

3966     /*
3967      * Setup MAC loopback
3968      */
3969     reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_HLREG0);
3970     reg |= IXGBE_HLREG0_LPBK;
3971     IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_HLREG0, reg);

3973     reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_AUTOC);
3974     reg &= ~IXGBE_AUTOC_LMS_MASK;
3975     IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_AUTOC, reg);

3977     /*
3978      * Disable Atlas Tx lanes to keep packets in loopback and not on wire
3979      */
3980     switch (hw->mac.type) {
3981     case ixgbe_mac_82598EB:
3982         (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_LPBK,
3983                                     &atlas);
3984         atlas |= IXGBE_ATLAS_PDN_TX_REG_EN;
3985         (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_LPBK,
3986                                     atlas);

3988         (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_10G,
3989                                     &atlas);
3990         atlas |= IXGBE_ATLAS_PDN_TX_10G_QL_ALL;
3991         (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_10G,
3992                                     atlas);

3994         (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_1G,
3995                                     &atlas);
3996         atlas |= IXGBE_ATLAS_PDN_TX_1G_QL_ALL;
3997         (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_1G,
3998                                     atlas);

4000         (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_AN,

```

```

4001         &atlas);
4002         atlas |= IXGBE_ATLAS_PDN_TX_AN_QL_ALL;
4003         (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_AN,
4004         atlas);
4005         break;

4007     case ixgbe_mac_82599EB:
4008     case ixgbe_mac_X540:
4009         reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_AUTOC);
4010         reg |= (IXGBE_AUTOC_FLU |
4011         IXGBE_AUTOC_10G_KX4);
4012         IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_AUTOC, reg);

4014         (void) ixgbe_setup_link(&ixgbe->hw, IXGBE_LINK_SPEED_10GB_FULL,
4015         B_FALSE, B_TRUE);
4016         break;

4018     default:
4019         break;
4020 }
4021 }
_____ unchanged_portion_omitted _____

4142 /*
4143  * ixgbe_intr_legacy - Interrupt handler for legacy interrupts.
4144  */
4145 static uint_t
4146 ixgbe_intr_legacy(void *arg1, void *arg2)
4147 {
4148     ixgbe_t *ixgbe = (ixgbe_t *)arg1;
4149     struct ixgbe_hw *hw = &ixgbe->hw;
4150     ixgbe_tx_ring_t *tx_ring;
4151     ixgbe_rx_ring_t *rx_ring;
4152     uint32_t eicr;
4153     mblk_t *mp;
4154     boolean_t tx_reschedule;
4155     uint_t result;

4157     _NOTE(ARGUNUSED(arg2));

4159     mutex_enter(&ixgbe->gen_lock);
4160     if (ixgbe->ixgbe_state & IXGBE_SUSPENDED) {
4161         mutex_exit(&ixgbe->gen_lock);
4162         return (DDI_INTR_UNCLAIMED);
4163     }

4165     mp = NULL;
4166     tx_reschedule = B_FALSE;

4168     /*
4169      * Any bit set in eicr: claim this interrupt
4170      */
4171     eicr = IXGBE_READ_REG(hw, IXGBE_EICR);

4173     if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) != DDI_FM_OK) {
4174         mutex_exit(&ixgbe->gen_lock);
4175         ddi_fm_service_impact(ixgbe->dip, DDI_SERVICE_DEGRADED);
4176         atomic_or_32(&ixgbe->ixgbe_state, IXGBE_ERROR);
4177         return (DDI_INTR_CLAIMED);
4178     }

4180     if (eicr) {
4181         /*
4182          * For legacy interrupt, we have only one interrupt,
4183          * so we have only one rx ring and one tx ring enabled.
4184          */

```

```

4185     ASSERT(ixgbe->num_rx_rings == 1);
4186     ASSERT(ixgbe->num_tx_rings == 1);

4188     /*
4189      * For legacy interrupt, rx rings[0] will use RTxQ[0].
4190      */
4191     if (eicr & 0x1) {
4192         ixgbe->eimc |= IXGBE_EICR_RTX_QUEUE;
4193         IXGBE_WRITE_REG(hw, IXGBE_EIMC, ixgbe->eimc);
4194         ixgbe->eims |= IXGBE_EICR_RTX_QUEUE;
4195         /*
4196          * Clean the rx descriptors
4197          */
4198         rx_ring = &ixgbe->rx_rings[0];
4199         mp = ixgbe_ring_rx(rx_ring, IXGBE_POLL_NULL);
4200     }

4202     /*
4203      * For legacy interrupt, tx rings[0] will use RTxQ[1].
4204      */
4205     if (eicr & 0x2) {
4206         /*
4207          * Recycle the tx descriptors
4208          */
4209         tx_ring = &ixgbe->tx_rings[0];
4210         tx_ring->tx_recycle(tx_ring);

4212         /*
4213          * Schedule the re-transmit
4214          */
4215         tx_reschedule = (tx_ring->reschedule &&
4216         (tx_ring->tbd_free >= ixgbe->tx_resched_thresh));
4217     }

4219     /* any interrupt type other than tx/rx */
4220     if (eicr & ixgbe->capab->other_intr) {
4221         switch (hw->mac.type) {
4222         case ixgbe_mac_82598EB:
4223             ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4224             break;

4226         case ixgbe_mac_82599EB:
4227         case ixgbe_mac_X540:
4228             ixgbe->eimc = IXGBE_82599_OTHER_INTR;
4229             IXGBE_WRITE_REG(hw, IXGBE_EIMC, ixgbe->eimc);
4230             break;

4232         default:
4233             break;
4234         }
4235         ixgbe_intr_other_work(ixgbe, eicr);
4236         ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4237     }

4239     mutex_exit(&ixgbe->gen_lock);

4241     result = DDI_INTR_CLAIMED;
4242 } else {
4243     mutex_exit(&ixgbe->gen_lock);

4245     /*
4246      * No interrupt cause bits set: don't claim this interrupt.
4247      */
4248     result = DDI_INTR_UNCLAIMED;
4249 }

```

```

4251      /* re-enable the interrupts which were automasked */
4252      IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);

4254      /*
4255       * Do the following work outside of the gen_lock
4256       */
4257      if (mp != NULL) {
4258          mac_rx_ring(rx_ring->ixgbe->mac_hdl, rx_ring->ring_handle, mp,
4259                    rx_ring->ring_gen_num);
4260      }

4262      if (tx_reschedule) {
4263          tx_ring->reschedule = B_FALSE;
4264          mac_tx_ring_update(ixgbe->mac_hdl, tx_ring->ring_handle);
4265          IXGBE_DEBUG_STAT(tx_ring->stat_reschedule);
4266      }

4268      return (result);
4269 }

4271 /*
4272  * ixgbe_intr_msi - Interrupt handler for MSI.
4273  */
4274 static uint_t
4275 ixgbe_intr_msi(void *arg1, void *arg2)
4276 {
4277     ixgbe_t *ixgbe = (ixgbe_t *)arg1;
4278     struct ixgbe_hw *hw = &ixgbe->hw;
4279     uint32_t eicr;

4281     _NOTE(ARGUNUSED(arg2));

4283     eicr = IXGBE_READ_REG(hw, IXGBE_EICR);

4285     if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) != DDI_FM_OK) {
4286         ddi_fm_service_impact(ixgbe->dip, DDI_SERVICE_DEGRADED);
4287         atomic_or_32(&ixgbe->ixgbe_state, IXGBE_ERROR);
4288         return (DDI_INTR_CLAIMED);
4289     }

4291     /*
4292      * For MSI interrupt, we have only one vector,
4293      * so we have only one rx ring and one tx ring enabled.
4294      */
4295     ASSERT(ixgbe->num_rx_rings == 1);
4296     ASSERT(ixgbe->num_tx_rings == 1);

4298     /*
4299      * For MSI interrupt, rx rings[0] will use RTxQ[0].
4300      */
4301     if (eicr & 0x1) {
4302         ixgbe_intr_rx_work(&ixgbe->rx_rings[0]);
4303     }

4305     /*
4306      * For MSI interrupt, tx rings[0] will use RTxQ[1].
4307      */
4308     if (eicr & 0x2) {
4309         ixgbe_intr_tx_work(&ixgbe->tx_rings[0]);
4310     }

4312     /* any interrupt type other than tx/rx */
4313     if (eicr & ixgbe->capab->other_intr) {
4314         mutex_enter(&ixgbe->gen_lock);
4315         switch (hw->mac.type) {
4316             case ixgbe_mac_82598EB:

```

```

4317         ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4318         break;

4320         case ixgbe_mac_82599EB:
4321         case ixgbe_mac_X540:
4322             ixgbe->eims = IXGBE_82599_OTHER_INTR;
4323             IXGBE_WRITE_REG(hw, IXGBE_EIMC, ixgbe->eims);
4324             break;

4326         default:
4327             break;
4328     }
4329     ixgbe_intr_other_work(ixgbe, eicr);
4330     ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4331     mutex_exit(&ixgbe->gen_lock);
4332 }

4334     /* re-enable the interrupts which were automasked */
4335     IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);

4337     return (DDI_INTR_CLAIMED);
4338 }

4340 /*
4341  * ixgbe_intr_msix - Interrupt handler for MSI-X.
4342  */
4343 static uint_t
4344 ixgbe_intr_msix(void *arg1, void *arg2)
4345 {
4346     ixgbe_intr_vector_t *vect = (ixgbe_intr_vector_t *)arg1;
4347     ixgbe_t *ixgbe = vect->ixgbe;
4348     struct ixgbe_hw *hw = &ixgbe->hw;
4349     uint32_t eicr;
4350     int r_idx = 0;

4352     _NOTE(ARGUNUSED(arg2));

4354     /*
4355      * Clean each rx ring that has its bit set in the map
4356      */
4357     r_idx = bt_getlowbit(vect->rx_map, 0, (ixgbe->num_rx_rings - 1));
4358     while (r_idx >= 0) {
4359         ixgbe_intr_rx_work(&ixgbe->rx_rings[r_idx]);
4360         r_idx = bt_getlowbit(vect->rx_map, (r_idx + 1),
4361                             (ixgbe->num_rx_rings - 1));
4362     }

4364     /*
4365      * Clean each tx ring that has its bit set in the map
4366      */
4367     r_idx = bt_getlowbit(vect->tx_map, 0, (ixgbe->num_tx_rings - 1));
4368     while (r_idx >= 0) {
4369         ixgbe_intr_tx_work(&ixgbe->tx_rings[r_idx]);
4370         r_idx = bt_getlowbit(vect->tx_map, (r_idx + 1),
4371                             (ixgbe->num_tx_rings - 1));
4372     }

4375     /*
4376      * Clean other interrupt (link change) that has its bit set in the map
4377      */
4378     if (BT_TEST(vect->other_map, 0) == 1) {
4379         eicr = IXGBE_READ_REG(hw, IXGBE_EICR);

4381         if (ixgbe_check_acc_handle(ixgbe->osdep.reg_handle) !=
4382             DDI_FM_OK) {

```

```

4383         ddi_fm_service_impact(ixgbe->dip,
4384             DDI_SERVICE_DEGRADED);
4385         atomic_or_32(&ixgbe->ixgbe_state, IXGBE_ERROR);
4386         return (DDI_INTR_CLAIMED);
4387     }
4388
4389     /*
4390     * Check "other" cause bits: any interrupt type other than tx/rx
4391     */
4392     if (eicr & ixgbe->capab->other_intr) {
4393         mutex_enter(&ixgbe->gen_lock);
4394         switch (hw->mac.type) {
4395             case ixgbe_mac_82598EB:
4396                 ixgbe->eims &= ~(eicr & IXGBE_OTHER_INTR);
4397                 ixgbe_intr_other_work(ixgbe, eicr);
4398                 break;
4399
4400             case ixgbe_mac_82599EB:
4401             case ixgbe_mac_X540:
4402                 ixgbe->eims |= IXGBE_EICR_RTX_QUEUE;
4403                 ixgbe_intr_other_work(ixgbe, eicr);
4404                 break;
4405
4406             default:
4407                 break;
4408         }
4409         mutex_exit(&ixgbe->gen_lock);
4410     }
4411
4412     /* re-enable the interrupts which were automasked */
4413     IXGBE_WRITE_REG(hw, IXGBE_EIMS, ixgbe->eims);
4414 }
4415
4416 return (DDI_INTR_CLAIMED);
4417 }

```

unchanged_portion_omitted

```

4773 /*
4774 * ixgbe_setup_ivar - Set the given entry in the given interrupt vector
4775 * allocation register (IVAR).
4776 * cause:
4777 *   -1 : other cause
4778 *   0 : rx
4779 *   1 : tx
4780 */
4781 static void
4782 ixgbe_setup_ivar(ixgbe_t *ixgbe, uint16_t intr_alloc_entry, uint8_t msix_vector,
4783     int8_t cause)
4784 {
4785     struct ixgbe_hw *hw = &ixgbe->hw;
4786     u32 ivar, index;
4787
4788     switch (hw->mac.type) {
4789     case ixgbe_mac_82598EB:
4790         msix_vector |= IXGBE_IVAR_ALLOC_VAL;
4791         if (cause == -1) {
4792             cause = 0;
4793         }
4794         index = (((cause * 64) + intr_alloc_entry) >> 2) & 0x1F;
4795         ivar = IXGBE_READ_REG(hw, IXGBE_IVAR(index));
4796         ivar &= ~(0xFF << (8 * (intr_alloc_entry & 0x3)));
4797         ivar |= (msix_vector << (8 * (intr_alloc_entry & 0x3)));
4798         IXGBE_WRITE_REG(hw, IXGBE_IVAR(index), ivar);
4799         break;
4800
4801     case ixgbe_mac_82599EB:

```

```

4802     case ixgbe_mac_X540:
4803         if (cause == -1) {
4804             /* other causes */
4805             msix_vector |= IXGBE_IVAR_ALLOC_VAL;
4806             index = (intr_alloc_entry & 1) * 8;
4807             ivar = IXGBE_READ_REG(hw, IXGBE_IVAR_MISC);
4808             ivar &= ~(0xFF << index);
4809             ivar |= (msix_vector << index);
4810             IXGBE_WRITE_REG(hw, IXGBE_IVAR_MISC, ivar);
4811         } else {
4812             /* tx or rx causes */
4813             msix_vector |= IXGBE_IVAR_ALLOC_VAL;
4814             index = ((16 * (intr_alloc_entry & 1)) + (8 * cause));
4815             ivar = IXGBE_READ_REG(hw,
4816                 IXGBE_IVAR(intr_alloc_entry >> 1));
4817             ivar &= ~(0xFF << index);
4818             ivar |= (msix_vector << index);
4819             IXGBE_WRITE_REG(hw, IXGBE_IVAR(intr_alloc_entry >> 1),
4820                 ivar);
4821         }
4822         break;
4823
4824     default:
4825         break;
4826     }
4827 }
4828
4829 /*
4830 * ixgbe_enable_ivar - Enable the given entry by setting the VAL bit of
4831 * given interrupt vector allocation register (IVAR).
4832 * cause:
4833 *   -1 : other cause
4834 *   0 : rx
4835 *   1 : tx
4836 */
4837 static void
4838 ixgbe_enable_ivar(ixgbe_t *ixgbe, uint16_t intr_alloc_entry, int8_t cause)
4839 {
4840     struct ixgbe_hw *hw = &ixgbe->hw;
4841     u32 ivar, index;
4842
4843     switch (hw->mac.type) {
4844     case ixgbe_mac_82598EB:
4845         if (cause == -1) {
4846             cause = 0;
4847         }
4848         index = (((cause * 64) + intr_alloc_entry) >> 2) & 0x1F;
4849         ivar = IXGBE_READ_REG(hw, IXGBE_IVAR(index));
4850         ivar |= (IXGBE_IVAR_ALLOC_VAL << (8 *
4851             (intr_alloc_entry & 0x3)));
4852         IXGBE_WRITE_REG(hw, IXGBE_IVAR(index), ivar);
4853         break;
4854
4855     case ixgbe_mac_82599EB:
4856     case ixgbe_mac_X540:
4857         if (cause == -1) {
4858             /* other causes */
4859             index = (intr_alloc_entry & 1) * 8;
4860             ivar = IXGBE_READ_REG(hw, IXGBE_IVAR_MISC);
4861             ivar |= (IXGBE_IVAR_ALLOC_VAL << index);
4862             IXGBE_WRITE_REG(hw, IXGBE_IVAR_MISC, ivar);
4863         } else {
4864             /* tx or rx causes */
4865             index = ((16 * (intr_alloc_entry & 1)) + (8 * cause));
4866             ivar = IXGBE_READ_REG(hw,
4867                 IXGBE_IVAR(intr_alloc_entry >> 1));

```

```

4868             ivar |= (IXGBE_IVAR_ALLOC_VAL << index);
4869             IXGBE_WRITE_REG(hw, IXGBE_IVAR(intr_alloc_entry >> 1),
4870                 ivar);
4871         }
4872         break;

4874     default:
4875         break;
4876 }
4877 }

4879 /*
4880  * ixgbe_disable_ivar - Disble the given entry by clearing the VAL bit of
4881  * given interrupt vector allocation register (IVAR).
4882  * cause:
4883  *   -1 : other cause
4884  *   0  : rx
4885  *   1  : tx
4886  */
4887 static void
4888 ixgbe_disable_ivar(ixgbe_t *ixgbe, uint16_t intr_alloc_entry, int8_t cause)
4889 {
4890     struct ixgbe_hw *hw = &ixgbe->hw;
4891     u32 ivar, index;

4893     switch (hw->mac.type) {
4894     case ixgbe_mac_82598EB:
4895         if (cause == -1) {
4896             cause = 0;
4897         }
4898         index = (((cause * 64) + intr_alloc_entry) >> 2) & 0x1F;
4899         ivar = IXGBE_READ_REG(hw, IXGBE_IVAR(index));
4900         ivar &= ~(IXGBE_IVAR_ALLOC_VAL << (8 *
4901             (intr_alloc_entry & 0x3)));
4902         IXGBE_WRITE_REG(hw, IXGBE_IVAR(index), ivar);
4903         break;

4905     case ixgbe_mac_82599EB:
4906     case ixgbe_mac_X540:
4907         if (cause == -1) {
4908             /* other causes */
4909             index = (intr_alloc_entry & 1) * 8;
4910             ivar = IXGBE_READ_REG(hw, IXGBE_IVAR_MISC);
4911             ivar &= ~(IXGBE_IVAR_ALLOC_VAL << index);
4912             IXGBE_WRITE_REG(hw, IXGBE_IVAR_MISC, ivar);
4913         } else {
4914             /* tx or rx causes */
4915             index = ((16 * (intr_alloc_entry & 1)) + (8 * cause));
4916             ivar = IXGBE_READ_REG(hw,
4917                 IXGBE_IVAR(intr_alloc_entry >> 1));
4918             ivar &= ~(IXGBE_IVAR_ALLOC_VAL << index);
4919             IXGBE_WRITE_REG(hw, IXGBE_IVAR(intr_alloc_entry >> 1),
4920                 ivar);
4921         }
4922         break;

4924     default:
4925         break;
4926 }
4927 }

4929 /*
4930  * Convert the rx ring index driver maintained to the rx ring index
4931  * in h/w.
4932  */
4933 static uint32_t

```

```

4934 ixgbe_get_hw_rx_index(ixgbe_t *ixgbe, uint32_t sw_rx_index)
4935 {
4937     struct ixgbe_hw *hw = &ixgbe->hw;
4938     uint32_t rx_ring_per_group, hw_rx_index;

4940     if (ixgbe->classify_mode == IXGBE_CLASSIFY_RSS ||
4941         ixgbe->classify_mode == IXGBE_CLASSIFY_NONE) {
4942         return (sw_rx_index);
4943     } else if (ixgbe->classify_mode == IXGBE_CLASSIFY_VMDQ) {
4944         switch (hw->mac.type) {
4945             case ixgbe_mac_82598EB:
4946                 return (sw_rx_index);

4948             case ixgbe_mac_82599EB:
4949             case ixgbe_mac_X540:
4950                 return (sw_rx_index * 2);

4952             default:
4953                 break;
4954         }
4955     } else if (ixgbe->classify_mode == IXGBE_CLASSIFY_VMDQ_RSS) {
4956         rx_ring_per_group = ixgbe->num_rx_rings / ixgbe->num_rx_groups;

4958         switch (hw->mac.type) {
4959             case ixgbe_mac_82598EB:
4960                 hw_rx_index = (sw_rx_index / rx_ring_per_group) *
4961                     16 + (sw_rx_index % rx_ring_per_group);
4962                 return (hw_rx_index);

4964             case ixgbe_mac_82599EB:
4965             case ixgbe_mac_X540:
4966                 if (ixgbe->num_rx_groups > 32) {
4967                     hw_rx_index = (sw_rx_index /
4968                         rx_ring_per_group) * 2 +
4969                         (sw_rx_index % rx_ring_per_group);
4970                 } else {
4971                     hw_rx_index = (sw_rx_index /
4972                         rx_ring_per_group) * 4 +
4973                         (sw_rx_index % rx_ring_per_group);
4974                 }
4975                 return (hw_rx_index);

4977             default:
4978                 break;
4979         }
4980     }

4982     /*
4983      * Should never reach. Just to make compiler happy.
4984      */
4985     return (sw_rx_index);
4986 }

4987 unchanged_portion_omitted

5045 /*
5046  * ixgbe_setup_adapter_vector - Setup the adapter interrupt vector(s).
5047  *
5048  * This relies on ring/vector mapping already set up in the
5049  * vect_map[] structures
5050  */
5051 static void
5052 ixgbe_setup_adapter_vector(ixgbe_t *ixgbe)
5053 {
5054     struct ixgbe_hw *hw = &ixgbe->hw;
5055     ixgbe_intr_vector_t *vect; /* vector bitmap */

```

```

5056     int r_idx;      /* ring index */
5057     int v_idx;      /* vector index */
5058     uint32_t hw_index;

5060     /*
5061      * Clear any previous entries
5062      */
5063     switch (hw->mac.type) {
5064     case ixgbe_mac_82598EB:
5065         for (v_idx = 0; v_idx < 25; v_idx++)
5066             IXGBE_WRITE_REG(hw, IXGBE_IVAR(v_idx), 0);
5067         break;

5069     case ixgbe_mac_82599EB:
5070     case ixgbe_mac_X540:
5071         for (v_idx = 0; v_idx < 64; v_idx++)
5072             IXGBE_WRITE_REG(hw, IXGBE_IVAR(v_idx), 0);
5073             IXGBE_WRITE_REG(hw, IXGBE_IVAR_MISC, 0);
5074         break;

5076     default:
5077         break;
5078     }

5080     /*
5081      * For non MSI-X interrupt, rx rings[0] will use RTxQ[0], and
5082      * tx rings[0] will use RTxQ[1].
5083      */
5084     if (ixgbe->intr_type != DDI_INTR_TYPE_MSIX) {
5085         ixgbe_setup_ivar(ixgbe, 0, 0, 0);
5086         ixgbe_setup_ivar(ixgbe, 0, 1, 1);
5087         return;
5088     }

5090     /*
5091      * For MSI-X interrupt, "Other" is always on vector[0].
5092      */
5093     ixgbe_setup_ivar(ixgbe, IXGBE_IVAR_OTHER_CAUSES_INDEX, 0, -1);

5095     /*
5096      * For each interrupt vector, populate the IVAR table
5097      */
5098     for (v_idx = 0; v_idx < ixgbe->intr_cnt; v_idx++) {
5099         vect = &ixgbe->vect_map[v_idx];

5101         /*
5102          * For each rx ring bit set
5103          */
5104         r_idx = bt_getlowbit(vect->rx_map, 0,
5105             (ixgbe->num_rx_rings - 1));

5107         while (r_idx >= 0) {
5108             hw_index = ixgbe->rx_rings[r_idx].hw_index;
5109             ixgbe_setup_ivar(ixgbe, hw_index, v_idx, 0);
5110             r_idx = bt_getlowbit(vect->rx_map, (r_idx + 1),
5111                 (ixgbe->num_rx_rings - 1));
5112         }

5114         /*
5115          * For each tx ring bit set
5116          */
5117         r_idx = bt_getlowbit(vect->tx_map, 0,
5118             (ixgbe->num_tx_rings - 1));

5120         while (r_idx >= 0) {
5121             ixgbe_setup_ivar(ixgbe, r_idx, v_idx, 1);

```

```

5122         r_idx = bt_getlowbit(vect->tx_map, (r_idx + 1),
5123             (ixgbe->num_tx_rings - 1));
5124     }
5125 }
5126 }
_____unchanged_portion_omitted_____

```

```

*****
19731 Fri Jul 20 10:59:38 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_mbx.c
XXX Intel X540 support
*****
1 /*****

3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD$*/
34
35 #include "ixgbe_type.h"
36 #include "ixgbe_mbx.h"
37
38 /**
39 * ixgbe_read_mbx - Reads a message from the mailbox
40 * @hw: pointer to the HW structure
41 * @msg: The message buffer
42 * @size: Length of buffer
43 * @mbx_id: id of mailbox to read
44 *
45 * returns SUCCESS if it successfully read message from buffer
46 */
47 s32 ixgbe_read_mbx(struct ixgbe_hw *hw, u32 *msg, u16 size, u16 mbx_id)
48 {
49     struct ixgbe_mbx_info *mbx = &hw->mbx;
50     s32 ret_val = IXGBE_ERR_MBX;
51
52     DEBUGFUNC("ixgbe_read_mbx");
53
54     /* limit read to size of mailbox */
55     if (size > mbx->size)
56         size = mbx->size;
57
58     if (mbx->ops.read)
59         ret_val = mbx->ops.read(hw, msg, size, mbx_id);
60
61     return ret_val;

```

```

62 }
63
64 /**
65 * ixgbe_write_mbx - Write a message to the mailbox
66 * @hw: pointer to the HW structure
67 * @msg: The message buffer
68 * @size: Length of buffer
69 * @mbx_id: id of mailbox to write
70 *
71 * returns SUCCESS if it successfully copied message into the buffer
72 */
73 s32 ixgbe_write_mbx(struct ixgbe_hw *hw, u32 *msg, u16 size, u16 mbx_id)
74 {
75     struct ixgbe_mbx_info *mbx = &hw->mbx;
76     s32 ret_val = IXGBE_SUCCESS;
77
78     DEBUGFUNC("ixgbe_write_mbx");
79
80     if (size > mbx->size)
81         ret_val = IXGBE_ERR_MBX;
82
83     else if (mbx->ops.write)
84         ret_val = mbx->ops.write(hw, msg, size, mbx_id);
85
86     return ret_val;
87 }
88
89 /**
90 * ixgbe_check_for_msg - checks to see if someone sent us mail
91 * @hw: pointer to the HW structure
92 * @mbx_id: id of mailbox to check
93 *
94 * returns SUCCESS if the Status bit was found or else ERR_MBX
95 */
96 s32 ixgbe_check_for_msg(struct ixgbe_hw *hw, u16 mbx_id)
97 {
98     struct ixgbe_mbx_info *mbx = &hw->mbx;
99     s32 ret_val = IXGBE_ERR_MBX;
100
101     DEBUGFUNC("ixgbe_check_for_msg");
102
103     if (mbx->ops.check_for_msg)
104         ret_val = mbx->ops.check_for_msg(hw, mbx_id);
105
106     return ret_val;
107 }
108
109 /**
110 * ixgbe_check_for_ack - checks to see if someone sent us ACK
111 * @hw: pointer to the HW structure
112 * @mbx_id: id of mailbox to check
113 *
114 * returns SUCCESS if the Status bit was found or else ERR_MBX
115 */
116 s32 ixgbe_check_for_ack(struct ixgbe_hw *hw, u16 mbx_id)
117 {
118     struct ixgbe_mbx_info *mbx = &hw->mbx;
119     s32 ret_val = IXGBE_ERR_MBX;
120
121     DEBUGFUNC("ixgbe_check_for_ack");
122
123     if (mbx->ops.check_for_ack)
124         ret_val = mbx->ops.check_for_ack(hw, mbx_id);
125
126     return ret_val;
127 }

```

```

129 /**
130  * ixgbe_check_for_rst - checks to see if other side has reset
131  * @hw: pointer to the HW structure
132  * @mbx_id: id of mailbox to check
133  *
134  * returns SUCCESS if the Status bit was found or else ERR_MBX
135  */
136 s32 ixgbe_check_for_rst(struct ixgbe_hw *hw, u16 mbx_id)
137 {
138     struct ixgbe_mbx_info *mbx = &hw->mbx;
139     s32 ret_val = IXGBE_ERR_MBX;
140
141     DEBUGFUNC("ixgbe_check_for_rst");
142
143     if (mbx->ops.check_for_rst)
144         ret_val = mbx->ops.check_for_rst(hw, mbx_id);
145
146     return ret_val;
147 }
148
149 /**
150  * ixgbe_poll_for_msg - Wait for message notification
151  * @hw: pointer to the HW structure
152  * @mbx_id: id of mailbox to write
153  *
154  * returns SUCCESS if it successfully received a message notification
155  */
156 static s32 ixgbe_poll_for_msg(struct ixgbe_hw *hw, u16 mbx_id)
157 {
158     struct ixgbe_mbx_info *mbx = &hw->mbx;
159     int countdown = mbx->timeout;
160
161     DEBUGFUNC("ixgbe_poll_for_msg");
162
163     if (!countdown || !mbx->ops.check_for_msg)
164         goto out;
165
166     while (countdown && mbx->ops.check_for_msg(hw, mbx_id)) {
167         countdown--;
168         if (!countdown)
169             break;
170         usec_delay(mbx->usec_delay);
171     }
172
173 out:
174     return countdown ? IXGBE_SUCCESS : IXGBE_ERR_MBX;
175 }
176
177 /**
178  * ixgbe_poll_for_ack - Wait for message acknowledgement
179  * @hw: pointer to the HW structure
180  * @mbx_id: id of mailbox to write
181  *
182  * returns SUCCESS if it successfully received a message acknowledgement
183  */
184 static s32 ixgbe_poll_for_ack(struct ixgbe_hw *hw, u16 mbx_id)
185 {
186     struct ixgbe_mbx_info *mbx = &hw->mbx;
187     int countdown = mbx->timeout;
188
189     DEBUGFUNC("ixgbe_poll_for_ack");
190
191     if (!countdown || !mbx->ops.check_for_ack)
192         goto out;

```

```

194     while (countdown && mbx->ops.check_for_ack(hw, mbx_id)) {
195         countdown--;
196         if (!countdown)
197             break;
198         usec_delay(mbx->usec_delay);
199     }
200
201 out:
202     return countdown ? IXGBE_SUCCESS : IXGBE_ERR_MBX;
203 }
204
205 /**
206  * ixgbe_read_posted_mbx - Wait for message notification and receive message
207  * @hw: pointer to the HW structure
208  * @msg: The message buffer
209  * @size: Length of buffer
210  * @mbx_id: id of mailbox to write
211  *
212  * returns SUCCESS if it successfully received a message notification and
213  * copied it into the receive buffer.
214  */
215 s32 ixgbe_read_posted_mbx(struct ixgbe_hw *hw, u32 *msg, u16 size, u16 mbx_id)
216 {
217     struct ixgbe_mbx_info *mbx = &hw->mbx;
218     s32 ret_val = IXGBE_ERR_MBX;
219
220     DEBUGFUNC("ixgbe_read_posted_mbx");
221
222     if (!mbx->ops.read)
223         goto out;
224
225     ret_val = ixgbe_poll_for_msg(hw, mbx_id);
226
227     /* if ack received read message, otherwise we timed out */
228     if (!ret_val)
229         ret_val = mbx->ops.read(hw, msg, size, mbx_id);
230 out:
231     return ret_val;
232 }
233
234 /**
235  * ixgbe_write_posted_mbx - Write a message to the mailbox, wait for ack
236  * @hw: pointer to the HW structure
237  * @msg: The message buffer
238  * @size: Length of buffer
239  * @mbx_id: id of mailbox to write
240  *
241  * returns SUCCESS if it successfully copied message into the buffer and
242  * received an ack to that message within delay * timeout period
243  */
244 s32 ixgbe_write_posted_mbx(struct ixgbe_hw *hw, u32 *msg, u16 size,
245                             u16 mbx_id)
246 {
247     struct ixgbe_mbx_info *mbx = &hw->mbx;
248     s32 ret_val = IXGBE_ERR_MBX;
249
250     DEBUGFUNC("ixgbe_write_posted_mbx");
251
252     /* exit if either we can't write or there isn't a defined timeout */
253     if (!mbx->ops.write || !mbx->timeout)
254         goto out;
255
256     /* send msg */
257     ret_val = mbx->ops.write(hw, msg, size, mbx_id);
258
259     /* 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_PFAK)) {
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 successfully 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_PFBMEM(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_PFBMEM(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 Fri Jul 20 10:59:39 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_mbx.h
XXX Intel X540 support
*****
1 /*****

3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD$*/
34
35 #ifndef _IXGBE_MBX_H_
36 #define _IXGBE_MBX_H_
37
38 #include "ixgbe_type.h"
39
40 #define IXGBE_VFMAILBOX_SIZE 16 /* 16 32 bit words - 64 bytes */
41 #define IXGBE_ERR_MBX -100
42
43 #define IXGBE_VFMAILBOX 0x002FC
44 #define IXGBE_VFMBMEM 0x00200
45
46 /* Define mailbox register bits */
47 #define IXGBE_VFMAILBOX_REQ 0x00000001 /* Request for PF Ready bit */
48 #define IXGBE_VFMAILBOX_ACK 0x00000002 /* Ack PF message received */
49 #define IXGBE_VFMAILBOX_VFU 0x00000004 /* VF owns the mailbox buffer */
50 #define IXGBE_VFMAILBOX_PFU 0x00000008 /* PF owns the mailbox buffer */
51 #define IXGBE_VFMAILBOX_PFSTS 0x00000010 /* PF wrote a message in the MB */
52 #define IXGBE_VFMAILBOX_PFAK 0x00000020 /* PF ack the previous VF msg */
53 #define IXGBE_VFMAILBOX_RSTI 0x00000040 /* PF has reset indication */
54 #define IXGBE_VFMAILBOX_RSTD 0x00000080 /* PF has indicated reset done */
55 #define IXGBE_VFMAILBOX_R2C_BITS 0x000000B0 /* All read to clear bits */
56
57 #define IXGBE_PFMAILBOX_STS 0x00000001 /* Initiate message send to VF */
58 #define IXGBE_PFMAILBOX_ACK 0x00000002 /* Ack message rcv'd from VF */
59 #define IXGBE_PFMAILBOX_VFU 0x00000004 /* VF owns the mailbox buffer */
60 #define IXGBE_PFMAILBOX_PFU 0x00000008 /* PF owns the mailbox buffer */
61 #define IXGBE_PFMAILBOX_RVU 0x00000010 /* Reset VFU - used when VF stuck */

```

```

63 #define IXGBE_MBVFICR_VFREQ_MASK 0x0000FFFF /* bits for VF messages */
64 #define IXGBE_MBVFICR_VFREQ_VF1 0x00000001 /* bit for VF 1 message */
65 #define IXGBE_MBVFICR_VFACK_MASK 0xFFFF0000 /* bits for VF acks */
66 #define IXGBE_MBVFICR_VFACK_VF1 0x00010000 /* bit for VF 1 ack */
67
68 /* If it's a IXGBE_VF_* msg then it originates in the VF and is sent to the
69 * PF. The reverse is TRUE if it is IXGBE_PF_*.
70 * Message ACK's are the value or'd with 0xF0000000
71 */
72 #define IXGBE_VT_MSGTYPE_ACK 0x80000000 /* Messages below or'd with
73 * this are the ACK */
74 #define IXGBE_VT_MSGTYPE_NACK 0x40000000 /* Messages below or'd with
75 * this are the NACK */
76 #define IXGBE_VT_MSGTYPE_CTS 0x20000000 /* Indicates that VF is still
77 * clear to send requests */
78 #define IXGBE_VT_MSGINFO_SHIFT 16
79 /* bits 23:16 are used for extra info for certain messages */
80 #define IXGBE_VT_MSGINFO_MASK (0xFF << IXGBE_VT_MSGINFO_SHIFT)
81
82 #define IXGBE_VF_RESET 0x01 /* VF requests reset */
83 #define IXGBE_VF_SET_MAC_ADDR 0x02 /* VF requests PF to set MAC addr */
84 #define IXGBE_VF_SET_MULTICAST 0x03 /* VF requests PF to set MC addr */
85 #define IXGBE_VF_SET_VLAN 0x04 /* VF requests PF to set VLAN */
86 #define IXGBE_VF_SET_LPE 0x05 /* VF requests PF to set VMOLR.LPE */
87 #define IXGBE_VF_SET_MACVLAN 0x06 /* VF requests PF for unicast filter */
88
89 /* length of permanent address message returned from PF */
90 #define IXGBE_VF_PERMADDR_MSG_LEN 4
91 /* word in permanent address message with the current multicast type */
92 #define IXGBE_VF_MC_TYPE_WORD 3
93
94 #define IXGBE_PF_CONTROL_MSG 0x0100 /* PF control message */
95
96 #define IXGBE_VF_MBX_INIT_TIMEOUT 2000 /* number of retries on mailbox */
97 #define IXGBE_VF_MBX_INIT_DELAY 500 /* microseconds between retries */
98
99 s32 ixgbe_read_mbx(struct ixgbe_hw *, u32 *, u16, u16);
100 s32 ixgbe_write_mbx(struct ixgbe_hw *, u32 *, u16, u16);
101 s32 ixgbe_read_posted_mbx(struct ixgbe_hw *, u32 *, u16, u16);
102 s32 ixgbe_write_posted_mbx(struct ixgbe_hw *, u32 *, u16, u16);
103 s32 ixgbe_check_for_msg(struct ixgbe_hw *, u16);
104 s32 ixgbe_check_for_ack(struct ixgbe_hw *, u16);
105 s32 ixgbe_check_for_rst(struct ixgbe_hw *, u16);
106 void ixgbe_init_mbx_ops_generic(struct ixgbe_hw *hw);
107 void ixgbe_init_mbx_params_vf(struct ixgbe_hw *);
108 void ixgbe_init_mbx_params_pf(struct ixgbe_hw *);
109
110 #endif /* _IXGBE_MBX_H_ */

```

new/usr/src/uts/common/io/ixgbe/ixgbe_osdep.h

1

```
*****
3898 Fri Jul 20 10:59:39 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_osdep.h
XXX 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 */
29 /*
30  * Copyright (c) 2012, Joyent, Inc. All rights reserved.
31 */

33 #ifndef _IXGBE_OSDEP_H
34 #define _IXGBE_OSDEP_H

36 #ifdef __cplusplus
37 extern "C" {
38 #endif

40 #include <sys/types.h>
41 #include <sys/byteorder.h>
42 #include <sys/conf.h>
43 #include <sys/debug.h>
44 #include <sys/stropts.h>
45 #include <sys/stream.h>
46 #include <sys/strlog.h>
47 #include <sys/kmem.h>
48 #include <sys/stat.h>
49 #include <sys/kstat.h>
50 #include <sys/modctl.h>
51 #include <sys/errno.h>
52 #include <sys/ddi.h>
53 #include <sys/dditypes.h>
54 #include <sys/sunddi.h>
55 #include <sys/pci.h>
56 #include <sys/atomic.h>
57 #include <sys/note.h>
58 #include "ixgbe_debug.h"

60 /* Cheesy hack for EWARN() */
61 #define EWARN(H, W, S) printf(W)
```

new/usr/src/uts/common/io/ixgbe/ixgbe_osdep.h

2

```
63 /* function declarations */
64 struct ixgbe_hw;
65 uint16_t ixgbe_read_pci_cfg(struct ixgbe_hw *, uint32_t);
66 void ixgbe_write_pci_cfg(struct ixgbe_hw *, uint32_t, uint32_t);

68 #define usec_delay(x)      drv_usecwait(x)
69 #define msec_delay(x)     drv_usecwait(x * 1000)

71 #define OS_DEP(hw)        ((struct ixgbe_osdep *)((hw)->back))

73 #define false             B_FALSE
74 #define true              B_TRUE
75 #define FALSE             B_FALSE
76 #define TRUE              B_TRUE

78 #define IXGBE_READ_PCIE_WORD    ixgbe_read_pci_cfg
79 #define IXGBE_WRITE_PCIE_WORD  ixgbe_write_pci_cfg
80 #define CMD_MEM_WRT_INVALIDATE 0x0010 /* BIT_4 */
81 #define PCI_COMMAND_REGISTER   0x04
82 #define PCI_EX_CONF_CAP        0xE0
83 #define SPEED_10GB             10000
84 #define SPEED_1GB              1000
85 #define SPEED_100              100
86 #define FULL_DUPLEX            2

88 #define IXGBE_WRITE_FLUSH(a)    (void) IXGBE_READ_REG(a, IXGBE_STATUS)

90 #define IXGBE_WRITE_REG(a, reg, value) \
91     ddi_put32((OS_DEP(a))->reg_handle, \
92     (uint32_t *)((uintptr_t)(a)->hw_addr + reg), (value))

94 #define IXGBE_WRITE_REG_ARRAY(a, reg, index, value) \
95     IXGBE_WRITE_REG(a, ((reg) + ((index) << 2)), (value))

97 #define IXGBE_READ_REG(a, reg) \
98     ddi_get32((OS_DEP(a))->reg_handle, \
99     (uint32_t *)((uintptr_t)(a)->hw_addr + reg))

101 #define IXGBE_READ_REG_ARRAY(a, reg, index) \
102     IXGBE_READ_REG(a, ((reg) + ((index) << 2)))

104 #define msec_delay_irq msec_delay
105 #define IXGBE_HTONL    htonl
106 #define IXGBE_NTOHL    ntohl
107 #define IXGBE_NTOHS    ntohs

109 #ifdef _BIG_ENDIAN
110 #define IXGBE_CPU_TO_LE32    BSWAP_32
111 #define IXGBE_LE32_TO_CPUS   BSWAP_32
112 #else
113 #define IXGBE_CPU_TO_LE32(x)    (x)
114 #define IXGBE_LE32_TO_CPUS(x)   (x)
115 #endif /* _BIG_ENDIAN */

117 #define UNREFERENCED_PARAMETER(x)    _NOTE(ARGUNUSED(x))
118 #define UNREFERENCED_1PARAMETER(_p)  UNREFERENCED_PARAMETER(_p)
119 #define UNREFERENCED_2PARAMETER(_p, _q)  _NOTE(ARGUNUSED(_p, _q))
120 #define UNREFERENCED_3PARAMETER(_p, _q, _r)  _NOTE(ARGUNUSED(_p, _q, _r))
121 #define UNREFERENCED_4PARAMETER(_p, _q, _r, _s)  _NOTE(ARGUNUSED(_p, _q, _r, _s))

125 typedef int8_t      s8;
126 typedef int16_t     s16;
127 typedef int32_t     s32;
```

new/usr/src/uts/common/io/ixgbe/ixgbe_osdep.h

3

```
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
```

new/usr/src/uts/common/io/ixgbe/ixgbe_phy.c

1

```
*****
51266 Fri Jul 20 10:59:40 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_phy.c
XXX Intel X540 support
*****
1 /*****
3 Copyright (c) 2001-2012, Intel Corporation
4 Copyright (c) 2001-2010, Intel Corporation
5 All rights reserved.
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_phy.c,v 1.13 2012/07/05 20:51:44 jfv Exp $*/
34 /*$FreeBSD$*/
35 #include "ixgbe_api.h"
36 #include "ixgbe_common.h"
37 #include "ixgbe_phy.h"
38
39 static void ixgbe_i2c_start(struct ixgbe_hw *hw);
40 static void ixgbe_i2c_stop(struct ixgbe_hw *hw);
41 static s32 ixgbe_clock_in_i2c_byte(struct ixgbe_hw *hw, u8 *data);
42 static s32 ixgbe_clock_out_i2c_byte(struct ixgbe_hw *hw, u8 data);
43 static s32 ixgbe_get_i2c_ack(struct ixgbe_hw *hw);
44 static s32 ixgbe_clock_in_i2c_bit(struct ixgbe_hw *hw, bool *data);
45 static s32 ixgbe_clock_out_i2c_bit(struct ixgbe_hw *hw, bool data);
46 static void ixgbe_raise_i2c_clk(struct ixgbe_hw *hw, u32 *i2ccctl);
47 static s32 ixgbe_raise_i2c_clk(struct ixgbe_hw *hw, u32 *i2ccctl);
48 static void ixgbe_lower_i2c_clk(struct ixgbe_hw *hw, u32 *i2ccctl);
49 static s32 ixgbe_set_i2c_data(struct ixgbe_hw *hw, u32 *i2ccctl, bool data);
50 bool ixgbe_get_i2c_data(u32 *i2ccctl);
51 void ixgbe_i2c_bus_clear(struct ixgbe_hw *hw);
52
53 /**
54 * ixgbe_init_phy_ops_generic - Inits PHY function ptrs
55 * @hw: pointer to the hardware structure
56 *
57 * Initialize the function pointers.
58 */
59 s32 ixgbe_init_phy_ops_generic(struct ixgbe_hw *hw)
```

new/usr/src/uts/common/io/ixgbe/ixgbe_phy.c

2

```
58 {
59     struct ixgbe_phy_info *phy = &hw->phy;
60
61     DEBUGFUNC("ixgbe_init_phy_ops_generic");
62
63     /* PHY */
64     phy->ops.identify = &ixgbe_identify_phy_generic;
65     phy->ops.reset = &ixgbe_reset_phy_generic;
66     phy->ops.read_reg = &ixgbe_read_phy_reg_generic;
67     phy->ops.write_reg = &ixgbe_write_phy_reg_generic;
68     phy->ops.setup_link = &ixgbe_setup_phy_link_generic;
69     phy->ops.setup_link_speed = &ixgbe_setup_phy_link_speed_generic;
70     phy->ops.check_link = NULL;
71     phy->ops.get_firmware_version = ixgbe_get_phy_firmware_version_generic;
72     phy->ops.read_i2c_byte = &ixgbe_read_i2c_byte_generic;
73     phy->ops.write_i2c_byte = &ixgbe_write_i2c_byte_generic;
74     phy->ops.read_i2c_eeprom = &ixgbe_read_i2c_eeprom_generic;
75     phy->ops.write_i2c_eeprom = &ixgbe_write_i2c_eeprom_generic;
76     phy->ops.i2c_bus_clear = &ixgbe_i2c_bus_clear;
77     phy->ops.identify_sfp = &ixgbe_identify_module_generic;
78     phy->ops.identify_sfp = &ixgbe_identify_sfp_module_generic;
79     phy->sfp_type = ixgbe_sfp_type_unknown;
80     phy->ops.check_overtmp = &ixgbe_tn_check_overtmp;
81     return IXGBE_SUCCESS;
82 }
83
84 /**
85 * ixgbe_identify_phy_generic - Get physical layer module
86 * @hw: pointer to hardware structure
87 *
88 * Determines the physical layer module found on the current adapter.
89 */
90 s32 ixgbe_identify_phy_generic(struct ixgbe_hw *hw)
91 {
92     s32 status = IXGBE_ERR_PHY_ADDR_INVALID;
93     u32 phy_addr;
94     u16 ext_ability = 0;
95
96     DEBUGFUNC("ixgbe_identify_phy_generic");
97
98     if (hw->phy.type == ixgbe_phy_unknown) {
99         for (phy_addr = 0; phy_addr < IXGBE_MAX_PHY_ADDR; phy_addr++) {
100             if (ixgbe_validate_phy_addr(hw, phy_addr)) {
101                 hw->phy.addr = phy_addr;
102                 ixgbe_get_phy_id(hw);
103                 (void) ixgbe_get_phy_id(hw);
104                 hw->phy.type =
105                     ixgbe_get_phy_type_from_id(hw->phy.id);
106
107                 if (hw->phy.type == ixgbe_phy_unknown) {
108                     hw->phy.ops.read_reg(hw,
109                         IXGBE_MDIO_PHY_EXT_ABILITY,
110                         IXGBE_MDIO_PMA_PMD_DEV_TYPE,
111                         &ext_ability);
112                     if (ext_ability &
113                         (IXGBE_MDIO_PHY_10GBASET_ABILITY |
114                         IXGBE_MDIO_PHY_1000BASET_ABILITY))
115                         hw->phy.type =
116                             ixgbe_phy_cu_unknown;
117                     else
118                         hw->phy.type =
119                             ixgbe_phy_generic;
120                 }
121
122                 status = IXGBE_SUCCESS;
123                 break;
124             }
125         }
126     }
127
128     return status;
129 }
```

new/usr/src/uts/common/io/ixgbe/ixgbe_phy.c

3

```
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 }
133 unchanged_portion_omitted
134 /**
135  * ixgbe_get_phy_type_from_id - Get the phy type
136  * @hw: pointer to hardware structure
137  */
138 enum ixgbe_phy_type ixgbe_get_phy_type_from_id(u32 phy_id)
139 {
140     enum ixgbe_phy_type phy_type;
141
142     DEBUGFUNC("ixgbe_get_phy_type_from_id");
143
144     switch (phy_id) {
145     case TN1010_PHY_ID:
146         phy_type = ixgbe_phy_tn;
147         break;
148     case X540_PHY_ID:
149     case AQ1002_PHY_ID:
150         phy_type = ixgbe_phy_aq;
151         break;
152     case QT2022_PHY_ID:
153         phy_type = ixgbe_phy_qt;
154         break;
155     case ATH_PHY_ID:
156         phy_type = ixgbe_phy_n1;
157         break;
158     default:
159         phy_type = ixgbe_phy_unknown;
160         break;
161     }
162
163     DEBUGOUT1("phy type found is %d\n", phy_type);
164     return phy_type;
165 }
166 unchanged_portion_omitted
167 /**
168  * ixgbe_read_phy_reg_generic - Reads a value from a specified PHY register
169  * @hw: pointer to hardware structure
170  * @reg_addr: 32 bit address of PHY register to read
171  * @phy_data: Pointer to read data from PHY register
172  */
173 s32 ixgbe_read_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
174                               u32 device_type, u16 *phy_data)
175 {
176     u32 command;
177     u32 i;
178     u32 data;
179     s32 status = IXGBE_SUCCESS;
180     u16 gssr;
181
182     DEBUGFUNC("ixgbe_read_phy_reg_generic");
183
184     if (IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_LAN_ID_1)
```

new/usr/src/uts/common/io/ixgbe/ixgbe_phy.c

4

```
290         gssr = IXGBE_GSSR_PHY1_SM;
291     else
292         gssr = IXGBE_GSSR_PHY0_SM;
293
294     if (hw->mac.ops.acquire_swfw_sync(hw, gssr) != IXGBE_SUCCESS)
295     if (ixgbe_acquire_swfw_sync(hw, gssr) != IXGBE_SUCCESS)
296         status = IXGBE_ERR_SWFW_SYNC;
297
298     if (status == IXGBE_SUCCESS) {
299         /* Setup and write the address cycle command */
300         command = ((reg_addr << IXGBE_MSCA_NP_ADDR_SHIFT) |
301                  (device_type << IXGBE_MSCA_DEV_TYPE_SHIFT) |
302                  (hw->phy.addr << IXGBE_MSCA_PHY_ADDR_SHIFT) |
303                  (IXGBE_MSCA_ADDR_CYCLE | IXGBE_MSCA_MDI_COMMAND));
304
305         IXGBE_WRITE_REG(hw, IXGBE_MSCA, command);
306
307         /*
308          * Check every 10 usec to see if the address cycle completed.
309          * The MDI Command bit will clear when the operation is
310          * complete
311          */
312         for (i = 0; i < IXGBE_MDIO_COMMAND_TIMEOUT; i++) {
313             usec_delay(10);
314
315             command = IXGBE_READ_REG(hw, IXGBE_MSCA);
316
317             if ((command & IXGBE_MSCA_MDI_COMMAND) == 0)
318                 break;
319         }
320
321         if ((command & IXGBE_MSCA_MDI_COMMAND) != 0) {
322             DEBUGOUT("PHY address command did not complete.\n");
323             status = IXGBE_ERR_PHY;
324         }
325
326         if (status == IXGBE_SUCCESS) {
327             /*
328              * Address cycle complete, setup and write the read
329              * command
330              */
331             command = ((reg_addr << IXGBE_MSCA_NP_ADDR_SHIFT) |
332                      (device_type << IXGBE_MSCA_DEV_TYPE_SHIFT) |
333                      (hw->phy.addr << IXGBE_MSCA_PHY_ADDR_SHIFT) |
334                      (IXGBE_MSCA_READ | IXGBE_MSCA_MDI_COMMAND));
335
336             IXGBE_WRITE_REG(hw, IXGBE_MSCA, command);
337
338             /*
339              * Check every 10 usec to see if the address cycle
340              * completed. The MDI Command bit will clear when the
341              * operation is complete
342              */
343             for (i = 0; i < IXGBE_MDIO_COMMAND_TIMEOUT; i++) {
344                 usec_delay(10);
345
346                 command = IXGBE_READ_REG(hw, IXGBE_MSCA);
347
348                 if ((command & IXGBE_MSCA_MDI_COMMAND) == 0)
349                     break;
350             }
351
352             if ((command & IXGBE_MSCA_MDI_COMMAND) != 0) {
353                 DEBUGOUT("PHY read command didn't complete\n");
354                 status = IXGBE_ERR_PHY;
355             } 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);
367 }

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);
461     }

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);
489
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;
493
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     }
498
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);
505
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;
509
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     }
515
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);
521
522         autoneg_reg &= ~(IXGBE_MII_100BASE_T_ADVERTISE |
523                         IXGBE_MII_100BASE_T_ADVERTISE_HALF);
524         if (hw->phy.autoneg_advertised & IXGBE_LINK_SPEED_100_FULL)
525             autoneg_reg |= IXGBE_MII_100BASE_T_ADVERTISE;
526
527         hw->phy.ops.write_reg(hw, IXGBE_MII_AUTONEG_ADVERTISE_REG,
528                             IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
529                             autoneg_reg);
530     }
531
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);
535
536     autoneg_reg |= IXGBE_MII_RESTART;
537
538     hw->phy.ops.write_reg(hw, IXGBE_MDIO_AUTO_NEG_CONTROL,
539                         IXGBE_MDIO_AUTO_NEG_DEV_TYPE, autoneg_reg);
540
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)
551             if (autoneg_reg == IXGBE_MII_AUTONEG_COMPLETE) {
552                 break;
553             }
554     }
555     if (time_out == max_time_out) {
556         status = IXGBE_ERR_LINK_SETUP;
557         DEBUGOUT("ixgbe_setup_phy_link_generic: time out");
558     }
559     return status;
560 }
561
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);
576
577     DEBUGFUNC("ixgbe_setup_phy_link_speed_generic");
578
579     /*
580      * Clear autoneg_advertised and set new values based on input link
581      * speed.
582      */
583     hw->phy.autoneg_advertised = 0;
584
585     if (speed & IXGBE_LINK_SPEED_10GB_FULL)
586         hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_10GB_FULL;
587
588     if (speed & IXGBE_LINK_SPEED_1GB_FULL)
589         hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_1GB_FULL;
590
591     if (speed & IXGBE_LINK_SPEED_100_FULL)
592         hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_100_FULL;
593
594     /* Setup link based on the new speed settings */
595     hw->phy.ops.setup_link(hw);
596
597     return IXGBE_SUCCESS;
598 }
599
600 unchanged_portion_omitted
601
602 /**
603  * ixgbe_check_phy_link_tnx - Determine link and speed status
604  * @hw: pointer to hardware structure
605  *
606  * Reads the VS1 register to determine if link is up and the current speed for
607  * the PHY.
608  */
609 s32 ixgbe_check_phy_link_tnx(struct ixgbe_hw *hw, ixgbe_link_speed *speed,
610                             bool *link_up)
611 {
612     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;
669         phy_link = phy_data &
670                 IXGBE_MDIO_VENDOR_SPECIFIC_1_LINK_STATUS;
671         phy_speed = phy_data &
672                 IXGBE_MDIO_VENDOR_SPECIFIC_1_SPEED_STATUS;
673         if (phy_link == IXGBE_MDIO_VENDOR_SPECIFIC_1_LINK_STATUS) {
674             *link_up = TRUE;
675             if (phy_speed ==
676                 IXGBE_MDIO_VENDOR_SPECIFIC_1_SPEED_STATUS)
677                 *speed = IXGBE_LINK_SPEED_1GB_FULL;
678             break;
679         }
680     }
681     return status;
682 }
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);
705     if (speed & IXGBE_LINK_SPEED_10GB_FULL) {
706         /* Set or unset auto-negotiation 10G advertisement */
707         hw->phy.ops.read_reg(hw, IXGBE_MII_10GBASE_T_AUTONEG_CTRL_REG,
708                             IXGBE_MDIO_AUTO_NEG_DEV_TYPE,
709                             &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)
766         if (autoneg_reg == IXGBE_MII_AUTONEG_COMPLETE) {
767             break;
768         }
769     }
771     if (time_out == max_time_out) {
772         status = IXGBE_ERR_LINK_SETUP;
773         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");
876         DEBUGOUT("DATA:  \n");
877         data_offset++;
878         hw->eeprom.ops.read(hw, data_offset++,
879                             &phy_offset);
880         for (i = 0; i < edata; i++) {
881             hw->eeprom.ops.read(hw, data_offset, &eword);
882             hw->phy.ops.write_reg(hw, phy_offset,
883                                 IXGBE_TWIXAX_DEV, eword);
884             DEBUGOUT2("Wrote %4.4x to %4.4x\n", eword,
885                       phy_offset);
886             data_offset++;
887             phy_offset++;
888         }
889         break;
890     case IXGBE_CONTROL_NL:
891         data_offset++;
892         DEBUGOUT("CONTROL:\n");
893         DEBUGOUT("CONTROL:  \n");
894         if (edata == IXGBE_CONTROL_EOL_NL) {
895             DEBUGOUT("EOL\n");
896             end_data = TRUE;
897         } else if (edata == IXGBE_CONTROL_SOL_NL) {
898             DEBUGOUT("SOL\n");
899         } else {
900             DEBUGOUT("Bad control value\n");
901             ret_val = IXGBE_ERR_PHY;
902             goto out;
903         }
904     default:
905         DEBUGOUT("Bad control type\n");
906         ret_val = IXGBE_ERR_PHY;
907         goto out;
908     }
909 }

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 }

941 /**
942  * ixgbe_identify_sfp_module_generic - Identifies SFP modules
943  * @hw: pointer to hardware structure
944  * Searches for and identifies the SFP module and assigns appropriate PHY type.
945  */
946 s32 ixgbe_identify_sfp_module_generic(struct ixgbe_hw *hw)
947 {
948     s32 status = IXGBE_ERR_PHY_ADDR_INVALID;
949     u32 vendor_oui = 0;
950     enum ixgbe_sfp_type stored_sfp_type = hw->phy.sfp_type;
951     u8 identifier = 0;
952     u8 comp_codes_1g = 0;
953     u8 comp_codes_10g = 0;
954     u8 oui_bytes[3] = {0, 0, 0};
955     u8 cable_tech = 0;
956     u8 cable_spec = 0;
957     u16 enforce_sfp = 0;

960     DEBUGFUNC("ixgbe_identify_sfp_module_generic");

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     }

968     status = hw->phy.ops.read_i2c_eeprom(hw,
969                                         IXGBE_SFF_IDENTIFIER,
970                                         &identifier);

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;

977     /* LAN ID is needed for sfp_type determination */
978     hw->mac.ops.set_lan_id(hw);

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);

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;

993         status = hw->phy.ops.read_i2c_eeprom(hw,
994                                             IXGBE_SFF_10GBE_COMP_CODES,
995                                             &comp_codes_10g);

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;

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_sr1r_core0;
1065             else
1066                 hw->phy.sfp_type =
1067                     ixgbe_sfp_type_sr1r_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_n1) {
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)) {
1177     hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0)) {
1178         hw->phy.type = ixgbe_phy_sfp_unsupported;
1179         status = IXGBE_ERR_SFP_NOT_SUPPORTED;
1180         goto out;
1181     }

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);
1189 (void) ixgbe_get_device_caps(hw, &enforce_sfp);
1190 if (!(enforce_sfp & IXGBE_DEVICE_CAPS_ALLOW_ANY_SFP) &&
1191     ((hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0) ||
1192     (hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1) ||
1193     (hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core0) ||
1194     (hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core1))) {
1195     /* Make sure we're a supported PHY type */
1196     if (hw->phy.type == ixgbe_phy_sfp_intel) {
1197         status = IXGBE_SUCCESS;
1198     } else {
1199         if (hw->allow_unsupported_sfp == TRUE) {
1200             EWARN(hw, "WARNING: Intel (R) Network "

```

```

1200         "Connections are quality tested "
1201         "using Intel (R) Ethernet Optics."
1202         " Using untested modules is not "
1203         "supported and may cause unstable"
1204         " operation or damage to the "
1205         "module or the adapter. Intel "
1206         "Corporation is not responsible "
1207         "for any harm caused by using "
1208         "untested modules.\n", status);
1209         status = IXGBE_SUCCESS;
1210     } else {
1211         DEBUGOUT("SFP+ module not supported\n");
1212         hw->phy.type =
1213             ixgbe_phy_sfp_unsupported;
1214         hw->phy.type = ixgbe_phy_sfp_unsupported;
1215         status = IXGBE_ERR_SFP_NOT_SUPPORTED;
1216     }
1217 } else {
1218     status = IXGBE_SUCCESS;
1219 }
1220 }

1222 out:
1223     return status;

1225 err_read_i2c_eeprom:
1226     hw->phy.sfp_type = ixgbe_sfp_type_not_present;
1227     if (hw->phy.type != ixgbe_phy_nl) {
1228         hw->phy.id = 0;
1229         hw->phy.type = ixgbe_phy_unknown;
1230     }
1231     return IXGBE_ERR_SFP_NOT_PRESENT;
1232 }

1236 /**
1237  * ixgbe_get_sfp_init_sequence_offsets - Provides offset of PHY init sequence
1238  * @hw: pointer to hardware structure
1239  * @list_offset: offset to the SFP ID list
1240  * @data_offset: offset to the SFP data block
1241  *
1242  * Checks the MAC's EEPROM to see if it supports a given SFP+ module type, if
1243  * so it returns the offsets to the phy init sequence block.
1244  */
1245 s32 ixgbe_get_sfp_init_sequence_offsets(struct ixgbe_hw *hw,
1246                                         u16 *list_offset,
1247                                         u16 *data_offset)
1248 {
1249     u16 sfp_id;
1250     u16 sfp_type = hw->phy.sfp_type;

1252     DEBUGFUNC("ixgbe_get_sfp_init_sequence_offsets");

1254     if (hw->phy.sfp_type == ixgbe_sfp_type_unknown)
1255         return IXGBE_ERR_SFP_NOT_SUPPORTED;

1257     if (hw->phy.sfp_type == ixgbe_sfp_type_not_present)
1258         return IXGBE_ERR_SFP_NOT_PRESENT;

1260     if ((hw->device_id == IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM) &&
1261         (hw->phy.sfp_type == ixgbe_sfp_type_da_cu))
1262         return IXGBE_ERR_SFP_NOT_SUPPORTED;

1264     /*

```

```

1265     * Limiting active cables and 1G Phys must be initialized as
1266     * SR modules
1267     */
1268     if (sfp_type == ixgbe_sfp_type_da_act_lmt_core0 ||
1269         sfp_type == ixgbe_sfp_type_1g_cu_core0 ||
1270         sfp_type == ixgbe_sfp_type_1g_sx_core0)
1271         sfp_type = ixgbe_sfp_type_1g_cu_core0;
1272     else if (sfp_type == ixgbe_sfp_type_da_act_lmt_core1 ||
1273             sfp_type == ixgbe_sfp_type_1g_cu_core1 ||
1274             sfp_type == ixgbe_sfp_type_1g_sx_core1)
1275         sfp_type = ixgbe_sfp_type_1g_cu_core1;
1276         sfp_type = ixgbe_sfp_type_sr1r_core1;

1277     /* Read offset to PHY init contents */
1278     hw->eeprom.ops.read(hw, IXGBE_PHY_INIT_OFFSET_NL, list_offset);

1280     if ((list_offset) || (*list_offset == 0xFFFF))
1281         return IXGBE_ERR_SFP_NO_INIT_SEQ_PRESENT;

1283     /* Shift offset to first ID word */
1284     (*list_offset)++;

1286     /*
1287     * Find the matching SFP ID in the EEPROM
1288     * and program the init sequence
1289     */
1290     hw->eeprom.ops.read(hw, *list_offset, &sfp_id);

1292     while (sfp_id != IXGBE_PHY_INIT_END_NL) {
1293         if (sfp_id == sfp_type) {
1294             (*list_offset)++;
1295             hw->eeprom.ops.read(hw, *list_offset, data_offset);
1296             if ((*data_offset) || (*data_offset == 0xFFFF)) {
1297                 DEBUGOUT("SFP+ module not supported\n");
1298                 return IXGBE_ERR_SFP_NOT_SUPPORTED;
1299             } else {
1300                 break;
1301             }
1302         } else {
1303             (*list_offset) += 2;
1304             if (hw->eeprom.ops.read(hw, *list_offset, &sfp_id))
1305                 return IXGBE_ERR_PHY;
1306         }
1307     }

1309     if (sfp_id == IXGBE_PHY_INIT_END_NL) {
1310         DEBUGOUT("No matching SFP+ module found\n");
1311         return IXGBE_ERR_SFP_NOT_SUPPORTED;
1312     }

1314     return IXGBE_SUCCESS;
1315 }

1316 unchanged_portion_omitted

1353 /**
1354  * ixgbe_read_i2c_byte_generic - Reads 8 bit word over I2C
1355  * @hw: pointer to hardware structure
1356  * @byte_offset: byte offset to read
1357  * @data: value read
1358  *
1359  * Performs byte read operation to SFP module's EEPROM over I2C interface at
1360  * a specified device address.
1361  * a specified device address.
1362  */
1362 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;
1371
1372     DEBUGFUNC("ixgbe_read_i2c_byte_generic");
1373
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;
1378
1379     do {
1380         if (hw->mac.ops.acquire_swfw_sync(hw, swfw_mask)
1381             != IXGBE_SUCCESS) {
1382             if (ixgbe_acquire_swfw_sync(hw, swfw_mask) != IXGBE_SUCCESS) {
1383                 status = IXGBE_ERR_SWFW_SYNC;
1384                 goto read_byte_out;
1385             }
1386
1387             ixgbe_i2c_start(hw);
1388
1389             /* Device Address and write indication */
1390             status = ixgbe_clock_out_i2c_byte(hw, dev_addr);
1391             if (status != IXGBE_SUCCESS)
1392                 goto fail;
1393
1394             status = ixgbe_get_i2c_ack(hw);
1395             if (status != IXGBE_SUCCESS)
1396                 goto fail;
1397
1398             status = ixgbe_clock_out_i2c_byte(hw, byte_offset);
1399             if (status != IXGBE_SUCCESS)
1400                 goto fail;
1401
1402             status = ixgbe_get_i2c_ack(hw);
1403             if (status != IXGBE_SUCCESS)
1404                 goto fail;
1405
1406             ixgbe_i2c_start(hw);
1407
1408             /* Device Address and read indication */
1409             status = ixgbe_clock_out_i2c_byte(hw, (dev_addr | 0x1));
1410             if (status != IXGBE_SUCCESS)
1411                 goto fail;
1412
1413             status = ixgbe_get_i2c_ack(hw);
1414             if (status != IXGBE_SUCCESS)
1415                 goto fail;
1416
1417             status = ixgbe_clock_in_i2c_byte(hw, data);
1418             if (status != IXGBE_SUCCESS)
1419                 goto fail;
1420
1421             status = ixgbe_clock_out_i2c_bit(hw, nack);
1422             if (status != IXGBE_SUCCESS)
1423                 goto fail;
1424
1425             ixgbe_i2c_stop(hw);
1426             break;
1427 fail:

```

1427 fail:

```

1428         hw->mac.ops.release_swfw_sync(hw, swfw_mask);
1429         ixgbe_release_swfw_sync(hw, swfw_mask);
1430         msec_delay(100);
1431         ixgbe_i2c_bus_clear(hw);
1432         retry++;
1433         if (retry < max_retry)
1434             DEBUGOUT("I2C byte read error - Retrying.\n");
1435         else
1436             DEBUGOUT("I2C byte read error.\n");
1437     } while (retry < max_retry);
1438
1439     hw->mac.ops.release_swfw_sync(hw, swfw_mask);
1440     ixgbe_release_swfw_sync(hw, swfw_mask);
1441
1442 read_byte_out:
1443     return status;
1444 }
1445
1446 /**
1447  * ixgbe_write_i2c_byte_generic - Writes 8 bit word over I2C
1448  * @hw: pointer to hardware structure
1449  * @byte_offset: byte offset to write
1450  * @data: value to write
1451  *
1452  * Performs byte write operation to SFP module's EEPROM over I2C interface at
1453  * a specified device address.
1454  */
1455 s32 ixgbe_write_i2c_byte_generic(struct ixgbe_hw *hw, u8 byte_offset,
1456                                u8 dev_addr, u8 data)
1457 {
1458     s32 status = IXGBE_SUCCESS;
1459     u32 max_retry = 1;
1460     u32 retry = 0;
1461     u16 swfw_mask = 0;
1462
1463     DEBUGFUNC("ixgbe_write_i2c_byte_generic");
1464
1465     if (IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_LAN_ID_1)
1466         swfw_mask = IXGBE_GSSR_PHY1_SM;
1467     else
1468         swfw_mask = IXGBE_GSSR_PHY0_SM;
1469
1470     if (hw->mac.ops.acquire_swfw_sync(hw, swfw_mask) != IXGBE_SUCCESS) {
1471         if (ixgbe_acquire_swfw_sync(hw, swfw_mask) != IXGBE_SUCCESS) {
1472             status = IXGBE_ERR_SWFW_SYNC;
1473             goto write_byte_out;
1474         }
1475     }
1476
1477     do {
1478         ixgbe_i2c_start(hw);
1479
1480         status = ixgbe_clock_out_i2c_byte(hw, dev_addr);
1481         if (status != IXGBE_SUCCESS)
1482             goto fail;
1483
1484         status = ixgbe_get_i2c_ack(hw);
1485         if (status != IXGBE_SUCCESS)
1486             goto fail;
1487
1488         status = ixgbe_clock_out_i2c_byte(hw, byte_offset);
1489         if (status != IXGBE_SUCCESS)
1490             goto fail;
1491
1492         status = ixgbe_get_i2c_ack(hw);
1493         if (status != IXGBE_SUCCESS)
1494             goto fail;
1495
1496         status = ixgbe_clock_in_i2c_byte(hw, data);
1497         if (status != IXGBE_SUCCESS)
1498             goto fail;
1499
1500         status = ixgbe_clock_out_i2c_bit(hw, nack);
1501         if (status != IXGBE_SUCCESS)
1502             goto fail;
1503
1504         ixgbe_i2c_stop(hw);
1505         break;
1506 fail:

```

```

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);
1546     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 i2ccctl = 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, &i2ccctl, 1);
1533     ixgbe_raise_i2c_clk(hw, &i2ccctl);
1545     (void) ixgbe_set_i2c_data(hw, &i2ccctl, 1);
1546     (void) ixgbe_raise_i2c_clk(hw, &i2ccctl);

1535     /* Setup time for start condition (4.7us) */
1536     usec_delay(IXGBE_I2C_T_SU_STA);

1538     ixgbe_set_i2c_data(hw, &i2ccctl, 0);
1541     (void) ixgbe_set_i2c_data(hw, &i2ccctl, 0);

1540     /* Hold time for start condition (4us) */
1541     usec_delay(IXGBE_I2C_T_HD_STA);

1543     ixgbe_lower_i2c_clk(hw, &i2ccctl);

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 i2ccctl = 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, &i2ccctl, 0);
1564     ixgbe_raise_i2c_clk(hw, &i2ccctl);
1566     (void) ixgbe_set_i2c_data(hw, &i2ccctl, 0);
1567     (void) ixgbe_raise_i2c_clk(hw, &i2ccctl);

1566     /* Setup time for stop condition (4us) */
1567     usec_delay(IXGBE_I2C_T_SU_STO);

1569     ixgbe_set_i2c_data(hw, &i2ccctl, 1);
1572     (void) ixgbe_set_i2c_data(hw, &i2ccctl, 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 {
1584     s32 status = IXGBE_SUCCESS;
1585     s32 i;
1586     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);
1593         status = ixgbe_clock_in_i2c_bit(hw, &bit);
1594         *data |= bit << i;

1597         if (status != IXGBE_SUCCESS)
1598             break;
1599     }

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 i2ccctl;
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;
1639     s32 i2cctl;
1640     u32 i = 0;
1641     u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1642     u32 timeout = 10;
1643     bool ack = 1;
1644     DEBUGFUNC("ixgbe_get_i2c_ack");
1646     ixgbe_raise_i2c_clk(hw, &i2cctl);
1647     status = ixgbe_raise_i2c_clk(hw, &i2cctl);
1649     if (status != IXGBE_SUCCESS)
1650         goto out;
1652     /* Minimum high period of clock is 4us */
1653     usec_delay(IXGBE_I2C_T_HIGH);
1655     /* Poll for ACK. Note that ACK in I2C spec is
1656      * transition from 1 to 0 */
1657     for (i = 0; i < timeout; i++) {
1658         i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1659         ack = ixgbe_get_i2c_data(&i2cctl);
1660         usec_delay(1);
1661         if (ack == 0)
1662             break;
1663     }
1664     if (ack == 1) {
1665         DEBUGOUT("I2C ack was not received.\n");
1666         status = IXGBE_ERR_I2C;
1667     }
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:
1622     return status;
1623 }
1624 }
1626 /**
1627  * ixgbe_clock_in_i2c_bit - Clocks in one bit via I2C data/clock
1628  * @hw: pointer to hardware structure
1629  * @data: read data value
1630  *
1631  * Clocks in one bit via I2C data/clock
1632  */
1633 static s32 ixgbe_clock_in_i2c_bit(struct ixgbe_hw *hw, bool *data)
1634 {
1635     s32 status;
1636     u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1638     DEBUGFUNC("ixgbe_clock_in_i2c_bit");
1639     ixgbe_raise_i2c_clk(hw, &i2cctl);
1640     status = ixgbe_raise_i2c_clk(hw, &i2cctl);
1642     /* Minimum high period of clock is 4us */
1643     usec_delay(IXGBE_I2C_T_HIGH);
1645     i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1646     *data = ixgbe_get_i2c_data(&i2cctl);
1648     ixgbe_lower_i2c_clk(hw, &i2cctl);
1650     /* Minimum low period of clock is 4.7 us */
1651     usec_delay(IXGBE_I2C_T_LOW);
1653     return IXGBE_SUCCESS;
1654     return status;
1655 }
1656 }
1658 /**
1659  * ixgbe_clock_out_i2c_bit - Clocks in/out one bit via I2C data/clock
1660  * @hw: pointer to hardware structure
1661  * @data: data value to write
1662  *
1663  * Clocks out one bit via I2C data/clock
1664  */
1665 static s32 ixgbe_clock_out_i2c_bit(struct ixgbe_hw *hw, bool data)
1666 {
1667     s32 status;
1668     u32 i2cctl = IXGBE_READ_REG(hw, IXGBE_I2CCTL);
1670     DEBUGFUNC("ixgbe_clock_out_i2c_bit");
1672     status = ixgbe_set_i2c_data(hw, &i2cctl, data);
1673     if (status == IXGBE_SUCCESS) {
1674         ixgbe_raise_i2c_clk(hw, &i2cctl);
1675         status = ixgbe_raise_i2c_clk(hw, &i2cctl);
1677         /* Minimum high period of clock is 4us */
1678         usec_delay(IXGBE_I2C_T_HIGH);
1680         ixgbe_lower_i2c_clk(hw, &i2cctl);
1682         /* Minimum low period of clock is 4.7 us.
1683          * This also takes care of the data hold time.
1684          */
1685         usec_delay(IXGBE_I2C_T_LOW);
1686     } 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     }
1766     return status;

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 }
1823 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 occurred.
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_L0M)
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
```

new/usr/src/uts/common/io/ixgbe/ixgbe_phy.h

1

```
*****
5811 Fri Jul 20 10:59:41 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_phy.h
XXX Intel X540 support
*****
1 /*****

3 Copyright (c) 2001-2012, Intel Corporation
3 Copyright (c) 2001-2010, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.

32 *****/
33 /*$FreeBSD$*/

35 #ifndef _IXGBE_PHY_H_
36 #define _IXGBE_PHY_H_

38 #include "ixgbe_type.h"
39 #define IXGBE_I2C_EEPROM_DEV_ADDR 0xA0

41 /* EEPROM byte offsets */
42 #define IXGBE_SFF_IDENTIFIER 0x0
43 #define IXGBE_SFF_IDENTIFIER_SFP 0x3
44 #define IXGBE_SFF_VENDOR_OUI_BYTE0 0x25
45 #define IXGBE_SFF_VENDOR_OUI_BYTE1 0x26
46 #define IXGBE_SFF_VENDOR_OUI_BYTE2 0x27
47 #define IXGBE_SFF_1GBE_COMP_CODES 0x6
48 #define IXGBE_SFF_10GBE_COMP_CODES 0x3
49 #define IXGBE_SFF_CABLE_TECHNOLOGY 0x8
50 #define IXGBE_SFF_CABLE_SPEC_COMP 0x3C

52 /* Bitmasks */
53 #define IXGBE_SFF_DA_PASSIVE_CABLE 0x4
54 #define IXGBE_SFF_DA_ACTIVE_CABLE 0x8
55 #define IXGBE_SFF_DA_SPEC_ACTIVE_LIMITING 0x4
56 #define IXGBE_SFF_1GBASESX_CAPABLE 0x1
57 #define IXGBE_SFF_1GBASELX_CAPABLE 0x2
58 #define IXGBE_SFF_1GBASET_CAPABLE 0x8
59 #define IXGBE_SFF_10GBASESR_CAPABLE 0x10
60 #define IXGBE_SFF_10GBASELR_CAPABLE 0x20
```

new/usr/src/uts/common/io/ixgbe/ixgbe_phy.h

2

```
61 #define IXGBE_I2C_EEPROM_READ_MASK 0x100
62 #define IXGBE_I2C_EEPROM_STATUS_MASK 0x3
63 #define IXGBE_I2C_EEPROM_STATUS_NO_OPERATION 0x0
64 #define IXGBE_I2C_EEPROM_STATUS_PASS 0x1
65 #define IXGBE_I2C_EEPROM_STATUS_FAIL 0x2
66 #define IXGBE_I2C_EEPROM_STATUS_IN_PROGRESS 0x3

68 /* Flow control defines */
69 #define IXGBE_TAF_SYM_PAUSE 0x400
70 #define IXGBE_TAF_ASM_PAUSE 0x800

72 /* Bit-shift macros */
73 #define IXGBE_SFF_VENDOR_OUI_BYTE0_SHIFT 24
74 #define IXGBE_SFF_VENDOR_OUI_BYTE1_SHIFT 16
75 #define IXGBE_SFF_VENDOR_OUI_BYTE2_SHIFT 8

77 /* Vendor OUIs: format of OUI is 0x[byte0][byte1][byte2][00] */
78 #define IXGBE_SFF_VENDOR_OUI_TYCO 0x00407600
79 #define IXGBE_SFF_VENDOR_OUI_FTL 0x00906500
80 #define IXGBE_SFF_VENDOR_OUI_AVAGO 0x00176A00
81 #define IXGBE_SFF_VENDOR_OUI_INTEL 0x001B2100

83 /* I2C SDA and SCL timing parameters for standard mode */
84 #define IXGBE_I2C_T_HD_STA 4
85 #define IXGBE_I2C_T_LOW 5
86 #define IXGBE_I2C_T_HIGH 4
87 #define IXGBE_I2C_T_SU_STA 5
88 #define IXGBE_I2C_T_HD_DATA 5
89 #define IXGBE_I2C_T_SU_DATA 1
90 #define IXGBE_I2C_T_RISE 1
91 #define IXGBE_I2C_T_FALL 1
92 #define IXGBE_I2C_T_SU_STO 4
93 #define IXGBE_I2C_T_BUF 5

95 #define IXGBE_TN_LASI_STATUS_REG 0x9005
96 #define IXGBE_TN_LASI_STATUS_TEMP_ALARM 0x0008

98 s32 ixgbe_init_phy_ops_generic(struct ixgbe_hw *hw);
99 bool ixgbe_validate_phy_addr(struct ixgbe_hw *hw, u32 phy_addr);
100 enum ixgbe_phy_type ixgbe_get_phy_type_from_id(u32 phy_id);
101 s32 ixgbe_get_phy_id(struct ixgbe_hw *hw);
102 s32 ixgbe_identify_phy_generic(struct ixgbe_hw *hw);
103 s32 ixgbe_reset_phy_generic(struct ixgbe_hw *hw);
104 s32 ixgbe_read_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
105 u32 device_type, u16 *phy_data);
106 s32 ixgbe_write_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
107 u32 device_type, u16 phy_data);
108 s32 ixgbe_setup_phy_link_generic(struct ixgbe_hw *hw);
109 s32 ixgbe_setup_phy_link_speed_generic(struct ixgbe_hw *hw,
110 ixgbe_link_speed speed,
111 bool autoneg,
112 bool autoneg_wait_to_complete);
113 s32 ixgbe_get_copper_link_capabilities_generic(struct ixgbe_hw *hw,
114 ixgbe_link_speed *speed,
115 bool *autoneg);

117 /* PHY specific */
118 s32 ixgbe_check_phy_link_tnx(struct ixgbe_hw *hw,
119 ixgbe_link_speed *speed,
120 bool *link_up);
121 s32 ixgbe_setup_phy_link_tnx(struct ixgbe_hw *hw);
122 s32 ixgbe_get_phy_firmware_version_tnx(struct ixgbe_hw *hw,
123 u16 *firmware_version);
124 s32 ixgbe_get_phy_firmware_version_generic(struct ixgbe_hw *hw,
125 u16 *firmware_version);
```

```
127 s32 ixgbe_reset_phy_n1(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_ */
```

new/usr/src/uts/common/io/ixgbe/ixgbe_stat.c

1

```
*****
24248 Fri Jul 20 10:59:41 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_stat.c
XXX 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 */

30 #include "ixgbe_sw.h"

32 /*
33  * Update driver private statistics.
34 */
35 static int
36 ixgbe_update_stats(kstat_t *ks, int rw)
37 {
38     ixgbe_t *ixgbe;
39     struct ixgbe_hw *hw;
40     ixgbe_stat_t *ixgbe_ks;
41     int i;

43     if (rw == KSTAT_WRITE)
44         return (EACCES);

46     ixgbe = (ixgbe_t *)ks->ks_private;
47     ixgbe_ks = (ixgbe_stat_t *)ks->ks_data;
48     hw = &ixgbe->hw;

50     mutex_enter(&ixgbe->gen_lock);

52     /*
53      * Basic information
54      */
55     ixgbe_ks->link_speed.value.ui64 = ixgbe->link_speed;
56     ixgbe_ks->reset_count.value.ui64 = ixgbe->reset_count;
57     ixgbe_ks->lroc.value.ui64 = ixgbe->lro_pkt_count;

59 #ifdef IXGBE_DEBUG
60     ixgbe_ks->rx_frame_error.value.ui64 = 0;
61     ixgbe_ks->rx_cksum_error.value.ui64 = 0;
```

new/usr/src/uts/common/io/ixgbe/ixgbe_stat.c

2

```
62     ixgbe_ks->rx_exceed_pkt.value.ui64 = 0;
63     for (i = 0; i < ixgbe->num_rx_rings; i++) {
64         ixgbe_ks->rx_frame_error.value.ui64 +=
65             ixgbe->rx_rings[i].stat_frame_error;
66         ixgbe_ks->rx_cksum_error.value.ui64 +=
67             ixgbe->rx_rings[i].stat_cksum_error;
68         ixgbe_ks->rx_exceed_pkt.value.ui64 +=
69             ixgbe->rx_rings[i].stat_exceed_pkt;
70     }

72     ixgbe_ks->tx_overload.value.ui64 = 0;
73     ixgbe_ks->tx_fail_no_tbd.value.ui64 = 0;
74     ixgbe_ks->tx_fail_no_tcb.value.ui64 = 0;
75     ixgbe_ks->tx_fail_dma_bind.value.ui64 = 0;
76     ixgbe_ks->tx_reschedule.value.ui64 = 0;
77     for (i = 0; i < ixgbe->num_tx_rings; i++) {
78         ixgbe_ks->tx_overload.value.ui64 +=
79             ixgbe->tx_rings[i].stat_overload;
80         ixgbe_ks->tx_fail_no_tbd.value.ui64 +=
81             ixgbe->tx_rings[i].stat_fail_no_tbd;
82         ixgbe_ks->tx_fail_no_tcb.value.ui64 +=
83             ixgbe->tx_rings[i].stat_fail_no_tcb;
84         ixgbe_ks->tx_fail_dma_bind.value.ui64 +=
85             ixgbe->tx_rings[i].stat_fail_dma_bind;
86         ixgbe_ks->tx_reschedule.value.ui64 +=
87             ixgbe->tx_rings[i].stat_reschedule;
88     }
89 #endif

91     /*
92      * Hardware calculated statistics.
93      */
94     ixgbe_ks->gprc.value.ui64 = 0;
95     ixgbe_ks->gptc.value.ui64 = 0;
96     ixgbe_ks->tor.value.ui64 = 0;
97     ixgbe_ks->tot.value.ui64 = 0;
98     for (i = 0; i < 16; i++) {
99         ixgbe_ks->qprc[i].value.ui64 +=
100             IXGBE_READ_REG(hw, IXGBE_QPRC(i));
101         ixgbe_ks->gprc.value.ui64 += ixgbe_ks->qprc[i].value.ui64;
102         ixgbe_ks->qptc[i].value.ui64 +=
103             IXGBE_READ_REG(hw, IXGBE_QPTC(i));
104         ixgbe_ks->gptc.value.ui64 += ixgbe_ks->qptc[i].value.ui64;
105         ixgbe_ks->qbrc[i].value.ui64 +=
106             IXGBE_READ_REG(hw, IXGBE_QBRC(i));
107         ixgbe_ks->tor.value.ui64 += ixgbe_ks->qbrc[i].value.ui64;
108         switch (hw->mac.type) {
109             case ixgbe_mac_82598EB:
110                 ixgbe_ks->qbtc[i].value.ui64 +=
111                     IXGBE_READ_REG(hw, IXGBE_QBTC(i));
112                 break;

114             case ixgbe_mac_82599EB:
115                 case ixgbe_mac_X540:
116                     ixgbe_ks->qbtc[i].value.ui64 +=
117                         IXGBE_READ_REG(hw, IXGBE_QBTC_L(i));
118                     ixgbe_ks->qbtc[i].value.ui64 +=
119                         ((uint64_t)(IXGBE_READ_REG(hw,
120                             IXGBE_QBTC_H(i))) & 0xF) << 32);
121                     break;

123             default:
124                 break;
125         }
126         ixgbe_ks->tot.value.ui64 += ixgbe_ks->qbtc[i].value.ui64;
127     }
```

```

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->mrhc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_MRHC);
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->rhc.value.ui64 += IXGBE_READ_REG(hw, IXGBE_RHC);
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 (EI0);
768     }

770     return (0);
771 }

```

unchanged portion omitted

new/usr/src/uts/common/io/ixgbe/ixgbe_tx.c

1

```
*****
41329 Fri Jul 20 10:59:42 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_tx.c
XXX 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
```

new/usr/src/uts/common/io/ixgbe/ixgbe_tx.c

2

```
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.
1083  * 82599 requires the packet length in paylen field with or without
1084  * LSO and 82598 will ignore it in non-LSO mode.
1085  */
```

```

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

new/usr/src/uts/common/io/ixgbe/ixgbe_type.h

1

```
*****
124904 Fri Jul 20 10:59:43 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_type.h
XXX Intel X540 support
*****
1 /*****

3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_type.h,v 1.14 2012/07/05 20:51:44 jfv Exp $*/
34
35 #ifndef _IXGBE_TYPE_H_
36 #define _IXGBE_TYPE_H_
37
38 #include "ixgbe_osdep.h"
39
40
41 /* Vendor ID */
42 #define IXGBE_INTEL_VENDOR_ID 0x8086
43
44 /* Device IDs */
45 #define IXGBE_DEV_ID_82598 0x10B6
46 #define IXGBE_DEV_ID_82598_BX 0x1508
47 #define IXGBE_DEV_ID_82598AF_DUAL_PORT 0x10C6
48 #define IXGBE_DEV_ID_82598AF_SINGLE_PORT 0x10C7
49 #define IXGBE_DEV_ID_82598AT 0x10C8
50 #define IXGBE_DEV_ID_82598AT2 0x150B
51 #define IXGBE_DEV_ID_82598EB_SFP_LOM 0x10DB
52 #define IXGBE_DEV_ID_82598EB_CX4 0x10DD
53 #define IXGBE_DEV_ID_82598_CX4_DUAL_PORT 0x10EC
54 #define IXGBE_DEV_ID_82598_DA_DUAL_PORT 0x10F1
55 #define IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM 0x10E1
56 #define IXGBE_DEV_ID_82598EB_XF_LR 0x10F4
57 #define IXGBE_DEV_ID_82599_KX4 0x10F7
58 #define IXGBE_DEV_ID_82599_KX4_MEZZ 0x1514
59 #define IXGBE_DEV_ID_82599_KR 0x1517
60 #define IXGBE_DEV_ID_82599_COMBO_BACKPLANE 0x10F8
61 #define IXGBE_SUBDEV_ID_82599_KX4_KR_MEZZ 0x000C
```

new/usr/src/uts/common/io/ixgbe/ixgbe_type.h

2

```
62 #define IXGBE_DEV_ID_82599_CX4 0x10F9
63 #define IXGBE_DEV_ID_82599_SFP 0x10FB
64 #define IXGBE_SUBDEV_ID_82599_SFP 0x11A9
65 #define IXGBE_SUBDEV_ID_82599_560FLR 0x17D0
66 #define IXGBE_DEV_ID_82599_BACKPLANE_FCOE 0x152A
67 #define IXGBE_DEV_ID_82599_SFP_FCOE 0x1529
68 #define IXGBE_DEV_ID_82599_SFP_EM 0x1507
69 #define IXGBE_DEV_ID_82599_SFP_SF2 0x154D
70 #define IXGBE_DEV_ID_82599EN_SFP 0x1557
71 #define IXGBE_DEV_ID_82599_XAUI_LOM 0x10FC
72 #define IXGBE_DEV_ID_82599_T3_LOM 0x151C
73 #define IXGBE_DEV_ID_82599_VF 0x10ED
74 #define IXGBE_DEV_ID_X540_VF 0x1515
75 #define IXGBE_DEV_ID_X540T 0x1528
76 #define IXGBE_DEV_ID_X540T1 0x1560

77
78 /* General Registers */
79 #define IXGBE_CTRL 0x00000
80 #define IXGBE_STATUS 0x00008
81 #define IXGBE_CTRL_EXT 0x00018
82 #define IXGBE_ESDP 0x00020
83 #define IXGBE_EODSDP 0x00028
84 #define IXGBE_I2CCTL 0x00028
85 #define IXGBE_PHY_GPIO 0x00028
86 #define IXGBE_MAC_GPIO 0x00030
87 #define IXGBE_PHYINT_STATUS0 0x00100
88 #define IXGBE_PHYINT_STATUS1 0x00104
89 #define IXGBE_PHYINT_STATUS2 0x00108
90 #define IXGBE_LEDCTL 0x00200
91 #define IXGBE_FRTIMER 0x00048
92 #define IXGBE_TCTIMER 0x0004C
93 #define IXGBE_CORESPARE 0x00600
94 #define IXGBE_EXVET 0x05078

95
96 /* NVM Registers */
97 #define IXGBE_EEC 0x10010
98 #define IXGBE_EERD 0x10014
99 #define IXGBE_EEWR 0x10018
100 #define IXGBE_FLA 0x1001C
101 #define IXGBE_EEMNGCTL 0x10110
102 #define IXGBE_EEMNGDATA 0x10114
103 #define IXGBE_FLMNGCTL 0x10118
104 #define IXGBE_FLMNGDATA 0x1011C
105 #define IXGBE_FLMNGCNT 0x10120
106 #define IXGBE_FLOP 0x1013C
107 #define IXGBE_GRC 0x10200
108 #define IXGBE_SRAMREL 0x10210
109 #define IXGBE_PHYDBG 0x10218

110
111 /* General Receive Control */
112 #define IXGBE_GRC_MNG 0x00000001 /* Manageability Enable */
113 #define IXGBE_GRC_APME 0x00000002 /* APM enabled in EEPROM */
114
115 #define IXGBE_VPDDIAG0 0x10204
116 #define IXGBE_VPDDIAG1 0x10208

117
118 /* I2CCTL Bit Masks */
119 #define IXGBE_I2C_CLK_IN 0x00000001
120 #define IXGBE_I2C_CLK_OUT 0x00000002
121 #define IXGBE_I2C_DATA_IN 0x00000004
122 #define IXGBE_I2C_DATA_OUT 0x00000008
123 #define IXGBE_I2C_CLOCK_STRETCHING_TIMEOUT 500

124
125
126 /* Interrupt Registers */
127 #define IXGBE_EICR 0x00800
```

```

128 #define IXGBE_EICS          0x00808
129 #define IXGBE_EIMS          0x00880
130 #define IXGBE_EIMC          0x00888
131 #define IXGBE_EIAC          0x00810
132 #define IXGBE_EIAM          0x00890
133 #define IXGBE_EICS_EX(_i)   ((0x00A90 + (_i) * 4)
134 #define IXGBE_EIMS_EX(_i)   ((0x00AA0 + (_i) * 4)
135 #define IXGBE_EIMC_EX(_i)   ((0x00AB0 + (_i) * 4)
136 #define IXGBE_EIAM_EX(_i)   ((0x00AD0 + (_i) * 4)
137 /* 82599 EITR is only 12 bits, with the lower 3 always zero */
138 /*
139 * 82598 EITR is 16 bits but set the limits based on the max
140 * supported by all ixgbe hardware
141 */
142 #define IXGBE_MAX_INT_RATE   488281
143 #define IXGBE_MIN_INT_RATE   956
144 #define IXGBE_MAX_EITR      0x0000FF8
145 #define IXGBE_MIN_EITR      8
146 #define IXGBE_EITR(_i)      (((_i) <= 23) ? (0x00820 + ((_i) * 4)) : \
147                               (0x012300 + (((_i) - 24) * 4)))
148 #define IXGBE_EITR_ITR_INT_MASK 0x0000FF8
149 #define IXGBE_EITR_LLI_MOD    0x00008000
150 #define IXGBE_EITR_CNT_WDIS   0x80000000
151 #define IXGBE_IVAR(_i)        ((0x00900 + ((_i) * 4)) /* 24 at 0x900-0x960 */
152 #define IXGBE_IVAR_MISC      0x00A00 /* misc MSI-X interrupt causes */
153 #define IXGBE_EITRSEL        0x00894
154 #define IXGBE_MSIXT          0x00000 /* MSI-X Table. 0x0000 - 0x01C */
155 #define IXGBE_MSIXPBA        0x02000 /* MSI-X Pending bit array */
156 #define IXGBE_PBAACL(_i) (((_i) == 0) ? (0x11068) : (0x110C0 + ((_i) * 4)))
157 #define IXGBE_GPIE           0x00898

159 /* Flow Control Registers */
160 #define IXGBE_FCADBUL        0x03210
161 #define IXGBE_FCADBUH        0x03214
162 #define IXGBE_FCAMACL        0x04328
163 #define IXGBE_FCAMACH        0x0432C
164 #define IXGBE_FCRTH_82599(_i) (0x03260 + ((_i) * 4)) /* 8 of these (0-7) */
165 #define IXGBE_FCRTH_82599(_i) (0x03220 + ((_i) * 4)) /* 8 of these (0-7) */
166 #define IXGBE_PFCCTOP        0x03008
167 #define IXGBE_FCTTV(_i)      (0x03200 + ((_i) * 4)) /* 4 of these (0-3) */
168 #define IXGBE_FCRTH(_i)      (0x03220 + ((_i) * 8)) /* 8 of these (0-7) */
169 #define IXGBE_FCRTH(_i)      (0x03260 + ((_i) * 8)) /* 8 of these (0-7) */
170 #define IXGBE_FCRTH         0x032A0
171 #define IXGBE_FCCFG          0x03D00
172 #define IXGBE_TFCS           0x0CE00

174 /* Receive DMA Registers */
175 #define IXGBE_RDBAL(_i) (((_i) < 64) ? (0x01000 + ((_i) * 0x40)) : \
176                           (0x0D000 + (((_i) - 64) * 0x40)))
177 #define IXGBE_RDBAH(_i) (((_i) < 64) ? (0x01004 + ((_i) * 0x40)) : \
178                           (0x0D004 + (((_i) - 64) * 0x40)))
179 #define IXGBE_RDLEN(_i) (((_i) < 64) ? (0x01008 + ((_i) * 0x40)) : \
180                           (0x0D008 + (((_i) - 64) * 0x40)))
181 #define IXGBE_RDH(_i) (((_i) < 64) ? (0x01010 + ((_i) * 0x40)) : \
182                           (0x0D010 + (((_i) - 64) * 0x40)))
183 #define IXGBE_RDT(_i) (((_i) < 64) ? (0x01018 + ((_i) * 0x40)) : \
184                           (0x0D018 + (((_i) - 64) * 0x40)))
185 #define IXGBE_RXDCTL(_i) (((_i) < 64) ? (0x01028 + ((_i) * 0x40)) : \
186                           (0x0D028 + (((_i) - 64) * 0x40)))
187 #define IXGBE_RSCCTL(_i) (((_i) < 64) ? (0x0102C + ((_i) * 0x40)) : \

```

```

188                               (0x0D02C + (((_i) - 64) * 0x40)))
189 #define IXGBE_RSCDBU        0x03028
190 #define IXGBE_RDDCC         0x02F20
191 #define IXGBE_RXMEMWRAP     0x03190
192 #define IXGBE_STARCTRL      0x03024
193 /*
194 * Split and Replication Receive Control Registers
195 * 00-15 : 0x02100 + n*4
196 * 16-64 : 0x01014 + n*0x40
197 * 64-127: 0x0D014 + (n-64)*0x40
198 */
199 #define IXGBE_SRRCTL(_i) (((_i) <= 15) ? (0x02100 + ((_i) * 4)) : \
200                               (((_i) < 64) ? (0x01014 + (((_i) * 0x40)) : \
201                               (0x0D014 + (((_i) - 64) * 0x40))))))
187                               (0x0D014 + (((_i) - 64) * 0x40)))
202 /*
203 * Rx DCA Control Register:
204 * 00-15 : 0x02200 + n*4
205 * 16-64 : 0x0100C + n*0x40
206 * 64-127: 0x0D00C + (n-64)*0x40
207 */
208 #define IXGBE_DCA_RXCTRL(_i) (((_i) <= 15) ? (0x02200 + ((_i) * 4)) : \
209                               (((_i) < 64) ? (0x0100C + (((_i) * 0x40)) : \
210                               (0x0D00C + (((_i) - 64) * 0x40))))))
196                               (0x0D00C + (((_i) - 64) * 0x40)))
211 #define IXGBE_RDRXCTL      0x02F00
212 #define IXGBE_RDRXCTL_RSC_PUSH 0x80
213 /* 8 of these 0x03C00 - 0x03C1C */
214 #define IXGBE_RXPBSIZE(_i) (0x03C00 + ((_i) * 4))
200                               /* 8 of these 0x03C00 - 0x03C1C */
215 #define IXGBE_RXCTRL       0x03000
216 #define IXGBE_DROPEN       0x03D04
217 #define IXGBE_RXPBSIZE_SHIFT 10

219 /* Receive Registers */
220 #define IXGBE_RXCSUM        0x05000
221 #define IXGBE_RFCTL         0x05008
222 #define IXGBE_DRECCCTL     0x02F08
223 #define IXGBE_DRECCCTL_DISABLE 0
224 #define IXGBE_DRECCCTL2    0x02F8C

226 /* Multicast Table Array - 128 entries */
227 #define IXGBE_MTA(_i)      (0x05200 + ((_i) * 4))
228 #define IXGBE_RAL(_i)      (((_i) <= 15) ? (0x05400 + ((_i) * 8)) : \
229                               (0x0A200 + ((_i) * 8)))
230 #define IXGBE_RAH(_i)      (((_i) <= 15) ? (0x05404 + ((_i) * 8)) : \
231                               (0x0A204 + ((_i) * 8)))
232 #define IXGBE_MPSAR_LO(_i) (0x0A600 + ((_i) * 8))
233 #define IXGBE_MPSAR_HI(_i) (0x0A604 + ((_i) * 8))
234 /* Packet split receive type */
235 #define IXGBE_PSRTYPE(_i) (((_i) <= 15) ? (0x05480 + ((_i) * 4)) : \
236                               (0x0EA00 + ((_i) * 4)))
237 /* array of 4096 1-bit vlan filters */
238 #define IXGBE_VFTA(_i)      (0x0A000 + ((_i) * 4))
239 /* array of 4096 4-bit vlan vmdq indices */
240 #define IXGBE_VFTAIND(_j, _i) (0x0A200 + ((_j) * 0x200) + ((_i) * 4))
241 #define IXGBE_FCTRL         0x05080
242 #define IXGBE_VLNCTRL       0x05088
243 #define IXGBE_MCSTCTRL      0x05090
244 #define IXGBE_MRQC          0x05818
245 #define IXGBE_SAQF(_i)      (0x0E000 + ((_i) * 4)) /* Source Address Queue Filter */
246 #define IXGBE_DAF(_i)       (0x0E200 + ((_i) * 4)) /* Dest. Address Queue Filter */
247 #define IXGBE_SDPQF(_i)      (0x0E400 + ((_i) * 4)) /* Src Dest. Addr Queue Filter */
248 #define IXGBE_FTQF(_i)       (0x0E600 + ((_i) * 4)) /* Five Tuple Queue Filter */
249 #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_MTOC 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_PFMALBOX(_i) (0x04B00 + (4 * (_i))) /* 64 total */
259 /* 64 Mailboxes, 16 DW each */
260 #define IXGBE_PFMEM(_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_VMLR(_i) (0x0F000 + ((_i) * 4)) /* 64 total */
269 #define IXGBE_UTA(_i) (0x0F400 + ((_i) * 4))
270 #define IXGBE_MRCTL(_i) (0x0F600 + ((_i) * 4))
271 #define IXGBE_VMRCTL(_i) (0x0F600 + ((_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_RSSA(_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) */
313 #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_PVFPSP00F(_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_PVFPSP00F_REG_COUNT 8
343 /* 16 of these (0-15) */
344 #define IXGBE_DCA_TXCTRL(_i) (0x07200 + ((_i) * 4))
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))
366 #define IXGBE_FHFT_EXT(_n) (0x09800 + (_n * 0x100)) /* Ext Flexible Host 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 0xFF /* 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 0x00F0000 /* 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_RTPBMR 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_LSECRXUNSA 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_PNTHRS_MASK 0xFFFFF00
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 0xFFFFF33

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_TXLQ(_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 */
566 #define IXGBE_RTTBCNRM 0x04980
567 #define IXGBE_RTTBCNRS 0x04988
568 #define IXGBE_RTTBCNCR 0x08B00
569 #define IXGBE_RTTBCNACH 0x08B04
570 #define IXGBE_RTTBCNACL 0x08B08
571 #define IXGBE_RTTBCNTG 0x04A90
572 #define IXGBE_RTTBCNDX 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 */
583 #define IXGBE_FCINVST0 0x03FC0 /* FC Invalid DMA Context Status Reg 0 */
584 #define IXGBE_FCINVST(_i) (IXGBE_FCINVST0 + ((_i) * 4))
585 #define IXGBE_FCBUFF_VALID (1 << 0) /* DMA Context Valid */
586 #define IXGBE_FCBUFF_BUFSIZE (3 << 3) /* User Buffer Size */
587 #define IXGBE_FCBUFF_WRCNTX (1 << 7) /* 0: Initiator, 1: Target */
588 #define IXGBE_FCBUFF_BUFFCNT 0x0000fff0 /* Number of User Buffers */
589 #define IXGBE_FCBUFF_OFFSET 0xffff0000 /* User Buffer Offset */
590 #define IXGBE_FCBUFF_BUFSIZE_SHIFT 3
591 #define IXGBE_FCBUFF_BUFFCNT_SHIFT 8
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 S0F/E0F */
598 #define IXGBE_TE0FF 0x04A94 /* Tx FC E0F */
599 #define IXGBE_TS0FF 0x04A98 /* Tx FC S0F */
600 #define IXGBE_RE0FF 0x05158 /* Rx FC E0F */
601 #define IXGBE_RS0FF 0x051F8 /* Rx FC S0F */
602 /* FCoE Filter Context Registers */
603 #define IXGBE_FCFILT 0x05108 /* FC FLT Context */
604 #define IXGBE_FCFLTRW 0x05110 /* FC Filter RW Control */
605 #define IXGBE_FCPARAM 0x05108 /* FC Offset Parameter */
606 #define IXGBE_FCFILT_VALID (1 << 0) /* Filter Context Valid */
607 #define IXGBE_FCFILT_FIRST (1 << 1) /* Filter First */
608 #define IXGBE_FCFILT_SEQID 0x00ff0000 /* Sequence ID */
609 #define IXGBE_FCFILT_SEQCNT 0xff000000 /* Sequence Count */
610 #define IXGBE_FCFLTRW_RVALID (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 0x00000fff /* 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 */
708 #define IXGBE_FCCRC 0x05118 /* Count of Good Eth CRC w/ Bad FC CRC */
709 #define IXGBE_FCOERPDC 0x0241C /* FCoE Rx Packets Dropped Count */
710 #define IXGBE_FCLAST 0x02424 /* FCoE Last Error Count */
711 #define IXGBE_FCOEPRC 0x02428 /* Number of FCoE Packets Received */
712 #define IXGBE_FCOEDWRC 0x0242C /* Number of FCoE DWords Received */
713 #define IXGBE_FCOEPTC 0x08784 /* Number of FCoE Packets Transmitted */
714 #define IXGBE_FCOEDWTC 0x08788 /* Number of FCoE DWords Transmitted */
715 #define IXGBE_FCCRC_CNT_MASK 0x0000FFFF /* CRC_CNT: bit 0 - 15 */
716 #define IXGBE_FCLAST_CNT_MASK 0x0000FFFF /* Last_CNT: bit 0 - 15 */
717 #define IXGBE_O2BSPC 0x041C4
718 #define IXGBE_B2OSPC 0x087B0
719 #define IXGBE_B2OAPC 0x041C0
720 #define IXGBE_B2OAPC 0x02F90
721 #define IXGBE_BUPRC 0x04180
722 #define IXGBE_BMPRC 0x04184
723 #define IXGBE_BPPRC 0x04188
724 #define IXGBE_BUPTC 0x0418C
725 #define IXGBE_BMPTC 0x04190
726 #define IXGBE_BBPCTC 0x04194
727 #define IXGBE_BRCERRS 0x04198
728 #define IXGBE_BXONRXC 0x0419C
729 #define IXGBE_BXOFFRXC 0x041E0
730 #define IXGBE_BXONTXC 0x041E4
731 #define IXGBE_PCRC8ECL 0x0E810
732 #define IXGBE_PCRC8ECH 0x0E811
733 #define IXGBE_PCRC8ECH_MASK 0x1F
734 #define IXGBE_LDPCACL 0x0E820
735 #define IXGBE_LDPCACL 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_CIAID_82599        0x1108C
816 #define IXGBE_PICAUSE            0x110B0
817 #define IXGBE_PIANA             0x110B8
818 #define IXGBE_INTRPT_CSR_82599  0x110B0
819 #define IXGBE_INTRPT_MASK_82599 0x110B8
820 #define IXGBE_CDQ_MBR_82599      0x110B4
821 #define IXGBE_PCIESPARE          0x110BC
822 #define IXGBE_MISC_REG_82599     0x110F0
823 #define IXGBE_ECC_CTRL_0_82599  0x11100
824 #define IXGBE_ECC_CTRL_1_82599  0x11104
825 #define IXGBE_ECC_STATUS_82599  0x110E0
826 #define IXGBE_BAR_CTRL_82599     0x110F4

```

```

826 /* PCI Express Control */
827 #define IXGBE_GCR_CMPL_TMOU_MASK 0x0000F000
828 #define IXGBE_GCR_CMPL_TMOU_10ms 0x00001000
829 #define IXGBE_GCR_CMPL_TMOU_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_RXSTMP_L          0x051E8 /* Rx timestamp Low - RO */
844 #define IXGBE_RXSTMP_H          0x051A4 /* Rx timestamp High - RO */
845 #define IXGBE_RXSAT_L           0x051A0 /* Rx timestamp attribute low - RO */
846 #define IXGBE_RXSAT_H           0x051A8 /* Rx timestamp attribute high - RO */
847 #define IXGBE_RXMTRL            0x05120 /* RX message type register low - RW */
848 #define IXGBE_TXSTMP_L          0x08C04 /* Tx timestamp value Low - RO */
849 #define IXGBE_TXSTMP_H          0x08C08 /* Tx timestamp value High - RO */
850 #define IXGBE_SYSTIM_L          0x08C0C /* System time register Low - RO */
851 #define IXGBE_SYSTIM_H          0x08C10 /* System time register High - RO */
852 #define IXGBE_TIMINCA           0x08C14 /* Increment attributes register - RW */
853 #define IXGBE_TIMADJ_L          0x08C18 /* Time Adjustment Offset register Low - RW */
854 #define IXGBE_TIMADJ_H          0x08C1C /* Time Adjustment Offset register High - RW */
855 #define IXGBE_TSAUXC            0x08C20 /* TimeSync Auxiliary Control register - RW */
856 #define IXGBE_TRGTTIM_L0        0x08C24 /* Target Time Register 0 Low - RW */
857 #define IXGBE_TRGTTIM_H0        0x08C28 /* Target Time Register 0 High - RW */
858 #define IXGBE_TRGTTIM_L1        0x08C2C /* Target Time Register 1 Low - RW */
859 #define IXGBE_TRGTTIM_H1        0x08C30 /* Target Time Register 1 High - RW */
860 #define IXGBE_CLKTIM_L          0x08C34 /* Clock Out Time Register Low - RW */
861 #define IXGBE_CLKTIM_H          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_AUXSTMP_L0        0x08C3C /* Auxiliary Time Stamp 0 register Low - RO */
865 #define IXGBE_AUXSTMP_H0        0x08C40 /* Auxiliary Time Stamp 0 register High - RO */
866 #define IXGBE_AUXSTMP_L1        0x08C44 /* Auxiliary Time Stamp 1 register Low - RO */
867 #define IXGBE_AUXSTMP_H1        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         0x3600
891 #define IXGBE_RXBUFDATA0        0x3610
892 #define IXGBE_RXBUFDATA1        0x3614
893 #define IXGBE_RXBUFDATA2        0x3618
894 #define IXGBE_RXBUFDATA3        0x361C
895 #define IXGBE_PCIE_DIAG(_i)     (0x11090 + ((_i) * 4)) /* 8 of these */
896 #define IXGBE_RFVAL             0x50A4

```

```

897 #define IXGBE_MDFTC1      0x042B8
898 #define IXGBE_MDFTC2      0x042C0
899 #define IXGBE_MDFTFIF01    0x042C4
900 #define IXGBE_MDFTFIF02    0x042C8
901 #define IXGBE_MDFTS        0x042CC
902 #define IXGBE_RXDATAWRPTR(_i) (0x03700 + ((_i) * 4)) /* 8 of these 3700-370C*/
903 #define IXGBE_RXDESCWRPTR(_i) (0x03710 + ((_i) * 4)) /* 8 of these 3710-371C*/
904 #define IXGBE_RXDATAARDPTR(_i) (0x03720 + ((_i) * 4)) /* 8 of these 3720-372C*/
905 #define IXGBE_RXDESCRDPTR(_i) (0x03730 + ((_i) * 4)) /* 8 of these 3730-373C*/
906 #define IXGBE_TXDATAWRPTR(_i) (0x0C700 + ((_i) * 4)) /* 8 of these C700-C70C*/
907 #define IXGBE_TXDESCWRPTR(_i) (0x0C710 + ((_i) * 4)) /* 8 of these C710-C71C*/
908 #define IXGBE_TXDATAARDPTR(_i) (0x0C720 + ((_i) * 4)) /* 8 of these C720-C72C*/
909 #define IXGBE_TXDESCRDPTR(_i) (0x0C730 + ((_i) * 4)) /* 8 of these C730-C73C*/
910 #define IXGBE_PCIEECCTL      0x1106C
911 #define IXGBE_RXWRPTR(_i) (0x03100 + ((_i) * 4)) /* 8 of these 3100-310C*/
912 #define IXGBE_RXUSED(_i) (0x03120 + ((_i) * 4)) /* 8 of these 3120-312C*/
913 #define IXGBE_RXRDPTR(_i) (0x03140 + ((_i) * 4)) /* 8 of these 3140-314C*/
914 #define IXGBE_RXRDWRPTR(_i) (0x03160 + ((_i) * 4)) /* 8 of these 3160-310C*/
915 #define IXGBE_TXWRPTR(_i) (0x0C100 + ((_i) * 4)) /* 8 of these C100-C10C*/
916 #define IXGBE_TXUSED(_i) (0x0C120 + ((_i) * 4)) /* 8 of these C120-C12C*/
917 #define IXGBE_TXRDPTR(_i) (0x0C140 + ((_i) * 4)) /* 8 of these C140-C14C*/
918 #define IXGBE_TXRDWRPTR(_i) (0x0C160 + ((_i) * 4)) /* 8 of these C160-C10C*/
919 #define IXGBE_PCIEECCTL0      0x11100
920 #define IXGBE_PCIEECCTL1      0x11104
921 #define IXGBE_RXDBUECC        0x03F70
922 #define IXGBE_TXDBUECC        0x0CF70
923 #define IXGBE_RXDBUEST        0x03F74
924 #define IXGBE_TXDBUEST        0x0CF74
925 #define IXGBE_PBTXECC         0x0C300
926 #define IXGBE_PBRXECC         0x03300
927 #define IXGBE_GHECCR          0x110B0

929 /* MAC Registers */
930 #define IXGBE_PCS1GCFIG        0x04200
931 #define IXGBE_PCS1GLCTL        0x04208
932 #define IXGBE_PCS1GLSTA        0x0420C
933 #define IXGBE_PCS1GDBG0        0x04210
934 #define IXGBE_PCS1GDBG1        0x04214
935 #define IXGBE_PCS1GANA        0x04218
936 #define IXGBE_PCS1GANLP        0x0421C
937 #define IXGBE_PCS1GANNP        0x04220
938 #define IXGBE_PCS1GANLNP        0x04224
939 #define IXGBE_HLREG0           0x04240
940 #define IXGBE_HLREG1           0x04244
941 #define IXGBE_PAP              0x04248
942 #define IXGBE_MACA             0x0424C
943 #define IXGBE_APAE             0x04250
944 #define IXGBE_ARD              0x04254
945 #define IXGBE_AIS              0x04258
946 #define IXGBE_MSCA            0x0425C
947 #define IXGBE_MSRWD            0x04260
948 #define IXGBE_MLADD            0x04264
949 #define IXGBE_MHADD            0x04268
950 #define IXGBE_MAXFRS           0x04268
951 #define IXGBE_TREG             0x0426C
952 #define IXGBE_PCSS1            0x04288
953 #define IXGBE_PCSS2            0x0428C
954 #define IXGBE_XPCSS           0x04290
955 #define IXGBE_MFLCN            0x04294
956 #define IXGBE_SERDESC          0x04298
957 #define IXGBE_MACS             0x0429C
958 #define IXGBE_AUTOC            0x042A0
959 #define IXGBE_LINKS            0x042A4
960 #define IXGBE_LINKS2           0x04324
961 #define IXGBE_AUTOC2           0x042A8
962 #define IXGBE_AUTOC3           0x042AC

```

```

963 #define IXGBE_ANLP1          0x042B0
964 #define IXGBE_ANLP2          0x042B4
965 #define IXGBE_MACC           0x04330
966 #define IXGBE_ATLASCTL       0x04800
967 #define IXGBE_MMNGC          0x042D0
968 #define IXGBE_ANLPNP1        0x042D4
969 #define IXGBE_ANLPNP2        0x042D8
970 #define IXGBE_KRPCSFC        0x042E0
971 #define IXGBE_KRPCSS         0x042E4
972 #define IXGBE_FECs1          0x042E8
973 #define IXGBE_FECs2          0x042EC
974 #define IXGBE_SMADARCTL      0x14F10
975 #define IXGBE_MPVC           0x04318
976 #define IXGBE_SGMIIC         0x04314

978 /* Statistics Registers */
979 #define IXGBE_RXNFGPC         0x041B0
980 #define IXGBE_RXNFGBCL       0x041B4
981 #define IXGBE_RXNFGBCH       0x041B8
982 #define IXGBE_RXDGPC         0x02F50
983 #define IXGBE_RXDGBCL        0x02F54
984 #define IXGBE_RXDGBCH        0x02F58
985 #define IXGBE_RXDDGPC        0x02F5C
986 #define IXGBE_RXDDGBCL       0x02F60
987 #define IXGBE_RXDDGBCH       0x02F64
988 #define IXGBE_RXLPBKGPC      0x02F68
989 #define IXGBE_RXLPBKGBCL     0x02F6C
990 #define IXGBE_RXLPBKGBCH     0x02F70
991 #define IXGBE_RXDLPBKGPC      0x02F74
992 #define IXGBE_RXDLPBKGBCL     0x02F78
993 #define IXGBE_RXDLPBKGBCH     0x02F7C
994 #define IXGBE_TXDGPC         0x087A0
995 #define IXGBE_TXDGBCL        0x087A4
996 #define IXGBE_TXDGBCH        0x087A8

998 #define IXGBE_RXDSTATCTRL     0x02F40

1000 /* Copper Pond 2 link timeout */
1001 #define IXGBE_VALIDATE_LINK_READY_TIMEOUT 50

1003 /* Omer CORECTL */
1004 #define IXGBE_CORECTL         0x014F00
1005 /* BARCTRL */
1006 #define IXGBE_BARCTRL         0x110F4
1007 #define IXGBE_BARCTRL_FLSIZE  0x0700
1008 #define IXGBE_BARCTRL_FLSIZE_SHIFT 8
1009 #define IXGBE_BARCTRL_CSRSIZE 0x2000

1011 /* RSCCTL Bit Masks */
1012 #define IXGBE_RSCCTL_RSCEN     0x01
1013 #define IXGBE_RSCCTL_MAXDESC_1 0x00
1014 #define IXGBE_RSCCTL_MAXDESC_4 0x04
1015 #define IXGBE_RSCCTL_MAXDESC_8 0x08
1016 #define IXGBE_RSCCTL_MAXDESC_16 0x0C

1018 /* RSCDBU Bit Masks */
1019 #define IXGBE_RSCDBU_RSCSMALDIS_MASK 0x0000007F
1020 #define IXGBE_RSCDBU_RSCACKDIS     0x00000080

1022 /* RDRXCTL Bit Masks */
1023 #define IXGBE_RDRXCTL_RDMTS_1_2 0x00000000 /* Rx Desc Min THLD Size */
1024 #define IXGBE_RDRXCTL_RDMTS_1_2 0x00000000 /* Rx Desc Min Threshold Size */
1024 #define IXGBE_RDRXCTL_CRCSTRIP 0x00000002 /* CRC Strip */
1025 #define IXGBE_RDRXCTL_MVMEN     0x00000020
1026 #define IXGBE_RDRXCTL_DMAIDONE  0x00000008 /* DMA init cycle done */
1027 #define IXGBE_RDRXCTL_AGGDIS    0x00010000 /* Aggregation disable */

```

```

1028 #define IXGBE_RDRXCTL_RSCFRSTSIZE 0x003E0000 /* RSC First packet size */
1029 #define IXGBE_RDRXCTL_RSCLLIDIS 0x00800000 /* Disabl RSC compl on LLI */
1030 #define IXGBE_RDRXCTL_RSCACKC 0x02000000 /* must set 1 when RSC ena */
1031 #define IXGBE_RDRXCTL_FCOE_WRFIX 0x04000000 /* must set 1 when RSC ena */
1032 #define IXGBE_RDRXCTL_RSCLLIDIS 0x00800000 /* Disable RSC compl on LLI */
1033 #define IXGBE_RDRXCTL_RSCACKC 0x02000000 /* must set 1 when RSC enabled */
1034 #define IXGBE_RDRXCTL_FCOE_WRFIX 0x04000000 /* must set 1 when RSC enabled */

1033 /* RQTC Bit Masks and Shifts */
1034 #define IXGBE_RQTC_SHIFT_TC(i) ((i) * 4)
1035 #define IXGBE_RQTC_TC0_MASK (0x7 << 0)
1036 #define IXGBE_RQTC_TC1_MASK (0x7 << 4)
1037 #define IXGBE_RQTC_TC2_MASK (0x7 << 8)
1038 #define IXGBE_RQTC_TC3_MASK (0x7 << 12)
1039 #define IXGBE_RQTC_TC4_MASK (0x7 << 16)
1040 #define IXGBE_RQTC_TC5_MASK (0x7 << 20)
1041 #define IXGBE_RQTC_TC6_MASK (0x7 << 24)
1042 #define IXGBE_RQTC_TC7_MASK (0x7 << 28)

1044 /* PSRTYPE.RQPL Bit masks and shift */
1045 #define IXGBE_PSRTYPE_RQPL_MASK 0x7
1046 #define IXGBE_PSRTYPE_RQPL_SHIFT 29

1048 /* CTRL Bit Masks */
1049 #define IXGBE_CTRL_GIO_DIS 0x00000004 /* Global IO Master Disable bit */
1050 #define IXGBE_CTRL_LNK_RST 0x00000008 /* Link Reset. Resets everything. */
1051 #define IXGBE_CTRL_RST 0x04000000 /* Reset (SW) */
1052 #define IXGBE_CTRL_RST_MASK (IXGBE_CTRL_LNK_RST | IXGBE_CTRL_RST)

1054 /* FACTPS */
1055 #define IXGBE_FACTPS_LFS 0x40000000 /* LAN Function Select */

1057 /* MHADD Bit Masks */
1058 #define IXGBE_MHADD_MFS_MASK 0xFFFF0000
1059 #define IXGBE_MHADD_MFS_SHIFT 16

1061 /* Extended Device Control */
1062 #define IXGBE_CTRL_EXT_PFRSTD 0x00004000 /* Physical Function Reset Done */
1063 #define IXGBE_CTRL_EXT_NS_DIS 0x00010000 /* No Snoop disable */
1064 #define IXGBE_CTRL_EXT_RO_DIS 0x00020000 /* Relaxed Ordering disable */
1065 #define IXGBE_CTRL_EXT_DRV_LOAD 0x10000000 /* Driver loaded bit for FW */

1067 /* Direct Cache Access (DCA) definitions */
1068 #define IXGBE_DCA_CTRL_DCA_ENABLE 0x00000000 /* DCA Enable */
1069 #define IXGBE_DCA_CTRL_DCA_DISABLE 0x00000001 /* DCA Disable */

1071 #define IXGBE_DCA_CTRL_DCA_MODE_CB1 0x00 /* DCA Mode CB1 */
1072 #define IXGBE_DCA_CTRL_DCA_MODE_CB2 0x02 /* DCA Mode CB2 */

1074 #define IXGBE_DCA_RXCTRL_CPUID_MASK 0x0000001F /* Rx CPUID Mask */
1075 #define IXGBE_DCA_RXCTRL_CPUID_MASK_82599 0xFF000000 /* Rx CPUID Mask */
1076 #define IXGBE_DCA_RXCTRL_CPUID_SHIFT_82599 24 /* Rx CPUID Shift */
1077 #define IXGBE_DCA_RXCTRL_DESC_DCA_EN (1 << 5) /* Rx Desc enable */
1078 #define IXGBE_DCA_RXCTRL_HEAD_DCA_EN (1 << 6) /* Rx Desc header ena */
1079 #define IXGBE_DCA_RXCTRL_DATA_DCA_EN (1 << 7) /* Rx Desc payload ena */
1080 #define IXGBE_DCA_RXCTRL_DESC_RRO_EN (1 << 9) /* Rx rd Desc Relax Order */
1081 #define IXGBE_DCA_RXCTRL_DATA_RRO_EN (1 << 13) /* Rx wr data Relax Order */
1082 #define IXGBE_DCA_RXCTRL_HEAD_WRO_EN (1 << 15) /* Rx wr header R0 */
1083 #define IXGBE_DCA_RXCTRL_DESC_DCA_EN (1 << 5) /* DCA Rx Desc enable */
1084 #define IXGBE_DCA_RXCTRL_HEAD_DCA_EN (1 << 6) /* DCA Rx Desc header enable */
1085 #define IXGBE_DCA_RXCTRL_DATA_DCA_EN (1 << 7) /* DCA Rx Desc payload enable */
1086 #define IXGBE_DCA_RXCTRL_DESC_RRO_EN (1 << 9) /* DCA Rx rd Desc Relax Order */
1087 #define IXGBE_DCA_RXCTRL_DESC_WRO_EN (1 << 13) /* DCA Rx wr Desc Relax Order */
1088 #define IXGBE_DCA_RXCTRL_DESC_HSR0_EN (1 << 15) /* DCA Rx Split Header R0 */

1084 #define IXGBE_DCA_TXCTRL_CPUID_MASK 0x0000001F /* Tx CPUID Mask */

```

```

1085 #define IXGBE_DCA_TXCTRL_CPUID_MASK_82599 0xFF000000 /* Tx CPUID Mask */
1086 #define IXGBE_DCA_TXCTRL_CPUID_SHIFT_82599 24 /* Tx CPUID Shift */
1087 #define IXGBE_DCA_TXCTRL_DESC_DCA_EN (1 << 5) /* DCA Tx Desc enable */
1088 #define IXGBE_DCA_TXCTRL_DESC_RRO_EN (1 << 9) /* Tx rd Desc Relax Order */
1089 #define IXGBE_DCA_TXCTRL_DESC_WRO_EN (1 << 11) /* Tx Desc writeback R0 bit */
1090 #define IXGBE_DCA_TXCTRL_DATA_RRO_EN (1 << 13) /* Tx rd data Relax Order */
1091 #define IXGBE_DCA_TXCTRL_TX_WB_R0_EN (1 << 11) /* Tx Desc writeback R0 bit */
1092 #define IXGBE_DCA_MAX_QUEUES_82598 16 /* DCA regs only on 16 queues */

1093 /* MSA Bit Masks */
1094 #define IXGBE_MSCA_NP_ADDR_MASK 0x0000FFFF /* MDI Addr (new prot) */
1095 #define IXGBE_MSCA_NP_ADDR_MASK 0x0000FFFF /* MDI Address (new protocol) */
1096 #define IXGBE_MSCA_NP_ADDR_SHIFT 0
1097 #define IXGBE_MSCA_DEV_TYPE_MASK 0x001F0000 /* Dev Type (new prot) */
1098 #define IXGBE_MSCA_DEV_TYPE_SHIFT 16 /* Register Address (old prot) */
1099 #define IXGBE_MSCA_DEV_TYPE_MASK 0x001F0000 /* Device Type (new protocol) */
1100 #define IXGBE_MSCA_DEV_TYPE_SHIFT 16 /* Register Address (old protocol) */
1101 #define IXGBE_MSCA_PHY_ADDR_MASK 0x03E00000 /* PHY Address mask */
1102 #define IXGBE_MSCA_PHY_ADDR_SHIFT 21 /* PHY Address shift */
1103 #define IXGBE_MSCA_OP_CODE_MASK 0x0C000000 /* OP CODE mask */
1104 #define IXGBE_MSCA_OP_CODE_SHIFT 26 /* OP CODE shift */
1105 #define IXGBE_MSCA_ADDR_CYCLE 0x00000000 /* OP CODE 00 (addr cycle) */
1106 #define IXGBE_MSCA_WRITE 0x04000000 /* OP CODE 01 (wr) */
1107 #define IXGBE_MSCA_READ 0x0C000000 /* OP CODE 11 (rd) */
1108 #define IXGBE_MSCA_READ_AUTOINC 0x08000000 /* OP CODE 10 (rd auto inc) */
1109 #define IXGBE_MSCA_WRITE 0x04000000 /* OP CODE 01 (write) */
1110 #define IXGBE_MSCA_READ 0x0C000000 /* OP CODE 11 (read) */
1111 #define IXGBE_MSCA_READ_AUTOINC 0x08000000 /* OP CODE 10 (read, auto inc) */
1112 #define IXGBE_MSCA_ST_CODE_MASK 0x30000000 /* ST Code mask */
1113 #define IXGBE_MSCA_ST_CODE_SHIFT 28 /* ST Code shift */
1114 #define IXGBE_MSCA_NEW_PROTOCOL 0x00000000 /* ST CODE 00 (new prot) */
1115 #define IXGBE_MSCA_OLD_PROTOCOL 0x10000000 /* ST CODE 01 (old prot) */
1116 #define IXGBE_MSCA_NEW_PROTOCOL 0x00000000 /* ST CODE 00 (new protocol) */
1117 #define IXGBE_MSCA_OLD_PROTOCOL 0x10000000 /* ST CODE 01 (old protocol) */
1118 #define IXGBE_MSCA_MDI_COMMAND 0x40000000 /* Initiate MDI command */
1119 #define IXGBE_MSCA_MDI_IN_PROG_EN 0x80000000 /* MDI in progress ena */
1120 #define IXGBE_MSCA_MDI_IN_PROG_EN 0x80000000 /* MDI in progress enable */

1113 /* MSRWD bit masks */
1114 #define IXGBE_MSRWD_WRITE_DATA_MASK 0x0000FFFF
1115 #define IXGBE_MSRWD_WRITE_DATA_SHIFT 0
1116 #define IXGBE_MSRWD_READ_DATA_MASK 0xFFFF0000
1117 #define IXGBE_MSRWD_READ_DATA_SHIFT 16

1119 /* Atlas registers */
1120 #define IXGBE_ATLAS_PDN_LPBK 0x24
1121 #define IXGBE_ATLAS_PDN_10G 0xB
1122 #define IXGBE_ATLAS_PDN_1G 0xC
1123 #define IXGBE_ATLAS_PDN_AN 0xD

1125 /* Atlas bit masks */
1126 #define IXGBE_ATLASCTL_WRITE_CMD 0x00010000
1127 #define IXGBE_ATLAS_PDN_TX_REG_EN 0x10
1128 #define IXGBE_ATLAS_PDN_TX_10G_QL_ALL 0xF0
1129 #define IXGBE_ATLAS_PDN_TX_1G_QL_ALL 0xF0
1130 #define IXGBE_ATLAS_PDN_TX_AN_QL_ALL 0xF0

1132 /* Omer bit masks */
1133 #define IXGBE_CORECTL_WRITE_CMD 0x00010000

1135 /* Device Type definitions for new protocol MDIO commands */
1136 #define IXGBE_MDIO_PMA_PMD_DEV_TYPE 0x1
1137 #define IXGBE_MDIO_PCS_DEV_TYPE 0x3
1138 #define IXGBE_MDIO_PHY_XS_DEV_TYPE 0x4
1139 #define IXGBE_MDIO_AUTO_NEG_DEV_TYPE 0x7
1140 #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_ADVDT 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_100GBASET_ABILITY 0x0020 /* 100GBaseT capable */
1167 #define IXGBE_MDIO_PHY_100GBASET_TX_ABILITY 0x0080 /* 100GBaseTX 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 0xFFFFFFFF
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     PBA_STRATEGY_EQUAL = PBA_STRATEGY_EQUAL
1252     PBA_STRATEGY_WEIGHTED = 1, /* Weight front half of TCs */
1253     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

```

new/usr/src/uts/common/io/ixgbe/ixgbe_type.h

```

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_PRI_PAUSEEN 0x01000000 /* bit 24 */
1287 #define IXGBE_HLREG0_RXPAUSERECDA 0x06000000 /* bits 25-26 */
1288 #define IXGBE_HLREG0_RXLNGBTHERREN 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_RDICDREQ 0x00800000
1317 #define IXGBE_RDHMPN_RDICADDR_SHIFT 11
1318 #define IXGBE_TDHMPN_TDICADDR 0x003FF800
1319 #define IXGBE_TDHMPN_TDICDREQ 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

```

21

new/usr/src/uts/common/io/ixgbe/ixgbe_type.h

```

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_FCRTL_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 */
1370 #define IXGBE_RMCS_RRM 0x00000002 /* Receive Recycle Mode enable */
1371 /* 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
1373 #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 */

```

22

```

1400 #define IXGBE_EICR_DHER      0x20000000 /* Descriptor Handler Error */
1401 #define IXGBE_EICR_TCP_TIMER  0x40000000 /* TCP Timer */
1402 #define IXGBE_EICR_OTHER      0x80000000 /* Interrupt Cause Active */

1404 /* Extended Interrupt Cause Set */
1405 #define IXGBE_EICS_RTX_QUEUE IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
1406 #define IXGBE_EICS_FLOW_DIR  IXGBE_EICR_FLOW_DIR /* FDir Exception */
1407 #define IXGBE_EICS_RX_MISS   IXGBE_EICR_RX_MISS /* Pkt Buffer Overrun */
1408 #define IXGBE_EICS_PCI       IXGBE_EICR_PCI /* PCI Exception */
1409 #define IXGBE_EICS_MAILBOX   IXGBE_EICR_MAILBOX /* VF to PF Mailbox Int */
1410 #define IXGBE_EICS_LSC      IXGBE_EICR_LSC /* Link Status Change */
1411 #define IXGBE_EICS_MNG      IXGBE_EICR_MNG /* MNG Event Interrupt */
1412 #define IXGBE_EICS_TIMESYNC IXGBE_EICR_TIMESYNC /* Timesync Event */
1413 #define IXGBE_EICS_GPI_SDP0 IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
1414 #define IXGBE_EICS_GPI_SDP1 IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
1415 #define IXGBE_EICS_GPI_SDP2 IXGBE_EICR_GPI_SDP2 /* SDP2 Gen Purpose Int */
1416 #define IXGBE_EICS_ECC      IXGBE_EICR_ECC /* ECC Error */
1417 #define IXGBE_EICS_PBUR     IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
1418 #define IXGBE_EICS_DHER     IXGBE_EICR_DHER /* Desc Handler Error */
1419 #define IXGBE_EICS_TCP_TIMER IXGBE_EICR_TCP_TIMER /* TCP Timer */
1420 #define IXGBE_EICS_OTHER    IXGBE_EICR_OTHER /* INT Cause Active */

1422 /* Extended Interrupt Mask Set */
1423 #define IXGBE_EIMS_RTX_QUEUE IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
1424 #define IXGBE_EIMS_FLOW_DIR  IXGBE_EICR_FLOW_DIR /* FDir Exception */
1425 #define IXGBE_EIMS_RX_MISS   IXGBE_EICR_RX_MISS /* Packet Buffer Overrun */
1426 #define IXGBE_EIMS_PCI       IXGBE_EICR_PCI /* PCI Exception */
1427 #define IXGBE_EIMS_MAILBOX   IXGBE_EICR_MAILBOX /* VF to PF Mailbox Int */
1428 #define IXGBE_EIMS_LSC      IXGBE_EICR_LSC /* Link Status Change */
1429 #define IXGBE_EIMS_MNG      IXGBE_EICR_MNG /* MNG Event Interrupt */
1430 #define IXGBE_EIMS_TS        IXGBE_EICR_TS /* Thermal Sensor Event */
1431 #define IXGBE_EIMS_TIMESYNC IXGBE_EICR_TIMESYNC /* Timesync Event */
1432 #define IXGBE_EIMS_GPI_SDP0 IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
1433 #define IXGBE_EIMS_GPI_SDP1 IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
1434 #define IXGBE_EIMS_GPI_SDP2 IXGBE_EICR_GPI_SDP2 /* SDP2 Gen Purpose Int */
1435 #define IXGBE_EIMS_ECC      IXGBE_EICR_ECC /* ECC Error */
1436 #define IXGBE_EIMS_PBUR     IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
1437 #define IXGBE_EIMS_DHER     IXGBE_EICR_DHER /* Descr Handler Error */
1438 #define IXGBE_EIMS_TCP_TIMER IXGBE_EICR_TCP_TIMER /* TCP Timer */
1439 #define IXGBE_EIMS_OTHER    IXGBE_EICR_OTHER /* INT Cause Active */

1441 /* Extended Interrupt Mask Clear */
1442 #define IXGBE_EIMC_RTX_QUEUE IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
1443 #define IXGBE_EIMC_FLOW_DIR  IXGBE_EICR_FLOW_DIR /* FDir Exception */
1444 #define IXGBE_EIMC_RX_MISS   IXGBE_EICR_RX_MISS /* Packet Buffer Overrun */
1445 #define IXGBE_EIMC_PCI       IXGBE_EICR_PCI /* PCI Exception */
1446 #define IXGBE_EIMC_MAILBOX   IXGBE_EICR_MAILBOX /* VF to PF Mailbox Int */
1447 #define IXGBE_EIMC_LSC      IXGBE_EICR_LSC /* Link Status Change */
1448 #define IXGBE_EIMC_MNG      IXGBE_EICR_MNG /* MNG Event Interrupt */
1449 #define IXGBE_EIMC_TIMESYNC IXGBE_EICR_TIMESYNC /* Timesync Event */
1450 #define IXGBE_EIMC_GPI_SDP0 IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
1451 #define IXGBE_EIMC_GPI_SDP1 IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
1452 #define IXGBE_EIMC_GPI_SDP2 IXGBE_EICR_GPI_SDP2 /* SDP2 Gen Purpose Int */
1453 #define IXGBE_EIMC_ECC      IXGBE_EICR_ECC /* ECC Error */
1454 #define IXGBE_EIMC_PBUR     IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
1455 #define IXGBE_EIMC_DHER     IXGBE_EICR_DHER /* Descr Handler Err */
1456 #define IXGBE_EIMC_TCP_TIMER IXGBE_EICR_TCP_TIMER /* TCP Timer */
1457 #define IXGBE_EIMC_OTHER    IXGBE_EICR_OTHER /* INT Cause Active */

1459 #define IXGBE_EIMS_ENABLE_MASK ( \
1460     IXGBE_EIMS_RTX_QUEUE | \
1461     IXGBE_EIMS_LSC | \
1462     IXGBE_EIMS_TCP_TIMER | \
1463     IXGBE_EIMS_OTHER)

1465 /* Immediate Interrupt Rx (A.K.A. Low Latency Interrupt) */

```

```

1466 #define IXGBE_IMIR_PORT_IM_EN 0x00010000 /* TCP port enable */
1467 #define IXGBE_IMIR_PORT_BP     0x00020000 /* TCP port check bypass */
1468 #define IXGBE_IMIREXT_SIZE_BP  0x00001000 /* Packet size bypass */
1469 #define IXGBE_IMIREXT_CTRL_URG 0x00002000 /* Check URG bit in header */
1470 #define IXGBE_IMIREXT_CTRL_ACK 0x00004000 /* Check ACK bit in header */
1471 #define IXGBE_IMIREXT_CTRL_PSH 0x00008000 /* Check PSH bit in header */
1472 #define IXGBE_IMIREXT_CTRL_RST 0x00010000 /* Check RST bit in header */
1473 #define IXGBE_IMIREXT_CTRL_SYN 0x00020000 /* Check SYN bit in header */
1474 #define IXGBE_IMIREXT_CTRL_FIN 0x00040000 /* Check FIN bit in header */
1475 #define IXGBE_IMIREXT_CTRL_BP  0x00080000 /* Bypass check of control bits */
1476 #define IXGBE_IMIR_SIZE_BP     0x00001000 /* Packet size bypass */
1477 #define IXGBE_IMIR_CTRL_URG_82599 0x00002000 /* Check URG bit in header */
1478 #define IXGBE_IMIR_CTRL_ACK_82599 0x00004000 /* Check ACK bit in header */
1479 #define IXGBE_IMIR_CTRL_PSH_82599 0x00008000 /* Check PSH bit in header */
1480 #define IXGBE_IMIR_CTRL_RST_82599 0x00010000 /* Check RST bit in header */
1481 #define IXGBE_IMIR_CTRL_SYN_82599 0x00020000 /* Check SYN bit in header */
1482 #define IXGBE_IMIR_CTRL_FIN_82599 0x00040000 /* Check FIN bit in header */
1483 #define IXGBE_IMIR_CTRL_BP_82599 0x00080000 /* Bypass chk of ctrl bits */
1484 #define IXGBE_IMIR_CTRL_BP_82599 0x00080000 /* Bypass check of control bits */
1485 #define IXGBE_IMIR_LLI_EN_82599 0x00100000 /* Enables low latency Int */
1486 #define IXGBE_IMIR_RX_QUEUE_MASK_82599 0x0000007F /* Rx Queue Mask */
1487 #define IXGBE_IMIRVP_PRIORITY_MASK 21 /* Rx Queue Shift */
1488 #define IXGBE_IMIRVP_PRIORITY_EN 0x00000007 /* VLAN priority mask */
1489 #define IXGBE_MAX_FTQF_FILTERS 128
1490 #define IXGBE_FTQF_PROTOCOL_MASK 0x00000003
1491 #define IXGBE_FTQF_PROTOCOL_TCP 0x00000000
1492 #define IXGBE_FTQF_PROTOCOL_UDP 0x00000001
1493 #define IXGBE_FTQF_PROTOCOL_SCTP 2
1494 #define IXGBE_FTQF_PRIORITY_MASK 0x00000007
1495 #define IXGBE_FTQF_PRIORITY_SHIFT 2
1496 #define IXGBE_FTQF_POOL_MASK 0x0000003F
1497 #define IXGBE_FTQF_POOL_SHIFT 8
1498 #define IXGBE_FTQF_STUPLE_MASK_MASK 0x0000001F
1499 #define IXGBE_FTQF_STUPLE_MASK_SHIFT 25
1500 #define IXGBE_FTQF_SOURCE_ADDR_MASK 0x1E
1501 #define IXGBE_FTQF_DEST_ADDR_MASK 0x1D
1502 #define IXGBE_FTQF_SOURCE_PORT_MASK 0x1B
1503 #define IXGBE_FTQF_DEST_PORT_MASK 0x17
1504 #define IXGBE_FTQF_PROTOCOL_COMP_MASK 0x0F
1505 #define IXGBE_FTQF_POOL_MASK_EN 0x40000000
1506 #define IXGBE_FTQF_QUEUE_ENABLE 0x80000000

1509 /* Interrupt clear mask */
1510 #define IXGBE_IRQ_CLEAR_MASK 0xFFFFFFFF

1512 /* Interrupt Vector Allocation Registers */
1513 #define IXGBE_IVAR_REG_NUM 25
1514 #define IXGBE_IVAR_REG_NUM_82599 64
1515 #define IXGBE_IVAR_TXRX_ENTRY 96
1516 #define IXGBE_IVAR_RX_ENTRY 64
1517 #define IXGBE_IVAR_RX_QUEUE(_i) (0 + (_i))
1518 #define IXGBE_IVAR_TX_QUEUE(_i) (64 + (_i))
1519 #define IXGBE_IVAR_TX_ENTRY 32

1521 #define IXGBE_IVAR_TCP_TIMER_INDEX 96 /* 0 based index */
1522 #define IXGBE_IVAR_OTHER_CAUSES_INDEX 97 /* 0 based index */

1524 #define IXGBE_MSIX_VECTOR(_i) (0 + (_i))

1526 #define IXGBE_IVAR_ALLOC_VAL 0x80 /* Interrupt Allocation valid */

1528 /* ETYPE Queue Filter/Select Bit Masks */
1529 #define IXGBE_MAX_ETQF_FILTERS 8
1530 #define IXGBE_ETQF_FCOE 0x08000000 /* bit 27 */

```

```

1531 #define IXGBE_ETQF_BCN 0x10000000 /* bit 28 */
1532 #define IXGBE_ETQF_1588 0x40000000 /* bit 30 */
1533 #define IXGBE_ETQF_FILTER_EN 0x80000000 /* bit 31 */
1534 #define IXGBE_ETQF_POOL_ENABLE (1 << 26) /* bit 26 */
1535 #define IXGBE_ETQF_POOL_SHIFT 20

1537 #define IXGBE_ETQS_RX_QUEUE 0x007F0000 /* bits 22:16 */
1538 #define IXGBE_ETQS_RX_QUEUE_SHIFT 16
1539 #define IXGBE_ETQS_LLI 0x20000000 /* bit 29 */
1540 #define IXGBE_ETQS_QUEUE_EN 0x80000000 /* bit 31 */

1542 /*
1543  * ETQF filter list: one static filter per filter consumer. This is
1544  * to avoid filter collisions later. Add new filters
1545  * here!!
1546  *
1547  * Current filters:
1548  *   EAPOL 802.1x (0x888e): Filter 0
1549  *   FCoE (0x8906): Filter 2
1550  *   1588 (0x88f7): Filter 3
1551  *   FIP (0x8914): Filter 4
1552  */
1553 #define IXGBE_ETQF_FILTER_EAPOL 0
1554 #define IXGBE_ETQF_FILTER_FCOE 2
1555 #define IXGBE_ETQF_FILTER_1588 3
1556 #define IXGBE_ETQF_FILTER_FIP 4
1557 /* VLAN Control Bit Masks */
1558 #define IXGBE_VLCTRL_VET 0x0000FFFF /* bits 0-15 */
1559 #define IXGBE_VLCTRL_CFI 0x10000000 /* bit 28 */
1560 #define IXGBE_VLCTRL_CFIEN 0x20000000 /* bit 29 */
1561 #define IXGBE_VLCTRL_VFE 0x40000000 /* bit 30 */
1562 #define IXGBE_VLCTRL_VME 0x80000000 /* bit 31 */

1564 /* VLAN pool filtering masks */
1565 #define IXGBE_VLVF_VIEN 0x80000000 /* filter is valid */
1566 #define IXGBE_VLVF_ENTRIES 64
1567 #define IXGBE_VLVF_VLANID_MASK 0x000000FF
1568 /* Per VF Port VLAN insertion rules */
1569 #define IXGBE_VMVIR_VLANA_DEFAULT 0x40000000 /* Always use default VLAN */
1570 #define IXGBE_VMVIR_VLANA_NEVER 0x80000000 /* Never insert VLAN tag */

1572 #define IXGBE_ETHERNET_IEEE_VLAN_TYPE 0x8100 /* 802.1q protocol */

1574 /* STATUS Bit Masks */
1575 #define IXGBE_STATUS_LAN_ID 0x0000000C /* LAN ID */
1576 #define IXGBE_STATUS_LAN_ID_SHIFT 2 /* LAN ID Shift */
1577 #define IXGBE_STATUS_GIO 0x00080000 /* GIO Master Ena Status */
1436 #define IXGBE_STATUS_GIO 0x00080000 /* GIO Master Enable Status */

1579 #define IXGBE_STATUS_LAN_ID_0 0x00000000 /* LAN ID 0 */
1580 #define IXGBE_STATUS_LAN_ID_1 0x00000004 /* LAN ID 1 */

1582 /* ESDP Bit Masks */
1583 #define IXGBE_ESDP_SDP0 0x00000001 /* SDP0 Data Value */
1584 #define IXGBE_ESDP_SDP1 0x00000002 /* SDP1 Data Value */
1585 #define IXGBE_ESDP_SDP2 0x00000004 /* SDP2 Data Value */
1586 #define IXGBE_ESDP_SDP3 0x00000008 /* SDP3 Data Value */
1587 #define IXGBE_ESDP_SDP4 0x00000010 /* SDP4 Data Value */
1588 #define IXGBE_ESDP_SDP5 0x00000020 /* SDP5 Data Value */
1589 #define IXGBE_ESDP_SDP6 0x00000040 /* SDP6 Data Value */
1590 #define IXGBE_ESDP_SDP7 0x00000080 /* SDP7 Data Value */
1591 #define IXGBE_ESDP_SDP0_DIR 0x00000100 /* SDP0 IO direction */
1592 #define IXGBE_ESDP_SDP1_DIR 0x00000200 /* SDP1 IO direction */
1593 #define IXGBE_ESDP_SDP3_DIR 0x00000800 /* SDP3 IO direction */
1594 #define IXGBE_ESDP_SDP4_DIR 0x00001000 /* SDP4 IO direction */
1449 #define IXGBE_ESDP_SDP4_DIR 0x00000004 /* SDP4 IO direction */

```

```

1595 #define IXGBE_ESDP_SDP5_DIR 0x00002000 /* SDP5 IO direction */
1596 #define IXGBE_ESDP_SDP6_DIR 0x00004000 /* SDP6 IO direction */
1597 #define IXGBE_ESDP_SDP7_DIR 0x00008000 /* SDP7 IO direction */
1598 #define IXGBE_ESDP_SDP0_NATIVE 0x00010000 /* SDP0 IO mode */
1599 #define IXGBE_ESDP_SDP1_NATIVE 0x00020000 /* SDP1 IO mode */

1602 /* LEDCTL Bit Masks */
1603 #define IXGBE_LED_IVRT_BASE 0x00000040
1604 #define IXGBE_LED_BLINK_BASE 0x00000080
1605 #define IXGBE_LED_MODE_MASK_BASE 0x0000000F
1606 #define IXGBE_LED_OFFSET(_base, _i) (_base << (8 * (_i)))
1607 #define IXGBE_LED_MODE_SHIFT(_i) (8 * (_i))
1608 #define IXGBE_LED_IVRT(_i) IXGBE_LED_OFFSET(IXGBE_LED_IVRT_BASE, _i)
1609 #define IXGBE_LED_BLINK(_i) IXGBE_LED_OFFSET(IXGBE_LED_BLINK_BASE, _i)
1610 #define IXGBE_LED_MODE_MASK(_i) IXGBE_LED_OFFSET(IXGBE_LED_MODE_MASK_BASE, _i)

1612 /* LED modes */
1613 #define IXGBE_LED_LINK_UP 0x0
1614 #define IXGBE_LED_LINK_10G 0x1
1615 #define IXGBE_LED_MAC 0x2
1616 #define IXGBE_LED_FILTER 0x3
1617 #define IXGBE_LED_LINK_ACTIVE 0x4
1618 #define IXGBE_LED_LINK_1G 0x5
1619 #define IXGBE_LED_ON 0xE
1620 #define IXGBE_LED_OFF 0xF

1622 /* AUTOC Bit Masks */
1623 #define IXGBE_AUTOC_KX4_KX_SUPP_MASK 0xC0000000
1624 #define IXGBE_AUTOC_KX4_SUPP 0x80000000
1625 #define IXGBE_AUTOC_KX_SUPP 0x40000000
1626 #define IXGBE_AUTOC_PAUSE 0x30000000
1627 #define IXGBE_AUTOC_ASM_PAUSE 0x20000000
1628 #define IXGBE_AUTOC_SYM_PAUSE 0x10000000
1629 #define IXGBE_AUTOC_RF 0x08000000
1630 #define IXGBE_AUTOC_PD_TMR 0x06000000
1631 #define IXGBE_AUTOC_AN_RX_LOOSE 0x01000000
1632 #define IXGBE_AUTOC_AN_RX_DRIFT 0x00800000
1633 #define IXGBE_AUTOC_AN_RX_ALIGN 0x007C0000
1634 #define IXGBE_AUTOC_FECA 0x00040000
1635 #define IXGBE_AUTOC_FECR 0x00020000
1636 #define IXGBE_AUTOC_KR_SUPP 0x00010000
1637 #define IXGBE_AUTOC_AN_RESTART 0x00001000
1638 #define IXGBE_AUTOC_FLU 0x00000001
1639 #define IXGBE_AUTOC_LMS_SHIFT 13
1640 #define IXGBE_AUTOC_LMS_10G_SERIAL (0x3 << IXGBE_AUTOC_LMS_SHIFT)
1641 #define IXGBE_AUTOC_LMS_KX4_KX_KR (0x4 << IXGBE_AUTOC_LMS_SHIFT)
1642 #define IXGBE_AUTOC_LMS_SGMII_1G_100M (0x5 << IXGBE_AUTOC_LMS_SHIFT)
1643 #define IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN (0x6 << IXGBE_AUTOC_LMS_SHIFT)
1644 #define IXGBE_AUTOC_LMS_KX4_KX_KR_SGMII (0x7 << IXGBE_AUTOC_LMS_SHIFT)
1645 #define IXGBE_AUTOC_LMS_MASK (0x7 << IXGBE_AUTOC_LMS_SHIFT)
1646 #define IXGBE_AUTOC_LMS_1G_LINK_NO_AN (0x0 << IXGBE_AUTOC_LMS_SHIFT)
1647 #define IXGBE_AUTOC_LMS_10G_LINK_NO_AN (0x1 << IXGBE_AUTOC_LMS_SHIFT)
1648 #define IXGBE_AUTOC_LMS_1G_AN (0x2 << IXGBE_AUTOC_LMS_SHIFT)
1649 #define IXGBE_AUTOC_LMS_KX4_AN (0x4 << IXGBE_AUTOC_LMS_SHIFT)
1650 #define IXGBE_AUTOC_LMS_KX4_AN_1G_AN (0x6 << IXGBE_AUTOC_LMS_SHIFT)
1651 #define IXGBE_AUTOC_LMS_ATTACH_TYPE (0x7 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)

1653 #define IXGBE_AUTOC_1G_PMA_PMD_MASK 0x00000200
1654 #define IXGBE_AUTOC_1G_PMA_PMD_SHIFT 9
1655 #define IXGBE_AUTOC_10G_PMA_PMD_MASK 0x00000180
1656 #define IXGBE_AUTOC_10G_PMA_PMD_SHIFT 7
1657 #define IXGBE_AUTOC_10G_XAUI (0x0 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
1658 #define IXGBE_AUTOC_10G_KX4 (0x1 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
1659 #define IXGBE_AUTOC_10G_CX4 (0x2 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
1660 #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 0x1000
1708 #define IXGBE_PCS1GLSTA_AN_PAGE_RX 0x2000
1709 #define IXGBE_PCS1GLSTA_AN_TIMED_OUT 0x4000
1710 #define IXGBE_PCS1GLSTA_AN_REMOTE_FAULT 0x8000
1711 #define IXGBE_PCS1GLSTA_AN_ERROR_RWS 0x10000

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 allows 14 bits for addr. */

1768 #define IXGBE_EEC_SIZE_SHIFT 11
1769 #define IXGBE_EEPROM_WORD_SIZE_SHIFT 6
1770 #define IXGBE_EEPROM_WORD_SIZE_BASE_SHIFT 6
1771 #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

```

```

1792 #define IXGBE_CSR1_CONFIG_PTR    0x0E
1793 #define IXGBE_FW_PTR              0x0F
1794 #define IXGBE_PBANUM0_PTR         0x15
1795 #define IXGBE_PBANUM1_PTR         0x16
1796 #define IXGBE_ALT_MAC_ADDR_PTR    0x37
1797 #define IXGBE_FREE_SPACE_PTR      0x3E

1799 #define IXGBE_SAN_MAC_ADDR_PTR     0x28
1800 #define IXGBE_DEVICE_CAPS         0x2C
1801 #define IXGBE_SERIAL_NUMBER_MAC_ADDR 0x11
1802 #define IXGBE_PCIE_MSIX_82599_CAPS 0x72
1803 #define IXGBE_MAX_MSIX_VECTORS_82599 0x40
1804 #define IXGBE_PCIE_MSIX_82598_CAPS 0x62
1805 #define IXGBE_MAX_MSIX_VECTORS_82598 0x13

1807 /* MSI-X capability fields masks */
1808 #define IXGBE_PCIE_MSIX_TBL_SZ_MASK 0x7FF

1810 /* Legacy EEPROM word offsets */
1811 #define IXGBE_ISCSI_B00T_CAPS      0x0033
1812 #define IXGBE_ISCSI_SETUP_PORT_0   0x0030
1813 #define IXGBE_ISCSI_SETUP_PORT_1   0x0034

1815 /* EEPROM Commands - SPI */
1816 #define IXGBE_EEPROM_MAX_RETRY_SPI 5000 /* Max wait 5ms for RDY signal */
1817 #define IXGBE_EEPROM_STATUS_RDY_SPI 0x01
1818 #define IXGBE_EEPROM_READ_OPCODE_SPI 0x03 /* EEPROM read opcode */
1819 #define IXGBE_EEPROM_WRITE_OPCODE_SPI 0x02 /* EEPROM write opcode */
1820 #define IXGBE_EEPROM_A8_OPCODE_SPI 0x08 /* opcode bit-3 = addr bit-8 */
1821 #define IXGBE_EEPROM_WREN_OPCODE_SPI 0x06 /* EEPROM set Write Ena latch */
1822 /* EEPROM reset Write Enable latch */
1823 #define IXGBE_EEPROM_WRTDI_OPCODE_SPI 0x04
1824 #define IXGBE_EEPROM_RDSR_OPCODE_SPI 0x05 /* EEPROM read Status reg */
1825 #define IXGBE_EEPROM_WRSR_OPCODE_SPI 0x01 /* EEPROM write Status reg */
1826 #define IXGBE_EEPROM_ERASE4K_OPCODE_SPI 0x20 /* EEPROM ERASE 4KB */
1827 #define IXGBE_EEPROM_ERASE64K_OPCODE_SPI 0xD8 /* EEPROM ERASE 64KB */
1828 #define IXGBE_EEPROM_ERASE256_OPCODE_SPI 0xDB /* EEPROM ERASE 256B */

1830 /* EEPROM Read Register */
1831 #define IXGBE_EEPROM_RW_REG_DATA 16 /* data offset in EEPROM read reg */
1832 #define IXGBE_EEPROM_RW_REG_DONE 2 /* Offset to READ done bit */
1833 #define IXGBE_EEPROM_RW_REG_START 1 /* First bit to start operation */
1834 #define IXGBE_EEPROM_RW_ADDR_SHIFT 2 /* Shift to the address bits */
1835 #define IXGBE_NVM_POLL_WRITE 1 /* Flag for polling for wr complete */
1836 #define IXGBE_NVM_POLL_READ 0 /* Flag for polling for rd complete */
1837 #define IXGBE_NVM_POLL_WRITE 1 /* Flag for polling for write complete */
1838 #define IXGBE_NVM_POLL_READ 0 /* Flag for polling for read complete */

1838 #define IXGBE_ETH_LENGTH_OF_ADDRESS 6

1840 #define IXGBE_EEPROM_PAGE_SIZE_MAX 128
1841 #define IXGBE_EEPROM_RD_BUFFER_MAX_COUNT 512 /* words rd in burst */
1842 #define IXGBE_EEPROM_WR_BUFFER_MAX_COUNT 256 /* words wr in burst */

1844 #ifndef IXGBE_EEPROM_GRANT_ATTEMPTS
1845 #define IXGBE_EEPROM_GRANT_ATTEMPTS 1000 /* EEPROM attempts to gain grant */
1846 #endif

1848 /* Number of 5 microseconds we wait for EERD read and
1849 * EERW write to complete */
1850 #define IXGBE_EERD_EEWR_ATTEMPTS 100000

1852 /* # attempts we wait for flush update to complete */
1853 #define IXGBE_FLUDONE_ATTEMPTS 20000

```

```

1855 #define IXGBE_PCIE_CTRL2         0x5 /* PCIe Control 2 Offset */
1856 #define IXGBE_PCIE_CTRL2_DUMMY_ENABLE 0x8 /* Dummy Function Enable */
1857 #define IXGBE_PCIE_CTRL2_LAN_DISABLE 0x2 /* LAN PCI Disable */
1858 #define IXGBE_PCIE_CTRL2_DISABLE_SELECT 0x1 /* LAN Disable Select */

1860 #define IXGBE_SAN_MAC_ADDR_PORT0_OFFSET 0x0
1861 #define IXGBE_SAN_MAC_ADDR_PORT1_OFFSET 0x3
1862 #define IXGBE_DEVICE_CAPS_ALLOW_ANY_SFP 0x1
1863 #define IXGBE_DEVICE_CAPS_FCOE_OFFLOADS 0x2
1864 #define IXGBE_FW_LESM_PARAMETERS_PTR 0x2
1865 #define IXGBE_FW_LESM_STATE_1 0x1
1866 #define IXGBE_FW_LESM_STATE_ENABLED 0x8000 /* LESM Enable bit */
1867 #define IXGBE_FW_PASSTHROUGH_PATCH_CONFIG_PTR 0x4
1868 #define IXGBE_FW_PATCH_VERSION_4 0x7
1869 #define IXGBE_FCOE_IBA_CAPS_BLK_PTR 0x33 /* iSCSI/FCOE block */
1870 #define IXGBE_FCOE_IBA_CAPS_FCOE 0x20 /* FCOE flags */
1871 #define IXGBE_ISCSI_FCOE_BLK_PTR 0x17 /* iSCSI/FCOE block */
1872 #define IXGBE_ISCSI_FCOE_FLAGS_OFFSET 0x0 /* FCOE flags */
1873 #define IXGBE_ISCSI_FCOE_FLAGS_ENABLE 0x1 /* FCOE flags enable bit */
1874 #define IXGBE_ALT_SAN_MAC_ADDR_BLK_PTR 0x27 /* Alt. SAN MAC block */
1875 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_OFFSET 0x0 /* Alt SAN MAC capability */
1876 #define IXGBE_ALT_SAN_MAC_ADDR_PORT0_OFFSET 0x1 /* Alt SAN MAC 0 offset */
1877 #define IXGBE_ALT_SAN_MAC_ADDR_PORT1_OFFSET 0x4 /* Alt SAN MAC 1 offset */
1878 #define IXGBE_ALT_SAN_MAC_ADDR_WNN_OFFSET 0x7 /* Alt WNN prefix offset */
1879 #define IXGBE_ALT_SAN_MAC_ADDR_WPN_OFFSET 0x8 /* Alt WPN prefix offset */
1880 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_SANMAC 0x0 /* Alt SAN MAC exists */
1881 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_ALTWNN 0x1 /* Alt WNN base exists */
1882 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_OFFSET 0x0 /* Alt. SAN MAC capability */
1883 #define IXGBE_ALT_SAN_MAC_ADDR_PORT0_OFFSET 0x1 /* Alt. SAN MAC 0 offset */
1884 #define IXGBE_ALT_SAN_MAC_ADDR_PORT1_OFFSET 0x4 /* Alt. SAN MAC 1 offset */
1885 #define IXGBE_ALT_SAN_MAC_ADDR_WNN_OFFSET 0x7 /* Alt. WNN prefix offset */
1886 #define IXGBE_ALT_SAN_MAC_ADDR_WPN_OFFSET 0x8 /* Alt. WPN prefix offset */
1887 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_SANMAC 0x0 /* Alt. SAN MAC exists */
1888 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_ALTWNN 0x1 /* Alt. WNN base exists */

1883 #define IXGBE_DEVICE_CAPS_WOL_PORT0_1 0x4 /* WoL supported on ports 0 & 1 */
1884 #define IXGBE_DEVICE_CAPS_WOL_PORT0 0x8 /* WoL supported on port 0 */
1885 #define IXGBE_DEVICE_CAPS_WOL_MASK 0xC /* Mask for WoL capabilities */

1887 /* PCI Bus Info */
1888 #define IXGBE_PCI_DEVICE_STATUS 0xAA
1889 #define IXGBE_PCI_DEVICE_STATUS_TRANSACTION_PENDING 0x0020
1890 #define IXGBE_PCI_LINK_STATUS 0xB2
1891 #define IXGBE_PCI_DEVICE_CONTROL2 0xC8
1892 #define IXGBE_PCI_LINK_WIDTH 0x3F0
1893 #define IXGBE_PCI_LINK_WIDTH_1 0x10
1894 #define IXGBE_PCI_LINK_WIDTH_2 0x20
1895 #define IXGBE_PCI_LINK_WIDTH_4 0x40
1896 #define IXGBE_PCI_LINK_WIDTH_8 0x80
1897 #define IXGBE_PCI_LINK_SPEED 0xF
1898 #define IXGBE_PCI_LINK_SPEED_2500 0x1
1899 #define IXGBE_PCI_LINK_SPEED_5000 0x2
1900 #define IXGBE_PCI_LINK_SPEED_8000 0x3
1901 #define IXGBE_PCI_HEADER_TYPE_REGISTER 0x0E
1902 #define IXGBE_PCI_HEADER_TYPE_MULTIFUNC 0x80
1903 #define IXGBE_PCI_DEVICE_CONTROL2_16ms 0x0005

1905 /* Number of 100 microseconds we wait for PCI Express master disable */
1906 #define IXGBE_PCI_MASTER_DISABLE_TIMEOUT 800

1908 /* Check whether address is multicast. This is little-endian specific check. */
1909 #define IXGBE_IS_MULTICAST(Address) \
1910     (bool)((((u8 *) (Address))[0] & ((u8) 0x01)))

1912 /* Check whether an address is broadcast. */
1913 #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 */
1944 #define IXGBE_TXDCTL_ENABLE        0x02000000 /* Enable specific Tx Queue */
1945 #define IXGBE_TXDCTL_SWFLSH        0x04000000 /* Tx Desc. write-back flushing */
1946 #define IXGBE_TXDCTL_WTHRESH_SHIFT 16 /* shift to WTHRESH bits */
1947 /* Enable short packet padding to 64 bytes */
1948 #define IXGBE_TX_PAD_ENABLE         0x00000400
1949 #define IXGBE_JUMBO_FRAME_ENABLE    0x00000004 /* Allow jumbo frames */
1950 /* This allows for 16K packets + 4k for vlan */
1951 #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
1961 #define IXGBE_RXCTRL_DMBYPS        0x00000002 /* Descriptor Monitor Bypass */
1962 #define IXGBE_RXDCTL_ENABLE        0x02000000 /* Enable specific Rx Queue */
1963 #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       0x00000000

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 0xffff0000 /* 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_FCOEFE 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_FCOEFS 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_FCP_RSP 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 /* SRCTL bit definitions */
2138 #define IXGBE_SRCTL_BSIZEPKT_SHIFT 10 /* so many KBs */
2139 #define IXGBE_SRCTL_RDMTS_SHIFT 22
2140 #define IXGBE_SRCTL_RDMTS_MASK 0x01C00000
2141 #define IXGBE_SRCTL_DROP_EN 0x10000000
2142 #define IXGBE_SRCTL_BSIZEPKT_MASK 0x0000007F
2143 #define IXGBE_SRCTL_BSIZEHDR_MASK 0x00003F00
2144 #define IXGBE_SRCTL_DESCTYPE_LEGACY 0x00000000
2145 #define IXGBE_SRCTL_DESCTYPE_ADV_ONEBUF 0x02000000
2146 #define IXGBE_SRCTL_DESCTYPE_HDR_SPLIT 0x04000000
2147 #define IXGBE_SRCTL_DESCTYPE_HDR_REPLICATION_LARGE_PKT 0x08000000
2148 #define IXGBE_SRCTL_DESCTYPE_HDR_SPLIT_ALWAYS 0x0A000000
2149 #define IXGBE_SRCTL_DESCTYPE_MASK 0x0E000000

2151 #define IXGBE_RXDPS_HDRSTAT_HDRSP 0x00008000
2152 #define IXGBE_RXDPS_HDRSTAT_HDRLEN_MASK 0x00003FFF

2154 #define IXGBE_RXDADV_RSSTYPE_MASK 0x0000000F
2155 #define IXGBE_RXDADV_PKTTYPE_MASK 0x0000FFFF
2156 #define IXGBE_RXDADV_PKTTYPE_MASK_EX 0x0001FFFF
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))
2234 #define IXGBE_MBVFICR(_i) (0x00710 + (_i * 4))
2235 #define IXGBE_VFLRE(_i) (((_i & 1) ? 0x001C0 : 0x00600))
2236 #define IXGBE_VFLREC(_i) (0x00700 + ((_i) * 4))
2237 #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
2246 #endif

2247 #endif
2248 #ifndef __be16
2249 /* Big Endian defines */
2250 #define __be16 u16
2251 #define __be32 u32
2252 #define __be64 u64
2253 #endif

2254 #endif
2255 enum ixgbe_fdir_pballoctype {
2256     IXGBE_FDIR_PBALLOC_NONE = 0,
2257     IXGBE_FDIR_PBALLOC_64K = 1,
2258     IXGBE_FDIR_PBALLOC_128K = 2,
2259     IXGBE_FDIR_PBALLOC_256K = 3,
2260     IXGBE_FDIR_PBALLOC_64K = 0,
2261     IXGBE_FDIR_PBALLOC_128K,
2262     IXGBE_FDIR_PBALLOC_256K,
2263 };
2264 #define IXGBE_FDIR_PBALLOC_SIZE_SHIFT 16

2266 /* Flow Director register values */
2267 #define IXGBE_FDIRCTRL_PBALLOC_64K 0x00000001
2268 #define IXGBE_FDIRCTRL_PBALLOC_128K 0x00000002
2269 #define IXGBE_FDIRCTRL_PBALLOC_256K 0x00000003
2270 #define IXGBE_FDIRCTRL_INIT_DONE 0x00000008
2271 #define IXGBE_FDIRCTRL_PERFECT_MATCH 0x00000010
2272 #define IXGBE_FDIRCTRL_REPORT_STATUS 0x00000020
2273 #define IXGBE_FDIRCTRL_REPORT_STATUS_ALWAYS 0x00000080
2274 #define IXGBE_FDIRCTRL_DROP_Q_SHIFT 8
2275 #define IXGBE_FDIRCTRL_FLEX_SHIFT 16
2276 #define IXGBE_FDIRCTRL_SEARCHLIM 0x00800000
2277 #define IXGBE_FDIRCTRL_MAX_LENGTH_SHIFT 24
2278 #define IXGBE_FDIRCTRL_FULL_THRESH_MASK 0xF0000000
2279 #define IXGBE_FDIRCTRL_FULL_THRESH_SHIFT 28

2281 #define IXGBE_FDIRTCM_DPORTM_SHIFT 16
2282 #define IXGBE_FDIRUDPM_DPORTM_SHIFT 16
2283 #define IXGBE_FDIRIP6M_DIPM_SHIFT 16
2284 #define IXGBE_FDIRM_VLANID 0x00000001
2285 #define IXGBE_FDIRM_VLANP 0x00000002
2286 #define IXGBE_FDIRM_POOL 0x00000004
2287 #define IXGBE_FDIRM_L4P 0x00000008
2288 #define IXGBE_FDIRM_FLEX 0x00000010
2289 #define IXGBE_FDIRM_DIPV6 0x00000020

2291 #define IXGBE_FDIRFREE_FREE_MASK 0xFFFF
2292 #define IXGBE_FDIRFREE_FREE_SHIFT 0
2293 #define IXGBE_FDIRFREE_COLL_MASK 0x7FFF0000
2294 #define IXGBE_FDIRFREE_COLL_SHIFT 16
2295 #define IXGBE_FDIRLEN_MAXLEN_MASK 0x3F
2296 #define IXGBE_FDIRLEN_MAXLEN_SHIFT 0
2297 #define IXGBE_FDIRLEN_MAXHASH_MASK 0x7FFF0000
2298 #define IXGBE_FDIRLEN_MAXHASH_SHIFT 16
2299 #define IXGBE_FDIRUSTAT_ADD_MASK 0xFFFF
2300 #define IXGBE_FDIRUSTAT_ADD_SHIFT 0
2301 #define IXGBE_FDIRUSTAT_REMOVE_MASK 0xFFFF0000
2302 #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
2313 #define IXGBE_FDIRCMD_CMD_QUERY_REM_HASH 0x00000007
2314 #define IXGBE_FDIRCMD_FILTER_UPDATE      0x00000008
2315 #define IXGBE_FDIRCMD_IPv6DMATCH         0x00000010
2316 #define IXGBE_FDIRCMD_L4TYPE_UDP         0x00000020
2317 #define IXGBE_FDIRCMD_L4TYPE_TCP         0x00000040
2318 #define IXGBE_FDIRCMD_L4TYPE_SCTP       0x00000060
2319 #define IXGBE_FDIRCMD_IPV6               0x00000080
2320 #define IXGBE_FDIRCMD_CLEARHT            0x00000100
2321 #define IXGBE_FDIRCMD_DROP               0x00000200
2322 #define IXGBE_FDIRCMD_INT                0x00000400
2323 #define IXGBE_FDIRCMD_LAST               0x00000800
2324 #define IXGBE_FDIRCMD_COLLISION          0x00001000
2325 #define IXGBE_FDIRCMD_QUEUE_EN           0x00008000
2326 #define IXGBE_FDIRCMD_FLOW_TYPE_SHIFT    5
2327 #define IXGBE_FDIRCMD_RX_QUEUE_SHIFT     16
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 /* Manageability 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 };
2392 /* 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 */
2465 #define IXGBE_ADVTXD_IPSEC_SA_INDEX_MASK 0x00200000 /* Advanced Context Desc */
2466 #define IXGBE_ADVTXD_DTYP_DATA           0x00300000 /* Advanced Data Descriptor */
2467 #define IXGBE_ADVTXD_DCMD_EOP             IXGBE_TXD_CMD_EOP /* End of Packet */
2468 #define IXGBE_ADVTXD_DCMD_IFCS            IXGBE_TXD_CMD_IFCS /* Insert FCS */
2469 #define IXGBE_ADVTXD_DCMD_RS              IXGBE_TXD_CMD_RS /* Report Status */
2470 #define IXGBE_ADVTXD_DCMD_DDTYP_ISCSI     0x10000000 /* DDP hdr type or iSCSI */
2471 #define IXGBE_ADVTXD_DCMD_DEXT            IXGBE_TXD_CMD_DEXT /* Desc ext 1=Adv */
2472 #define IXGBE_ADVTXD_DCMD_DEXT            IXGBE_TXD_CMD_DEXT /* Desc ext (1=Adv) */
2473 #define IXGBE_ADVTXD_DCMD_VLE             IXGBE_TXD_CMD_VLE /* VLAN pkt enable */
2474 #define IXGBE_ADVTXD_DCMD_TSE             0x80000000 /* TCP Seg enable */
2475 #define IXGBE_ADVTXD_STAT_DD              IXGBE_TXD_STAT_DD /* Descriptor Done */
2476 #define IXGBE_ADVTXD_STAT_SN_CRC          0x00000002 /* NXTSEQ/SEED pres in WB */
2477 #define IXGBE_ADVTXD_STAT_RSV             0x0000000C /* STA Reserved */
2478 #define IXGBE_ADVTXD_IDX_SHIFT            4 /* Adv desc Index shift */
2479 #define IXGBE_ADVTXD_CC                   0x00000080 /* Check Context */
2480 #define IXGBE_ADVTXD_POPTS_SHIFT          8 /* Adv desc POPTS shift */
2481 #define IXGBE_ADVTXD_POPTS_IJSM           (IXGBE_TXD_POPTS_IJSM << \
IXGBE_ADVTXD_POPTS_SHIFT)
2482 #define IXGBE_ADVTXD_POPTS_TXSM           (IXGBE_TXD_POPTS_TXSM << \
IXGBE_ADVTXD_POPTS_SHIFT)
2483 #define IXGBE_ADVTXD_POPTS_ISCO_1ST      0x00000000 /* 1st TSO of iSCSI PDU */
2484 #define IXGBE_ADVTXD_POPTS_ISCO_MDL      0x00000800 /* Middle TSO of iSCSI PDU */
2485 #define IXGBE_ADVTXD_POPTS_ISCO_LAST     0x00001000 /* Last TSO of iSCSI PDU */
2486 #define IXGBE_ADVTXD_POPTS_ISCO_FULL     0x00001800 /* 1st&Last TSO-full iSCSI PDU */
2487 #define IXGBE_ADVTXD_POPTS_ISCO_FULL     0x00001800 /* 1st&Last TSO-full iSCSI PDU */
2488 #define IXGBE_ADVTXD_POPTS_RS            0x00002000 /* POPTS Reserved */

```

```

2488 #define IXGBE_ADVTXD_PAYLEN_SHIFT 14 /* Adv desc PAYLEN shift */
2489 #define IXGBE_ADVTXD_MACLEN_SHIFT 9 /* Adv ctxt desc mac len shift */
2490 #define IXGBE_ADVTXD_VLAN_SHIFT 16 /* Adv ctxt vlan tag shift */
2491 #define IXGBE_ADVTXD_TUCMD_IPV4 0x000000400 /* IP Packet Type: 1=IPv4 */
2492 #define IXGBE_ADVTXD_TUCMD_IPV6 0x000000000 /* IP Packet Type: 0=IPv6 */
2493 #define IXGBE_ADVTXD_TUCMD_L4T_UDP 0x000000000 /* L4 Packet TYPE of UDP */
2494 #define IXGBE_ADVTXD_TUCMD_L4T_TCP 0x000000800 /* L4 Packet TYPE of TCP */
2495 #define IXGBE_ADVTXD_TUCMD_L4T_SCTP 0x000001000 /* L4 Packet TYPE of SCTP */
2496 #define IXGBE_ADVTXD_TUCMD_MKRREQ 0x000002000 /* req Markers and CRC */
2497 #define IXGBE_ADVTXD_POPTS_IPSEC 0x00000400 /* IPsec offload request */
2498 #define IXGBE_ADVTXD_TUCMD_IPSEC_TYPE_ESP 0x00002000 /* IPsec Type ESP */
2499 #define IXGBE_ADVTXD_TUCMD_IPSEC_ENCRYPT_EN 0x00004000 /* ESP Encrypt Enable */
2500 #define IXGBE_ADVTXD_TUCMD_FCOE 0x00008000 /* FCoE Frame Type */
2501 #define IXGBE_ADVTXD_FCOEF_EOF_MASK (0x3 << 10) /* FC EOF index */
2502 #define IXGBE_ADVTXD_FCOEF_SOF ((1 << 2) << 10) /* FC SOF index */
2503 #define IXGBE_ADVTXD_FCOEF_PARINC ((1 << 3) << 10) /* Rel_Off in F_CTL */
2504 #define IXGBE_ADVTXD_FCOEF_ORIE ((1 << 4) << 10) /* Orientation End */
2505 #define IXGBE_ADVTXD_FCOEF_ORIS ((1 << 5) << 10) /* Orientation Start */
2506 #define IXGBE_ADVTXD_FCOEF_ORIE ((1 << 4) << 10) /* Orientation: End */
2507 #define IXGBE_ADVTXD_FCOEF_ORIS ((1 << 5) << 10) /* Orientation: Start */
2508 #define IXGBE_ADVTXD_FCOEF_EOF_N (0x0 << 10) /* 00: EOFn */
2509 #define IXGBE_ADVTXD_FCOEF_EOF_T (0x1 << 10) /* 01: EOFt */
2510 #define IXGBE_ADVTXD_FCOEF_EOF_NI (0x2 << 10) /* 10: EOFni */
2511 #define IXGBE_ADVTXD_FCOEF_EOF_A (0x3 << 10) /* 11: EOFa */
2512 #define IXGBE_ADVTXD_L4LEN_SHIFT 8 /* Adv ctxt L4LEN shift */
2513 #define IXGBE_ADVTXD_MSS_SHIFT 16 /* Adv ctxt MSS shift */

2513 /* Autonegotiation advertised speeds */
2514 typedef u32 ixgbe_autoneg_advertised;
2515 /* Link speed */
2516 typedef u32 ixgbe_link_speed;
2517 #define IXGBE_LINK_SPEED_UNKNOWN 0
2518 #define IXGBE_LINK_SPEED_100_FULL 0x0008
2519 #define IXGBE_LINK_SPEED_1GB_FULL 0x0020
2520 #define IXGBE_LINK_SPEED_10GB_FULL 0x0080
2521 #define IXGBE_LINK_SPEED_82598_AUTONEG (IXGBE_LINK_SPEED_1GB_FULL | \
2522 IXGBE_LINK_SPEED_10GB_FULL)
2523 #define IXGBE_LINK_SPEED_82599_AUTONEG (IXGBE_LINK_SPEED_100_FULL | \
2524 IXGBE_LINK_SPEED_1GB_FULL | \
2525 IXGBE_LINK_SPEED_10GB_FULL)

2528 /* Physical layer type */
2529 typedef u32 ixgbe_physical_layer;
2530 #define IXGBE_PHYSICAL_LAYER_UNKNOWN 0
2531 #define IXGBE_PHYSICAL_LAYER_10GBASE_T 0x0001
2532 #define IXGBE_PHYSICAL_LAYER_1000BASE_T 0x0002
2533 #define IXGBE_PHYSICAL_LAYER_100BASE_TX 0x0004
2534 #define IXGBE_PHYSICAL_LAYER_SFP_PLUS_CU 0x0008
2535 #define IXGBE_PHYSICAL_LAYER_10GBASE_LR 0x0010
2536 #define IXGBE_PHYSICAL_LAYER_10GBASE_LRM 0x0020
2537 #define IXGBE_PHYSICAL_LAYER_10GBASE_SR 0x0040
2538 #define IXGBE_PHYSICAL_LAYER_10GBASE_KX4 0x0080
2539 #define IXGBE_PHYSICAL_LAYER_10GBASE_CX4 0x0100
2540 #define IXGBE_PHYSICAL_LAYER_1000BASE_KX 0x0200
2541 #define IXGBE_PHYSICAL_LAYER_1000BASE_BX 0x0400
2542 #define IXGBE_PHYSICAL_LAYER_10GBASE_KR 0x0800
2543 #define IXGBE_PHYSICAL_LAYER_10GBASE_XAUI 0x1000
2544 #define IXGBE_PHYSICAL_LAYER_SFP_ACTIVE_DA 0x2000
2545 #define IXGBE_PHYSICAL_LAYER_1000BASE_SX 0x4000

2547 /* Flow Control Data Sheet defined values
2548 * Calculation and defines taken from 802.1bb Annex 0
2549 */
2548 /* Flow Control Macros */

```

```

2285 #define PAUSE_RTT 8
2286 #define PAUSE_MTU(MTU) ((MTU + 1024 - 1) / 1024)

2551 /* BitTimes (BT) conversion */
2552 #define IXGBE_BT2KB(BT) ((BT + (8 * 1024 - 1)) / (8 * 1024))
2553 #define IXGBE_B2BT(BT) (BT * 8)
2288 #define FC_HIGH_WATER(MTU) (((PAUSE_RTT + PAUSE_MTU(MTU)) * 144) + 99) / 100 + \
2289 PAUSE_MTU(MTU))
2290 #define FC_LOW_WATER(MTU) (2 * (2 * PAUSE_MTU(MTU) + PAUSE_RTT))

2555 /* Calculate Delay to respond to PFC */
2556 #define IXGBE_PFC_D 672

2558 /* Calculate Cable Delay */
2559 #define IXGBE_CABLE_DC 5556 /* Delay Copper */
2560 #define IXGBE_CABLE_DO 5000 /* Delay Optical */

2562 /* Calculate Interface Delay X540 */
2563 #define IXGBE_PHY_DC 25600 /* Delay 10G BASET */
2564 #define IXGBE_MAC_DC 8192 /* Delay Copper XAUI interface */
2565 #define IXGBE_XAUI_DC (2 * 2048) /* Delay Copper Phy */

2567 #define IXGBE_ID_X540 (IXGBE_MAC_DC + IXGBE_XAUI_DC + IXGBE_PHY_DC)

2569 /* Calculate Interface Delay 82598, 82599 */
2570 #define IXGBE_PHY_D 12800
2571 #define IXGBE_MAC_D 4096
2572 #define IXGBE_XAUI_D (2 * 1024)

2574 #define IXGBE_ID (IXGBE_MAC_D + IXGBE_XAUI_D + IXGBE_PHY_D)

2576 /* Calculate Delay incurred from higher layer */
2577 #define IXGBE_HD 6144

2579 /* Calculate PCI Bus delay for low thresholds */
2580 #define IXGBE_PCI_DELAY 10000

2582 /* Calculate X540 delay value in bit times */
2583 #define IXGBE_DV_X540(_max_frame_link, _max_frame_tc) \
2584 ((36 * \
2585 (IXGBE_B2BT(_max_frame_link) + \
2586 IXGBE_PFC_D + \
2587 (2 * IXGBE_CABLE_DC) + \
2588 (2 * IXGBE_ID_X540) + \
2589 IXGBE_HD) / 25 + 1) + \
2590 2 * IXGBE_B2BT(_max_frame_tc))

2592 /* Calculate 82599, 82598 delay value in bit times */
2593 #define IXGBE_DV(_max_frame_link, _max_frame_tc) \
2594 ((36 * \
2595 (IXGBE_B2BT(_max_frame_link) + \
2596 IXGBE_PFC_D + \
2597 (2 * IXGBE_CABLE_DC) + \
2598 (2 * IXGBE_ID) + \
2599 IXGBE_HD) / 25 + 1) + \
2600 2 * IXGBE_B2BT(_max_frame_tc))

2602 /* Calculate low threshold delay values */
2603 #define IXGBE_LOW_DV_X540(_max_frame_tc) \
2604 (2 * IXGBE_B2BT(_max_frame_tc) + \
2605 (36 * IXGBE_PCI_DELAY / 25) + 1)
2606 #define IXGBE_LOW_DV(_max_frame_tc) \
2607 (2 * IXGBE_LOW_DV_X540(_max_frame_tc))

2609 /* Software ATR hash keys */
2610 #define IXGBE_ATR_BUCKET_HASH_KEY 0x3DAD14E2

```

```

2611 #define IXGBE_ATR_SIGNATURE_HASH_KEY    0x174D3614

2613 /* Software ATR input stream values and masks */
2614 #define IXGBE_ATR_HASH_MASK              0x7fff
2615 #define IXGBE_ATR_L4TYPE_MASK            0x3
2616 #define IXGBE_ATR_L4TYPE_UDP              0x1
2617 #define IXGBE_ATR_L4TYPE_TCP              0x2
2618 #define IXGBE_ATR_L4TYPE_SCTP             0x3
2619 #define IXGBE_ATR_L4TYPE_IPV6_MASK        0x4
2620 enum ixgbe_atr_flow_type {
2621     IXGBE_ATR_FLOW_TYPE_IPV4              = 0x0,
2622     IXGBE_ATR_FLOW_TYPE_UDPV4              = 0x1,
2623     IXGBE_ATR_FLOW_TYPE_TCPV4              = 0x2,
2624     IXGBE_ATR_FLOW_TYPE_SCTPV4             = 0x3,
2625     IXGBE_ATR_FLOW_TYPE_IPV6              = 0x4,
2626     IXGBE_ATR_FLOW_TYPE_UDPV6              = 0x5,
2627     IXGBE_ATR_FLOW_TYPE_TCPV6              = 0x6,
2628     IXGBE_ATR_FLOW_TYPE_SCTPV6             = 0x7,
2629 };

2631 /* Flow Director ATR input struct. */
2632 union ixgbe_atr_input {
2633     /*
2634      * Byte layout in order, all values with MSB first:
2635      *
2636      * vm_pool      - 1 byte
2637      * flow_type    - 1 byte
2638      * vlan_id      - 2 bytes
2639      * src_ip       - 16 bytes
2640      * dst_ip       - 16 bytes
2641      * src_port     - 2 bytes
2642      * dst_port     - 2 bytes
2643      * flex_bytes   - 2 bytes
2644      * bkt_hash     - 2 bytes
2645      * rsvd0        - 2 bytes - space reserved must be 0.
2646      */
2647     struct {
2648         u8 vm_pool;
2649         u8 flow_type;
2650         __be16 vlan_id;
2651         __be32 dst_ip[4];
2652         __be32 src_ip[4];
2653         __be16 src_port;
2654         __be16 dst_port;
2655         __be16 flex_bytes;
2656         __be16 bkt_hash;
2657         __be16 rsvd0;
2658     } formatted;
2659     __be32 dword_stream[11];
2660 };

```

unchanged portion omitted

```

2360 struct ixgbe_atr_input_masks {
2361     __be16 rsvd0;
2362     __be16 vlan_id_mask;
2363     __be32 dst_ip_mask[4];
2364     __be32 src_ip_mask[4];
2365     __be16 src_port_mask;
2366     __be16 dst_port_mask;
2367     __be16 flex_mask;
2368 };

```

```

2677 /*
2678  * Unavailable: The FCoE Boot Option ROM is not present in the flash.
2679  * Disabled: Present; boot order is not set for any targets on the port.

```

```

2680  * Enabled: Present; boot order is set for at least one target on the port.
2681  */
2682 enum ixgbe_fcoe_boot_status {
2683     ixgbe_fcoe_bootstatus_disabled = 0,
2684     ixgbe_fcoe_bootstatus_enabled = 1,
2685     ixgbe_fcoe_bootstatus_unavailable = 0xFFFF
2686 };

```

unchanged portion omitted

```

2695 enum ixgbe_mac_type {
2696     ixgbe_mac_unknown = 0,
2697     ixgbe_mac_82598EB,
2698     ixgbe_mac_82599EB,
2699     ixgbe_mac_82599_vf,
2700     ixgbe_mac_X540,
2701     ixgbe_mac_X540_vf,
2702     ixgbe_num_macs
2703 };

```

unchanged portion omitted

```

2726 /*
2727  * SFP+ module type IDs:
2728  *
2729  * ID      Module Type
2730  * =====
2731  * 0       SFP_DA_CU
2732  * 1       SFP_SR
2733  * 2       SFP_LR
2734  * 3       SFP_DA_CU_CORE0 - 82599-specific
2735  * 4       SFP_DA_CU_CORE1 - 82599-specific
2736  * 5       SFP_SR/LR_CORE0 - 82599-specific
2737  * 6       SFP_SR/LR_CORE1 - 82599-specific
2738  */
2739 enum ixgbe_sfp_type {
2740     ixgbe_sfp_type_da_cu = 0,
2741     ixgbe_sfp_type_sr = 1,
2742     ixgbe_sfp_type_lr = 2,
2743     ixgbe_sfp_type_da_cu_core0 = 3,
2744     ixgbe_sfp_type_da_cu_core1 = 4,
2745     ixgbe_sfp_type_srlr_core0 = 5,
2746     ixgbe_sfp_type_srlr_core1 = 6,
2747     ixgbe_sfp_type_da_act_lmt_core0 = 7,
2748     ixgbe_sfp_type_da_act_lmt_core1 = 8,
2749     ixgbe_sfp_type_1g_cu_core0 = 9,
2750     ixgbe_sfp_type_1g_cu_core1 = 10,
2751     ixgbe_sfp_type_1g_sx_core0 = 11,
2752     ixgbe_sfp_type_1g_sx_core1 = 12,
2753     ixgbe_sfp_type_not_present = 0xFFFE,
2754     ixgbe_sfp_type_unknown = 0xFFFF
2755 };

```

unchanged portion omitted

```

2792 /* PCI bus speeds */
2793 enum ixgbe_bus_speed {
2794     ixgbe_bus_speed_unknown = 0,
2795     ixgbe_bus_speed_33      = 33,
2796     ixgbe_bus_speed_66      = 66,
2797     ixgbe_bus_speed_100     = 100,
2798     ixgbe_bus_speed_120     = 120,
2799     ixgbe_bus_speed_133     = 133,
2800     ixgbe_bus_speed_2500    = 2500,
2801     ixgbe_bus_speed_5000    = 5000,
2802     ixgbe_bus_speed_8000    = 8000,
2803     ixgbe_bus_speed_reserved
2804 };

```

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 */
2840     u32 high_water; /* Flow Control High-water */
2841     u32 low_water; /* Flow Control Low-water */
2842     u16 pause_time; /* Flow Control Pause timer */
2843     bool send_xon; /* Flow control send XON */
2844     bool strict_ieee; /* Strict IEEE mode */
2845     bool disable_fc_autoneg; /* Do not autonegotiate FC */
2846     bool fc_was_autonegged; /* Is current_mode the result of autonegging? */
2847     enum ixgbe_fc_mode current_mode; /* FC mode in effect */
2848     enum ixgbe_fc_mode requested_mode; /* FC mode requested by caller */
2849 };
2850
2851 /* Statistics counters collected by the MAC */
2852 struct ixgbe_hw_stats {
2853     u64 crcerrs;
2854     u64 illerrc;
2855     u64 errbc;
2856     u64 mspdc;
2857     u64 mpctotal;
2858     u64 mpc[8];
2859     u64 mlfc;
2860     u64 mrfc;
2861     u64 rlec;
2862     u64 lxontxc;
2863     u64 lxonrxc;
2864     u64 lxofftxc;
2865     u64 lxoffrxc;
2866     u64 pxontxc[8];
2867     u64 pxonrxc[8];
2868     u64 pxofftxc[8];
2869     u64 pxoffrxc[8];
2870     u64 prc64;
2871     u64 prc127;
2872     u64 prc255;
2873     u64 prc511;
2874     u64 prc1023;
2875     u64 prc1522;
2876     u64 gprc;
2877     u64 bprc;
2878     u64 mprc;
2879     u64 gptc;
2880     u64 gorc;
2881     u64 gotc;
2882     u64 rnbtc[8];
2883     u64 ruc;
2884     u64 rfc;
2885     u64 roc;
2886     u64 rjc;
2887     u64 mngprc;
2888     u64 mngpdc;
2889     u64 mngptc;
2890     u64 tor;
2891     u64 tpr;
2892     u64 tpt;
2893     u64 ptc64;
2894     u64 ptc127;
2895     u64 ptc255;
2896     u64 ptc511;
2897     u64 ptc1023;
2898     u64 ptc1522;
2899     u64 mptc;
2900     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 fcldst;
2914     u64 fcoerpc;
2915     u64 fcoeprc;
2916     u64 fcoepdc;
2917     u64 fcoedwrc;
2918     u64 fcoedwrc;
2919     u64 fcoe_noddp;
2920     u64 fcoe_noddp_ext_buff;
2921     u64 ldpcc;
2922     u64 pcrc8ec;
2923     u64 b2ospc;
2924     u64 b2ogprc;
2925     u64 o2bgprc;
2926     u64 o2bspc;
2927 };
2928
2929 /* forward declaration */
2930 struct ixgbe_hw;
2931
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);
2935
2936 /* Function pointer table */
2937 struct ixgbe_eeeprom_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 };
2947
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 rst;
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_eeeprom_info eeeprom;
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 Fri Jul 20 10:59:44 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_x540.c
XXX Intel X540 support
*****
1 /*****
3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD: src/sys/dev/ixgbe/ixgbe_x540.c,v 1.2 2012/07/05 20:51:44 jfv Exp $*/
34
35 #include "ixgbe_x540.h"
36 #include "ixgbe_type.h"
37 #include "ixgbe_api.h"
38 #include "ixgbe_common.h"
39 #include "ixgbe_phy.h"
40
41 static s32 ixgbe_update_flash_X540(struct ixgbe_hw *hw);
42 static s32 ixgbe_poll_flash_update_done_X540(struct ixgbe_hw *hw);
43 static s32 ixgbe_get_swfw_sync_semaphore(struct ixgbe_hw *hw);
44 static void ixgbe_release_swfw_sync_semaphore(struct ixgbe_hw *hw);
45
46 /**
47 * ixgbe_init_ops_X540 - Inits func ptrs and MAC type
48 * @hw: pointer to hardware structure
49 *
50 * Initialize the function pointers and assign the MAC type for X540.
51 * Does not touch the hardware.
52 */
53 s32 ixgbe_init_ops_X540(struct ixgbe_hw *hw)
54 {
55     struct ixgbe_mac_info *mac = &hw->mac;
56     struct ixgbe_phy_info *phy = &hw->phy;
57     struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
58     s32 ret_val;
59
60     DEBUGFUNC("ixgbe_init_ops_X540");

```

```

62     ret_val = ixgbe_init_phy_ops_generic(hw);
63     ret_val = ixgbe_init_ops_generic(hw);
64
65     /* EEPROM */
66     eeprom->ops.init_params = &ixgbe_init_eeprom_params_X540;
67     eeprom->ops.read = &ixgbe_read_eerd_X540;
68     eeprom->ops.read_buffer = &ixgbe_read_eerd_buffer_X540;
69     eeprom->ops.write = &ixgbe_write_eewr_X540;
70     eeprom->ops.write_buffer = &ixgbe_write_eewr_buffer_X540;
71     eeprom->ops.update_checksum = &ixgbe_update_eeprom_checksum_X540;
72     eeprom->ops.validate_checksum = &ixgbe_validate_eeprom_checksum_X540;
73     eeprom->ops.calc_checksum = &ixgbe_calc_eeprom_checksum_X540;
74
75     /* PHY */
76     phy->ops.init = &ixgbe_init_phy_ops_generic;
77     phy->ops.reset = NULL;
78
79     /* MAC */
80     mac->ops.reset_hw = &ixgbe_reset_hw_X540;
81     mac->ops.enable_relaxed_ordering = &ixgbe_enable_relaxed_ordering_gen2;
82     mac->ops.get_media_type = &ixgbe_get_media_type_X540;
83     mac->ops.get_supported_physical_layer =
84         &ixgbe_get_supported_physical_layer_X540;
85     mac->ops.read_analog_reg8 = NULL;
86     mac->ops.write_analog_reg8 = NULL;
87     mac->ops.start_hw = &ixgbe_start_hw_X540;
88     mac->ops.get_san_mac_addr = &ixgbe_get_san_mac_addr_generic;
89     mac->ops.set_san_mac_addr = &ixgbe_set_san_mac_addr_generic;
90     mac->ops.get_device_caps = &ixgbe_get_device_caps_generic;
91     mac->ops.get_wnn_prefix = &ixgbe_get_wnn_prefix_generic;
92     mac->ops.get_fcqe_boot_status = &ixgbe_get_fcqe_boot_status_generic;
93     mac->ops.acquire_swfw_sync = &ixgbe_acquire_swfw_sync_X540;
94     mac->ops.release_swfw_sync = &ixgbe_release_swfw_sync_X540;
95     mac->ops.disable_sec_rx_path = &ixgbe_disable_sec_rx_path_generic;
96     mac->ops.enable_sec_rx_path = &ixgbe_enable_sec_rx_path_generic;
97
98     /* RAR, Multicast, VLAN */
99     mac->ops.set_vmdq = &ixgbe_set_vmdq_generic;
100     mac->ops.set_vmdq_san_mac = &ixgbe_set_vmdq_san_mac_generic;
101     mac->ops.clear_vmdq = &ixgbe_clear_vmdq_generic;
102     mac->ops.insert_mac_addr = &ixgbe_insert_mac_addr_generic;
103     mac->rar_highwater = 1;
104     mac->ops.set_vfta = &ixgbe_set_vfta_generic;
105     mac->ops.set_vlvf = &ixgbe_set_vlvf_generic;
106     mac->ops.clear_vfta = &ixgbe_clear_vfta_generic;
107     mac->ops.init_uta_tables = &ixgbe_init_uta_tables_generic;
108     mac->ops.set_mac_anti_spoofing = &ixgbe_set_mac_anti_spoofing;
109     mac->ops.set_vlan_anti_spoofing = &ixgbe_set_vlan_anti_spoofing;
110
111     /* Link */
112     mac->ops.get_link_capabilities =
113         &ixgbe_get_copper_link_capabilities_generic;
114     mac->ops.setup_link = &ixgbe_setup_mac_link_X540;
115     mac->ops.setup_rxpba = &ixgbe_set_rxpba_generic;
116     mac->ops.check_link = &ixgbe_check_mac_link_generic;
117
118     mac->mcft_size = 128;
119     mac->vft_size = 128;
120     mac->num_rar_entries = 128;
121     mac->rx_pb_size = 384;
122     mac->max_tx_queues = 128;
123     mac->max_rx_queues = 128;
124     mac->max_msix_vectors = ixgbe_get_pcie_msix_count_generic(hw);
125
126     /*

```

```

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);
265      }
266
267      /* Save the SAN MAC RAR index */
268      hw->mac.san_mac_rar_index = hw->mac.num_rar_entries - 1;
269
270      /* Reserve the last RAR for the SAN MAC address */
271      hw->mac.num_rar_entries--;
272  }
273
274  /* Store the alternative WNN/WPN prefix */
275  hw->mac.ops.get_wnn_prefix(hw, &hw->mac.wnn_prefix,
276                          &hw->mac.wpn_prefix);
277
278  reset_hw_out:
279      return status;
280  }
281
282  /**
283   * ixgbe_start_hw_X540 - Prepare hardware for Tx/Rx
284   * @hw: pointer to hardware structure
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;
292
293      DEBUGFUNC("ixgbe_start_hw_X540");
294
295      ret_val = ixgbe_start_hw_generic(hw);
296      if (ret_val != IXGBE_SUCCESS)
297          goto out;
298
299      ret_val = ixgbe_start_hw_gen2(hw);
300
301  out:
302      return ret_val;
303  }
304
305  /**
306   * ixgbe_get_supported_physical_layer_X540 - Returns physical layer type
307   * @hw: pointer to hardware structure
308   * Determines physical layer capabilities of the current configuration.
309   */
310  u32 ixgbe_get_supported_physical_layer_X540(struct ixgbe_hw *hw)
311  {
312      u32 physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
313      u16 ext_ability = 0;
314
315      DEBUGFUNC("ixgbe_get_supported_physical_layer_X540");
316
317      hw->phy.ops.read_reg(hw, IXGBE_MDIO_PHY_EXT_ABILITY,
318                          IXGBE_MDIO_PMA_PMD_DEV_TYPE, &ext_ability);
319      if (ext_ability & IXGBE_MDIO_PHY_10GBASET_ABILITY)
320          physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_T;
321      if (ext_ability & IXGBE_MDIO_PHY_1000BASET_ABILITY)
322          physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_T;
323      if (ext_ability & IXGBE_MDIO_PHY_100BASETX_ABILITY)
324          physical_layer |= IXGBE_PHYSICAL_LAYER_100BASE_TX;
325  }

```

```

327      return physical_layer;
328  }
329
330  /**
331   * ixgbe_init_eeprom_params_X540 - Initialize EEPROM params
332   * @hw: pointer to hardware structure
333   * Initializes the EEPROM parameters ixgbe_eeprom_info within the
334   * ixgbe_hw struct in order to set up EEPROM access.
335   */
336  s32 ixgbe_init_eeprom_params_X540(struct ixgbe_hw *hw)
337  {
338      struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
339      u32 eec;
340      u16 eeprom_size;
341
342      DEBUGFUNC("ixgbe_init_eeprom_params_X540");
343
344      if (eeprom->type == ixgbe_eeprom_uninitialized) {
345          eeprom->semaphore_delay = 10;
346          eeprom->type = ixgbe_flash;
347
348          eec = IXGBE_READ_REG(hw, IXGBE_EEC);
349          eeprom_size = (u16)((eec & IXGBE_EEC_SIZE) >>
350                          IXGBE_EEC_SIZE_SHIFT);
351          eeprom->word_size = 1 << (eeprom_size +
352                          IXGBE_EEPROM_WORD_SIZE_SHIFT);
353
354          DEBUGOUT2("Eeprom params: type = %d, size = %d\n",
355                  eeprom->type, eeprom->word_size);
356      }
357
358      return IXGBE_SUCCESS;
359  }
360
361  /**
362   * ixgbe_read_eerd_X540 - Read EEPROM word using EERD
363   * @hw: pointer to hardware structure
364   * @offset: offset of word in the EEPROM to read
365   * @data: word read from the EEPROM
366   * Reads a 16 bit word from the EEPROM using the EERD register.
367   */
368  s32 ixgbe_read_eerd_X540(struct ixgbe_hw *hw, u16 offset, u16 *data)
369  {
370      s32 status = IXGBE_SUCCESS;
371
372      DEBUGFUNC("ixgbe_read_eerd_X540");
373      if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
374          IXGBE_SUCCESS)
375          status = ixgbe_read_eerd_generic(hw, offset, data);
376      else
377          status = IXGBE_ERR_SWFW_SYNC;
378
379      hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
380      return status;
381  }
382
383  /**
384   * ixgbe_read_eerd_buffer_X540 - Read EEPROM word(s) using EERD
385   * @hw: pointer to hardware structure
386   * @offset: offset of word in the EEPROM to read
387   * @words: number of words
388   * @data: word(s) read from the EEPROM
389   */

```

```

392  * Reads a 16 bit word(s) from the EEPROM using the EERD register.
393  **/
394  s32 ixgbe_read_eerd_buffer_X540(struct ixgbe_hw *hw,
395                                u16 offset, u16 words, u16 *data)
396  {
397      s32 status = IXGBE_SUCCESS;
398
399      DEBUGFUNC("ixgbe_read_eerd_buffer_X540");
400      if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
401          IXGBE_SUCCESS)
402          status = ixgbe_read_eerd_buffer_generic(hw, offset,
403                                                  words, data);
404      else
405          status = IXGBE_ERR_SWFW_SYNC;
406
407      hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
408      return status;
409  }
410
411  /**
412   * ixgbe_write_eewr_X540 - Write EEPROM word using EEWL
413   * @hw: pointer to hardware structure
414   * @offset: offset of word in the EEPROM to write
415   * @data: word write to the EEPROM
416   *
417   * Write a 16 bit word to the EEPROM using the EEWL register.
418   **/
419  s32 ixgbe_write_eewr_X540(struct ixgbe_hw *hw, u16 offset, u16 data)
420  {
421      s32 status = IXGBE_SUCCESS;
422
423      DEBUGFUNC("ixgbe_write_eewr_X540");
424      if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
425          IXGBE_SUCCESS)
426          status = ixgbe_write_eewr_generic(hw, offset, data);
427      else
428          status = IXGBE_ERR_SWFW_SYNC;
429
430      hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
431      return status;
432  }
433
434  /**
435   * ixgbe_write_eewr_buffer_X540 - Write EEPROM word(s) using EEWL
436   * @hw: pointer to hardware structure
437   * @offset: offset of word in the EEPROM to write
438   * @words: number of words
439   * @data: word(s) write to the EEPROM
440   *
441   * Write a 16 bit word(s) to the EEPROM using the EEWL register.
442   **/
443  s32 ixgbe_write_eewr_buffer_X540(struct ixgbe_hw *hw,
444                                   u16 offset, u16 words, u16 *data)
445  {
446      s32 status = IXGBE_SUCCESS;
447
448      DEBUGFUNC("ixgbe_write_eewr_buffer_X540");
449      if (hw->mac.ops.acquire_swfw_sync(hw, IXGBE_GSSR_EEP_SM) ==
450          IXGBE_SUCCESS)
451          status = ixgbe_write_eewr_buffer_generic(hw, offset,
452                                                  words, data);
453      else
454          status = IXGBE_ERR_SWFW_SYNC;
455
456      hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_EEP_SM);
457      return status;

```

```

458  }
459
460  /**
461   * ixgbe_calc_eeprom_checksum_X540 - Calculates and returns the checksum
462   *
463   * This function does not use synchronization for EERD and EEWL. It can
464   * be used internally by function which utilize ixgbe_acquire_swfw_sync_X540.
465   *
466   * @hw: pointer to hardware structure
467   **/
468  u16 ixgbe_calc_eeprom_checksum_X540(struct ixgbe_hw *hw)
469  {
470      u16 i;
471      u16 j;
472      u16 checksum = 0;
473      u16 length = 0;
474      u16 pointer = 0;
475      u16 word = 0;
476
477      /*
478       * Do not use hw->eeprom.ops.read because we do not want to take
479       * the synchronization semaphores here. Instead use
480       * ixgbe_read_eerd_generic
481       */
482
483      DEBUGFUNC("ixgbe_calc_eeprom_checksum_X540");
484
485      /* Include 0x0-0x3F in the checksum */
486      for (i = 0; i < IXGBE_EEPROM_CHECKSUM; i++) {
487          if (ixgbe_read_eerd_generic(hw, i, &word) != IXGBE_SUCCESS) {
488              DEBUGOUT("EEPROM read failed\n");
489              break;
490          }
491          checksum += word;
492      }
493
494      /*
495       * Include all data from pointers 0x3, 0x6-0xE. This excludes the
496       * FW, PHY module, and PCIe Expansion/Option ROM pointers.
497       */
498      for (i = IXGBE_PCIE_ANALOG_PTR; i < IXGBE_FW_PTR; i++) {
499          if (i == IXGBE_PHY_PTR || i == IXGBE_OPTION_ROM_PTR)
500              continue;
501
502          if (ixgbe_read_eerd_generic(hw, i, &pointer) != IXGBE_SUCCESS) {
503              DEBUGOUT("EEPROM read failed\n");
504              break;
505          }
506
507          /* Skip pointer section if the pointer is invalid. */
508          if (pointer == 0xFFFF || pointer == 0 ||
509              pointer >= hw->eeprom.word_size)
510              continue;
511
512          if (ixgbe_read_eerd_generic(hw, pointer, &length) !=
513              IXGBE_SUCCESS) {
514              DEBUGOUT("EEPROM read failed\n");
515              break;
516          }
517
518          /* Skip pointer section if length is invalid. */
519          if (length == 0xFFFF || length == 0 ||
520              (pointer + length) >= hw->eeprom.word_size)
521              continue;
522
523          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 EEW 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;

922     DEBUGFUNC("ixgbe_blink_led_start_X540");

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);

942     return IXGBE_SUCCESS;
943 }

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 static ixgbe_blink_led_stop_X540(struct ixgbe_hw *hw, u32 index)
954 {
955     u32 macc_reg;
956     u32 ledctl_reg;

958     DEBUGFUNC("ixgbe_blink_led_stop_X540");

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);

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);

973     return IXGBE_SUCCESS;
974 }

```

new/usr/src/uts/common/io/ixgbe/ixgbe_x540.h

1

```
*****
3172 Fri Jul 20 10:59:44 2012
new/usr/src/uts/common/io/ixgbe/ixgbe_x540.h
XXX Intel X540 support
*****
1 /*****
3 Copyright (c) 2001-2012, Intel Corporation
4 All rights reserved.
5
6 Redistribution and use in source and binary forms, with or without
7 modification, are permitted provided that the following conditions are met:
8
9 1. Redistributions of source code must retain the above copyright notice,
10 this list of conditions and the following disclaimer.
11
12 2. Redistributions in binary form must reproduce the above copyright
13 notice, this list of conditions and the following disclaimer in the
14 documentation and/or other materials provided with the distribution.
15
16 3. Neither the name of the Intel Corporation nor the names of its
17 contributors may be used to endorse or promote products derived from
18 this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
21 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
23 ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
24 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
30 POSSIBILITY OF SUCH DAMAGE.
31
32 *****/
33 /*$FreeBSD$*/
34
35 #ifndef _IXGBE_X540_H_
36 #define _IXGBE_X540_H_
37
38 #include "ixgbe_type.h"
39
40 s32 ixgbe_get_link_capabilities_X540(struct ixgbe_hw *hw,
41                                     ixgbe_link_speed *speed, bool *autoneg);
42 enum ixgbe_media_type ixgbe_get_media_type_X540(struct ixgbe_hw *hw);
43 s32 ixgbe_setup_mac_link_X540(struct ixgbe_hw *hw, ixgbe_link_speed speed,
44                               bool autoneg, bool link_up_wait_to_complete);
45 s32 ixgbe_reset_hw_X540(struct ixgbe_hw *hw);
46 s32 ixgbe_start_hw_X540(struct ixgbe_hw *hw);
47 u32 ixgbe_get_supported_physical_layer_X540(struct ixgbe_hw *hw);
48
49 s32 ixgbe_init_eeprom_params_X540(struct ixgbe_hw *hw);
50 s32 ixgbe_read_eerd_X540(struct ixgbe_hw *hw, u16 offset, u16 *data);
51 s32 ixgbe_read_eerd_buffer_X540(struct ixgbe_hw *hw, u16 offset, u16 words,
52                                 u16 *data);
53 s32 ixgbe_write_eewr_X540(struct ixgbe_hw *hw, u16 offset, u16 data);
54 s32 ixgbe_write_eewr_buffer_X540(struct ixgbe_hw *hw, u16 offset, u16 words,
55                                 u16 *data);
56 s32 ixgbe_update_eeprom_checksum_X540(struct ixgbe_hw *hw);
57 s32 ixgbe_validate_eeprom_checksum_X540(struct ixgbe_hw *hw, u16 *checksum_val);
58 u16 ixgbe_calc_eeprom_checksum_X540(struct ixgbe_hw *hw);
59
60 s32 ixgbe_acquire_swfw_sync_X540(struct ixgbe_hw *hw, u16 mask);
61 void ixgbe_release_swfw_sync_X540(struct ixgbe_hw *hw, u16 mask);
```

new/usr/src/uts/common/io/ixgbe/ixgbe_x540.h

2

```
63 s32 ixgbe_blink_led_start_X540(struct ixgbe_hw *hw, u32 index);
64 s32 ixgbe_blink_led_stop_X540(struct ixgbe_hw *hw, u32 index);
65 #endif /* _IXGBE_X540_H_ */
```