```
**********************************************************
   40996 Tue Apr 30 09:54:21 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_82598.c
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
```

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

```
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

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

static s32 ixgbe_get_link_capabilities_82598(struct ixgbe_hw *hw,
                                             ixgbe_link_speed *speed,
                                             bool *autoneg);
static enum ixgbe_media_type ixgbe_get_media_type_82598(struct ixgbe_hw *hw);
static s32 ixgbe_start_mac_link_82598(struct ixgbe_hw *hw,
                                      bool autoneg_wait_to_complete);
static s32 ixgbe_check_mac_link_82598(struct ixgbe_hw *hw,
                                      ixgbe_link_speed *speed, bool *link_up,
                                      bool link_up_wait_to_complete);
static s32 ixgbe_setup_mac_link_82598(struct ixgbe_hw *hw,
                                      ixgbe_link_speed speed,
                                      bool autoneg,
                                      bool autoneg_wait_to_complete);
static s32 ixgbe_setup_copper_link_82598(struct ixgbe_hw *hw,
                                         ixgbe_link_speed speed,
                                         bool autoneg,
                                         bool autoneg_wait_to_complete);
static s32 ixgbe_reset_hw_82598(struct ixgbe_hw *hw);
static s32 ixgbe_clear_vmdq_82598(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
static s32 ixgbe_clear_vfta_82598(struct ixgbe_hw *hw);
```



```
static void ixgbe_set_rxpba_82598(struct ixgbe_hw *hw, int num_pb,
                                  u32 headroom, int strategy);
static s32 ixgbe_read_i2c_sff8472_82598(struct ixgbe_hw *hw, u8 byte_offset,
                                        u8 *sff8472_data);

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

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

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

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

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

        DEBUGFUNC("ixgbe_init_ops_82598");

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

        /* PHY */
```

```
124         phy->ops.init = &ixgbe_init_phy_ops_82598;
125
126         /* MAC */
127         mac->ops.start_hw = &ixgbe_start_hw_82598;
128         mac->ops.enable_relaxed_ordering = &ixgbe_enable_relaxed_ordering_82598;
129         mac->ops.reset_hw = &ixgbe_reset_hw_82598;
130         mac->ops.get_media_type = &ixgbe_get_media_type_82598;
131         mac->ops.get_supported_physical_layer =
132                                 &ixgbe_get_supported_physical_layer_82598;
133         mac->ops.read_analog_reg8 = &ixgbe_read_analog_reg8_82598;
134         mac->ops.write_analog_reg8 = &ixgbe_write_analog_reg8_82598;
135         mac->ops.set_lan_id = &ixgbe_set_lan_id_multi_port_pcie_82598;
136
137         /* RAR, Multicast, VLAN */
138         mac->ops.set_vmdq = &ixgbe_set_vmdq_82598;
139         mac->ops.clear_vmdq = &ixgbe_clear_vmdq_82598;
140         mac->ops.set_vfta = &ixgbe_set_vfta_82598;
141         mac->ops.set_vlvf = NULL;
142         mac->ops.clear_vfta = &ixgbe_clear_vfta_82598;
143
144         /* Flow Control */
145         mac->ops.fc_enable = &ixgbe_fc_enable_82598;
146
147         mac->mcft_size          = 128;
148         mac->vft_size           = 128;
149         mac->num_rar_entries    = 16;
150         mac->rx_pb_size         = 512;
151         mac->max_tx_queues      = 32;
152         mac->max_rx_queues      = 64;
153         mac->max_msix_vectors   = ixgbe_get_pcie_msix_count_generic(hw);
154
155         /* SFP+ Module */
156         phy->ops.read_i2c_eeprom = &ixgbe_read_i2c_eeprom_82598;
157         phy->ops.read_i2c_sff8472 = &ixgbe_read_i2c_sff8472_82598;
158
159         /* Link */
160         mac->ops.check_link = &ixgbe_check_mac_link_82598;
161         mac->ops.setup_link = &ixgbe_setup_mac_link_82598;
162         mac->ops.flap_tx_laser = NULL;
163         mac->ops.get_link_capabilities = &ixgbe_get_link_capabilities_82598;
164         mac->ops.setup_rxpba = &ixgbe_set_rxpba_82598;
165
166         /* Manageability interface */
167         mac->ops.set_fw_drv_ver = NULL;
168
169         return ret_val;
170 }
```

*unchanged_portion_omitted_*

```
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,
720                                       ixgbe_link_speed speed, bool autoneg,
721                                       bool autoneg_wait_to_complete)
722 {
723         bool autoneg = FALSE;
724         s32 status;
725         ixgbe_link_speed link_capabilities = IXGBE_LINK_SPEED_UNKNOWN;
```



```
726         u32 curr_autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
727         u32 autoc = curr_autoc;
728         u32 link_mode = autoc & IXGBE_AUTOC_LMS_MASK;
729
730         DEBUGFUNC("ixgbe_setup_mac_link_82598");
731
732         /* Check to see if speed passed in is supported. */
733         status = ixgbe_get_link_capabilities(hw, &link_capabilities, &autoneg);
734         if (status != IXGBE_SUCCESS)
735                 return (status);
736         speed &= link_capabilities;
737
738         if (speed == IXGBE_LINK_SPEED_UNKNOWN)
739                 status = IXGBE_ERR_LINK_SETUP;
740
741         /* Set KX4/KX support according to speed requested */
742         else if (link_mode == IXGBE_AUTOC_LMS_KX4_AN ||
743                  link_mode == IXGBE_AUTOC_LMS_KX4_AN_1G_AN) {
744                 autoc &= ~IXGBE_AUTOC_KX4_KX_SUPP_MASK;
745                 if (speed & IXGBE_LINK_SPEED_10GB_FULL)
746                         autoc |= IXGBE_AUTOC_KX4_SUPP;
747                 if (speed & IXGBE_LINK_SPEED_1GB_FULL)
748                         autoc |= IXGBE_AUTOC_KX_SUPP;
749                 if (autoc != curr_autoc)
750                         IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc);
751         }
752
753         if (status == IXGBE_SUCCESS) {
754                 /*
755                  * Setup and restart the link based on the new values in
756                  * ixgbe_hw This will write the AUTOC register based on the new
757                  * stored values
758                  */
759                 status = ixgbe_start_mac_link_82598(hw,
760                                                     autoneg_wait_to_complete);
761         }
762
763         return status;
764 }
765
766
767 /**
768  *  ixgbe_setup_copper_link_82598 - Set the PHY autoneg advertised field
769  *  @hw: pointer to hardware structure
770  *  @speed: new link speed
770  *  @autoneg: TRUE if autonegotiation enabled
771  *  @autoneg_wait_to_complete: TRUE if waiting is needed to complete
772  *
773  *  Sets the link speed in the AUTOC register in the MAC and restarts link.
774  **/
775 static s32 ixgbe_setup_copper_link_82598(struct ixgbe_hw *hw,
776                                          ixgbe_link_speed speed,
777                                          bool autoneg,
777                                          bool autoneg_wait_to_complete)
778 {
779         s32 status;
780
781         DEBUGFUNC("ixgbe_setup_copper_link_82598");
782
783         /* Setup the PHY according to input speed */
784         status = hw->phy.ops.setup_link_speed(hw, speed,
785         status = hw->phy.ops.setup_link_speed(hw, speed, autoneg,
785                                               autoneg_wait_to_complete);
786         if (status == IXGBE_SUCCESS) {
787                 /* Set up MAC */
788                 status =
```

```
 789                     ixgbe_start_mac_link_82598(hw, autoneg_wait_to_complete);
 790         }

 792         return status;
 793 }
_____unchanged_portion_omitted_

1107 /**
1108  *  ixgbe_read_i2c_phy_82598 - Reads 8 bit word over I2C interface.
1109  *  ixgbe_read_i2c_eeprom_82598 - Reads 8 bit word over I2C interface.
1109  *  @hw: pointer to hardware structure
1110  *  @dev_addr: address to read from
1111  *  @byte_offset: byte offset to read from dev_addr
1111  *  @byte_offset: EEPROM byte offset to read
1112  *  @eeprom_data: value read
1113  *
1114  *  Performs 8 byte read operation to SFP module's EEPROM over I2C interface.
1115  **/
1116 static s32 ixgbe_read_i2c_phy_82598(struct ixgbe_hw *hw, u8 dev_addr,
1117                                     u8 byte_offset, u8 *eeprom_data)
1116 s32 ixgbe_read_i2c_eeprom_82598(struct ixgbe_hw *hw, u8 byte_offset,
1117                                 u8 *eeprom_data)
1118 {
1119         s32 status = IXGBE_SUCCESS;
1120         u16 sfp_addr = 0;
1121         u16 sfp_data = 0;
1122         u16 sfp_stat = 0;
1123         u32 i;

1125         DEBUGFUNC("ixgbe_read_i2c_phy_82598");
1125         DEBUGFUNC("ixgbe_read_i2c_eeprom_82598");

1127         if (hw->phy.type == ixgbe_phy_nl) {
1128                 /*
1129                  * NetLogic phy SDA/SCL registers are at addresses 0xC30A to
1130                  * 0xC30D. These registers are used to talk to the SFP+
1131                  * module's EEPROM through the SDA/SCL (I2C) interface.
1132                  */
1133                 sfp_addr = (dev_addr << 8) + byte_offset;
1133                 sfp_addr = (IXGBE_I2C_EEPROM_DEV_ADDR << 8) + byte_offset;
1134                 sfp_addr = (sfp_addr | IXGBE_I2C_EEPROM_READ_MASK);
1135                 hw->phy.ops.write_reg(hw,
1136                                       IXGBE_MDIO_PMA_PMD_SDA_SCL_ADDR,
1137                                       IXGBE_MDIO_PMA_PMD_DEV_TYPE,
1138                                       sfp_addr);

1140                 /* Poll status */
1141                 for (i = 0; i < 100; i++) {
1142                         hw->phy.ops.read_reg(hw,
1143                                              IXGBE_MDIO_PMA_PMD_SDA_SCL_STAT,
1144                                              IXGBE_MDIO_PMA_PMD_DEV_TYPE,
1145                                              &sfp_stat);
1146                         sfp_stat = sfp_stat & IXGBE_I2C_EEPROM_STATUS_MASK;
1147                         if (sfp_stat != IXGBE_I2C_EEPROM_STATUS_IN_PROGRESS)
1148                                 break;
1149                         msec_delay(10);
1150                 }

1152                 if (sfp_stat != IXGBE_I2C_EEPROM_STATUS_PASS) {
1153                         DEBUGOUT("EEPROM read did not pass.\n");
1154                         status = IXGBE_ERR_SFP_NOT_PRESENT;
1155                         goto out;
1156                 }

1158                 /* Read data */
1159                 hw->phy.ops.read_reg(hw, IXGBE_MDIO_PMA_PMD_SDA_SCL_DATA,
```



```
1160                                      IXGBE_MDIO_PMA_PMD_DEV_TYPE, &sfp_data);

1162                 *eeprom_data = (u8)(sfp_data >> 8);
1163         } else {
1164                 status = IXGBE_ERR_PHY;
1165                 goto out;
1165         }

1167 out:
1168         return status;
1169 }

1171 /**
1172  *  ixgbe_read_i2c_eeprom_82598 - Reads 8 bit word over I2C interface.
1173  *  @hw: pointer to hardware structure
1174  *  @byte_offset: EEPROM byte offset to read
1175  *  @eeprom_data: value read
1176  *
1177  *  Performs 8 byte read operation to SFP module's EEPROM over I2C interface.
1178  **/
1179 s32 ixgbe_read_i2c_eeprom_82598(struct ixgbe_hw *hw, u8 byte_offset,
1180                                 u8 *eeprom_data)
1181 {
1182         return ixgbe_read_i2c_phy_82598(hw, IXGBE_I2C_EEPROM_DEV_ADDR,
1183                                         byte_offset, eeprom_data);
1184 }

1186 /**
1187  *  ixgbe_read_i2c_sff8472_82598 - Reads 8 bit word over I2C interface.
1188  *  @hw: pointer to hardware structure
1189  *  @byte_offset: byte offset at address 0xA2
1190  *  @eeprom_data: value read
1191  *
1192  *  Performs 8 byte read operation to SFP module's SFF-8472 data over I2C
1193  **/
1194 static s32 ixgbe_read_i2c_sff8472_82598(struct ixgbe_hw *hw, u8 byte_offset,
1195                                         u8 *sff8472_data)
1196 {
1197         return ixgbe_read_i2c_phy_82598(hw, IXGBE_I2C_EEPROM_DEV_ADDR2,
1198                                         byte_offset, sff8472_data);
1199 }

1201 /**
1202  *  ixgbe_get_supported_physical_layer_82598 - Returns physical layer type
1203  *  @hw: pointer to hardware structure
1204  *
1205  *  Determines physical layer capabilities of the current configuration.
1206  **/
1207 u32 ixgbe_get_supported_physical_layer_82598(struct ixgbe_hw *hw)
1208 {
1209         u32 physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
1210         u32 autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
1211         u32 pma_pmd_10g = autoc & IXGBE_AUTOC_10G_PMA_PMD_MASK;
1212         u32 pma_pmd_1g = autoc & IXGBE_AUTOC_1G_PMA_PMD_MASK;
1213         u16 ext_ability = 0;

1215         DEBUGFUNC("ixgbe_get_supported_physical_layer_82598");

1217         hw->phy.ops.identify(hw);

1219         /* Copper PHY must be checked before AUTOC LMS to determine correct
1220          * physical layer because 10GBase-T PHYs use LMS = KX4/KX */
1221         switch (hw->phy.type) {
1222         case ixgbe_phy_tn:
1223         case ixgbe_phy_cu_unknown:
1224                 hw->phy.ops.read_reg(hw, IXGBE_MDIO_PHY_EXT_ABILITY,
```

```
1225                IXGBE_MDIO_PMA_PMD_DEV_TYPE, &ext_ability);
1226                if (ext_ability & IXGBE_MDIO_PHY_10GBASET_ABILITY)
1227                        physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_T;
1228                if (ext_ability & IXGBE_MDIO_PHY_1000BASET_ABILITY)
1229                        physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_T;
1230                if (ext_ability & IXGBE_MDIO_PHY_100BASETX_ABILITY)
1231                        physical_layer |= IXGBE_PHYSICAL_LAYER_100BASE_TX;
1232                goto out;
1233        default:
1234                break;
1235        }

1237        switch (autoc & IXGBE_AUTOC_LMS_MASK) {
1238        case IXGBE_AUTOC_LMS_1G_AN:
1239        case IXGBE_AUTOC_LMS_1G_LINK_NO_AN:
1240                if (pma_pmd_1g == IXGBE_AUTOC_1G_KX)
1241                        physical_layer = IXGBE_PHYSICAL_LAYER_1000BASE_KX;
1242                else
1243                        physical_layer = IXGBE_PHYSICAL_LAYER_1000BASE_BX;
1244                break;
1245        case IXGBE_AUTOC_LMS_10G_LINK_NO_AN:
1246                if (pma_pmd_10g == IXGBE_AUTOC_10G_CX4)
1247                        physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_CX4;
1248                else if (pma_pmd_10g == IXGBE_AUTOC_10G_KX4)
1249                        physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_KX4;
1250                else /* XAUI */
1251                        physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
1252                break;
1253        case IXGBE_AUTOC_LMS_KX4_AN:
1254        case IXGBE_AUTOC_LMS_KX4_AN_1G_AN:
1255                if (autoc & IXGBE_AUTOC_KX_SUPP)
1256                        physical_layer |= IXGBE_PHYSICAL_LAYER_1000BASE_KX;
1257                if (autoc & IXGBE_AUTOC_KX4_SUPP)
1258                        physical_layer |= IXGBE_PHYSICAL_LAYER_10GBASE_KX4;
1259                break;
1260        default:
1261                break;
1262        }

1264        if (hw->phy.type == ixgbe_phy_nl) {
1265                hw->phy.ops.identify_sfp(hw);

1267                switch (hw->phy.sfp_type) {
1268                case ixgbe_sfp_type_da_cu:
1269                        physical_layer = IXGBE_PHYSICAL_LAYER_SFP_PLUS_CU;
1270                        break;
1271                case ixgbe_sfp_type_sr:
1272                        physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_SR;
1273                        break;
1274                case ixgbe_sfp_type_lr:
1275                        physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_LR;
1276                        break;
1277                default:
1278                        physical_layer = IXGBE_PHYSICAL_LAYER_UNKNOWN;
1279                        break;
1280                }
1281        }

1283        switch (hw->device_id) {
1284        case IXGBE_DEV_ID_82598_DA_DUAL_PORT:
1285                physical_layer = IXGBE_PHYSICAL_LAYER_SFP_PLUS_CU;
1286                break;
1287        case IXGBE_DEV_ID_82598AF_DUAL_PORT:
1288        case IXGBE_DEV_ID_82598AF_SINGLE_PORT:
1289        case IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM:
1290                physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_SR;
```



```
1291                break;
1292        case IXGBE_DEV_ID_82598EB_XF_LR:
1293                physical_layer = IXGBE_PHYSICAL_LAYER_10GBASE_LR;
1294                break;
1295        default:
1296                break;
1297        }

1299 out:
1300        return physical_layer;
1301 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   75299 Tue Apr 30 09:54:22 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_82599.c
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
```

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

```
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

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

static s32 ixgbe_setup_copper_link_82599(struct ixgbe_hw *hw,
                                         ixgbe_link_speed speed,
                                         bool autoneg,
                                         bool autoneg_wait_to_complete);
static s32 ixgbe_verify_fw_version_82599(struct ixgbe_hw *hw);
static s32 ixgbe_read_eeprom_82599(struct ixgbe_hw *hw,
                                   u16 offset, u16 *data);
static s32 ixgbe_read_eeprom_buffer_82599(struct ixgbe_hw *hw, u16 offset,
                                          u16 words, u16 *data);

static bool ixgbe_mng_enabled(struct ixgbe_hw *hw)
{
        u32 fwsm, manc, factps;

        fwsm = IXGBE_READ_REG(hw, IXGBE_FWSM);
        if ((fwsm & IXGBE_FWSM_MODE_MASK) != IXGBE_FWSM_FW_MODE_PT)
                return FALSE;

        manc = IXGBE_READ_REG(hw, IXGBE_MANC);
        if (!(manc & IXGBE_MANC_RCV_TCO_EN))
```



```
                return FALSE;

        factps = IXGBE_READ_REG(hw, IXGBE_FACTPS);
        if (factps & IXGBE_FACTPS_MNGCG)
                return FALSE;

        return TRUE;
}

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

        DEBUGFUNC("ixgbe_init_mac_link_ops_82599");

        /*
         * enable the laser control functions for SFP+ fiber
         * and MNG not enabled
         */
        if ((mac->ops.get_media_type(hw) == ixgbe_media_type_fiber) &&
            !(ixgbe_mng_enabled(hw))) {
        /* enable the laser control functions for SFP+ fiber */
        if (mac->ops.get_media_type(hw) == ixgbe_media_type_fiber) {
                mac->ops.disable_tx_laser =
                                       &ixgbe_disable_tx_laser_multispeed_fiber;
                mac->ops.enable_tx_laser =
                                        &ixgbe_enable_tx_laser_multispeed_fiber;
                mac->ops.flap_tx_laser = &ixgbe_flap_tx_laser_multispeed_fiber;

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

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

s32 ixgbe_setup_sfp_modules_82599(struct ixgbe_hw *hw)
{
        s32 ret_val = IXGBE_SUCCESS;
        u32 reg_anlp1 = 0;
        u32 i = 0;
        u16 list_offset, data_offset, data_value;
        bool got_lock = FALSE;

        DEBUGFUNC("ixgbe_setup_sfp_modules_82599");

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

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

```
170                 ret_val = ixgbe_get_sfp_init_sequence_offsets(hw, &list_offset,
171                                                               &data_offset);
172                 if (ret_val != IXGBE_SUCCESS)
173                         goto setup_sfp_out;
174
175                 /* PHY config will finish before releasing the semaphore */
176                 ret_val = hw->mac.ops.acquire_swfw_sync(hw,
177                                                         IXGBE_GSSR_MAC_CSR_SM);
178                 if (ret_val != IXGBE_SUCCESS) {
179                         ret_val = IXGBE_ERR_SWFW_SYNC;
180                         goto setup_sfp_out;
181                 }
182
183                 hw->eeprom.ops.read(hw, ++data_offset, &data_value);
184                 while (data_value != 0xffff) {
185                         IXGBE_WRITE_REG(hw, IXGBE_CORECTL, data_value);
186                         IXGBE_WRITE_FLUSH(hw);
187                         hw->eeprom.ops.read(hw, ++data_offset, &data_value);
188                 }
189
190                 /* Release the semaphore */
191                 hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_MAC_CSR_SM);
192                 /* Delay obtaining semaphore again to allow FW access */
193                 msec_delay(hw->eeprom.semaphore_delay);
194
195                 /* Need SW/FW semaphore around AUTOC writes if LESM on,
196                  * likewise reset_pipeline requires lock as it also writes
197                  * AUTOC.
198                  */
199                 if (ixgbe_verify_lesm_fw_enabled_82599(hw)) {
200                         ret_val = hw->mac.ops.acquire_swfw_sync(hw,
201                                                                 IXGBE_GSSR_MAC_CSR_SM);
202                         if (ret_val != IXGBE_SUCCESS) {
203                                 ret_val = IXGBE_ERR_SWFW_SYNC;
204                                 goto setup_sfp_out;
205                         }
174                 /* Now restart DSP by setting Restart_AN and clearing LMS */
175                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, ((IXGBE_READ_REG(hw,
176                                 IXGBE_AUTOC) & ~IXGBE_AUTOC_LMS_MASK) |
177                                 IXGBE_AUTOC_AN_RESTART));
178
207                         got_lock = TRUE;
179                 /* Wait for AN to leave state 0 */
180                 for (i = 0; i < 10; i++) {
181                         msec_delay(4);
182                         reg_anlp1 = IXGBE_READ_REG(hw, IXGBE_ANLP1);
183                         if (reg_anlp1 & IXGBE_ANLP1_AN_STATE_MASK)
184                                 break;
185                 }
208
210                 /* Restart DSP and set SFI mode */
211                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, ((hw->mac.orig_autoc) |
212                                 IXGBE_AUTOC_LMS_10G_SERIAL));
213                 hw->mac.cached_autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
214                 ret_val = ixgbe_reset_pipeline_82599(hw);
215
216                 if (got_lock) {
217                         hw->mac.ops.release_swfw_sync(hw,
218                                                       IXGBE_GSSR_MAC_CSR_SM);
219                         got_lock = FALSE;
220                 }
221
222                 if (ret_val) {
186                 if (!(reg_anlp1 & IXGBE_ANLP1_AN_STATE_MASK)) {
223                         DEBUGOUT("sfp module setup not complete\n");
224                         ret_val = IXGBE_ERR_SFP_SETUP_NOT_COMPLETE;
```



```
225                         goto setup_sfp_out;
226                 }
227
192                 /* Restart DSP by setting Restart_AN and return to SFI mode */
193                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, (IXGBE_READ_REG(hw,
194                                 IXGBE_AUTOC) | IXGBE_AUTOC_LMS_10G_SERIAL |
195                                 IXGBE_AUTOC_AN_RESTART));
228         }
229
230 setup_sfp_out:
231         return ret_val;
232 }
233
234 /**
235  *  ixgbe_init_ops_82599 - Inits func ptrs and MAC type
236  *  @hw: pointer to hardware structure
237  *
238  *  Initialize the function pointers and assign the MAC type for 82599.
239  *  Does not touch the hardware.
240  **/
241
242 s32 ixgbe_init_ops_82599(struct ixgbe_hw *hw)
243 {
244         struct ixgbe_mac_info *mac = &hw->mac;
245         struct ixgbe_phy_info *phy = &hw->phy;
246         struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
247         s32 ret_val;
248
249         DEBUGFUNC("ixgbe_init_ops_82599");
250
251         (void) ixgbe_init_phy_ops_generic(hw);
219         ret_val = ixgbe_init_phy_ops_generic(hw);
252         ret_val = ixgbe_init_ops_generic(hw);
253
254         /* PHY */
255         phy->ops.identify = &ixgbe_identify_phy_82599;
256         phy->ops.init = &ixgbe_init_phy_ops_82599;
257
258         /* MAC */
259         mac->ops.reset_hw = &ixgbe_reset_hw_82599;
260         mac->ops.enable_relaxed_ordering = &ixgbe_enable_relaxed_ordering_gen2;
261         mac->ops.get_media_type = &ixgbe_get_media_type_82599;
262         mac->ops.get_supported_physical_layer =
263                                     &ixgbe_get_supported_physical_layer_82599;
264         mac->ops.disable_sec_rx_path = &ixgbe_disable_sec_rx_path_generic;
265         mac->ops.enable_sec_rx_path = &ixgbe_enable_sec_rx_path_generic;
266         mac->ops.enable_rx_dma = &ixgbe_enable_rx_dma_82599;
267         mac->ops.read_analog_reg8 = &ixgbe_read_analog_reg8_82599;
268         mac->ops.write_analog_reg8 = &ixgbe_write_analog_reg8_82599;
269         mac->ops.start_hw = &ixgbe_start_hw_82599;
270         mac->ops.get_san_mac_addr = &ixgbe_get_san_mac_addr_generic;
271         mac->ops.set_san_mac_addr = &ixgbe_set_san_mac_addr_generic;
272         mac->ops.get_device_caps = &ixgbe_get_device_caps_generic;
273         mac->ops.get_wwn_prefix = &ixgbe_get_wwn_prefix_generic;
274         mac->ops.get_fcoe_boot_status = &ixgbe_get_fcoe_boot_status_generic;
275
276         /* RAR, Multicast, VLAN */
277         mac->ops.set_vmdq = &ixgbe_set_vmdq_generic;
278         mac->ops.set_vmdq_san_mac = &ixgbe_set_vmdq_san_mac_generic;
279         mac->ops.clear_vmdq = &ixgbe_clear_vmdq_generic;
280         mac->ops.insert_mac_addr = &ixgbe_insert_mac_addr_generic;
281         mac->rar_highwater = 1;
282         mac->ops.set_vfta = &ixgbe_set_vfta_generic;
283         mac->ops.set_vlvf = &ixgbe_set_vlvf_generic;
284         mac->ops.clear_vfta = &ixgbe_clear_vfta_generic;
285         mac->ops.init_uta_tables = &ixgbe_init_uta_tables_generic;
```

```
286         mac->ops.setup_sfp = &ixgbe_setup_sfp_modules_82599;
287         mac->ops.set_mac_anti_spoofing = &ixgbe_set_mac_anti_spoofing;
288         mac->ops.set_vlan_anti_spoofing = &ixgbe_set_vlan_anti_spoofing;

290         /* Link */
291         mac->ops.get_link_capabilities = &ixgbe_get_link_capabilities_82599;
292         mac->ops.check_link = &ixgbe_check_mac_link_generic;
293         mac->ops.setup_rxpba = &ixgbe_set_rxpba_generic;
294         ixgbe_init_mac_link_ops_82599(hw);

296         mac->mcft_size          = 128;
297         mac->vft_size           = 128;
298         mac->num_rar_entries    = 128;
299         mac->rx_pb_size         = 512;
300         mac->max_tx_queues      = 128;
301         mac->max_rx_queues      = 128;
302         mac->max_msix_vectors   = ixgbe_get_pcie_msix_count_generic(hw);

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

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

309         /* EEPROM */
310         eeprom->ops.read = &ixgbe_read_eeprom_82599;
311         eeprom->ops.read_buffer = &ixgbe_read_eeprom_buffer_82599;

313         /* Manageability interface */
314         mac->ops.set_fw_drv_ver = &ixgbe_set_fw_drv_ver_generic;


317         return ret_val;
318 }

320 /**
321  *  ixgbe_get_link_capabilities_82599 - Determines link capabilities
322  *  @hw: pointer to hardware structure
323  *  @speed: pointer to link speed
324  *  @autoneg: TRUE when autoneg or autotry is enabled
292  *  @negotiation: TRUE when autoneg or autotry is enabled
325  *
326  *  Determines the link capabilities by reading the AUTOC register.
327  **/
328 s32 ixgbe_get_link_capabilities_82599(struct ixgbe_hw *hw,
329                                       ixgbe_link_speed *speed,
330                                       bool *autoneg)
298                                       bool *negotiation)
331 {
332         s32 status = IXGBE_SUCCESS;
333         u32 autoc = 0;

335         DEBUGFUNC("ixgbe_get_link_capabilities_82599");


338         /* Check if 1G SFP module. */
339         if (hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0 ||
340             hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1 ||
341             hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core0 ||
342             hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core1) {
343                 *speed = IXGBE_LINK_SPEED_1GB_FULL;
344                 *autoneg = TRUE;
312                 *negotiation = TRUE;
345                 goto out;
346         }

348         /*
```



```
349          * Determine link capabilities based on the stored value of AUTOC,
350          * which represents EEPROM defaults.  If AUTOC value has not
351          * been stored, use the current register values.
352          */
353         if (hw->mac.orig_link_settings_stored)
354                 autoc = hw->mac.orig_autoc;
355         else
356                 autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);

358         switch (autoc & IXGBE_AUTOC_LMS_MASK) {
359         case IXGBE_AUTOC_LMS_1G_LINK_NO_AN:
360                 *speed = IXGBE_LINK_SPEED_1GB_FULL;
361                 *autoneg = FALSE;
329                 *negotiation = FALSE;
362                 break;

364         case IXGBE_AUTOC_LMS_10G_LINK_NO_AN:
365                 *speed = IXGBE_LINK_SPEED_10GB_FULL;
366                 *autoneg = FALSE;
334                 *negotiation = FALSE;
367                 break;

369         case IXGBE_AUTOC_LMS_1G_AN:
370                 *speed = IXGBE_LINK_SPEED_1GB_FULL;
371                 *autoneg = TRUE;
339                 *negotiation = TRUE;
372                 break;

374         case IXGBE_AUTOC_LMS_10G_SERIAL:
375                 *speed = IXGBE_LINK_SPEED_10GB_FULL;
376                 *autoneg = FALSE;
344                 *negotiation = FALSE;
377                 break;

379         case IXGBE_AUTOC_LMS_KX4_KX_KR:
380         case IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN:
381                 *speed = IXGBE_LINK_SPEED_UNKNOWN;
382                 if (autoc & IXGBE_AUTOC_KR_SUPP)
383                         *speed |= IXGBE_LINK_SPEED_10GB_FULL;
384                 if (autoc & IXGBE_AUTOC_KX4_SUPP)
385                         *speed |= IXGBE_LINK_SPEED_10GB_FULL;
386                 if (autoc & IXGBE_AUTOC_KX_SUPP)
387                         *speed |= IXGBE_LINK_SPEED_1GB_FULL;
388                 *autoneg = TRUE;
356                 *negotiation = TRUE;
389                 break;

391         case IXGBE_AUTOC_LMS_KX4_KX_KR_SGMII:
392                 *speed = IXGBE_LINK_SPEED_100_FULL;
393                 if (autoc & IXGBE_AUTOC_KR_SUPP)
394                         *speed |= IXGBE_LINK_SPEED_10GB_FULL;
395                 if (autoc & IXGBE_AUTOC_KX4_SUPP)
396                         *speed |= IXGBE_LINK_SPEED_10GB_FULL;
397                 if (autoc & IXGBE_AUTOC_KX_SUPP)
398                         *speed |= IXGBE_LINK_SPEED_1GB_FULL;
399                 *autoneg = TRUE;
367                 *negotiation = TRUE;
400                 break;

402         case IXGBE_AUTOC_LMS_SGMII_1G_100M:
403                 *speed = IXGBE_LINK_SPEED_1GB_FULL | IXGBE_LINK_SPEED_100_FULL;
404                 *autoneg = FALSE;
372                 *negotiation = FALSE;
405                 break;

407         default:
```

```
408                 status = IXGBE_ERR_LINK_SETUP;
409                 goto out;
410         }

412         if (hw->phy.multispeed_fiber) {
413                 *speed |= IXGBE_LINK_SPEED_10GB_FULL |
414                           IXGBE_LINK_SPEED_1GB_FULL;
415                 *autoneg = TRUE;
383                 *negotiation = TRUE;
416         }

418 out:
419         return status;
420 }

422 /**
423  *  ixgbe_get_media_type_82599 - Get media type
424  *  @hw: pointer to hardware structure
425  *
426  *  Returns the media type (fiber, copper, backplane)
427  **/
428 enum ixgbe_media_type ixgbe_get_media_type_82599(struct ixgbe_hw *hw)
429 {
430         enum ixgbe_media_type media_type;

432         DEBUGFUNC("ixgbe_get_media_type_82599");

434         /* Detect if there is a copper PHY attached. */
435         switch (hw->phy.type) {
436         case ixgbe_phy_cu_unknown:
437         case ixgbe_phy_tn:
438                 media_type = ixgbe_media_type_copper;
439                 goto out;
440         default:
441                 break;
442         }

444         switch (hw->device_id) {
445         case IXGBE_DEV_ID_82599_KX4:
446         case IXGBE_DEV_ID_82599_KX4_MEZZ:
447         case IXGBE_DEV_ID_82599_COMBO_BACKPLANE:
448         case IXGBE_DEV_ID_82599_KR:
449         case IXGBE_DEV_ID_82599_BACKPLANE_FCOE:
450         case IXGBE_DEV_ID_82599_XAUI_LOM:
451                 /* Default device ID is mezzanine card KX/KX4 */
452                 media_type = ixgbe_media_type_backplane;
453                 break;
454         case IXGBE_DEV_ID_82599_SFP:
455         case IXGBE_DEV_ID_82599_SFP_FCOE:
456         case IXGBE_DEV_ID_82599_SFP_EM:
457         case IXGBE_DEV_ID_82599_SFP_SF2:
458         case IXGBE_DEV_ID_82599_SFP_SF_QP:
459         case IXGBE_DEV_ID_82599EN_SFP:
460                 media_type = ixgbe_media_type_fiber;
461                 break;
462         case IXGBE_DEV_ID_82599_CX4:
463                 media_type = ixgbe_media_type_cx4;
464                 break;
465         case IXGBE_DEV_ID_82599_T3_LOM:
466                 media_type = ixgbe_media_type_copper;
467                 break;
468         case IXGBE_DEV_ID_82599_BYPASS:
469                 media_type = ixgbe_media_type_fiber_fixed;
470                 hw->phy.multispeed_fiber = TRUE;
471                 break;
472         default:
```



```
473                 media_type = ixgbe_media_type_unknown;
474                 break;
475         }
476 out:
477         return media_type;
478 }

480 /**
481  *  ixgbe_start_mac_link_82599 - Setup MAC link settings
482  *  @hw: pointer to hardware structure
483  *  @autoneg_wait_to_complete: TRUE when waiting for completion is needed
484  *
485  *  Configures link settings based on values in the ixgbe_hw struct.
486  *  Restarts the link.  Performs autonegotiation if needed.
487  **/
488 s32 ixgbe_start_mac_link_82599(struct ixgbe_hw *hw,
489                                bool autoneg_wait_to_complete)
490 {
491         u32 autoc_reg;
492         u32 links_reg;
493         u32 i;
494         s32 status = IXGBE_SUCCESS;
495         bool got_lock = FALSE;

497         DEBUGFUNC("ixgbe_start_mac_link_82599");


500         /*  reset_pipeline requires us to hold this lock as it writes to
501          *  AUTOC.
502          */
503         if (ixgbe_verify_lesm_fw_enabled_82599(hw)) {
504                 status = hw->mac.ops.acquire_swfw_sync(hw,
505                                                        IXGBE_GSSR_MAC_CSR_SM);
506                 if (status != IXGBE_SUCCESS)
507                         goto out;

509                 got_lock = TRUE;
510         }

512         /* Restart link */
513         (void) ixgbe_reset_pipeline_82599(hw);
463         autoc_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC);
464         autoc_reg |= IXGBE_AUTOC_AN_RESTART;
465         IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc_reg);

515         if (got_lock)
516                 hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_MAC_CSR_SM);

518         /* Only poll for autoneg to complete if specified to do so */
519         if (autoneg_wait_to_complete) {
520                 autoc_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC);
521                 if ((autoc_reg & IXGBE_AUTOC_LMS_MASK) ==
522                     IXGBE_AUTOC_LMS_KX4_KX_KR ||
523                     (autoc_reg & IXGBE_AUTOC_LMS_MASK) ==
524                     IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN ||
525                     (autoc_reg & IXGBE_AUTOC_LMS_MASK) ==
526                     IXGBE_AUTOC_LMS_KX4_KX_KR_SGMII) {
527                         links_reg = 0; /* Just in case Autoneg time = 0 */
528                         for (i = 0; i < IXGBE_AUTO_NEG_TIME; i++) {
529                                 links_reg = IXGBE_READ_REG(hw, IXGBE_LINKS);
530                                 if (links_reg & IXGBE_LINKS_KX_AN_COMP)
531                                         break;
532                                 msec_delay(100);
533                         }
534                         if (!(links_reg & IXGBE_LINKS_KX_AN_COMP)) {
535                                 status = IXGBE_ERR_AUTONEG_NOT_COMPLETE;
```

```
536                                 DEBUGOUT("Autoneg did not complete.\n");
537                         }
538                 }
539         }

541         /* Add delay to filter out noises during initial link setup */
542         msec_delay(50);

544 out:
545         return status;
546 }
```

*unchanged_portion_omitted_*

```
609 /**
610  *  ixgbe_set_fiber_fixed_speed - Set module link speed for fixed fiber
611  *  @hw: pointer to hardware structure
612  *  @speed: link speed to set
613  *
614  *  We set the module speed differently for fixed fiber.  For other
615  *  multi-speed devices we don't have an error value so here if we
616  *  detect an error we just log it and exit.
617  */
618 static void ixgbe_set_fiber_fixed_speed(struct ixgbe_hw *hw,
619                                         ixgbe_link_speed speed)
620 {
621         s32 status;
622         u8 rs, eeprom_data;

624         switch (speed) {
625         case IXGBE_LINK_SPEED_10GB_FULL:
626                 /* one bit mask same as setting on */
627                 rs = IXGBE_SFF_SOFT_RS_SELECT_10G;
628                 break;
629         case IXGBE_LINK_SPEED_1GB_FULL:
630                 rs = IXGBE_SFF_SOFT_RS_SELECT_1G;
631                 break;
632         default:
633                 DEBUGOUT("Invalid fixed module speed\n");
634                 return;
635         }

637         /* Set RS0 */
638         status = hw->phy.ops.read_i2c_byte(hw, IXGBE_SFF_SFF_8472_OSCB,
639                                            IXGBE_I2C_EEPROM_DEV_ADDR2,
640                                            &eeprom_data);
641         if (status) {
642                 DEBUGOUT("Failed to read Rx Rate Select RS0\n");
643                 goto out;
644         }

646         eeprom_data = (eeprom_data & ~IXGBE_SFF_SOFT_RS_SELECT_MASK) & rs;

648         status = hw->phy.ops.write_i2c_byte(hw, IXGBE_SFF_SFF_8472_OSCB,
649                                             IXGBE_I2C_EEPROM_DEV_ADDR2,
650                                             eeprom_data);
651         if (status) {
652                 DEBUGOUT("Failed to write Rx Rate Select RS0\n");
653                 goto out;
654         }

656         /* Set RS1 */
657         status = hw->phy.ops.read_i2c_byte(hw, IXGBE_SFF_SFF_8472_ESCB,
658                                            IXGBE_I2C_EEPROM_DEV_ADDR2,
659                                            &eeprom_data);
660         if (status) {
661                 DEBUGOUT("Failed to read Rx Rate Select RS1\n");
```



```
662                 goto out;
663         }

665         eeprom_data = (eeprom_data & ~IXGBE_SFF_SOFT_RS_SELECT_MASK) & rs;

667         status = hw->phy.ops.write_i2c_byte(hw, IXGBE_SFF_SFF_8472_ESCB,
668                                             IXGBE_I2C_EEPROM_DEV_ADDR2,
669                                             eeprom_data);
670         if (status) {
671                 DEBUGOUT("Failed to write Rx Rate Select RS1\n");
672                 goto out;
673         }
674 out:
675         return;
676 }

678 /**
679  *  ixgbe_setup_mac_link_multispeed_fiber - Set MAC link speed
680  *  @hw: pointer to hardware structure
681  *  @speed: new link speed
560  *  @autoneg: TRUE if autonegotiation enabled
682  *  @autoneg_wait_to_complete: TRUE when waiting for completion is needed
683  *
684  *  Set the link speed in the AUTOC register and restarts link.
685  **/
686 s32 ixgbe_setup_mac_link_multispeed_fiber(struct ixgbe_hw *hw,
687                                           ixgbe_link_speed speed,
566                                           ixgbe_link_speed speed, bool autoneg,
688                                           bool autoneg_wait_to_complete)
689 {
690         s32 status = IXGBE_SUCCESS;
691         ixgbe_link_speed link_speed = IXGBE_LINK_SPEED_UNKNOWN;
692         ixgbe_link_speed highest_link_speed = IXGBE_LINK_SPEED_UNKNOWN;
693         u32 speedcnt = 0;
694         u32 esdp_reg = IXGBE_READ_REG(hw, IXGBE_ESDP);
695         u32 i = 0;
696         bool autoneg, link_up = FALSE;
575         bool link_up = FALSE;
576         bool negotiation;

698         DEBUGFUNC("ixgbe_setup_mac_link_multispeed_fiber");

700         /* Mask off requested but non-supported speeds */
701         status = ixgbe_get_link_capabilities(hw, &link_speed, &autoneg);
581         status = ixgbe_get_link_capabilities(hw, &link_speed, &negotiation);
702         if (status != IXGBE_SUCCESS)
703                 return status;

705         speed &= link_speed;

707         /*
708          * Try each speed one by one, highest priority first.  We do this in
709          * software because 10gb fiber doesn't support speed autonegotiation.
710          */
711         if (speed & IXGBE_LINK_SPEED_10GB_FULL) {
712                 speedcnt++;
713                 highest_link_speed = IXGBE_LINK_SPEED_10GB_FULL;

715                 /* If we already have link at this speed, just jump out */
716                 status = ixgbe_check_link(hw, &link_speed, &link_up, FALSE);
717                 if (status != IXGBE_SUCCESS)
718                         return status;

720                 if ((link_speed == IXGBE_LINK_SPEED_10GB_FULL) && link_up)
721                         goto out;
```

```
723                 /* Set the module link speed */
724                 if (hw->phy.media_type == ixgbe_media_type_fiber_fixed) {
725                         ixgbe_set_fiber_fixed_speed(hw,
726                                                     IXGBE_LINK_SPEED_10GB_FULL);
727                 } else {
728                         esdp_reg |= (IXGBE_ESDP_SDP5_DIR | IXGBE_ESDP_SDP5);
729                         IXGBE_WRITE_REG(hw, IXGBE_ESDP, esdp_reg);
730                         IXGBE_WRITE_FLUSH(hw);
731                 }

733                 /* Allow module to change analog characteristics (1G->10G) */
734                 msec_delay(40);

736                 status = ixgbe_setup_mac_link_82599(hw,
737                                                     IXGBE_LINK_SPEED_10GB_FULL,
613                                                     autoneg,
738                                                     autoneg_wait_to_complete);
739                 if (status != IXGBE_SUCCESS)
740                         return status;

742                 /* Flap the tx laser if it has not already been done */
743                 ixgbe_flap_tx_laser(hw);

745                 /*
746                  * Wait for the controller to acquire link.  Per IEEE 802.3ap,
747                  * Section 73.10.2, we may have to wait up to 500ms if KR is
748                  * attempted.  82599 uses the same timing for 10g SFI.
749                  */
750                 for (i = 0; i < 5; i++) {
751                         /* Wait for the link partner to also set speed */
752                         msec_delay(100);

754                         /* If we have link, just jump out */
755                         status = ixgbe_check_link(hw, &link_speed,
756                                                   &link_up, FALSE);
757                         if (status != IXGBE_SUCCESS)
758                                 return status;

760                         if (link_up)
761                                 goto out;
762                 }
763         }

765         if (speed & IXGBE_LINK_SPEED_1GB_FULL) {
766                 speedcnt++;
767                 if (highest_link_speed == IXGBE_LINK_SPEED_UNKNOWN)
768                         highest_link_speed = IXGBE_LINK_SPEED_1GB_FULL;

770                 /* If we already have link at this speed, just jump out */
771                 status = ixgbe_check_link(hw, &link_speed, &link_up, FALSE);
772                 if (status != IXGBE_SUCCESS)
773                         return status;

775                 if ((link_speed == IXGBE_LINK_SPEED_1GB_FULL) && link_up)
776                         goto out;

778                 /* Set the module link speed */
779                 if (hw->phy.media_type == ixgbe_media_type_fiber_fixed) {
780                         ixgbe_set_fiber_fixed_speed(hw,
781                                                     IXGBE_LINK_SPEED_1GB_FULL);
782                 } else {
783                         esdp_reg &= ~IXGBE_ESDP_SDP5;
784                         esdp_reg |= IXGBE_ESDP_SDP5_DIR;
785                         IXGBE_WRITE_REG(hw, IXGBE_ESDP, esdp_reg);
786                         IXGBE_WRITE_FLUSH(hw);
787                 }
```



```
789                 /* Allow module to change analog characteristics (10G->1G) */
790                 msec_delay(40);

792                 status = ixgbe_setup_mac_link_82599(hw,
793                                                     IXGBE_LINK_SPEED_1GB_FULL,
665                                                     autoneg,
794                                                     autoneg_wait_to_complete);
795                 if (status != IXGBE_SUCCESS)
796                         return status;

798                 /* Flap the tx laser if it has not already been done */
799                 ixgbe_flap_tx_laser(hw);

801                 /* Wait for the link partner to also set speed */
802                 msec_delay(100);

804                 /* If we have link, just jump out */
805                 status = ixgbe_check_link(hw, &link_speed, &link_up, FALSE);
806                 if (status != IXGBE_SUCCESS)
807                         return status;

809                 if (link_up)
810                         goto out;
811         }

813         /*
814          * We didn't get link.  Configure back to the highest speed we tried,
815          * (if there was more than one).  We call ourselves back with just the
816          * single highest speed that the user requested.
817          */
818         if (speedcnt > 1)
819                 status = ixgbe_setup_mac_link_multispeed_fiber(hw,
820                         highest_link_speed, autoneg_wait_to_complete);
692                         highest_link_speed, autoneg, autoneg_wait_to_complete);

822 out:
823         /* Set autoneg_advertised value based on input link speed */
824         hw->phy.autoneg_advertised = 0;

826         if (speed & IXGBE_LINK_SPEED_10GB_FULL)
827                 hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_10GB_FULL;

829         if (speed & IXGBE_LINK_SPEED_1GB_FULL)
830                 hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_1GB_FULL;

832         return status;
833 }

835 /**
836  *  ixgbe_setup_mac_link_smartspeed - Set MAC link speed using SmartSpeed
837  *  @hw: pointer to hardware structure
838  *  @speed: new link speed
711  *  @autoneg: TRUE if autonegotiation enabled
839  *  @autoneg_wait_to_complete: TRUE when waiting for completion is needed
840  *
841  *  Implements the Intel SmartSpeed algorithm.
842  **/
843 s32 ixgbe_setup_mac_link_smartspeed(struct ixgbe_hw *hw,
844                                     ixgbe_link_speed speed,
717                                     ixgbe_link_speed speed, bool autoneg,
845                                     bool autoneg_wait_to_complete)
846 {
847         s32 status = IXGBE_SUCCESS;
848         ixgbe_link_speed link_speed = IXGBE_LINK_SPEED_UNKNOWN;
849         s32 i, j;
```

```
850         bool link_up = FALSE;
851         u32 autoc_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC);
852
853         DEBUGFUNC("ixgbe_setup_mac_link_smartspeed");
854
855          /* Set autoneg_advertised value based on input link speed */
856         hw->phy.autoneg_advertised = 0;
857
858         if (speed & IXGBE_LINK_SPEED_10GB_FULL)
859                 hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_10GB_FULL;
860
861         if (speed & IXGBE_LINK_SPEED_1GB_FULL)
862                 hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_1GB_FULL;
863
864         if (speed & IXGBE_LINK_SPEED_100_FULL)
865                 hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_100_FULL;
866
867         /*
868          * Implement Intel SmartSpeed algorithm.  SmartSpeed will reduce the
869          * autoneg advertisement if link is unable to be established at the
870          * highest negotiated rate.  This can sometimes happen due to integrity
871          * issues with the physical media connection.
872          */
873
874         /* First, try to get link with full advertisement */
875         hw->phy.smart_speed_active = FALSE;
876         for (j = 0; j < IXGBE_SMARTSPEED_MAX_RETRIES; j++) {
877                 status = ixgbe_setup_mac_link_82599(hw, speed,
750                 status = ixgbe_setup_mac_link_82599(hw, speed, autoneg,
878                                                     autoneg_wait_to_complete);
879                 if (status != IXGBE_SUCCESS)
880                         goto out;
881
882                 /*
883                  * Wait for the controller to acquire link.  Per IEEE 802.3ap,
884                  * Section 73.10.2, we may have to wait up to 500ms if KR is
885                  * attempted, or 200ms if KX/KX4/BX/BX4 is attempted, per
886                  * Table 9 in the AN MAS.
887                  */
888                 for (i = 0; i < 5; i++) {
889                         msec_delay(100);
890
891                         /* If we have link, just jump out */
892                         status = ixgbe_check_link(hw, &link_speed, &link_up,
893                                                   FALSE);
894                         if (status != IXGBE_SUCCESS)
895                                 goto out;
896
897                         if (link_up)
898                                 goto out;
899                 }
900         }
901
902         /*
903          * We didn't get link.  If we advertised KR plus one of KX4/KX
904          * (or BX4/BX), then disable KR and try again.
905          */
906         if (((autoc_reg & IXGBE_AUTOC_KR_SUPP) == 0) ||
907             ((autoc_reg & IXGBE_AUTOC_KX4_KX_SUPP_MASK) == 0))
908                 goto out;
909
910         /* Turn SmartSpeed on to disable KR support */
911         hw->phy.smart_speed_active = TRUE;
912         status = ixgbe_setup_mac_link_82599(hw, speed,
785         status = ixgbe_setup_mac_link_82599(hw, speed, autoneg,
913                                             autoneg_wait_to_complete);
```



```
914         if (status != IXGBE_SUCCESS)
915                 goto out;
916
917         /*
918          * Wait for the controller to acquire link.  600ms will allow for
919          * the AN link_fail_inhibit_timer as well for multiple cycles of
920          * parallel detect, both 10g and 1g. This allows for the maximum
921          * connect attempts as defined in the AN MAS table 73-7.
922          */
923         for (i = 0; i < 6; i++) {
924                 msec_delay(100);
925
926                 /* If we have link, just jump out */
927                 status = ixgbe_check_link(hw, &link_speed, &link_up, FALSE);
928                 if (status != IXGBE_SUCCESS)
929                         goto out;
930
931                 if (link_up)
932                         goto out;
933         }
934
935         /* We didn't get link.  Turn SmartSpeed back off. */
936         hw->phy.smart_speed_active = FALSE;
937         status = ixgbe_setup_mac_link_82599(hw, speed,
810         status = ixgbe_setup_mac_link_82599(hw, speed, autoneg,
938                                             autoneg_wait_to_complete);
939
940 out:
941         if (link_up && (link_speed == IXGBE_LINK_SPEED_1GB_FULL))
942                 DEBUGOUT("Smartspeed has downgraded the link speed "
943                 "from the maximum advertised\n");
944         return status;
945 }
946
947 /**
948  *  ixgbe_setup_mac_link_82599 - Set MAC link speed
949  *  @hw: pointer to hardware structure
950  *  @speed: new link speed
824  *  @autoneg: TRUE if autonegotiation enabled
951  *  @autoneg_wait_to_complete: TRUE when waiting for completion is needed
952  *
953  *  Set the link speed in the AUTOC register and restarts link.
954  **/
955 s32 ixgbe_setup_mac_link_82599(struct ixgbe_hw *hw,
956                                ixgbe_link_speed speed,
830                                ixgbe_link_speed speed, bool autoneg,
957                                bool autoneg_wait_to_complete)
958 {
959         bool autoneg = FALSE;
960         s32 status = IXGBE_SUCCESS;
961         u32 autoc, pma_pmd_1g, link_mode, start_autoc;
834         u32 autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
962         u32 autoc2 = IXGBE_READ_REG(hw, IXGBE_AUTOC2);
836         u32 start_autoc = autoc;
963         u32 orig_autoc = 0;
838         u32 link_mode = autoc & IXGBE_AUTOC_LMS_MASK;
839         u32 pma_pmd_1g = autoc & IXGBE_AUTOC_1G_PMA_PMD_MASK;
964         u32 pma_pmd_10g_serial = autoc2 & IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_MASK;
965         u32 links_reg;
966         u32 i;
967         ixgbe_link_speed link_capabilities = IXGBE_LINK_SPEED_UNKNOWN;
968         bool got_lock = FALSE;
969
970         DEBUGFUNC("ixgbe_setup_mac_link_82599");
971
972         /* Check to see if speed passed in is supported. */
```

```
 973         status = ixgbe_get_link_capabilities(hw, &link_capabilities, &autoneg);
 974         if (status)
 849         if (status != IXGBE_SUCCESS)
 975                 goto out;

 977         speed &= link_capabilities;

 979         if (speed == IXGBE_LINK_SPEED_UNKNOWN) {
 980                 status = IXGBE_ERR_LINK_SETUP;
 981                 goto out;
 982         }

 984         /* Use stored value (EEPROM defaults) of AUTOC to find KR/KX4 support*/
 985         if (hw->mac.orig_link_settings_stored)
 986                 autoc = hw->mac.orig_autoc;
 861                 orig_autoc = hw->mac.orig_autoc;
 987         else
 988                 autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);

 990         orig_autoc = autoc;
 991         start_autoc = hw->mac.cached_autoc;
 992         link_mode = autoc & IXGBE_AUTOC_LMS_MASK;
 993         pma_pmd_1g = autoc & IXGBE_AUTOC_1G_PMA_PMD_MASK;

 995         if (link_mode == IXGBE_AUTOC_LMS_KX4_KX_KR ||
 996             link_mode == IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN ||
 997             link_mode == IXGBE_AUTOC_LMS_KX4_KX_KR_SGMII) {
 998                 /* Set KX4/KX/KR support according to speed requested */
 999                 autoc &= ~(IXGBE_AUTOC_KX4_KX_SUPP_MASK | IXGBE_AUTOC_KR_SUPP);
1000                 if (speed & IXGBE_LINK_SPEED_10GB_FULL) {
1001                         if (orig_autoc & IXGBE_AUTOC_KX4_SUPP)
1002                                 autoc |= IXGBE_AUTOC_KX4_SUPP;
1003                         if ((orig_autoc & IXGBE_AUTOC_KR_SUPP) &&
1004                             (hw->phy.smart_speed_active == FALSE))
1005                                 autoc |= IXGBE_AUTOC_KR_SUPP;
1006                 }
1007                 if (speed & IXGBE_LINK_SPEED_1GB_FULL)
1008                         autoc |= IXGBE_AUTOC_KX_SUPP;
1009         } else if ((pma_pmd_1g == IXGBE_AUTOC_1G_SFI) &&
1010                    (link_mode == IXGBE_AUTOC_LMS_1G_LINK_NO_AN ||
1011                     link_mode == IXGBE_AUTOC_LMS_1G_AN)) {
1012                 /* Switch from 1G SFI to 10G SFI if requested */
1013                 if ((speed == IXGBE_LINK_SPEED_10GB_FULL) &&
1014                     (pma_pmd_10g_serial == IXGBE_AUTOC2_10G_SFI)) {
1015                         autoc &= ~IXGBE_AUTOC_LMS_MASK;
1016                         autoc |= IXGBE_AUTOC_LMS_10G_SERIAL;
1017                 }
1018         } else if ((pma_pmd_10g_serial == IXGBE_AUTOC2_10G_SFI) &&
1019                    (link_mode == IXGBE_AUTOC_LMS_10G_SERIAL)) {
1020                 /* Switch from 10G SFI to 1G SFI if requested */
1021                 if ((speed == IXGBE_LINK_SPEED_1GB_FULL) &&
1022                     (pma_pmd_1g == IXGBE_AUTOC_1G_SFI)) {
1023                         autoc &= ~IXGBE_AUTOC_LMS_MASK;
1024                         if (autoneg)
1025                                 autoc |= IXGBE_AUTOC_LMS_1G_AN;
1026                         else
1027                                 autoc |= IXGBE_AUTOC_LMS_1G_LINK_NO_AN;
1028                 }
1029         }

1031         if (autoc != start_autoc) {
1032                 /* Need SW/FW semaphore around AUTOC writes if LESM is on,
1033                  * likewise reset_pipeline requires us to hold this lock as
1034                  * it also writes to AUTOC.
1035                  */
1036                 if (ixgbe_verify_lesm_fw_enabled_82599(hw)) {
```



```
1037                         status = hw->mac.ops.acquire_swfw_sync(hw,
1038                                                                IXGBE_GSSR_MAC_CSR_SM);
1039                         if (status != IXGBE_SUCCESS) {
1040                                 status = IXGBE_ERR_SWFW_SYNC;
1041                                 goto out;
1042                         }

1044                         got_lock = TRUE;
1045                 }

1047                 /* Restart link */
 903                 autoc |= IXGBE_AUTOC_AN_RESTART;
1048                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc);
1049                 hw->mac.cached_autoc = autoc;
1050                 (void) ixgbe_reset_pipeline_82599(hw);

1052                 if (got_lock) {
1053                         hw->mac.ops.release_swfw_sync(hw,
1054                                                       IXGBE_GSSR_MAC_CSR_SM);
1055                         got_lock = FALSE;
1056                 }

1058                 /* Only poll for autoneg to complete if specified to do so */
1059                 if (autoneg_wait_to_complete) {
1060                         if (link_mode == IXGBE_AUTOC_LMS_KX4_KX_KR ||
1061                             link_mode == IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN ||
1062                             link_mode == IXGBE_AUTOC_LMS_KX4_KX_KR_SGMII) {
1063                                 links_reg = 0; /*Just in case Autoneg time=0*/
1064                                 for (i = 0; i < IXGBE_AUTO_NEG_TIME; i++) {
1065                                         links_reg =
1066                                               IXGBE_READ_REG(hw, IXGBE_LINKS);
1067                                         if (links_reg & IXGBE_LINKS_KX_AN_COMP)
1068                                                 break;
1069                                         msec_delay(100);
1070                                 }
1071                                 if (!(links_reg & IXGBE_LINKS_KX_AN_COMP)) {
1072                                         status =
1073                                                 IXGBE_ERR_AUTONEG_NOT_COMPLETE;
1074                                         DEBUGOUT("Autoneg did not complete.\n");
1075                                 }
1076                         }
1077                 }

1079                 /* Add delay to filter out noises during initial link setup */
1080                 msec_delay(50);
1081         }

1083 out:
1084         return status;
1085 }

1087 /**
1088  *  ixgbe_setup_copper_link_82599 - Set the PHY autoneg advertised field
1089  *  @hw: pointer to hardware structure
1090  *  @speed: new link speed
 939  *  @autoneg: TRUE if autonegotiation enabled
1091  *  @autoneg_wait_to_complete: TRUE if waiting is needed to complete
1092  *
1093  *  Restarts link on PHY and MAC based on settings passed in.
1094  **/
1095 static s32 ixgbe_setup_copper_link_82599(struct ixgbe_hw *hw,
1096                                          ixgbe_link_speed speed,
 946                                          bool autoneg,
1097                                          bool autoneg_wait_to_complete)
1098 {
1099         s32 status;
```

```
1101         DEBUGFUNC("ixgbe_setup_copper_link_82599");

1103         /* Setup the PHY according to input speed */
1104         status = hw->phy.ops.setup_link_speed(hw, speed,
 954         status = hw->phy.ops.setup_link_speed(hw, speed, autoneg,
1105                                               autoneg_wait_to_complete);
 956         if (status == IXGBE_SUCCESS) {
1106         /* Set up MAC */
1107         (void) ixgbe_start_mac_link_82599(hw, autoneg_wait_to_complete);
 958                 status =
 959                     ixgbe_start_mac_link_82599(hw, autoneg_wait_to_complete);
 960         }

1109         return status;
1110 }

1112 /**
1113  *  ixgbe_reset_hw_82599 - Perform hardware reset
1114  *  @hw: pointer to hardware structure
1115  *
1116  *  Resets the hardware by resetting the transmit and receive units, masks
1117  *  and clears all interrupts, perform a PHY reset, and perform a link (MAC)
1118  *  reset.
1119  **/
1120 s32 ixgbe_reset_hw_82599(struct ixgbe_hw *hw)
1121 {
1122         ixgbe_link_speed link_speed;
1123         s32 status;
1124         u32 ctrl, i, autoc, autoc2;
1125         bool link_up = FALSE;

1127         DEBUGFUNC("ixgbe_reset_hw_82599");

1129         /* Call adapter stop to disable tx/rx and clear interrupts */
1130         status = hw->mac.ops.stop_adapter(hw);
1131         if (status != IXGBE_SUCCESS)
1132                 goto reset_hw_out;

1134         /* flush pending Tx transactions */
1135         ixgbe_clear_tx_pending(hw);

1137         /* PHY ops must be identified and initialized prior to reset */

1139         /* Identify PHY and related function pointers */
1140         status = hw->phy.ops.init(hw);

1142         if (status == IXGBE_ERR_SFP_NOT_SUPPORTED)
1143                 goto reset_hw_out;

1145         /* Setup SFP module if there is one present. */
1146         if (hw->phy.sfp_setup_needed) {
1147                 status = hw->mac.ops.setup_sfp(hw);
1148                 hw->phy.sfp_setup_needed = FALSE;
1149         }

1151         if (status == IXGBE_ERR_SFP_NOT_SUPPORTED)
1152                 goto reset_hw_out;

1154         /* Reset PHY */
1155         if (hw->phy.reset_disable == FALSE && hw->phy.ops.reset != NULL)
1156                 hw->phy.ops.reset(hw);

1158 mac_reset_top:
1159         /*
1160          * Issue global reset to the MAC.  Needs to be SW reset if link is up.
```



```
1161          * If link reset is used when link is up, it might reset the PHY when
1162          * mng is using it.  If link is down or the flag to force full link
1163          * reset is set, then perform link reset.
1164          */
1165         ctrl = IXGBE_CTRL_LNK_RST;
1166         if (!hw->force_full_reset) {
1167                 hw->mac.ops.check_link(hw, &link_speed, &link_up, FALSE);
1168                 if (link_up)
1169                         ctrl = IXGBE_CTRL_RST;
1170         }

1172         ctrl |= IXGBE_READ_REG(hw, IXGBE_CTRL);
1173         IXGBE_WRITE_REG(hw, IXGBE_CTRL, ctrl);
1174         IXGBE_WRITE_FLUSH(hw);

1176         /* Poll for reset bit to self-clear indicating reset is complete */
1177         for (i = 0; i < 10; i++) {
1178                 usec_delay(1);
1179                 ctrl = IXGBE_READ_REG(hw, IXGBE_CTRL);
1180                 if (!(ctrl & IXGBE_CTRL_RST_MASK))
1181                         break;
1182         }

1184         if (ctrl & IXGBE_CTRL_RST_MASK) {
1185                 status = IXGBE_ERR_RESET_FAILED;
1186                 DEBUGOUT("Reset polling failed to complete.\n");
1187         }

1189         msec_delay(50);

1191         /*
1192          * Double resets are required for recovery from certain error
1193          * conditions.  Between resets, it is necessary to stall to allow time
1194          * for any pending HW events to complete.
1195          */
1196         if (hw->mac.flags & IXGBE_FLAGS_DOUBLE_RESET_REQUIRED) {
1197                 hw->mac.flags &= ~IXGBE_FLAGS_DOUBLE_RESET_REQUIRED;
1198                 goto mac_reset_top;
1199         }

1201         /*
1202          * Store the original AUTOC/AUTOC2 values if they have not been
1203          * stored off yet.  Otherwise restore the stored original
1204          * values since the reset operation sets back to defaults.
1205          */
1206         autoc = IXGBE_READ_REG(hw, IXGBE_AUTOC);
1207         autoc2 = IXGBE_READ_REG(hw, IXGBE_AUTOC2);

1209         /* Enable link if disabled in NVM */
1210         if (autoc2 & IXGBE_AUTOC2_LINK_DISABLE_MASK) {
1211                 autoc2 &= ~IXGBE_AUTOC2_LINK_DISABLE_MASK;
1212                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC2, autoc2);
1213                 IXGBE_WRITE_FLUSH(hw);
1214         }

1216         if (hw->mac.orig_link_settings_stored == FALSE) {
1217                 hw->mac.orig_autoc = autoc;
1218                 hw->mac.orig_autoc2 = autoc2;
1219                 hw->mac.cached_autoc = autoc;
1220                 hw->mac.orig_link_settings_stored = TRUE;
1221         } else {
1222                 if (autoc != hw->mac.orig_autoc) {
1223                         /* Need SW/FW semaphore around AUTOC writes if LESM is
1224                          * on, likewise reset_pipeline requires us to hold
1225                          * this lock as it also writes to AUTOC.
1226                          */
```

```
                        bool got_lock = FALSE;
                        if (ixgbe_verify_lesm_fw_enabled_82599(hw)) {
                                status = hw->mac.ops.acquire_swfw_sync(hw,
                                                IXGBE_GSSR_MAC_CSR_SM);
                                if (status != IXGBE_SUCCESS) {
                                        status = IXGBE_ERR_SWFW_SYNC;
                                        goto reset_hw_out;
                                }
                *if (autoc != hw->mac.orig_autoc)
                        IXGBE_WRITE_REG(hw, IXGBE_AUTOC, (hw->mac.orig_autoc |
                                        IXGBE_AUTOC_AN_RESTART));*

                                got_lock = TRUE;
                        }

                        IXGBE_WRITE_REG(hw, IXGBE_AUTOC, hw->mac.orig_autoc);
                        hw->mac.cached_autoc = hw->mac.orig_autoc;
                        (void) ixgbe_reset_pipeline_82599(hw);

                        if (got_lock)
                                hw->mac.ops.release_swfw_sync(hw,
                                                IXGBE_GSSR_MAC_CSR_SM);
                }

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

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

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

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

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

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

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

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

reset_hw_out:
        return status;
}
```



```
/**
 *  ixgbe_reinit_fdir_tables_82599 - Reinitialize Flow Director tables.
 *  @hw: pointer to hardware structure
 **/
s32 ixgbe_reinit_fdir_tables_82599(struct ixgbe_hw *hw)
{
        int i;
        u32 fdirctrl = IXGBE_READ_REG(hw, IXGBE_FDIRCTRL);
        fdirctrl &= ~IXGBE_FDIRCTRL_INIT_DONE;

        DEBUGFUNC("ixgbe_reinit_fdir_tables_82599");

        /*
         * Before starting reinitialization process,
         * FDIRCMD.CMD must be zero.
         */
        for (i = 0; i < IXGBE_FDIRCMD_CMD_POLL; i++) {
                if (!(IXGBE_READ_REG(hw, IXGBE_FDIRCMD) &
                      IXGBE_FDIRCMD_CMD_MASK))
                        break;
                usec_delay(10);
        }
        if (i >= IXGBE_FDIRCMD_CMD_POLL) {
                DEBUGOUT("Flow Director previous command isn't complete, "
                         "aborting table re-initialization.\n");
                return IXGBE_ERR_FDIR_REINIT_FAILED;
        }

        IXGBE_WRITE_REG(hw, IXGBE_FDIRFREE, 0);
        IXGBE_WRITE_FLUSH(hw);
        /*
         * 82599 adapters flow director init flow cannot be restarted,
         * Workaround 82599 silicon errata by performing the following steps
         * before re-writing the FDIRCTRL control register with the same value.
         * - write 1 to bit 8 of FDIRCMD register &
         * - write 0 to bit 8 of FDIRCMD register
         */
        IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD,
                        (IXGBE_READ_REG(hw, IXGBE_FDIRCMD) |
                         IXGBE_FDIRCMD_CLEARHT));
        IXGBE_WRITE_FLUSH(hw);
        IXGBE_WRITE_REG(hw, IXGBE_FDIRCMD,
                        (IXGBE_READ_REG(hw, IXGBE_FDIRCMD) &
                         ~IXGBE_FDIRCMD_CLEARHT));
        IXGBE_WRITE_FLUSH(hw);
        /*
         * Clear FDIR Hash register to clear any leftover hashes
         * waiting to be programmed.
         */
        IXGBE_WRITE_REG(hw, IXGBE_FDIRHASH, 0x00);
        IXGBE_WRITE_FLUSH(hw);

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

        /* Poll init-done after we write FDIRCTRL register */
        for (i = 0; i < IXGBE_FDIR_INIT_DONE_POLL; i++) {
                if (IXGBE_READ_REG(hw, IXGBE_FDIRCTRL) &
                                   IXGBE_FDIRCTRL_INIT_DONE)
                        break;
                *msec_delay(1);*
                *usec_delay(10);*
        }
        if (i >= IXGBE_FDIR_INIT_DONE_POLL) {
                DEBUGOUT("Flow Director Signature poll time exceeded!\n");
```

```
1355                 return IXGBE_ERR_FDIR_REINIT_FAILED;
1356         }

1358         /* Clear FDIR statistics registers (read to clear) */
1359         (void) IXGBE_READ_REG(hw, IXGBE_FDIRUSTAT);
1360         (void) IXGBE_READ_REG(hw, IXGBE_FDIRFSTAT);
1361         (void) IXGBE_READ_REG(hw, IXGBE_FDIRMATCH);
1362         (void) IXGBE_READ_REG(hw, IXGBE_FDIRMISS);
1363         (void) IXGBE_READ_REG(hw, IXGBE_FDIRLEN);

1365         return IXGBE_SUCCESS;
1366 }
______unchanged_portion_omitted_

2276 /**
2277  *  ixgbe_verify_fw_version_82599 - verify fw version for 82599
2278  *  @hw: pointer to hardware structure
2279  *
2280  *  Verifies that installed the firmware version is 0.6 or higher
2281  *  for SFI devices. All 82599 SFI devices should have version 0.6 or higher.
2282  *
2283  *  Returns IXGBE_ERR_EEPROM_VERSION if the FW is not present or
2284  *  if the FW version is not supported.
2285  **/
2286 s32 ixgbe_verify_fw_version_82599(struct ixgbe_hw *hw)
2108 static s32 ixgbe_verify_fw_version_82599(struct ixgbe_hw *hw)
2287 {
2288         s32 status = IXGBE_ERR_EEPROM_VERSION;
2289         u16 fw_offset, fw_ptp_cfg_offset;
2290         u16 fw_version = 0;

2292         DEBUGFUNC("ixgbe_verify_fw_version_82599");

2294         /* firmware check is only necessary for SFI devices */
2295         if (hw->phy.media_type != ixgbe_media_type_fiber) {
2296                 status = IXGBE_SUCCESS;
2297                 goto fw_version_out;
2298         }

2300         /* get the offset to the Firmware Module block */
2301         hw->eeprom.ops.read(hw, IXGBE_FW_PTR, &fw_offset);

2303         if ((fw_offset == 0) || (fw_offset == 0xFFFF))
2304                 goto fw_version_out;

2306         /* get the offset to the Pass Through Patch Configuration block */
2307         hw->eeprom.ops.read(hw, (fw_offset +
2308                                  IXGBE_FW_PASSTHROUGH_PATCH_CONFIG_PTR),
2309                                  &fw_ptp_cfg_offset);

2311         if ((fw_ptp_cfg_offset == 0) || (fw_ptp_cfg_offset == 0xFFFF))
2312                 goto fw_version_out;

2314         /* get the firmware version */
2315         hw->eeprom.ops.read(hw, (fw_ptp_cfg_offset +
2316                                  IXGBE_FW_PATCH_VERSION_4), &fw_version);

2318         if (fw_version > 0x5)
2319                 status = IXGBE_SUCCESS;

2321 fw_version_out:
2322         return status;
2323 }
______unchanged_portion_omitted_

2435 /**
```



```
2436  * ixgbe_reset_pipeline_82599 - perform pipeline reset
2437  *
2438  *  @hw: pointer to hardware structure
2439  *
2440  * Reset pipeline by asserting Restart_AN together with LMS change to ensure
2441  * full pipeline reset
2442  **/
2443 s32 ixgbe_reset_pipeline_82599(struct ixgbe_hw *hw)
2444 {
2445         s32 ret_val;
2446         u32 anlp1_reg = 0;
2447         u32 i, autoc_reg, autoc2_reg;

2449         /* Enable link if disabled in NVM */
2450         autoc2_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC2);
2451         if (autoc2_reg & IXGBE_AUTOC2_LINK_DISABLE_MASK) {
2452                 autoc2_reg &= ~IXGBE_AUTOC2_LINK_DISABLE_MASK;
2453                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC2, autoc2_reg);
2454                 IXGBE_WRITE_FLUSH(hw);
2455         }

2457         autoc_reg = hw->mac.cached_autoc;
2458         autoc_reg |= IXGBE_AUTOC_AN_RESTART;
2459         /* Write AUTOC register with toggled LMS[2] bit and Restart_AN */
2460         IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc_reg ^ IXGBE_AUTOC_LMS_1G_AN);
2461         /* Wait for AN to leave state 0 */
2462         for (i = 0; i < 10; i++) {
2463                 msec_delay(4);
2464                 anlp1_reg = IXGBE_READ_REG(hw, IXGBE_ANLP1);
2465                 if (anlp1_reg & IXGBE_ANLP1_AN_STATE_MASK)
2466                         break;
2467         }

2469         if (!(anlp1_reg & IXGBE_ANLP1_AN_STATE_MASK)) {
2470                 DEBUGOUT("auto negotiation not completed\n");
2471                 ret_val = IXGBE_ERR_RESET_FAILED;
2472                 goto reset_pipeline_out;
2473         }

2475         ret_val = IXGBE_SUCCESS;

2477 reset_pipeline_out:
2478         /* Write AUTOC register with original LMS field and Restart_AN */
2479         IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc_reg);
2480         IXGBE_WRITE_FLUSH(hw);

2482         return ret_val;
2483 }
```

```
**********************************************************
    3348 Tue Apr 30 09:54:22 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_82599.h
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
/******************************************************************************
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

#ifndef _IXGBE_82599_H_
#define _IXGBE_82599_H_

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



```
s32 ixgbe_write_analog_reg8_82599(struct ixgbe_hw *hw, u32 reg, u8 val);
s32 ixgbe_start_hw_82599(struct ixgbe_hw *hw);
s32 ixgbe_identify_phy_82599(struct ixgbe_hw *hw);
s32 ixgbe_init_phy_ops_82599(struct ixgbe_hw *hw);
u32 ixgbe_get_supported_physical_layer_82599(struct ixgbe_hw *hw);
s32 ixgbe_enable_rx_dma_82599(struct ixgbe_hw *hw, u32 regval);
bool ixgbe_verify_lesm_fw_enabled_82599(struct ixgbe_hw *hw);
#endif /* _IXGBE_82599_H_ */
```

```
**********************************************************
   35370 Tue Apr 30 09:54:22 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_api.c
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
/******************************************************************************
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

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

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

        DEBUGFUNC("ixgbe_init_shared_code");

        /*
         * Set the mac type
         */
        status = ixgbe_set_mac_type(hw);
        if (status != IXGBE_SUCCESS)
```



```
                return (status);

        switch (hw->mac.type) {
        case ixgbe_mac_82598EB:
                status = ixgbe_init_ops_82598(hw);
                break;
        case ixgbe_mac_82599EB:
                status = ixgbe_init_ops_82599(hw);
                break;
        case ixgbe_mac_X540:
                status = ixgbe_init_ops_X540(hw);
                break;
        default:
                status = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
                break;
        }

        return status;
}

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

        DEBUGFUNC("ixgbe_set_mac_type\n");

        if (hw->vendor_id == IXGBE_INTEL_VENDOR_ID) {
        switch (hw->device_id) {
        case IXGBE_DEV_ID_82598:
        case IXGBE_DEV_ID_82598_BX:
        case IXGBE_DEV_ID_82598AF_SINGLE_PORT:
        case IXGBE_DEV_ID_82598AF_DUAL_PORT:
        case IXGBE_DEV_ID_82598AT:
        case IXGBE_DEV_ID_82598AT2:
        case IXGBE_DEV_ID_82598EB_CX4:
        case IXGBE_DEV_ID_82598_CX4_DUAL_PORT:
        case IXGBE_DEV_ID_82598_DA_DUAL_PORT:
        case IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM:
        case IXGBE_DEV_ID_82598EB_XF_LR:
        case IXGBE_DEV_ID_82598EB_SFP_LOM:
                hw->mac.type = ixgbe_mac_82598EB;
                break;
        case IXGBE_DEV_ID_82599_KX4:
        case IXGBE_DEV_ID_82599_KX4_MEZZ:
        case IXGBE_DEV_ID_82599_XAUI_LOM:
        case IXGBE_DEV_ID_82599_COMBO_BACKPLANE:
        case IXGBE_DEV_ID_82599_KR:
        case IXGBE_DEV_ID_82599_SFP:
        case IXGBE_DEV_ID_82599_BACKPLANE_FCOE:
        case IXGBE_DEV_ID_82599_SFP_FCOE:
        case IXGBE_DEV_ID_82599_SFP_EM:
        case IXGBE_DEV_ID_82599_SFP_SF2:
        case IXGBE_DEV_ID_82599_SFP_SF_QP:
        case IXGBE_DEV_ID_82599EN_SFP:
        case IXGBE_DEV_ID_82599_CX4:
        case IXGBE_DEV_ID_82599_BYPASS:
        case IXGBE_DEV_ID_82599_T3_LOM:
                hw->mac.type = ixgbe_mac_82599EB;
                break;
```

```
126         case IXGBE_DEV_ID_82599_VF:
127         case IXGBE_DEV_ID_82599_VF_HV:
128                 hw->mac.type = ixgbe_mac_82599_vf;
129                 break;
130         case IXGBE_DEV_ID_X540_VF:
131         case IXGBE_DEV_ID_X540_VF_HV:
132                 hw->mac.type = ixgbe_mac_X540_vf;
133                 break;
134         case IXGBE_DEV_ID_X540T:
135         case IXGBE_DEV_ID_X540_BYPASS:
132                 case IXGBE_DEV_ID_X540T1:
136                 hw->mac.type = ixgbe_mac_X540;
137                 break;
138         default:
139                 ret_val = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
140                 break;
141         }
139         } else {
140                 ret_val = IXGBE_ERR_DEVICE_NOT_SUPPORTED;
141         }

143         DEBUGOUT2("ixgbe_set_mac_type found mac: %d, returns: %d\n",
144                   hw->mac.type, ret_val);
145         return ret_val;
146 }
```

unchanged_portion_omitted_

```
521 /**
522  *  ixgbe_setup_phy_link_speed - Set auto advertise
523  *  @hw: pointer to hardware structure
524  *  @speed: new link speed
525  *  @autoneg: TRUE if autonegotiation enabled
525  *
526  *  Sets the auto advertised capabilities
527  **/
528 s32 ixgbe_setup_phy_link_speed(struct ixgbe_hw *hw, ixgbe_link_speed speed,
530                                bool autoneg,
529                                bool autoneg_wait_to_complete)
530 {
531         return ixgbe_call_func(hw, hw->phy.ops.setup_link_speed, (hw, speed,
532                                autoneg_wait_to_complete),
534                                autoneg, autoneg_wait_to_complete),
533                                IXGBE_NOT_IMPLEMENTED);
534 }
```

unchanged_portion_omitted_

```
588 /**
589  *  ixgbe_setup_link - Set link speed
590  *  @hw: pointer to hardware structure
591  *  @speed: new link speed
594  *  @autoneg: TRUE if autonegotiation enabled
592  *
593  *  Configures link settings.  Restarts the link.
594  *  Performs autonegotiation if needed.
595  **/
596 s32 ixgbe_setup_link(struct ixgbe_hw *hw, ixgbe_link_speed speed,
600                      bool autoneg,
597                      bool autoneg_wait_to_complete)
598 {
599         return ixgbe_call_func(hw, hw->mac.ops.setup_link, (hw, speed,
600                                autoneg_wait_to_complete),
604                                autoneg, autoneg_wait_to_complete),
601                                IXGBE_NOT_IMPLEMENTED);
602 }
```

unchanged_portion_omitted_

```
**********************************************************
    8453 Tue Apr 30 09:54:22 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_api.h
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
/******************************************************************************

  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

#ifndef _IXGBE_API_H_
#define _IXGBE_API_H_

#include "ixgbe_type.h"

s32 ixgbe_init_shared_code(struct ixgbe_hw *hw);

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

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



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

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

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

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

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

```
125                           u8 ver);
126 void ixgbe_set_mta(struct ixgbe_hw *hw, u8 *mc_addr);
127 s32 ixgbe_get_phy_firmware_version(struct ixgbe_hw *hw,
128                                    u16 *firmware_version);
129 s32 ixgbe_read_analog_reg8(struct ixgbe_hw *hw, u32 reg, u8 *val);
130 s32 ixgbe_write_analog_reg8(struct ixgbe_hw *hw, u32 reg, u8 val);
131 s32 ixgbe_init_uta_tables(struct ixgbe_hw *hw);
132 s32 ixgbe_read_i2c_eeprom(struct ixgbe_hw *hw, u8 byte_offset, u8 *eeprom_data);
133 u32 ixgbe_get_supported_physical_layer(struct ixgbe_hw *hw);
134 s32 ixgbe_enable_rx_dma(struct ixgbe_hw *hw, u32 regval);
135 s32 ixgbe_disable_sec_rx_path(struct ixgbe_hw *hw);
136 s32 ixgbe_enable_sec_rx_path(struct ixgbe_hw *hw);
137 s32 ixgbe_reinit_fdir_tables_82599(struct ixgbe_hw *hw);
138 s32 ixgbe_init_fdir_signature_82599(struct ixgbe_hw *hw, u32 fdirctrl);
139 s32 ixgbe_init_fdir_perfect_82599(struct ixgbe_hw *hw, u32 fdirctrl);
140 s32 ixgbe_fdir_add_signature_filter_82599(struct ixgbe_hw *hw,
141                                           union ixgbe_atr_hash_dword input,
142                                           union ixgbe_atr_hash_dword common,
143                                           u8 queue);
144 s32 ixgbe_fdir_set_input_mask_82599(struct ixgbe_hw *hw,
145                                     union ixgbe_atr_input *input_mask);
146 s32 ixgbe_fdir_write_perfect_filter_82599(struct ixgbe_hw *hw,
147                                           union ixgbe_atr_input *input,
148                                           u16 soft_id, u8 queue);
149 s32 ixgbe_fdir_erase_perfect_filter_82599(struct ixgbe_hw *hw,
150                                           union ixgbe_atr_input *input,
151                                           u16 soft_id);
152 s32 ixgbe_fdir_add_perfect_filter_82599(struct ixgbe_hw *hw,
153                                         union ixgbe_atr_input *input,
154                                         union ixgbe_atr_input *mask,
155                                         u16 soft_id,
156                                         u8 queue);
157 void ixgbe_atr_compute_perfect_hash_82599(union ixgbe_atr_input *input,
158                                           union ixgbe_atr_input *mask);
159 u32 ixgbe_atr_compute_sig_hash_82599(union ixgbe_atr_hash_dword input,
160                                      union ixgbe_atr_hash_dword common);
161 bool ixgbe_verify_lesm_fw_enabled_82599(struct ixgbe_hw *hw);
162 s32 ixgbe_read_i2c_byte(struct ixgbe_hw *hw, u8 byte_offset, u8 dev_addr,
163                         u8 *data);
164 s32 ixgbe_write_i2c_byte(struct ixgbe_hw *hw, u8 byte_offset, u8 dev_addr,
165                          u8 data);
166 s32 ixgbe_write_i2c_eeprom(struct ixgbe_hw *hw, u8 byte_offset, u8 eeprom_data);
167 s32 ixgbe_get_san_mac_addr(struct ixgbe_hw *hw, u8 *san_mac_addr);
168 s32 ixgbe_set_san_mac_addr(struct ixgbe_hw *hw, u8 *san_mac_addr);
169 s32 ixgbe_get_device_caps(struct ixgbe_hw *hw, u16 *device_caps);
170 s32 ixgbe_acquire_swfw_semaphore(struct ixgbe_hw *hw, u16 mask);
171 void ixgbe_release_swfw_semaphore(struct ixgbe_hw *hw, u16 mask);
172 s32 ixgbe_get_wwn_prefix(struct ixgbe_hw *hw, u16 *wwnn_prefix,
173                          u16 *wwpn_prefix);
174 s32 ixgbe_get_fcoe_boot_status(struct ixgbe_hw *hw, u16 *bs);

176 #endif /* _IXGBE_API_H_ */
```

```
**********************************************************
  123824 Tue Apr 30 09:54:22 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_common.c
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
```

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

```
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

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

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

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



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

        DEBUGFUNC("ixgbe_init_ops_generic");

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

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

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

        /* RAR, Multicast, VLAN */
        mac->ops.set_rar = &ixgbe_set_rar_generic;
        mac->ops.clear_rar = &ixgbe_clear_rar_generic;
        mac->ops.insert_mac_addr = NULL;
        mac->ops.set_vmdq = NULL;
        mac->ops.clear_vmdq = NULL;
        mac->ops.init_rx_addrs = &ixgbe_init_rx_addrs_generic;
        mac->ops.update_uc_addr_list = &ixgbe_update_uc_addr_list_generic;
        mac->ops.update_mc_addr_list = &ixgbe_update_mc_addr_list_generic;
        mac->ops.enable_mc = &ixgbe_enable_mc_generic;
        mac->ops.disable_mc = &ixgbe_disable_mc_generic;
        mac->ops.clear_vfta = NULL;
        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 s32 ixgbe_device_supports_autoneg_fc(struct ixgbe_hw *hw)
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_82599_T3_LOM:
157         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;
159         default:
160                 return IXGBE_ERR_FC_NOT_SUPPORTED;
161         }
162 }

164 /**
165  *  ixgbe_setup_fc - Set up flow control
166  *  @hw: pointer to hardware structure
167  *
168  *  Called at init time to set up flow control.
169  **/
170 static s32 ixgbe_setup_fc(struct ixgbe_hw *hw)
171 {
172         s32 ret_val = IXGBE_SUCCESS;
173         u32 reg = 0, reg_bp = 0;
174         u16 reg_cu = 0;
175         bool got_lock = FALSE;

177         DEBUGFUNC("ixgbe_setup_fc");

179         /*
180          * Validate the requested mode.  Strict IEEE mode does not allow
181          * ixgbe_fc_rx_pause because it will cause us to fail at UNH.
182          */
183         if (hw->fc.strict_ieee && hw->fc.requested_mode == ixgbe_fc_rx_pause) {
184                 DEBUGOUT("ixgbe_fc_rx_pause not valid in strict IEEE mode\n");
185                 ret_val = IXGBE_ERR_INVALID_LINK_SETTINGS;
186                 goto out;
187         }
```



```
189         /*
190          * 10gig parts do not have a word in the EEPROM to determine the
191          * default flow control setting, so we explicitly set it to full.
192          */
193         if (hw->fc.requested_mode == ixgbe_fc_default)
194                 hw->fc.requested_mode = ixgbe_fc_full;

196         /*
197          * Set up the 1G and 10G flow control advertisement registers so the
198          * HW will be able to do fc autoneg once the cable is plugged in.  If
199          * we link at 10G, the 1G advertisement is harmless and vice versa.
200          */
201         switch (hw->phy.media_type) {
202         case ixgbe_media_type_fiber_fixed:
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                 /* Need the SW/FW semaphore around AUTOC writes if 82599 and
300                  * LESM is on, likewise reset_pipeline requries the lock as
301                  * it also writes AUTOC.
302                  */
303                 if ((hw->mac.type == ixgbe_mac_82599EB) &&
304                     ixgbe_verify_lesm_fw_enabled_82599(hw)) {
305                         ret_val = hw->mac.ops.acquire_swfw_sync(hw,
306                                                 IXGBE_GSSR_MAC_CSR_SM);
307                         if (ret_val != IXGBE_SUCCESS) {
308                                 ret_val = IXGBE_ERR_SWFW_SYNC;
309                                 goto out;
310                         }
311                         got_lock = TRUE;
312                 }
313
314                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, reg_bp);
315                 if (hw->mac.type == ixgbe_mac_82599EB)
316                         (void) ixgbe_reset_pipeline_82599(hw);
317
318                 if (got_lock)
319                         hw->mac.ops.release_swfw_sync(hw,
320                                                       IXGBE_GSSR_MAC_CSR_SM);
```



```
321         } else if ((hw->phy.media_type == ixgbe_media_type_copper) &&
322                    (ixgbe_device_supports_autoneg_fc(hw) == IXGBE_SUCCESS)) {
323                 hw->phy.ops.write_reg(hw, IXGBE_MDIO_AUTO_NEG_ADVT,
324                                       IXGBE_MDIO_AUTO_NEG_DEV_TYPE, reg_cu);
325         }
326
327         DEBUGOUT1("Set up FC; IXGBE_AUTOC = 0x%08X\n", reg);
328 out:
329         return ret_val;
330 }
_____unchanged_portion_omitted_

702 /**
703  *  ixgbe_read_pba_raw
704  *  @hw: pointer to the HW structure
705  *  @eeprom_buf: optional pointer to EEPROM image
706  *  @eeprom_buf_size: size of EEPROM image in words
707  *  @max_pba_block_size: PBA block size limit
708  *  @pba: pointer to output PBA structure
709  *
710  *  Reads PBA from EEPROM image when eeprom_buf is not NULL.
711  *  Reads PBA from physical EEPROM device when eeprom_buf is NULL.
712  *
713  **/
714 s32 ixgbe_read_pba_raw(struct ixgbe_hw *hw, u16 *eeprom_buf,
715                        u32 eeprom_buf_size, u16 max_pba_block_size,
716                        struct ixgbe_pba *pba)
717 {
718         s32 ret_val;
719         u16 pba_block_size;
720
721         if (pba == NULL)
722                 return IXGBE_ERR_PARAM;
723
724         if (eeprom_buf == NULL) {
725                 ret_val = hw->eeprom.ops.read_buffer(hw, IXGBE_PBANUM0_PTR, 2,
726                                                      &pba->word[0]);
727                 if (ret_val)
728                         return ret_val;
729         } else {
730                 if (eeprom_buf_size > IXGBE_PBANUM1_PTR) {
731                         pba->word[0] = eeprom_buf[IXGBE_PBANUM0_PTR];
732                         pba->word[1] = eeprom_buf[IXGBE_PBANUM1_PTR];
733                 } else {
734                         return IXGBE_ERR_PARAM;
735                 }
736         }
737
738         if (pba->word[0] == IXGBE_PBANUM_PTR_GUARD) {
739                 if (pba->pba_block == NULL)
740                         return IXGBE_ERR_PARAM;
741
742                 ret_val = ixgbe_get_pba_block_size(hw, eeprom_buf,
743                                                    eeprom_buf_size,
744                                                    &pba_block_size);
745                 if (ret_val)
746                         return ret_val;
747
748                 if (pba_block_size > max_pba_block_size)
749                         return IXGBE_ERR_PARAM;
750
751                 if (eeprom_buf == NULL) {
752                         ret_val = hw->eeprom.ops.read_buffer(hw, pba->word[1],
753                                                              pba_block_size,
754                                                              pba->pba_block);
755                         if (ret_val)
```

```
                                return ret_val;
                } else {
                        if (eeprom_buf_size > (u32)(pba->word[1] +
                                                    pba->pba_block[0])) {
                                (void) memcpy(pba->pba_block,
                                       &eeprom_buf[pba->word[1]],
                                       pba_block_size * sizeof(u16));
                        } else {
                                return IXGBE_ERR_PARAM;
                        }
                }
        }

        return IXGBE_SUCCESS;
}

/**
 *  ixgbe_write_pba_raw
 *  @hw: pointer to the HW structure
 *  @eeprom_buf: optional pointer to EEPROM image
 *  @eeprom_buf_size: size of EEPROM image in words
 *  @pba: pointer to PBA structure
 *
 *  Writes PBA to EEPROM image when eeprom_buf is not NULL.
 *  Writes PBA to physical EEPROM device when eeprom_buf is NULL.
 *
 **/
s32 ixgbe_write_pba_raw(struct ixgbe_hw *hw, u16 *eeprom_buf,
                        u32 eeprom_buf_size, struct ixgbe_pba *pba)
{
        s32 ret_val;

        if (pba == NULL)
                return IXGBE_ERR_PARAM;

        if (eeprom_buf == NULL) {
                ret_val = hw->eeprom.ops.write_buffer(hw, IXGBE_PBANUM0_PTR, 2,
                                                      &pba->word[0]);
                if (ret_val)
                        return ret_val;
        } else {
                if (eeprom_buf_size > IXGBE_PBANUM1_PTR) {
                        eeprom_buf[IXGBE_PBANUM0_PTR] = pba->word[0];
                        eeprom_buf[IXGBE_PBANUM1_PTR] = pba->word[1];
                } else {
                        return IXGBE_ERR_PARAM;
                }
        }

        if (pba->word[0] == IXGBE_PBANUM_PTR_GUARD) {
                if (pba->pba_block == NULL)
                        return IXGBE_ERR_PARAM;

                if (eeprom_buf == NULL) {
                        ret_val = hw->eeprom.ops.write_buffer(hw, pba->word[1],
                                                              pba->pba_block[0],
                                                              pba->pba_block);
                        if (ret_val)
                                return ret_val;
                } else {
                        if (eeprom_buf_size > (u32)(pba->word[1] +
                                                    pba->pba_block[0])) {
                                (void) memcpy(&eeprom_buf[pba->word[1]],
                                       pba->pba_block,
                                       pba->pba_block[0] * sizeof(u16));
                        } else {
```



```
                                return IXGBE_ERR_PARAM;
                        }
                }
        }

        return IXGBE_SUCCESS;
}

/**
 *  ixgbe_get_pba_block_size
 *  @hw: pointer to the HW structure
 *  @eeprom_buf: optional pointer to EEPROM image
 *  @eeprom_buf_size: size of EEPROM image in words
 *  @pba_data_size: pointer to output variable
 *
 *  Returns the size of the PBA block in words. Function operates on EEPROM
 *  image if the eeprom_buf pointer is not NULL otherwise it accesses physical
 *  EEPROM device.
 *
 **/
s32 ixgbe_get_pba_block_size(struct ixgbe_hw *hw, u16 *eeprom_buf,
                             u32 eeprom_buf_size, u16 *pba_block_size)
{
        s32 ret_val;
        u16 pba_word[2];
        u16 length;

        DEBUGFUNC("ixgbe_get_pba_block_size");

        if (eeprom_buf == NULL) {
                ret_val = hw->eeprom.ops.read_buffer(hw, IXGBE_PBANUM0_PTR, 2,
                                                     &pba_word[0]);
                if (ret_val)
                        return ret_val;
        } else {
                if (eeprom_buf_size > IXGBE_PBANUM1_PTR) {
                        pba_word[0] = eeprom_buf[IXGBE_PBANUM0_PTR];
                        pba_word[1] = eeprom_buf[IXGBE_PBANUM1_PTR];
                } else {
                        return IXGBE_ERR_PARAM;
                }
        }

        if (pba_word[0] == IXGBE_PBANUM_PTR_GUARD) {
                if (eeprom_buf == NULL) {
                        ret_val = hw->eeprom.ops.read(hw, pba_word[1] + 0,
                                                      &length);
                        if (ret_val)
                                return ret_val;
                } else {
                        if (eeprom_buf_size > pba_word[1])
                                length = eeprom_buf[pba_word[1] + 0];
                        else
                                return IXGBE_ERR_PARAM;
                }

                if (length == 0xFFFF || length == 0)
                        return IXGBE_ERR_PBA_SECTION;
        } else {
                /* PBA number in legacy format, there is no PBA Block. */
                length = 0;
        }

        if (pba_block_size != NULL)
                *pba_block_size = length;
```

```
 888         return IXGBE_SUCCESS;
 889 }

 891 /**
 892  *  ixgbe_get_mac_addr_generic - Generic get MAC address
 893  *  @hw: pointer to hardware structure
 894  *  @mac_addr: Adapter MAC address
 895  *
 896  *  Reads the adapter's MAC address from first Receive Address Register (RAR0)
 897  *  A reset of the adapter must be performed prior to calling this function
 898  *  in order for the MAC address to have been loaded from the EEPROM into RAR0
 899  **/
 900 s32 ixgbe_get_mac_addr_generic(struct ixgbe_hw *hw, u8 *mac_addr)
 901 {
 902         u32 rar_high;
 903         u32 rar_low;
 904         u16 i;

 906         DEBUGFUNC("ixgbe_get_mac_addr_generic");

 908         rar_high = IXGBE_READ_REG(hw, IXGBE_RAH(0));
 909         rar_low = IXGBE_READ_REG(hw, IXGBE_RAL(0));

 911         for (i = 0; i < 4; i++)
 912                 mac_addr[i] = (u8)(rar_low >> (i*8));

 914         for (i = 0; i < 2; i++)
 915                 mac_addr[i+4] = (u8)(rar_high >> (i*8));

 917         return IXGBE_SUCCESS;
 918 }
_____unchanged_portion_omitted_

1451 /**
1452  *  ixgbe_read_eerd_buffer_generic - Read EEPROM word(s) using EERD
1453  *  @hw: pointer to hardware structure
1454  *  @offset: offset of word in the EEPROM to read
1455  *  @words: number of word(s)
1456  *  @data: 16 bit word(s) from the EEPROM
1457  *
1458  *  Reads a 16 bit word(s) from the EEPROM using the EERD register.
1459  **/
1460 s32 ixgbe_read_eerd_buffer_generic(struct ixgbe_hw *hw, u16 offset,
1461                                    u16 words, u16 *data)
1462 {
1463         u32 eerd;
1464         s32 status = IXGBE_SUCCESS;
1465         u32 i;

1467         DEBUGFUNC("ixgbe_read_eerd_buffer_generic");

1469         hw->eeprom.ops.init_params(hw);

1471         if (words == 0) {
1472                 status = IXGBE_ERR_INVALID_ARGUMENT;
1473                 goto out;
1474         }

1476         if (offset >= hw->eeprom.word_size) {
1477                 status = IXGBE_ERR_EEPROM;
1478                 goto out;
1479         }

1481         for (i = 0; i < words; i++) {
1482                 eerd = ((offset + i) << IXGBE_EEPROM_RW_ADDR_SHIFT) |
1272                 eerd = ((offset + i) << IXGBE_EEPROM_RW_ADDR_SHIFT) +
```



```
1483                         IXGBE_EEPROM_RW_REG_START;

1485                 IXGBE_WRITE_REG(hw, IXGBE_EERD, eerd);
1486                 status = ixgbe_poll_eerd_eewr_done(hw, IXGBE_NVM_POLL_READ);

1488                 if (status == IXGBE_SUCCESS) {
1489                         data[i] = (IXGBE_READ_REG(hw, IXGBE_EERD) >>
1490                                    IXGBE_EEPROM_RW_REG_DATA);
1491                 } else {
1492                         DEBUGOUT("Eeprom read timed out\n");
1493                         goto out;
1494                 }
1495         }
1496 out:
1497         return status;
1498 }
_____unchanged_portion_omitted_

2902 /**
2903  *  ixgbe_fc_autoneg - Configure flow control
2904  *  @hw: pointer to hardware structure
2905  *
2906  *  Compares our advertised flow control capabilities to those advertised by
2907  *  our link partner, and determines the proper flow control mode to use.
2908  **/
2909 void ixgbe_fc_autoneg(struct ixgbe_hw *hw)
2910 {
2911         s32 ret_val = IXGBE_ERR_FC_NOT_NEGOTIATED;
2912         ixgbe_link_speed speed;
2913         bool link_up;

2915         DEBUGFUNC("ixgbe_fc_autoneg");

2917         /*
2918          * AN should have completed when the cable was plugged in.
2919          * Look for reasons to bail out.  Bail out if:
2920          * - FC autoneg is disabled, or if
2921          * - link is not up.
2922          */
2923         if (hw->fc.disable_fc_autoneg)
2924                 goto out;

2926         hw->mac.ops.check_link(hw, &speed, &link_up, FALSE);
2927         if (!link_up)
2928                 goto out;

2930         switch (hw->phy.media_type) {
2931         /* Autoneg flow control on fiber adapters */
2932         case ixgbe_media_type_fiber_fixed:
2933         case ixgbe_media_type_fiber:
2934                 if (speed == IXGBE_LINK_SPEED_1GB_FULL)
2935                         ret_val = ixgbe_fc_autoneg_fiber(hw);
2936                 break;

2938         /* Autoneg flow control on backplane adapters */
2939         case ixgbe_media_type_backplane:
2940                 ret_val = ixgbe_fc_autoneg_backplane(hw);
2941                 break;

2943         /* Autoneg flow control on copper adapters */
2944         case ixgbe_media_type_copper:
2945                 if (ixgbe_device_supports_autoneg_fc(hw) == IXGBE_SUCCESS)
2946                         ret_val = ixgbe_fc_autoneg_copper(hw);
2947                 break;

2949         default:
```

```
2950                 break;
2951         }

2953 out:
2954         if (ret_val == IXGBE_SUCCESS) {
2955                 hw->fc.fc_was_autonegged = TRUE;
2956         } else {
2957                 hw->fc.fc_was_autonegged = FALSE;
2958                 hw->fc.current_mode = hw->fc.requested_mode;
2959         }
2960 }
______unchanged_portion_omitted_

3168 /**
3169  *  ixgbe_blink_led_start_generic - Blink LED based on index.
3170  *  @hw: pointer to hardware structure
3171  *  @index: led number to blink
3172  **/
3173 s32 ixgbe_blink_led_start_generic(struct ixgbe_hw *hw, u32 index)
3174 {
3175         ixgbe_link_speed speed = 0;
3176         bool link_up = 0;
3177         u32 autoc_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC);
3178         u32 led_reg = IXGBE_READ_REG(hw, IXGBE_LEDCTL);
3179         s32 ret_val = IXGBE_SUCCESS;

3181         DEBUGFUNC("ixgbe_blink_led_start_generic");

3183         /*
3184          * Link must be up to auto-blink the LEDs;
3185          * Force it if link is down.
3186          */
3187         hw->mac.ops.check_link(hw, &speed, &link_up, FALSE);

3189         if (!link_up) {
3190                 /* Need the SW/FW semaphore around AUTOC writes if 82599 and
3191                  * LESM is on.
3192                  */
3193                 bool got_lock = FALSE;
3194                 if ((hw->mac.type == ixgbe_mac_82599EB) &&
3195                     ixgbe_verify_lesm_fw_enabled_82599(hw)) {
3196                         ret_val = hw->mac.ops.acquire_swfw_sync(hw,
3197                                                 IXGBE_GSSR_MAC_CSR_SM);
3198                         if (ret_val != IXGBE_SUCCESS) {
3199                                 ret_val = IXGBE_ERR_SWFW_SYNC;
3200                                 goto out;
3201                         }
3202                         got_lock = TRUE;
3203                 }

3205                 autoc_reg |= IXGBE_AUTOC_AN_RESTART;
3206                 autoc_reg |= IXGBE_AUTOC_FLU;
3207                 IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc_reg);
3208                 IXGBE_WRITE_FLUSH(hw);

3210                 if (got_lock)
3211                         hw->mac.ops.release_swfw_sync(hw,
3212                                                 IXGBE_GSSR_MAC_CSR_SM);
3213                 msec_delay(10);
3214         }

3216         led_reg &= ~IXGBE_LED_MODE_MASK(index);
3217         led_reg |= IXGBE_LED_BLINK(index);
3218         IXGBE_WRITE_REG(hw, IXGBE_LEDCTL, led_reg);
3219         IXGBE_WRITE_FLUSH(hw);
```



```
3221 out:
3222         return ret_val;
2990         return IXGBE_SUCCESS;
3223 }

3225 /**
3226  *  ixgbe_blink_led_stop_generic - Stop blinking LED based on index.
3227  *  @hw: pointer to hardware structure
3228  *  @index: led number to stop blinking
3229  **/
3230 s32 ixgbe_blink_led_stop_generic(struct ixgbe_hw *hw, u32 index)
3231 {
3232         u32 autoc_reg = IXGBE_READ_REG(hw, IXGBE_AUTOC);
3233         u32 led_reg = IXGBE_READ_REG(hw, IXGBE_LEDCTL);
3234         s32 ret_val = IXGBE_SUCCESS;
3235         bool got_lock = FALSE;

3237         DEBUGFUNC("ixgbe_blink_led_stop_generic");
3238         /* Need the SW/FW semaphore around AUTOC writes if 82599 and
3239          * LESM is on.
3240          */
3241         if ((hw->mac.type == ixgbe_mac_82599EB) &&
3242             ixgbe_verify_lesm_fw_enabled_82599(hw)) {
3243                 ret_val = hw->mac.ops.acquire_swfw_sync(hw,
3244                                                 IXGBE_GSSR_MAC_CSR_SM);
3245                 if (ret_val != IXGBE_SUCCESS) {
3246                         ret_val = IXGBE_ERR_SWFW_SYNC;
3247                         goto out;
3248                 }
3249                 got_lock = TRUE;
3250         }


3253         autoc_reg &= ~IXGBE_AUTOC_FLU;
3254         autoc_reg |= IXGBE_AUTOC_AN_RESTART;
3255         IXGBE_WRITE_REG(hw, IXGBE_AUTOC, autoc_reg);

3257         if (hw->mac.type == ixgbe_mac_82599EB)
3258                 (void) ixgbe_reset_pipeline_82599(hw);

3260         if (got_lock)
3261                 hw->mac.ops.release_swfw_sync(hw, IXGBE_GSSR_MAC_CSR_SM);

3263         led_reg &= ~IXGBE_LED_MODE_MASK(index);
3264         led_reg &= ~IXGBE_LED_BLINK(index);
3265         led_reg |= IXGBE_LED_LINK_ACTIVE << IXGBE_LED_MODE_SHIFT(index);
3266         IXGBE_WRITE_REG(hw, IXGBE_LEDCTL, led_reg);
3267         IXGBE_WRITE_FLUSH(hw);

3269 out:
3270         return ret_val;
3016         return IXGBE_SUCCESS;
3271 }
______unchanged_portion_omitted_

4132 /**
4133  *  ixgbe_calculate_checksum - Calculate checksum for buffer
4134  *  @buffer: pointer to EEPROM
4135  *  @length: size of EEPROM to calculate a checksum for
4136  *  Calculates the checksum for some buffer on a specified length.  The
4137  *  checksum calculated is returned.
4138  **/
4139 u8 ixgbe_calculate_checksum(u8 *buffer, u32 length)
3885 static u8 ixgbe_calculate_checksum(u8 *buffer, u32 length)
4140 {
4141         u32 i;
```

```
4142         u8 sum = 0;

4144         DEBUGFUNC("ixgbe_calculate_checksum");

4146         if (!buffer)
4147                 return 0;

4149         for (i = 0; i < length; i++)
4150                 sum += buffer[i];

4152         return (u8) (0 - sum);
4153 }

4155 /**
4156  *  ixgbe_host_interface_command - Issue command to manageability block
4157  *  @hw: pointer to the HW structure
4158  *  @buffer: contains the command to write and where the return status will
4159  *   be placed
4160  *  @length: length of buffer, must be multiple of 4 bytes
4161  *
4162  *  Communicates with the manageability block.  On success return IXGBE_SUCCESS
4163  *  else return IXGBE_ERR_HOST_INTERFACE_COMMAND.
4164  **/
4165 s32 ixgbe_host_interface_command(struct ixgbe_hw *hw, u32 *buffer,
3911 static s32 ixgbe_host_interface_command(struct ixgbe_hw *hw, u32 *buffer,
4166                                  u32 length)
4167 {
4168         u32 hicr, i, bi;
4169         u32 hdr_size = sizeof(struct ixgbe_hic_hdr);
4170         u8 buf_len, dword_len;

4172         s32 ret_val = IXGBE_SUCCESS;

4174         DEBUGFUNC("ixgbe_host_interface_command");

4176         if (length == 0 || length & 0x3 ||
4177             length > IXGBE_HI_MAX_BLOCK_BYTE_LENGTH) {
4178                 DEBUGOUT("Buffer length failure.\n");
4179                 ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;
4180                 goto out;
4181         }

4183         /* Check that the host interface is enabled. */
4184         hicr = IXGBE_READ_REG(hw, IXGBE_HICR);
4185         if ((hicr & IXGBE_HICR_EN) == 0) {
4186                 DEBUGOUT("IXGBE_HOST_EN bit disabled.\n");
4187                 ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;
4188                 goto out;
4189         }

4191         /* Calculate length in DWORDs */
4192         dword_len = length >> 2;

4194         /*
4195          * The device driver writes the relevant command block
4196          * into the ram area.
4197          */
4198         for (i = 0; i < dword_len; i++)
4199                 IXGBE_WRITE_REG_ARRAY(hw, IXGBE_FLEX_MNG,
4200                                       i, IXGBE_CPU_TO_LE32(buffer[i]));

4202         /* Setting this bit tells the ARC that a new command is pending. */
4203         IXGBE_WRITE_REG(hw, IXGBE_HICR, hicr | IXGBE_HICR_C);

4205         for (i = 0; i < IXGBE_HI_COMMAND_TIMEOUT; i++) {
4206                 hicr = IXGBE_READ_REG(hw, IXGBE_HICR);
```



```
4207                 if (!(hicr & IXGBE_HICR_C))
4208                         break;
4209                 msec_delay(1);
4210         }

4212         /* Check command successful completion. */
4213         if (i == IXGBE_HI_COMMAND_TIMEOUT ||
4214             (!(IXGBE_READ_REG(hw, IXGBE_HICR) & IXGBE_HICR_SV))) {
4215                 DEBUGOUT("Command has failed with no status valid.\n");
4216                 ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;
4217                 goto out;
4218         }

4220         /* Calculate length in DWORDs */
4221         dword_len = hdr_size >> 2;

4223         /* first pull in the header so we know the buffer length */
4224         for (bi = 0; bi < dword_len; bi++) {
4225                 buffer[bi] = IXGBE_READ_REG_ARRAY(hw, IXGBE_FLEX_MNG, bi);
4226                 buffer[bi] = IXGBE_LE32_TO_CPUS(buffer[bi]);
4227         }

4229         /* If there is any thing in data position pull it in */
4230         buf_len = ((struct ixgbe_hic_hdr *)buffer)->buf_len;
4231         if (buf_len == 0)
4232                 goto out;

4234         if (length < (buf_len + hdr_size)) {
4235                 DEBUGOUT("Buffer not large enough for reply message.\n");
4236                 ret_val = IXGBE_ERR_HOST_INTERFACE_COMMAND;
4237                 goto out;
4238         }

4240         /* Calculate length in DWORDs, add 3 for odd lengths */
4241         dword_len = (buf_len + 3) >> 2;

4243         /* Pull in the rest of the buffer (bi is where we left off)*/
4244         for (; bi <= dword_len; bi++) {
4245                 buffer[bi] = IXGBE_READ_REG_ARRAY(hw, IXGBE_FLEX_MNG, bi);
4246                 buffer[bi] = IXGBE_LE32_TO_CPUS(buffer[bi]);
4247         }

4249 out:
4250         return ret_val;
4251 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7722 Tue Apr 30 09:54:23 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_common.h
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
```

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

```
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

#ifndef _IXGBE_COMMON_H_
#define _IXGBE_COMMON_H_

#include "ixgbe_type.h"
#ifdef lint
/* Use "hw" somehow... */
#define IXGBE_WRITE_REG64(hw, reg, value) hw = hw
#else
#define IXGBE_WRITE_REG64(hw, reg, value) \
        do { \
                IXGBE_WRITE_REG(hw, reg, (u32) value); \
                IXGBE_WRITE_REG(hw, reg + 4, (u32) (value >> 32)); \
        } while (0)
#endif
#if !defined(NO_READ_PBA_RAW) || !defined(NO_WRITE_PBA_RAW)
struct ixgbe_pba {
        u16 word[2];
        u16 *pba_block;
};
#endif

u16 ixgbe_get_pcie_msix_count_generic(struct ixgbe_hw *hw);

s32 ixgbe_init_ops_generic(struct ixgbe_hw *hw);
s32 ixgbe_init_hw_generic(struct ixgbe_hw *hw);
s32 ixgbe_start_hw_generic(struct ixgbe_hw *hw);
```



```
s32 ixgbe_start_hw_gen2(struct ixgbe_hw *hw);
s32 ixgbe_clear_hw_cntrs_generic(struct ixgbe_hw *hw);
s32 ixgbe_read_pba_num_generic(struct ixgbe_hw *hw, u32 *pba_num);
s32 ixgbe_read_pba_string_generic(struct ixgbe_hw *hw, u8 *pba_num,
                                  u32 pba_num_size);
s32 ixgbe_read_pba_raw(struct ixgbe_hw *hw, u16 *eeprom_buf,
                       u32 eeprom_buf_size, u16 max_pba_block_size,
                       struct ixgbe_pba *pba);
s32 ixgbe_write_pba_raw(struct ixgbe_hw *hw, u16 *eeprom_buf,
                        u32 eeprom_buf_size, struct ixgbe_pba *pba);
s32 ixgbe_get_pba_block_size(struct ixgbe_hw *hw, u16 *eeprom_buf,
                             u32 eeprom_buf_size, u16 *pba_block_size);
s32 ixgbe_get_mac_addr_generic(struct ixgbe_hw *hw, u8 *mac_addr);
s32 ixgbe_get_bus_info_generic(struct ixgbe_hw *hw);
void ixgbe_set_lan_id_multi_port_pcie(struct ixgbe_hw *hw);
s32 ixgbe_stop_adapter_generic(struct ixgbe_hw *hw);

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

s32 ixgbe_init_eeprom_params_generic(struct ixgbe_hw *hw);
s32 ixgbe_write_eeprom_generic(struct ixgbe_hw *hw, u16 offset, u16 data);
s32 ixgbe_write_eeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
                                               u16 words, u16 *data);
s32 ixgbe_read_eerd_generic(struct ixgbe_hw *hw, u16 offset, u16 *data);
s32 ixgbe_read_eerd_buffer_generic(struct ixgbe_hw *hw, u16 offset,
                                   u16 words, u16 *data);
s32 ixgbe_write_eewr_generic(struct ixgbe_hw *hw, u16 offset, u16 data);
s32 ixgbe_write_eewr_buffer_generic(struct ixgbe_hw *hw, u16 offset,
                                    u16 words, u16 *data);
s32 ixgbe_read_eeprom_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
                                       u16 *data);
s32 ixgbe_read_eeprom_buffer_bit_bang_generic(struct ixgbe_hw *hw, u16 offset,
                                              u16 words, u16 *data);
u16 ixgbe_calc_eeprom_checksum_generic(struct ixgbe_hw *hw);
s32 ixgbe_validate_eeprom_checksum_generic(struct ixgbe_hw *hw,
                                           u16 *checksum_val);
s32 ixgbe_update_eeprom_checksum_generic(struct ixgbe_hw *hw);
s32 ixgbe_poll_eerd_eewr_done(struct ixgbe_hw *hw, u32 ee_reg);

s32 ixgbe_set_rar_generic(struct ixgbe_hw *hw, u32 index, u8 *addr, u32 vmdq,
                          u32 enable_addr);
s32 ixgbe_clear_rar_generic(struct ixgbe_hw *hw, u32 index);
s32 ixgbe_init_rx_addrs_generic(struct ixgbe_hw *hw);
s32 ixgbe_update_mc_addr_list_generic(struct ixgbe_hw *hw, u8 *mc_addr_list,
                                      u32 mc_addr_count,
                                      ixgbe_mc_addr_itr func, bool clear);
s32 ixgbe_update_uc_addr_list_generic(struct ixgbe_hw *hw, u8 *addr_list,
                                      u32 addr_count, ixgbe_mc_addr_itr func);
s32 ixgbe_enable_mc_generic(struct ixgbe_hw *hw);
s32 ixgbe_disable_mc_generic(struct ixgbe_hw *hw);
s32 ixgbe_enable_rx_dma_generic(struct ixgbe_hw *hw, u32 regval);
s32 ixgbe_disable_sec_rx_path_generic(struct ixgbe_hw *hw);
s32 ixgbe_enable_sec_rx_path_generic(struct ixgbe_hw *hw);

s32 ixgbe_fc_enable_generic(struct ixgbe_hw *hw);
s32 ixgbe_device_supports_autoneg_fc(struct ixgbe_hw *hw);
void ixgbe_fc_autoneg(struct ixgbe_hw *hw);

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

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

```
127 s32 ixgbe_get_san_mac_addr_generic(struct ixgbe_hw *hw, u8 *san_mac_addr);
128 s32 ixgbe_set_san_mac_addr_generic(struct ixgbe_hw *hw, u8 *san_mac_addr);

130 s32 ixgbe_set_vmdq_generic(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
131 s32 ixgbe_set_vmdq_san_mac_generic(struct ixgbe_hw *hw, u32 vmdq);
132 s32 ixgbe_clear_vmdq_generic(struct ixgbe_hw *hw, u32 rar, u32 vmdq);
133 s32 ixgbe_insert_mac_addr_generic(struct ixgbe_hw *hw, u8 *addr, u32 vmdq);
134 s32 ixgbe_init_uta_tables_generic(struct ixgbe_hw *hw);
135 s32 ixgbe_set_vfta_generic(struct ixgbe_hw *hw, u32 vlan,
136                            u32 vind, bool vlan_on);
137 s32 ixgbe_set_vlvf_generic(struct ixgbe_hw *hw, u32 vlan, u32 vind,
138                            bool vlan_on, bool *vfta_changed);
139 s32 ixgbe_clear_vfta_generic(struct ixgbe_hw *hw);
140 s32 ixgbe_find_vlvf_slot(struct ixgbe_hw *hw, u32 vlan);

142 s32 ixgbe_check_mac_link_generic(struct ixgbe_hw *hw,
143                                  ixgbe_link_speed *speed,
144                                  bool *link_up, bool link_up_wait_to_complete);

146 s32 ixgbe_get_wwn_prefix_generic(struct ixgbe_hw *hw, u16 *wwnn_prefix,
147                                  u16 *wwpn_prefix);

149 s32 ixgbe_get_fcoe_boot_status_generic(struct ixgbe_hw *hw, u16 *bs);
150 void ixgbe_set_mac_anti_spoofing(struct ixgbe_hw *hw, bool enable, int pf);
151 void ixgbe_set_vlan_anti_spoofing(struct ixgbe_hw *hw, bool enable, int vf);
152 s32 ixgbe_get_device_caps_generic(struct ixgbe_hw *hw, u16 *device_caps);
153 void ixgbe_set_rxpba_generic(struct ixgbe_hw *hw, int num_pb, u32 headroom,
154                              int strategy);
155 void ixgbe_enable_relaxed_ordering_gen2(struct ixgbe_hw *hw);
156 s32 ixgbe_set_fw_drv_ver_generic(struct ixgbe_hw *hw, u8 maj, u8 min,
157                                  u8 build, u8 ver);
158 u8 ixgbe_calculate_checksum(u8 *buffer, u32 length);
159 s32 ixgbe_host_interface_command(struct ixgbe_hw *hw, u32 *buffer,
160                                  u32 length);
161 void ixgbe_clear_tx_pending(struct ixgbe_hw *hw);

163 extern s32 ixgbe_reset_pipeline_82599(struct ixgbe_hw *hw);

165 #endif /* IXGBE_COMMON */
```

```
**********************************************************
  143294 Tue Apr 30 09:54:23 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_main.c
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
______unchanged_portion_omitted_

3290 /*
3291  * ixgbe_driver_setup_link - Using the link properties to setup the link.
3292  */
3293 int
3294 ixgbe_driver_setup_link(ixgbe_t *ixgbe, boolean_t setup_hw)
3295 {
3296         u32 autoneg_advertised = 0;

3298         /*
3299          * No half duplex support with 10Gb parts
3300          */
3301         if (ixgbe->param_adv_10000fdx_cap == 1)
3302                 autoneg_advertised |= IXGBE_LINK_SPEED_10GB_FULL;

3304         if (ixgbe->param_adv_1000fdx_cap == 1)
3305                 autoneg_advertised |= IXGBE_LINK_SPEED_1GB_FULL;

3307         if (ixgbe->param_adv_100fdx_cap == 1)
3308                 autoneg_advertised |= IXGBE_LINK_SPEED_100_FULL;

3310         if (ixgbe->param_adv_autoneg_cap == 1 && autoneg_advertised == 0) {
3311                 ixgbe_notice(ixgbe, "Invalid link settings. Setup link "
3312                     "to autonegotiation with full link capabilities.");

3314                 autoneg_advertised = IXGBE_LINK_SPEED_10GB_FULL |
3315                     IXGBE_LINK_SPEED_1GB_FULL |
3316                     IXGBE_LINK_SPEED_100_FULL;
3317         }

3319         if (setup_hw) {
3320                 if (ixgbe_setup_link(&ixgbe->hw, autoneg_advertised,
3321                    B_TRUE) != IXGBE_SUCCESS) {
3321                     ixgbe->param_adv_autoneg_cap, B_TRUE) != IXGBE_SUCCESS) {
3322                         ixgbe_notice(ixgbe, "Setup link failed on this "
3323                             "device.");
3324                         return (IXGBE_FAILURE);
3325                 }
3326         }

3328         return (IXGBE_SUCCESS);
3329 }
______unchanged_portion_omitted_

3403 /*
3404  * ixgbe_sfp_check - sfp module processing done in taskq only for 82599.
3405  */
3406 static void
3407 ixgbe_sfp_check(void *arg)
3408 {
3409         ixgbe_t *ixgbe = (ixgbe_t *)arg;
3410         uint32_t eicr = ixgbe->eicr;
3411         struct ixgbe_hw *hw = &ixgbe->hw;

3413         mutex_enter(&ixgbe->gen_lock);
3414         if (eicr & IXGBE_EICR_GPI_SDP1) {
3415                 /* clear the interrupt */
3416                 IXGBE_WRITE_REG(hw, IXGBE_EICR, IXGBE_EICR_GPI_SDP1);

3418                 /* if link up, do multispeed fiber setup */
```



```
3419                 (void) ixgbe_setup_link(hw, IXGBE_LINK_SPEED_82599_AUTONEG,
3420                     B_TRUE);
3420                     B_TRUE, B_TRUE);
3421                 ixgbe_driver_link_check(ixgbe);
3422                 ixgbe_get_hw_state(ixgbe);
3423         } else if (eicr & IXGBE_EICR_GPI_SDP2) {
3424                 /* clear the interrupt */
3425                 IXGBE_WRITE_REG(hw, IXGBE_EICR, IXGBE_EICR_GPI_SDP2);

3427                 /* if link up, do sfp module setup */
3428                 (void) hw->mac.ops.setup_sfp(hw);

3430                 /* do multispeed fiber setup */
3431                 (void) ixgbe_setup_link(hw, IXGBE_LINK_SPEED_82599_AUTONEG,
3432                     B_TRUE);
3432                     B_TRUE, B_TRUE);
3433                 ixgbe_driver_link_check(ixgbe);
3434                 ixgbe_get_hw_state(ixgbe);
3435         }
3436         mutex_exit(&ixgbe->gen_lock);

3438         /*
3439          * We need to fully re-check the link later.
3440          */
3441         ixgbe->link_check_complete = B_FALSE;
3442         ixgbe->link_check_hrtime = gethrtime() +
3443             (IXGBE_LINK_UP_TIME * 100000000ULL);
3444 }
______unchanged_portion_omitted_

4010 /*
4011  * ixgbe_set_internal_mac_loopback - Set the internal MAC loopback mode.
4012  */
4013 static void
4014 ixgbe_set_internal_mac_loopback(ixgbe_t *ixgbe)
4015 {
4016         struct ixgbe_hw *hw;
4017         uint32_t reg;
4018         uint8_t atlas;

4020         hw = &ixgbe->hw;

4022         /*
4023          * Setup MAC loopback
4024          */
4025         reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_HLREG0);
4026         reg |= IXGBE_HLREG0_LPBK;
4027         IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_HLREG0, reg);

4029         reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_AUTOC);
4030         reg &= ~IXGBE_AUTOC_LMS_MASK;
4031         IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_AUTOC, reg);

4033         /*
4034          * Disable Atlas Tx lanes to keep packets in loopback and not on wire
4035          */
4036         switch (hw->mac.type) {
4037         case ixgbe_mac_82598EB:
4038                 (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_LPBK,
4039                     &atlas);
4040                 atlas |= IXGBE_ATLAS_PDN_TX_REG_EN;
4041                 (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_LPBK,
4042                     atlas);

4044                 (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_10G,
4045                     &atlas);
```

```
4046                atlas |= IXGBE_ATLAS_PDN_TX_10G_QL_ALL;
4047                (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_10G,
4048                    atlas);

4050                (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_1G,
4051                    &atlas);
4052                atlas |= IXGBE_ATLAS_PDN_TX_1G_QL_ALL;
4053                (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_1G,
4054                    atlas);

4056                (void) ixgbe_read_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_AN,
4057                    &atlas);
4058                atlas |= IXGBE_ATLAS_PDN_TX_AN_QL_ALL;
4059                (void) ixgbe_write_analog_reg8(&ixgbe->hw, IXGBE_ATLAS_PDN_AN,
4060                    atlas);
4061                break;

4063        case ixgbe_mac_82599EB:
4064        case ixgbe_mac_X540:
4065                reg = IXGBE_READ_REG(&ixgbe->hw, IXGBE_AUTOC);
4066                reg |= (IXGBE_AUTOC_FLU |
4067                    IXGBE_AUTOC_10G_KX4);
4068                IXGBE_WRITE_REG(&ixgbe->hw, IXGBE_AUTOC, reg);

4070                (void) ixgbe_setup_link(&ixgbe->hw, IXGBE_LINK_SPEED_10GB_FULL,
4071                    B_TRUE);
4071                    B_FALSE, B_TRUE);
4072                break;

4074        default:
4075                break;
4076        }
4077 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    5876 Tue Apr 30 09:54:23 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_mbx.h
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
/*****************************************************************************

  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

*****************************************************************************/
/*$FreeBSD$*/

#ifndef _IXGBE_MBX_H_
#define _IXGBE_MBX_H_

#include "ixgbe_type.h"

#define IXGBE_VFMAILBOX_SIZE    16 /* 16 32 bit words - 64 bytes */
#define IXGBE_ERR_MBX           -100

#define IXGBE_VFMAILBOX         0x002FC
#define IXGBE_VFMBMEM           0x00200

/* Define mailbox register bits */
#define IXGBE_VFMAILBOX_REQ     0x00000001 /* Request for PF Ready bit */
#define IXGBE_VFMAILBOX_ACK     0x00000002 /* Ack PF message received */
#define IXGBE_VFMAILBOX_VFU     0x00000004 /* VF owns the mailbox buffer */
#define IXGBE_VFMAILBOX_PFU     0x00000008 /* PF owns the mailbox buffer */
#define IXGBE_VFMAILBOX_PFSTS   0x00000010 /* PF wrote a message in the MB */
#define IXGBE_VFMAILBOX_PFACK   0x00000020 /* PF ack the previous VF msg */
#define IXGBE_VFMAILBOX_RSTI    0x00000040 /* PF has reset indication */
#define IXGBE_VFMAILBOX_RSTD    0x00000080 /* PF has indicated reset done */
#define IXGBE_VFMAILBOX_R2C_BITS        0x000000B0 /* All read to clear bits */

#define IXGBE_PFMAILBOX_STS     0x00000001 /* Initiate message send to VF */
#define IXGBE_PFMAILBOX_ACK     0x00000002 /* Ack message recv'd from VF */
#define IXGBE_PFMAILBOX_VFU     0x00000004 /* VF owns the mailbox buffer */
#define IXGBE_PFMAILBOX_PFU     0x00000008 /* PF owns the mailbox buffer */
```



```
#define IXGBE_PFMAILBOX_RVFU    0x00000010 /* Reset VFU - used when VF stuck */

#define IXGBE_MBVFICR_VFREQ_MASK        0x0000FFFF /* bits for VF messages */
#define IXGBE_MBVFICR_VFREQ_VF1         0x00000001 /* bit for VF 1 message */
#define IXGBE_MBVFICR_VFACK_MASK        0xFFFF0000 /* bits for VF acks */
#define IXGBE_MBVFICR_VFACK_VF1         0x00010000 /* bit for VF 1 ack */


/* If it's a IXGBE_VF_* msg then it originates in the VF and is sent to the
 * PF.  The reverse is TRUE if it is IXGBE_PF_*.
 * Message ACK's are the value or'd with 0xF0000000
 */
#define IXGBE_VT_MSGTYPE_ACK    0x80000000 /* Messages below or'd with
                                            * this are the ACK */
#define IXGBE_VT_MSGTYPE_NACK   0x40000000 /* Messages below or'd with
                                            * this are the NACK */
#define IXGBE_VT_MSGTYPE_CTS    0x20000000 /* Indicates that VF is still
                                            * clear to send requests */
#define IXGBE_VT_MSGINFO_SHIFT  16
/* bits 23:16 are used for extra info for certain messages */
#define IXGBE_VT_MSGINFO_MASK   (0xFF << IXGBE_VT_MSGINFO_SHIFT)

#define IXGBE_VF_RESET          0x01 /* VF requests reset */
#define IXGBE_VF_SET_MAC_ADDR   0x02 /* VF requests PF to set MAC addr */
#define IXGBE_VF_SET_MULTICAST  0x03 /* VF requests PF to set MC addr */
#define IXGBE_VF_SET_VLAN       0x04 /* VF requests PF to set VLAN */

/* mailbox API, version 1.0 VF requests */
#define IXGBE_VF_SET_LPE        0x05 /* VF requests PF to set VMOLR.LPE */
#define IXGBE_VF_SET_MACVLAN    0x06 /* VF requests PF for unicast filter */
#define IXGBE_VF_API_NEGOTIATE  0x08 /* negotiate API version */

/* mailbox API, version 1.1 VF requests */
#define IXGBE_VF_GET_QUEUES     0x09 /* get queue configuration */

/* GET_QUEUES return data indices within the mailbox */
#define IXGBE_VF_TX_QUEUES      1       /* number of Tx queues supported */
#define IXGBE_VF_RX_QUEUES      2       /* number of Rx queues supported */
#define IXGBE_VF_TRANS_VLAN     3       /* Indication of port vlan */
#define IXGBE_VF_DEF_QUEUE      4       /* Default queue offset */

/* length of permanent address message returned from PF */
#define IXGBE_VF_PERMADDR_MSG_LEN       4
/* word in permanent address message with the current multicast type */
#define IXGBE_VF_MC_TYPE_WORD           3

#define IXGBE_PF_CONTROL_MSG            0x0100 /* PF control message */


#define IXGBE_VF_MBX_INIT_TIMEOUT       2000 /* number of retries on mailbox */
#define IXGBE_VF_MBX_INIT_DELAY         500  /* microseconds between retries */

s32 ixgbe_read_mbx(struct ixgbe_hw *, u32 *, u16, u16);
s32 ixgbe_write_mbx(struct ixgbe_hw *, u32 *, u16, u16);
s32 ixgbe_read_posted_mbx(struct ixgbe_hw *, u32 *, u16, u16);
s32 ixgbe_write_posted_mbx(struct ixgbe_hw *, u32 *, u16, u16);
s32 ixgbe_check_for_msg(struct ixgbe_hw *, u16);
s32 ixgbe_check_for_ack(struct ixgbe_hw *, u16);
s32 ixgbe_check_for_rst(struct ixgbe_hw *, u16);
void ixgbe_init_mbx_ops_generic(struct ixgbe_hw *hw);
void ixgbe_init_mbx_params_vf(struct ixgbe_hw *);
void ixgbe_init_mbx_params_pf(struct ixgbe_hw *);

#endif /* _IXGBE_MBX_H_ */
```

```
**********************************************************
   51473 Tue Apr 30 09:54:24 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_phy.c
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
/******************************************************************************
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

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

static void ixgbe_i2c_start(struct ixgbe_hw *hw);
static void ixgbe_i2c_stop(struct ixgbe_hw *hw);
static s32 ixgbe_clock_in_i2c_byte(struct ixgbe_hw *hw, u8 *data);
static s32 ixgbe_clock_out_i2c_byte(struct ixgbe_hw *hw, u8 data);
static s32 ixgbe_get_i2c_ack(struct ixgbe_hw *hw);
static s32 ixgbe_clock_in_i2c_bit(struct ixgbe_hw *hw, bool *data);
static s32 ixgbe_clock_out_i2c_bit(struct ixgbe_hw *hw, bool data);
static void ixgbe_raise_i2c_clk(struct ixgbe_hw *hw, u32 *i2cctl);
static void ixgbe_lower_i2c_clk(struct ixgbe_hw *hw, u32 *i2cctl);
static s32 ixgbe_set_i2c_data(struct ixgbe_hw *hw, u32 *i2cctl, bool data);
static bool ixgbe_get_i2c_data(u32 *i2cctl);
static s32 ixgbe_read_i2c_sff8472_generic(struct ixgbe_hw *hw, u8 byte_offset,
                                          u8 *sff8472_data);

/**
 *  ixgbe_init_phy_ops_generic - Inits PHY function ptrs
 *  @hw: pointer to the hardware structure
 *
 *  Initialize the function pointers.
 **/
s32 ixgbe_init_phy_ops_generic(struct ixgbe_hw *hw)
{
```



```
        struct ixgbe_phy_info *phy = &hw->phy;

        DEBUGFUNC("ixgbe_init_phy_ops_generic");

        /* PHY */
        phy->ops.identify = &ixgbe_identify_phy_generic;
        phy->ops.reset = &ixgbe_reset_phy_generic;
        phy->ops.read_reg = &ixgbe_read_phy_reg_generic;
        phy->ops.write_reg = &ixgbe_write_phy_reg_generic;
        phy->ops.setup_link = &ixgbe_setup_phy_link_generic;
        phy->ops.setup_link_speed = &ixgbe_setup_phy_link_speed_generic;
        phy->ops.check_link = NULL;
        phy->ops.get_firmware_version = ixgbe_get_phy_firmware_version_generic;
        phy->ops.read_i2c_byte = &ixgbe_read_i2c_byte_generic;
        phy->ops.write_i2c_byte = &ixgbe_write_i2c_byte_generic;
        phy->ops.read_i2c_sff8472 = &ixgbe_read_i2c_sff8472_generic;
        phy->ops.read_i2c_eeprom = &ixgbe_read_i2c_eeprom_generic;
        phy->ops.write_i2c_eeprom = &ixgbe_write_i2c_eeprom_generic;
        phy->ops.i2c_bus_clear = &ixgbe_i2c_bus_clear;
        phy->ops.identify_sfp = &ixgbe_identify_module_generic;
        phy->sfp_type = ixgbe_sfp_type_unknown;
        phy->ops.check_overtemp = &ixgbe_tn_check_overtemp;
        return IXGBE_SUCCESS;
}
_____unchanged_portion_omitted_

/**
 *  ixgbe_setup_phy_link_speed_generic - Sets the auto advertised capabilities
 *  @hw: pointer to hardware structure
 *  @speed: new link speed
 *  @autoneg: TRUE if autonegotiation enabled
 **/
s32 ixgbe_setup_phy_link_speed_generic(struct ixgbe_hw *hw,
                                       ixgbe_link_speed speed,
                                       bool autoneg,
                                       bool autoneg_wait_to_complete)
{
        UNREFERENCED_1PARAMETER(autoneg_wait_to_complete);
        UNREFERENCED_2PARAMETER(autoneg, autoneg_wait_to_complete);

        DEBUGFUNC("ixgbe_setup_phy_link_speed_generic");

        /*
         * Clear autoneg_advertised and set new values based on input link
         * speed.
         */
        hw->phy.autoneg_advertised = 0;

        if (speed & IXGBE_LINK_SPEED_10GB_FULL)
                hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_10GB_FULL;

        if (speed & IXGBE_LINK_SPEED_1GB_FULL)
                hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_1GB_FULL;

        if (speed & IXGBE_LINK_SPEED_100_FULL)
                hw->phy.autoneg_advertised |= IXGBE_LINK_SPEED_100_FULL;

        /* Setup link based on the new speed settings */
        hw->phy.ops.setup_link(hw);

        return IXGBE_SUCCESS;
}
_____unchanged_portion_omitted_

/**
 *  ixgbe_identify_sfp_module_generic - Identifies SFP modules
```

```
 *  @hw: pointer to hardware structure
 *
 *  Searches for and identifies the SFP module and assigns appropriate PHY type.
 **/
s32 ixgbe_identify_sfp_module_generic(struct ixgbe_hw *hw)
{
        s32 status = IXGBE_ERR_PHY_ADDR_INVALID;
        u32 vendor_oui = 0;
        enum ixgbe_sfp_type stored_sfp_type = hw->phy.sfp_type;
        u8 identifier = 0;
        u8 comp_codes_1g = 0;
        u8 comp_codes_10g = 0;
        u8 oui_bytes[3] = {0, 0, 0};
        u8 cable_tech = 0;
        u8 cable_spec = 0;
        u16 enforce_sfp = 0;

        DEBUGFUNC("ixgbe_identify_sfp_module_generic");

        if (hw->mac.ops.get_media_type(hw) != ixgbe_media_type_fiber) {
                hw->phy.sfp_type = ixgbe_sfp_type_not_present;
                status = IXGBE_ERR_SFP_NOT_PRESENT;
                goto out;
        }

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

        if (status != IXGBE_SUCCESS)
        if (status == IXGBE_ERR_SWFW_SYNC ||
            status == IXGBE_ERR_I2C ||
            status == IXGBE_ERR_SFP_NOT_PRESENT)
                goto err_read_i2c_eeprom;

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

        if (identifier != IXGBE_SFF_IDENTIFIER_SFP) {
                hw->phy.type = ixgbe_phy_sfp_unsupported;
                status = IXGBE_ERR_SFP_NOT_SUPPORTED;
        } else {
                status = hw->phy.ops.read_i2c_eeprom(hw,
                                                     IXGBE_SFF_1GBE_COMP_CODES,
                                                     &comp_codes_1g);

                if (status != IXGBE_SUCCESS)
                if (status == IXGBE_ERR_SWFW_SYNC ||
                    status == IXGBE_ERR_I2C ||
                    status == IXGBE_ERR_SFP_NOT_PRESENT)
                        goto err_read_i2c_eeprom;

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

                if (status != IXGBE_SUCCESS)
                if (status == IXGBE_ERR_SWFW_SYNC ||
                    status == IXGBE_ERR_I2C ||
                    status == IXGBE_ERR_SFP_NOT_PRESENT)
                        goto err_read_i2c_eeprom;
                status = hw->phy.ops.read_i2c_eeprom(hw,
                                                     IXGBE_SFF_CABLE_TECHNOLOGY,
                                                     &cable_tech);

                if (status != IXGBE_SUCCESS)
```



```
                if (status == IXGBE_ERR_SWFW_SYNC ||
                    status == IXGBE_ERR_I2C ||
                    status == IXGBE_ERR_SFP_NOT_PRESENT)
                        goto err_read_i2c_eeprom;

                 /* ID Module
                  * =========
                  * 0   SFP_DA_CU
                  * 1   SFP_SR
                  * 2   SFP_LR
                  * 3   SFP_DA_CORE0 - 82599-specific
                  * 4   SFP_DA_CORE1 - 82599-specific
                  * 5   SFP_SR/LR_CORE0 - 82599-specific
                  * 6   SFP_SR/LR_CORE1 - 82599-specific
                  * 7   SFP_act_lmt_DA_CORE0 - 82599-specific
                  * 8   SFP_act_lmt_DA_CORE1 - 82599-specific
                  * 9   SFP_1g_cu_CORE0 - 82599-specific
                  * 10  SFP_1g_cu_CORE1 - 82599-specific
                  * 11  SFP_1g_sx_CORE0 - 82599-specific
                  * 12  SFP_1g_sx_CORE1 - 82599-specific
                  */
                if (hw->mac.type == ixgbe_mac_82598EB) {
                        if (cable_tech & IXGBE_SFF_DA_PASSIVE_CABLE)
                                hw->phy.sfp_type = ixgbe_sfp_type_da_cu;
                        else if (comp_codes_10g & IXGBE_SFF_10GBASESR_CAPABLE)
                                hw->phy.sfp_type = ixgbe_sfp_type_sr;
                        else if (comp_codes_10g & IXGBE_SFF_10GBASELR_CAPABLE)
                                hw->phy.sfp_type = ixgbe_sfp_type_lr;
                        else
                                hw->phy.sfp_type = ixgbe_sfp_type_unknown;
                } else if (hw->mac.type == ixgbe_mac_82599EB) {
                        if (cable_tech & IXGBE_SFF_DA_PASSIVE_CABLE) {
                                if (hw->bus.lan_id == 0)
                                        hw->phy.sfp_type =
                                                ixgbe_sfp_type_da_cu_core0;
                                else
                                        hw->phy.sfp_type =
                                                ixgbe_sfp_type_da_cu_core1;
                        } else if (cable_tech & IXGBE_SFF_DA_ACTIVE_CABLE) {
                                hw->phy.ops.read_i2c_eeprom(
                                                hw, IXGBE_SFF_CABLE_SPEC_COMP,
                                                &cable_spec);
                                if (cable_spec &
                                    IXGBE_SFF_DA_SPEC_ACTIVE_LIMITING) {
                                        if (hw->bus.lan_id == 0)
                                                hw->phy.sfp_type =
                                                ixgbe_sfp_type_da_act_lmt_core0;
                                        else
                                                hw->phy.sfp_type =
                                                ixgbe_sfp_type_da_act_lmt_core1;
                                } else {
                                        hw->phy.sfp_type =
                                                        ixgbe_sfp_type_unknown;
                                }
                        } else if (comp_codes_10g &
                                   (IXGBE_SFF_10GBASESR_CAPABLE |
                                    IXGBE_SFF_10GBASELR_CAPABLE)) {
                                if (hw->bus.lan_id == 0)
                                        hw->phy.sfp_type =
                                                        ixgbe_sfp_type_srlr_core0;
                                else
                                        hw->phy.sfp_type =
                                                        ixgbe_sfp_type_srlr_core1;
                        } else if (comp_codes_1g & IXGBE_SFF_1GBASET_CAPABLE) {
                                if (hw->bus.lan_id == 0)
                                        hw->phy.sfp_type =
```

```
1070                                            ixgbe_sfp_type_1g_cu_core0;
1071                                    else
1072                                            hw->phy.sfp_type =
1073                                                    ixgbe_sfp_type_1g_cu_core1;
1074                            } else if (comp_codes_1g & IXGBE_SFF_1GBASESX_CAPABLE) {
1075                                    if (hw->bus.lan_id == 0)
1076                                            hw->phy.sfp_type =
1077                                                    ixgbe_sfp_type_1g_sx_core0;
1078                                    else
1079                                            hw->phy.sfp_type =
1080                                                    ixgbe_sfp_type_1g_sx_core1;
1081                            } else {
1082                                    hw->phy.sfp_type = ixgbe_sfp_type_unknown;
1083                            }
1084                    }

1086                    if (hw->phy.sfp_type != stored_sfp_type)
1087                            hw->phy.sfp_setup_needed = TRUE;

1089                    /* Determine if the SFP+ PHY is dual speed or not. */
1090                    hw->phy.multispeed_fiber = FALSE;
1091                    if (((comp_codes_1g & IXGBE_SFF_1GBASESX_CAPABLE) &&
1092                       (comp_codes_10g & IXGBE_SFF_10GBASESR_CAPABLE)) ||
1093                       ((comp_codes_1g & IXGBE_SFF_1GBASELX_CAPABLE) &&
1094                       (comp_codes_10g & IXGBE_SFF_10GBASELR_CAPABLE)))
1095                            hw->phy.multispeed_fiber = TRUE;

1097                    /* Determine PHY vendor */
1098                    if (hw->phy.type != ixgbe_phy_nl) {
1099                            hw->phy.id = identifier;
1100                            status = hw->phy.ops.read_i2c_eeprom(hw,
1101                                                    IXGBE_SFF_VENDOR_OUI_BYTE0,
1102                                                    &oui_bytes[0]);

1104                            if (status != IXGBE_SUCCESS)
1111                            if (status == IXGBE_ERR_SWFW_SYNC ||
1112                                status == IXGBE_ERR_I2C ||
1113                                status == IXGBE_ERR_SFP_NOT_PRESENT)
1105                                    goto err_read_i2c_eeprom;

1107                            status = hw->phy.ops.read_i2c_eeprom(hw,
1108                                                    IXGBE_SFF_VENDOR_OUI_BYTE1,
1109                                                    &oui_bytes[1]);

1111                            if (status != IXGBE_SUCCESS)
1120                            if (status == IXGBE_ERR_SWFW_SYNC ||
1121                                status == IXGBE_ERR_I2C ||
1122                                status == IXGBE_ERR_SFP_NOT_PRESENT)
1112                                    goto err_read_i2c_eeprom;

1114                            status = hw->phy.ops.read_i2c_eeprom(hw,
1115                                                    IXGBE_SFF_VENDOR_OUI_BYTE2,
1116                                                    &oui_bytes[2]);

1118                            if (status != IXGBE_SUCCESS)
1129                            if (status == IXGBE_ERR_SWFW_SYNC ||
1130                                status == IXGBE_ERR_I2C ||
1131                                status == IXGBE_ERR_SFP_NOT_PRESENT)
1119                                    goto err_read_i2c_eeprom;

1121                            vendor_oui =
1122                              ((oui_bytes[0] << IXGBE_SFF_VENDOR_OUI_BYTE0_SHIFT) |
1123                               (oui_bytes[1] << IXGBE_SFF_VENDOR_OUI_BYTE1_SHIFT) |
1124                               (oui_bytes[2] << IXGBE_SFF_VENDOR_OUI_BYTE2_SHIFT));

1126                            switch (vendor_oui) {
```



```
1127                            case IXGBE_SFF_VENDOR_OUI_TYCO:
1128                                    if (cable_tech & IXGBE_SFF_DA_PASSIVE_CABLE)
1129                                            hw->phy.type =
1130                                                    ixgbe_phy_sfp_passive_tyco;
1131                                    break;
1132                            case IXGBE_SFF_VENDOR_OUI_FTL:
1133                                    if (cable_tech & IXGBE_SFF_DA_ACTIVE_CABLE)
1134                                            hw->phy.type = ixgbe_phy_sfp_ftl_active;
1135                                    else
1136                                            hw->phy.type = ixgbe_phy_sfp_ftl;
1137                                    break;
1138                            case IXGBE_SFF_VENDOR_OUI_AVAGO:
1139                                    hw->phy.type = ixgbe_phy_sfp_avago;
1140                                    break;
1141                            case IXGBE_SFF_VENDOR_OUI_INTEL:
1142                                    hw->phy.type = ixgbe_phy_sfp_intel;
1143                                    break;
1144                            default:
1145                                    if (cable_tech & IXGBE_SFF_DA_PASSIVE_CABLE)
1146                                            hw->phy.type =
1147                                                    ixgbe_phy_sfp_passive_unknown;
1148                                    else if (cable_tech & IXGBE_SFF_DA_ACTIVE_CABLE)
1149                                            hw->phy.type =
1150                                                    ixgbe_phy_sfp_active_unknown;
1151                                    else
1152                                            hw->phy.type = ixgbe_phy_sfp_unknown;
1153                                    break;
1154                            }
1155                    }

1157                    /* Allow any DA cable vendor */
1158                    if (cable_tech & (IXGBE_SFF_DA_PASSIVE_CABLE |
1159                        IXGBE_SFF_DA_ACTIVE_CABLE)) {
1160                            status = IXGBE_SUCCESS;
1161                            goto out;
1162                    }

1164                    /* Verify supported 1G SFP modules */
1165                    if (comp_codes_10g == 0 &&
1166                        !(hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1 ||
1167                          hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0 ||
1168                          hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core0 ||
1169                          hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core1)) {
1170                            hw->phy.type = ixgbe_phy_sfp_unsupported;
1171                            status = IXGBE_ERR_SFP_NOT_SUPPORTED;
1172                            goto out;
1173                    }

1175                    /* Anything else 82598-based is supported */
1176                    if (hw->mac.type == ixgbe_mac_82598EB) {
1177                            status = IXGBE_SUCCESS;
1178                            goto out;
1179                    }

1181                    (void) ixgbe_get_device_caps(hw, &enforce_sfp);
1182                    if (!(enforce_sfp & IXGBE_DEVICE_CAPS_ALLOW_ANY_SFP) &&
1183                        !((hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core0) ||
1184                          (hw->phy.sfp_type == ixgbe_sfp_type_1g_cu_core1) ||
1185                          (hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core0) ||
1186                          (hw->phy.sfp_type == ixgbe_sfp_type_1g_sx_core1))) {
1187                            /* Make sure we're a supported PHY type */
1188                            if (hw->phy.type == ixgbe_phy_sfp_intel) {
1189                                    status = IXGBE_SUCCESS;
1190                            } else {
1191                                    if (hw->allow_unsupported_sfp == TRUE) {
1192                                            EWARN(hw, "WARNING: Intel (R) Network "
```

```
1193                                            "Connections are quality tested "
1194                                            "using Intel (R) Ethernet Optics."
1195                                            " Using untested modules is not "
1196                                            "supported and may cause unstable"
1197                                            " operation or damage to the "
1198                                            "module or the adapter. Intel "
1199                                            "Corporation is not responsible "
1200                                            "for any harm caused by using "
1201                                            "untested modules.\n", status);
1202                                    status = IXGBE_SUCCESS;
1203                            } else {
1204                                    EWARN(hw, "SFP+ module not supported\n",
1205                                          status);
1206                                    hw->phy.type =
1207                                        ixgbe_phy_sfp_unsupported;
1208                                    status = IXGBE_ERR_SFP_NOT_SUPPORTED;
1209                            }
1210                    }
1211            } else {
1212                    status = IXGBE_SUCCESS;
1213            }
1214    }

1216 out:
1217    return status;

1219 err_read_i2c_eeprom:
1220    hw->phy.sfp_type = ixgbe_sfp_type_not_present;
1221    if (hw->phy.type != ixgbe_phy_nl) {
1222            hw->phy.id = 0;
1223            hw->phy.type = ixgbe_phy_unknown;
1224    }
1225    return IXGBE_ERR_SFP_NOT_PRESENT;
1226 }
_____unchanged_portion_omitted_

1329 /**
1330  *  ixgbe_read_i2c_sff8472_generic - Reads 8 bit word over I2C interface
1331  *  @hw: pointer to hardware structure
1332  *  @byte_offset: byte offset at address 0xA2
1333  *  @eeprom_data: value read
1334  *
1335  *  Performs byte read operation to SFP module's SFF-8472 data over I2C
1336  **/
1337 static s32 ixgbe_read_i2c_sff8472_generic(struct ixgbe_hw *hw, u8 byte_offset,
1338                                           u8 *sff8472_data)
1339 {
1340    return hw->phy.ops.read_i2c_byte(hw, byte_offset,
1341                                     IXGBE_I2C_EEPROM_DEV_ADDR2,
1342                                     sff8472_data);
1343 }

1345 /**
1346  *  ixgbe_write_i2c_eeprom_generic - Writes 8 bit EEPROM word over I2C interface
1347  *  @hw: pointer to hardware structure
1348  *  @byte_offset: EEPROM byte offset to write
1349  *  @eeprom_data: value to write
1350  *
1351  *  Performs byte write operation to SFP module's EEPROM over I2C interface.
1352  **/
1353 s32 ixgbe_write_i2c_eeprom_generic(struct ixgbe_hw *hw, u8 byte_offset,
1354                                    u8 eeprom_data)
1355 {
1356    DEBUGFUNC("ixgbe_write_i2c_eeprom_generic");

1358    return hw->phy.ops.write_i2c_byte(hw, byte_offset,
```



```
1359                                      IXGBE_I2C_EEPROM_DEV_ADDR,
1360                                      eeprom_data);
1361 }

1363 /**
1364  *  ixgbe_read_i2c_byte_generic - Reads 8 bit word over I2C
1365  *  @hw: pointer to hardware structure
1366  *  @byte_offset: byte offset to read
1367  *  @data: value read
1368  *
1369  *  Performs byte read operation to SFP module's EEPROM over I2C interface at
1370  *  a specified device address.
1371  **/
1372 s32 ixgbe_read_i2c_byte_generic(struct ixgbe_hw *hw, u8 byte_offset,
1373                                 u8 dev_addr, u8 *data)
1374 {
1375    s32 status = IXGBE_SUCCESS;
1376    u32 max_retry = 10;
1377    u32 retry = 0;
1378    u16 swfw_mask = 0;
1379    bool nack = 1;
1380    *data = 0;

1382    DEBUGFUNC("ixgbe_read_i2c_byte_generic");

1384    if (IXGBE_READ_REG(hw, IXGBE_STATUS) & IXGBE_STATUS_LAN_ID_1)
1385            swfw_mask = IXGBE_GSSR_PHY1_SM;
1386    else
1387            swfw_mask = IXGBE_GSSR_PHY0_SM;

1389    do {
1390            if (hw->mac.ops.acquire_swfw_sync(hw, swfw_mask)
1391                != IXGBE_SUCCESS) {
1392                status = IXGBE_ERR_SWFW_SYNC;
1393                goto read_byte_out;
1394            }

1396            ixgbe_i2c_start(hw);

1398            /* Device Address and write indication */
1399            status = ixgbe_clock_out_i2c_byte(hw, dev_addr);
1400            if (status != IXGBE_SUCCESS)
1401                    goto fail;

1403            status = ixgbe_get_i2c_ack(hw);
1404            if (status != IXGBE_SUCCESS)
1405                    goto fail;

1407            status = ixgbe_clock_out_i2c_byte(hw, byte_offset);
1408            if (status != IXGBE_SUCCESS)
1409                    goto fail;

1411            status = ixgbe_get_i2c_ack(hw);
1412            if (status != IXGBE_SUCCESS)
1413                    goto fail;

1415            ixgbe_i2c_start(hw);

1417            /* Device Address and read indication */
1418            status = ixgbe_clock_out_i2c_byte(hw, (dev_addr | 0x1));
1419            if (status != IXGBE_SUCCESS)
1420                    goto fail;

1422            status = ixgbe_get_i2c_ack(hw);
1423            if (status != IXGBE_SUCCESS)
1424                    goto fail;
```

```
1426                 status = ixgbe_clock_in_i2c_byte(hw, data);
1427                 if (status != IXGBE_SUCCESS)
1428                         goto fail;

1430                 status = ixgbe_clock_out_i2c_bit(hw, nack);
1431                 if (status != IXGBE_SUCCESS)
1432                         goto fail;

1434                 ixgbe_i2c_stop(hw);
1435                 break;

1437 fail:
1438                 ixgbe_i2c_bus_clear(hw);
1439                 hw->mac.ops.release_swfw_sync(hw, swfw_mask);
1440                 msec_delay(100);
1437                 ixgbe_i2c_bus_clear(hw);
1441                 retry++;
1442                 if (retry < max_retry)
1443                         DEBUGOUT("I2C byte read error - Retrying.\n");
1444                 else
1445                         DEBUGOUT("I2C byte read error.\n");

1447         } while (retry < max_retry);

1449         hw->mac.ops.release_swfw_sync(hw, swfw_mask);

1451 read_byte_out:
1452         return status;
1453 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    6410 Tue Apr 30 09:54:24 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_phy.h
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
/******************************************************************************
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

******************************************************************************/
/*$FreeBSD$*/

#ifndef _IXGBE_PHY_H_
#define _IXGBE_PHY_H_

#include "ixgbe_type.h"
#define IXGBE_I2C_EEPROM_DEV_ADDR       0xA0
#define IXGBE_I2C_EEPROM_DEV_ADDR2      0xA2
#define IXGBE_I2C_EEPROM_BANK_LEN       0xFF

/* EEPROM byte offsets */
#define IXGBE_SFF_IDENTIFIER            0x0
#define IXGBE_SFF_IDENTIFIER_SFP        0x3
#define IXGBE_SFF_VENDOR_OUI_BYTE0      0x25
#define IXGBE_SFF_VENDOR_OUI_BYTE1      0x26
#define IXGBE_SFF_VENDOR_OUI_BYTE2      0x27
#define IXGBE_SFF_1GBE_COMP_CODES       0x6
#define IXGBE_SFF_10GBE_COMP_CODES      0x3
#define IXGBE_SFF_CABLE_TECHNOLOGY      0x8
#define IXGBE_SFF_CABLE_SPEC_COMP       0x3C
#define IXGBE_SFF_SFF_8472_SWAP         0x5C
#define IXGBE_SFF_SFF_8472_COMP         0x5E
#define IXGBE_SFF_SFF_8472_OSCB         0x6E
#define IXGBE_SFF_SFF_8472_ESCB         0x76

/* Bitmasks */
#define IXGBE_SFF_DA_PASSIVE_CABLE      0x4
#define IXGBE_SFF_DA_ACTIVE_CABLE       0x8
```



```
#define IXGBE_SFF_DA_SPEC_ACTIVE_LIMITING       0x4
#define IXGBE_SFF_1GBASESX_CAPABLE      0x1
#define IXGBE_SFF_1GBASELX_CAPABLE      0x2
#define IXGBE_SFF_1GBASET_CAPABLE       0x8
#define IXGBE_SFF_10GBASESR_CAPABLE     0x10
#define IXGBE_SFF_10GBASELR_CAPABLE     0x20
#define IXGBE_SFF_SOFT_RS_SELECT_MASK   0x8
#define IXGBE_SFF_SOFT_RS_SELECT_10G    0x8
#define IXGBE_SFF_SOFT_RS_SELECT_1G     0x0
#define IXGBE_I2C_EEPROM_READ_MASK      0x100
#define IXGBE_I2C_EEPROM_STATUS_MASK    0x3
#define IXGBE_I2C_EEPROM_STATUS_NO_OPERATION    0x0
#define IXGBE_I2C_EEPROM_STATUS_PASS    0x1
#define IXGBE_I2C_EEPROM_STATUS_FAIL    0x2
#define IXGBE_I2C_EEPROM_STATUS_IN_PROGRESS     0x3

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

/* Bit-shift macros */
#define IXGBE_SFF_VENDOR_OUI_BYTE0_SHIFT        24
#define IXGBE_SFF_VENDOR_OUI_BYTE1_SHIFT        16
#define IXGBE_SFF_VENDOR_OUI_BYTE2_SHIFT        8

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

/* I2C SDA and SCL timing parameters for standard mode */
#define IXGBE_I2C_T_HD_STA      4
#define IXGBE_I2C_T_LOW         5
#define IXGBE_I2C_T_HIGH        4
#define IXGBE_I2C_T_SU_STA      5
#define IXGBE_I2C_T_HD_DATA     5
#define IXGBE_I2C_T_SU_DATA     1
#define IXGBE_I2C_T_RISE        1
#define IXGBE_I2C_T_FALL        1
#define IXGBE_I2C_T_SU_STO      4
#define IXGBE_I2C_T_BUF         5

#define IXGBE_TN_LASI_STATUS_REG        0x9005
#define IXGBE_TN_LASI_STATUS_TEMP_ALARM 0x0008

/* SFP+ SFF-8472 Compliance */
#define IXGBE_SFF_SFF_8472_UNSUP        0x00
#define IXGBE_SFF_SFF_8472_REV_9_3      0x01
#define IXGBE_SFF_SFF_8472_REV_9_5      0x02
#define IXGBE_SFF_SFF_8472_REV_10_2     0x03
#define IXGBE_SFF_SFF_8472_REV_10_4     0x04
#define IXGBE_SFF_SFF_8472_REV_11_0     0x05

s32 ixgbe_init_phy_ops_generic(struct ixgbe_hw *hw);
bool ixgbe_validate_phy_addr(struct ixgbe_hw *hw, u32 phy_addr);
enum ixgbe_phy_type ixgbe_get_phy_type_from_id(u32 phy_id);
s32 ixgbe_get_phy_id(struct ixgbe_hw *hw);
s32 ixgbe_identify_phy_generic(struct ixgbe_hw *hw);
s32 ixgbe_reset_phy_generic(struct ixgbe_hw *hw);
s32 ixgbe_read_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
                               u32 device_type, u16 *phy_data);
s32 ixgbe_write_phy_reg_generic(struct ixgbe_hw *hw, u32 reg_addr,
                                u32 device_type, u16 phy_data);
s32 ixgbe_setup_phy_link_generic(struct ixgbe_hw *hw);
s32 ixgbe_setup_phy_link_speed_generic(struct ixgbe_hw *hw,
```

```
127                                     ixgbe_link_speed speed,
111                                     bool autoneg,
128                                     bool autoneg_wait_to_complete);
129 s32 ixgbe_get_copper_link_capabilities_generic(struct ixgbe_hw *hw,
130                                                 ixgbe_link_speed *speed,
131                                                 bool *autoneg);

133 /* PHY specific */
134 s32 ixgbe_check_phy_link_tnx(struct ixgbe_hw *hw,
135                              ixgbe_link_speed *speed,
136                              bool *link_up);
137 s32 ixgbe_setup_phy_link_tnx(struct ixgbe_hw *hw);
138 s32 ixgbe_get_phy_firmware_version_tnx(struct ixgbe_hw *hw,
139                                        u16 *firmware_version);
140 s32 ixgbe_get_phy_firmware_version_generic(struct ixgbe_hw *hw,
141                                            u16 *firmware_version);

143 s32 ixgbe_reset_phy_nl(struct ixgbe_hw *hw);
144 s32 ixgbe_identify_module_generic(struct ixgbe_hw *hw);
145 s32 ixgbe_identify_sfp_module_generic(struct ixgbe_hw *hw);
146 s32 ixgbe_get_sfp_init_sequence_offsets(struct ixgbe_hw *hw,
147                                         u16 *list_offset,
148                                         u16 *data_offset);
149 s32 ixgbe_tn_check_overtemp(struct ixgbe_hw *hw);
150 s32 ixgbe_read_i2c_byte_generic(struct ixgbe_hw *hw, u8 byte_offset,
151                                 u8 dev_addr, u8 *data);
152 s32 ixgbe_write_i2c_byte_generic(struct ixgbe_hw *hw, u8 byte_offset,
153                                  u8 dev_addr, u8 data);
154 s32 ixgbe_read_i2c_eeprom_generic(struct ixgbe_hw *hw, u8 byte_offset,
155                                   u8 *eeprom_data);
156 s32 ixgbe_write_i2c_eeprom_generic(struct ixgbe_hw *hw, u8 byte_offset,
157                                    u8 eeprom_data);
158 void ixgbe_i2c_bus_clear(struct ixgbe_hw *hw);
159 #endif /* _IXGBE_PHY_H_ */
```

```
**********************************************************
  125813 Tue Apr 30 09:54:24 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_type.h
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
```

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

```
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

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

#ifndef _IXGBE_TYPE_H_
#define _IXGBE_TYPE_H_

#include "ixgbe_osdep.h"


/* Vendor ID */
#define IXGBE_INTEL_VENDOR_ID                   0x8086

/* Device IDs */
#define IXGBE_DEV_ID_82598                      0x10B6
#define IXGBE_DEV_ID_82598_BX                   0x1508
#define IXGBE_DEV_ID_82598AF_DUAL_PORT          0x10C6
#define IXGBE_DEV_ID_82598AF_SINGLE_PORT        0x10C7
#define IXGBE_DEV_ID_82598AT                    0x10C8
#define IXGBE_DEV_ID_82598AT2                   0x150B
#define IXGBE_DEV_ID_82598EB_SFP_LOM            0x10DB
#define IXGBE_DEV_ID_82598EB_CX4                0x10DD
#define IXGBE_DEV_ID_82598_CX4_DUAL_PORT        0x10EC
#define IXGBE_DEV_ID_82598_DA_DUAL_PORT         0x10F1
#define IXGBE_DEV_ID_82598_SR_DUAL_PORT_EM      0x10E1
#define IXGBE_DEV_ID_82598EB_XF_LR              0x10F4
#define IXGBE_DEV_ID_82599_KX4                  0x10F7
#define IXGBE_DEV_ID_82599_KX4_MEZZ             0x1514
#define IXGBE_DEV_ID_82599_KR                   0x1517
#define IXGBE_DEV_ID_82599_COMBO_BACKPLANE      0x10F8
```



```
#define IXGBE_SUBDEV_ID_82599_KX4_KR_MEZZ       0x000C
#define IXGBE_DEV_ID_82599_CX4                  0x10F9
#define IXGBE_DEV_ID_82599_SFP                  0x10FB
#define IXGBE_SUBDEV_ID_82599_SFP               0x11A9
#define IXGBE_SUBDEV_ID_82599_RNDC              0x1F72
#define IXGBE_SUBDEV_ID_82599_560FLR            0x17D0
#define IXGBE_SUBDEV_ID_82599_ECNA_DP           0x0470
#define IXGBE_DEV_ID_82599_BACKPLANE_FCOE       0x152A
#define IXGBE_DEV_ID_82599_SFP_FCOE             0x1529
#define IXGBE_DEV_ID_82599_SFP_EM               0x1507
#define IXGBE_DEV_ID_82599_SFP_SF2              0x154D
#define IXGBE_DEV_ID_82599_SFP_SF_QP            0x154A
#define IXGBE_DEV_ID_82599EN_SFP                0x1557
#define IXGBE_DEV_ID_82599_XAUI_LOM             0x10FC
#define IXGBE_DEV_ID_82599_T3_LOM               0x151C
#define IXGBE_DEV_ID_82599_VF                   0x10ED
#define IXGBE_DEV_ID_82599_VF_HV                0x152E
#define IXGBE_DEV_ID_82599_BYPASS               0x155D
#define IXGBE_DEV_ID_X540T                      0x1528
#define IXGBE_DEV_ID_X540_VF                    0x1515
#define IXGBE_DEV_ID_X540_VF_HV                 0x1530
#define IXGBE_DEV_ID_X540_BYPASS                0x155C
#define IXGBE_DEV_ID_X540T                      0x1528
#define IXGBE_DEV_ID_X540T1                     0x1560

/* General Registers */
#define IXGBE_CTRL              0x00000
#define IXGBE_STATUS            0x00008
#define IXGBE_CTRL_EXT          0x00018
#define IXGBE_ESDP              0x00020
#define IXGBE_EODSDP            0x00028
#define IXGBE_I2CCTL            0x00028
#define IXGBE_PHY_GPIO          0x00028
#define IXGBE_MAC_GPIO          0x00030
#define IXGBE_PHYINT_STATUS0    0x00100
#define IXGBE_PHYINT_STATUS1    0x00104
#define IXGBE_PHYINT_STATUS2    0x00108
#define IXGBE_LEDCTL            0x00200
#define IXGBE_FRTIMER           0x00048
#define IXGBE_TCPTIMER          0x0004C
#define IXGBE_CORESPARE         0x00600
#define IXGBE_EXVET             0x05078

/* NVM Registers */
#define IXGBE_EEC       0x10010
#define IXGBE_EERD      0x10014
#define IXGBE_EEWR      0x10018
#define IXGBE_FLA       0x1001C
#define IXGBE_EEMNGCTL  0x10110
#define IXGBE_EEMNGDATA 0x10114
#define IXGBE_FLMNGCTL  0x10118
#define IXGBE_FLMNGDATA 0x1011C
#define IXGBE_FLMNGCNT  0x10120
#define IXGBE_FLOP      0x1013C
#define IXGBE_GRC       0x10200
#define IXGBE_SRAMREL   0x10210
#define IXGBE_PHYDBG    0x10218

/* General Receive Control */
#define IXGBE_GRC_MNG   0x00000001 /* Manageability Enable */
#define IXGBE_GRC_APME  0x00000002 /* APM enabled in EEPROM */

#define IXGBE_VPDDIAG0  0x10204
#define IXGBE_VPDDIAG1  0x10208

/* I2CCTL Bit Masks */
```

```
#define IXGBE_I2C_CLK_IN        0x00000001
#define IXGBE_I2C_CLK_OUT       0x00000002
#define IXGBE_I2C_DATA_IN       0x00000004
#define IXGBE_I2C_DATA_OUT      0x00000008
#define IXGBE_I2C_CLOCK_STRETCHING_TIMEOUT      500


/* Interrupt Registers */
#define IXGBE_EICR              0x00800
#define IXGBE_EICS              0x00808
#define IXGBE_EIMS              0x00880
#define IXGBE_EIMC              0x00888
#define IXGBE_EIAC              0x00810
#define IXGBE_EIAM              0x00890
#define IXGBE_EICS_EX(_i)       (0x00A90 + (_i) * 4)
#define IXGBE_EIMS_EX(_i)       (0x00AA0 + (_i) * 4)
#define IXGBE_EIMC_EX(_i)       (0x00AB0 + (_i) * 4)
#define IXGBE_EIAM_EX(_i)       (0x00AD0 + (_i) * 4)
/* 82599 EITR is only 12 bits, with the lower 3 always zero */
/*
 * 82598 EITR is 16 bits but set the limits based on the max
 * supported by all ixgbe hardware
 */
#define IXGBE_MAX_INT_RATE      488281
#define IXGBE_MIN_INT_RATE      956
#define IXGBE_MAX_EITR          0x00000FF8
#define IXGBE_MIN_EITR          8
#define IXGBE_EITR(_i)          (((_i) <= 23) ? (0x00820 + ((_i) * 4)) : \
                                 (0x012300 + (((_i) - 24) * 4)))
#define IXGBE_EITR_ITR_INT_MASK 0x00000FF8
#define IXGBE_EITR_LLI_MOD      0x00008000
#define IXGBE_EITR_CNT_WDIS     0x80000000
#define IXGBE_IVAR(_i)          (0x00900 + ((_i) * 4)) /* 24 at 0x900-0x960 */
#define IXGBE_IVAR_MISC         0x00A00 /* misc MSI-X interrupt causes */
#define IXGBE_EITRSEL           0x00894
#define IXGBE_MSIXT             0x00000 /* MSI-X Table. 0x0000 - 0x01C */
#define IXGBE_MSIXPBA           0x02000 /* MSI-X Pending bit array */
#define IXGBE_PBACL(_i) (((_i) == 0) ? (0x11068) : (0x110C0 + ((_i) * 4)))
#define IXGBE_GPIE              0x00898

/* Flow Control Registers */
#define IXGBE_FCADBUL           0x03210
#define IXGBE_FCADBUH           0x03214
#define IXGBE_FCAMACL           0x04328
#define IXGBE_FCAMACH           0x0432C
#define IXGBE_FCRTH_82599(_i)   (0x03260 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_FCRTL_82599(_i)   (0x03220 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_PFCTOP            0x03008
#define IXGBE_FCTTV(_i)         (0x03200 + ((_i) * 4)) /* 4 of these (0-3) */
#define IXGBE_FCRTL(_i)         (0x03220 + ((_i) * 8)) /* 8 of these (0-7) */
#define IXGBE_FCRTH(_i)         (0x03260 + ((_i) * 8)) /* 8 of these (0-7) */
#define IXGBE_FCRTV             0x032A0
#define IXGBE_FCCFG             0x03D00
#define IXGBE_TFCS              0x0CE00

/* Receive DMA Registers */
#define IXGBE_RDBAL(_i) (((_i) < 64) ? (0x01000 + ((_i) * 0x40)) : \
                         (0x0D000 + (((_i) - 64) * 0x40)))
#define IXGBE_RDBAH(_i) (((_i) < 64) ? (0x01004 + ((_i) * 0x40)) : \
                         (0x0D004 + (((_i) - 64) * 0x40)))
#define IXGBE_RDLEN(_i) (((_i) < 64) ? (0x01008 + ((_i) * 0x40)) : \
                         (0x0D008 + (((_i) - 64) * 0x40)))
#define IXGBE_RDH(_i)   (((_i) < 64) ? (0x01010 + ((_i) * 0x40)) : \
                         (0x0D010 + (((_i) - 64) * 0x40)))
#define IXGBE_RDT(_i)   (((_i) < 64) ? (0x01018 + ((_i) * 0x40)) : \
                         (0x0D018 + (((_i) - 64) * 0x40)))
```



```
#define IXGBE_RXDCTL(_i)        (((_i) < 64) ? (0x01028 + ((_i) * 0x40)) : \
                                 (0x0D028 + (((_i) - 64) * 0x40)))
#define IXGBE_RSCCTL(_i)        (((_i) < 64) ? (0x0102C + ((_i) * 0x40)) : \
                                 (0x0D02C + (((_i) - 64) * 0x40)))
#define IXGBE_RSCDBU    0x03028
#define IXGBE_RDDCC     0x02F20
#define IXGBE_RXMEMWRAP 0x03190
#define IXGBE_STARCTRL  0x03024
/*
 * Split and Replication Receive Control Registers
 * 00-15 : 0x02100 + n*4
 * 16-64 : 0x01014 + n*0x40
 * 64-127: 0x0D014 + (n-64)*0x40
 */
#define IXGBE_SRRCTL(_i)        (((_i) <= 15) ? (0x02100 + ((_i) * 4)) : \
                                 (((_i) < 64) ? (0x01014 + ((_i) * 0x40)) : \
                                 (0x0D014 + (((_i) - 64) * 0x40))))
/*
 * Rx DCA Control Register:
 * 00-15 : 0x02200 + n*4
 * 16-64 : 0x0100C + n*0x40
 * 64-127: 0x0D00C + (n-64)*0x40
 */
#define IXGBE_DCA_RXCTRL(_i)    (((_i) <= 15) ? (0x02200 + ((_i) * 4)) : \
                                 (((_i) < 64) ? (0x0100C + ((_i) * 0x40)) : \
                                 (0x0D00C + (((_i) - 64) * 0x40))))
#define IXGBE_RDRXCTL           0x02F00
#define IXGBE_RDRXCTL_RSC_PUSH  0x80
/* 8 of these 0x03C00 - 0x03C1C */
#define IXGBE_RXPBSIZE(_i)      (0x03C00 + ((_i) * 4))
#define IXGBE_RXCTRL            0x03000
#define IXGBE_DROPEN            0x03D04
#define IXGBE_RXPBSIZE_SHIFT    10

/* Receive Registers */
#define IXGBE_RXCSUM            0x05000
#define IXGBE_RFCTL             0x05008
#define IXGBE_DRECCCTL          0x02F08
#define IXGBE_DRECCCTL_DISABLE  0
#define IXGBE_DRECCCTL2         0x02F8C

/* Multicast Table Array - 128 entries */
#define IXGBE_MTA(_i)           (0x05200 + ((_i) * 4))
#define IXGBE_RAL(_i)           (((_i) <= 15) ? (0x05400 + ((_i) * 8)) : \
                                 (0x0A200 + ((_i) * 8)))
#define IXGBE_RAH(_i)           (((_i) <= 15) ? (0x05404 + ((_i) * 8)) : \
                                 (0x0A204 + ((_i) * 8)))
#define IXGBE_MPSAR_LO(_i)      (0x0A600 + ((_i) * 8))
#define IXGBE_MPSAR_HI(_i)      (0x0A604 + ((_i) * 8))
/* Packet split receive type */
#define IXGBE_PSRTYPE(_i)       (((_i) <= 15) ? (0x05480 + ((_i) * 4)) : \
                                 (0x0EA00 + ((_i) * 4)))
/* array of 4096 1-bit vlan filters */
#define IXGBE_VFTA(_i)          (0x0A000 + ((_i) * 4))
/*array of 4096 4-bit vlan vmdq indices */
#define IXGBE_VFTAVIND(_j, _i)  (0x0A200 + ((_j) * 0x200) + ((_i) * 4))
#define IXGBE_FCTRL             0x05080
#define IXGBE_VLNCTRL           0x05088
#define IXGBE_MCSTCTRL          0x05090
#define IXGBE_MRQC              0x05818
#define IXGBE_SAQF(_i)  (0x0E000 + ((_i) * 4)) /* Source Address Queue Filter */
#define IXGBE_DAQF(_i)  (0x0E200 + ((_i) * 4)) /* Dest. Address Queue Filter */
#define IXGBE_SDPQF(_i) (0x0E400 + ((_i) * 4)) /* Src Dest. Addr Queue Filter */
#define IXGBE_FTQF(_i)  (0x0E600 + ((_i) * 4)) /* Five Tuple Queue Filter */
#define IXGBE_ETQF(_i)  (0x05128 + ((_i) * 4)) /* EType Queue Filter */
#define IXGBE_ETQS(_i)  (0x0EC00 + ((_i) * 4)) /* EType Queue Select */
```

```
254 #define IXGBE_SYNQF     0x0EC30 /* SYN Packet Queue Filter */
255 #define IXGBE_RQTC      0x0EC70
256 #define IXGBE_MTQC      0x08120
257 #define IXGBE_VLVF(_i)  (0x0F100 + ((_i) * 4))  /* 64 of these (0-63) */
258 #define IXGBE_VLVFB(_i) (0x0F200 + ((_i) * 4))  /* 128 of these (0-127) */
259 #define IXGBE_VMVIR(_i) (0x08000 + ((_i) * 4))  /* 64 of these (0-63) */
260 #define IXGBE_VT_CTL            0x051B0
261 #define IXGBE_PFMAILBOX(_i)     (0x04B00 + (4 * (_i))) /* 64 total */
262 /* 64 Mailboxes, 16 DW each */
263 #define IXGBE_PFMBMEM(_i)       (0x13000 + (64 * (_i)))
264 #define IXGBE_PFMBICR(_i)       (0x00710 + (4 * (_i))) /* 4 total */
265 #define IXGBE_PFMBIMR(_i)       (0x00720 + (4 * (_i))) /* 4 total */
266 #define IXGBE_VFRE(_i)          (0x051E0 + ((_i) * 4))
267 #define IXGBE_VFTE(_i)          (0x08110 + ((_i) * 4))
268 #define IXGBE_VMECM(_i)         (0x08790 + ((_i) * 4))
269 #define IXGBE_QDE               0x2F04
270 #define IXGBE_VMTXSW(_i)        (0x05180 + ((_i) * 4)) /* 2 total */
271 #define IXGBE_VMOLR(_i)         (0x0F000 + ((_i) * 4)) /* 64 total */
272 #define IXGBE_UTA(_i)           (0x0F400 + ((_i) * 4))
273 #define IXGBE_MRCTL(_i)         (0x0F600 + ((_i) * 4))
274 #define IXGBE_VMRVLAN(_i)       (0x0F610 + ((_i) * 4))
275 #define IXGBE_VMRVM(_i)         (0x0F630 + ((_i) * 4))
276 #define IXGBE_L34T_IMIR(_i)     (0x0E800 + ((_i) * 4)) /*128 of these (0-127)*/
277 #define IXGBE_RXFECCERR0        0x051B8
278 #define IXGBE_LLITHRESH         0x0EC90
279 #define IXGBE_IMIR(_i)          (0x05A80 + ((_i) * 4))  /* 8 of these (0-7) */
280 #define IXGBE_IMIREXT(_i)       (0x05AA0 + ((_i) * 4))  /* 8 of these (0-7) */
281 #define IXGBE_IMIRVP            0x05AC0
282 #define IXGBE_VMD_CTL           0x0581C
283 #define IXGBE_RETA(_i)          (0x05C00 + ((_i) * 4))  /* 32 of these (0-31) */
284 #define IXGBE_RSSRK(_i)         (0x05C80 + ((_i) * 4))  /* 10 of these (0-9) */

287 /* Flow Director registers */
288 #define IXGBE_FDIRCTRL  0x0EE00
289 #define IXGBE_FDIRHKEY  0x0EE68
290 #define IXGBE_FDIRSKEY  0x0EE6C
291 #define IXGBE_FDIRDIP4M 0x0EE3C
292 #define IXGBE_FDIRSIP4M 0x0EE40
293 #define IXGBE_FDIRTCPM  0x0EE44
294 #define IXGBE_FDIRUDPM  0x0EE48
295 #define IXGBE_FDIRIP6M  0x0EE74
296 #define IXGBE_FDIRM     0x0EE70

298 /* Flow Director Stats registers */
299 #define IXGBE_FDIRFREE  0x0EE38
300 #define IXGBE_FDIRLEN   0x0EE4C
301 #define IXGBE_FDIRUSTAT 0x0EE50
302 #define IXGBE_FDIRFSTAT 0x0EE54
303 #define IXGBE_FDIRMATCH 0x0EE58
304 #define IXGBE_FDIRMISS  0x0EE5C

306 /* Flow Director Programming registers */
307 #define IXGBE_FDIRSIPv6(_i) (0x0EE0C + ((_i) * 4)) /* 3 of these (0-2) */
308 #define IXGBE_FDIRIPSA  0x0EE18
309 #define IXGBE_FDIRIPDA  0x0EE1C
310 #define IXGBE_FDIRPORT  0x0EE20
311 #define IXGBE_FDIRVLAN  0x0EE24
312 #define IXGBE_FDIRHASH  0x0EE28
313 #define IXGBE_FDIRCMD   0x0EE2C

315 /* Transmit DMA registers */
316 #define IXGBE_TDBAL(_i)         (0x06000 + ((_i) * 0x40)) /* 32 of them (0-31)*/
317 #define IXGBE_TDBAH(_i)         (0x06004 + ((_i) * 0x40))
318 #define IXGBE_TDLEN(_i)         (0x06008 + ((_i) * 0x40))
319 #define IXGBE_TDH(_i)           (0x06010 + ((_i) * 0x40))
```



```
320 #define IXGBE_TDT(_i)           (0x06018 + ((_i) * 0x40))
321 #define IXGBE_TXDCTL(_i)        (0x06028 + ((_i) * 0x40))
322 #define IXGBE_TDWBAL(_i)        (0x06038 + ((_i) * 0x40))
323 #define IXGBE_TDWBAH(_i)        (0x0603C + ((_i) * 0x40))
324 #define IXGBE_DTXCTL            0x07E00

326 #define IXGBE_DMATXCTL          0x04A80
327 #define IXGBE_PFVFSPOOF(_i)     (0x08200 + ((_i) * 4)) /* 8 of these 0 - 7 */
328 #define IXGBE_PFDTXGSWC         0x08220
329 #define IXGBE_DTXMXSZRQ         0x08100
330 #define IXGBE_DTXTCPFLGL        0x04A88
331 #define IXGBE_DTXTCPFLGH        0x04A8C
332 #define IXGBE_LBDRPEN           0x0CA00
333 #define IXGBE_TXPBTHRESH(_i)    (0x04950 + ((_i) * 4)) /* 8 of these 0 - 7 */

335 #define IXGBE_DMATXCTL_TE       0x1 /* Transmit Enable */
336 #define IXGBE_DMATXCTL_NS       0x2 /* No Snoop LSO hdr buffer */
337 #define IXGBE_DMATXCTL_GDV      0x8 /* Global Double VLAN */
338 #define IXGBE_DMATXCTL_VT_SHIFT 16  /* VLAN EtherType */

340 #define IXGBE_PFDTXGSWC_VT_LBEN 0x1 /* Local L2 VT switch enable */

342 /* Anti-spoofing defines */
343 #define IXGBE_SPOOF_MACAS_MASK          0xFF
344 #define IXGBE_SPOOF_VLANAS_MASK         0xFF00
345 #define IXGBE_SPOOF_VLANAS_SHIFT        8
346 #define IXGBE_PFVFSPOOF_REG_COUNT       8
347 /* 16 of these (0-15) */
348 #define IXGBE_DCA_TXCTRL(_i)            (0x07200 + ((_i) * 4))
349 /* Tx DCA Control register : 128 of these (0-127) */
350 #define IXGBE_DCA_TXCTRL_82599(_i)      (0x0600C + ((_i) * 0x40))
351 #define IXGBE_TIPG                      0x0CB00
352 #define IXGBE_TXPBSIZE(_i)              (0x0CC00 + ((_i) * 4)) /* 8 of these */
353 #define IXGBE_MNGTXMAP                  0x0CD10
354 #define IXGBE_TIPG_FIBER_DEFAULT        3
355 #define IXGBE_TXPBSIZE_SHIFT            10

357 /* Wake up registers */
358 #define IXGBE_WUC       0x05800
359 #define IXGBE_WUFC      0x05808
360 #define IXGBE_WUS       0x05810
361 #define IXGBE_IPAV      0x05838
362 #define IXGBE_IP4AT     0x05840 /* IPv4 table 0x5840-0x5858 */
363 #define IXGBE_IP6AT     0x05880 /* IPv6 table 0x5880-0x588F */

365 #define IXGBE_WUPL      0x05900
366 #define IXGBE_WUPM      0x05A00 /* wake up pkt memory 0x5A00-0x5A7C */

368 #define IXGBE_FHFT(_n)  (0x09000 + (_n * 0x100)) /* Flex host filter table */
369 /* Ext Flexible Host Filter Table */
370 #define IXGBE_FHFT_EXT(_n)      (0x09800 + (_n * 0x100))

372 /* Four Flexible Filters are supported */
373 #define IXGBE_FLEXIBLE_FILTER_COUNT_MAX         4

375 /* Six Flexible Filters are supported */
376 #define IXGBE_FLEXIBLE_FILTER_COUNT_MAX_6       6
377 #define IXGBE_EXT_FLEXIBLE_FILTER_COUNT_MAX     2

379 /* Each Flexible Filter is at most 128 (0x80) bytes in length */
380 #define IXGBE_FLEXIBLE_FILTER_SIZE_MAX          128
381 #define IXGBE_FHFT_LENGTH_OFFSET                0xFC  /* Length byte in FHFT */
382 #define IXGBE_FHFT_LENGTH_MASK                  0x0FF /* Length in lower byte */

384 /* Definitions for power management and wakeup registers */
385 /* Wake Up Control */
```

```
386 #define IXGBE_WUC_PME_EN        0x00000002 /* PME Enable */
387 #define IXGBE_WUC_PME_STATUS    0x00000004 /* PME Status */
388 #define IXGBE_WUC_WKEN          0x00000010 /* Enable PE_WAKE_N pin assertion  */

390 /* Wake Up Filter Control */
391 #define IXGBE_WUFC_LNKC 0x00000001 /* Link Status Change Wakeup Enable */
392 #define IXGBE_WUFC_MAG  0x00000002 /* Magic Packet Wakeup Enable */
393 #define IXGBE_WUFC_EX   0x00000004 /* Directed Exact Wakeup Enable */
394 #define IXGBE_WUFC_MC   0x00000008 /* Directed Multicast Wakeup Enable */
395 #define IXGBE_WUFC_BC   0x00000010 /* Broadcast Wakeup Enable */
396 #define IXGBE_WUFC_ARP  0x00000020 /* ARP Request Packet Wakeup Enable */
397 #define IXGBE_WUFC_IPV4 0x00000040 /* Directed IPv4 Packet Wakeup Enable */
398 #define IXGBE_WUFC_IPV6 0x00000080 /* Directed IPv6 Packet Wakeup Enable */
399 #define IXGBE_WUFC_MNG  0x00000100 /* Directed Mgmt Packet Wakeup Enable */

401 #define IXGBE_WUFC_IGNORE_TCO   0x00008000 /* Ignore WakeOn TCO packets */
402 #define IXGBE_WUFC_FLX0 0x00010000 /* Flexible Filter 0 Enable */
403 #define IXGBE_WUFC_FLX1 0x00020000 /* Flexible Filter 1 Enable */
404 #define IXGBE_WUFC_FLX2 0x00040000 /* Flexible Filter 2 Enable */
405 #define IXGBE_WUFC_FLX3 0x00080000 /* Flexible Filter 3 Enable */
406 #define IXGBE_WUFC_FLX4 0x00100000 /* Flexible Filter 4 Enable */
407 #define IXGBE_WUFC_FLX5 0x00200000 /* Flexible Filter 5 Enable */
408 #define IXGBE_WUFC_FLX_FILTERS          0x000F0000 /* Mask for 4 flex filters */
409 /* Mask for Ext. flex filters */
410 #define IXGBE_WUFC_EXT_FLX_FILTERS      0x00300000
411 #define IXGBE_WUFC_ALL_FILTERS          0x000F00FF /* Mask all 4 flex filters */
412 #define IXGBE_WUFC_ALL_FILTERS_6        0x003F00FF /* Mask all 6 flex filters */
402 #define IXGBE_WUFC_ALL_FILTERS  0x003F00FF /* Mask for all wakeup filters */
413 #define IXGBE_WUFC_FLX_OFFSET   16 /* Offset to the Flexible Filters bits */

415 /* Wake Up Status */
416 #define IXGBE_WUS_LNKC          IXGBE_WUFC_LNKC
417 #define IXGBE_WUS_MAG           IXGBE_WUFC_MAG
418 #define IXGBE_WUS_EX            IXGBE_WUFC_EX
419 #define IXGBE_WUS_MC            IXGBE_WUFC_MC
420 #define IXGBE_WUS_BC            IXGBE_WUFC_BC
421 #define IXGBE_WUS_ARP           IXGBE_WUFC_ARP
422 #define IXGBE_WUS_IPV4          IXGBE_WUFC_IPV4
423 #define IXGBE_WUS_IPV6          IXGBE_WUFC_IPV6
424 #define IXGBE_WUS_MNG           IXGBE_WUFC_MNG
425 #define IXGBE_WUS_FLX0          IXGBE_WUFC_FLX0
426 #define IXGBE_WUS_FLX1          IXGBE_WUFC_FLX1
427 #define IXGBE_WUS_FLX2          IXGBE_WUFC_FLX2
428 #define IXGBE_WUS_FLX3          IXGBE_WUFC_FLX3
429 #define IXGBE_WUS_FLX4          IXGBE_WUFC_FLX4
430 #define IXGBE_WUS_FLX5          IXGBE_WUFC_FLX5
431 #define IXGBE_WUS_FLX_FILTERS   IXGBE_WUFC_FLX_FILTERS

423 /* Wake Up Packet Length */
433 #define IXGBE_WUPL_LENGTH_MASK  0xFFFF

435 /* DCB registers */
436 #define IXGBE_DCB_MAX_TRAFFIC_CLASS     8
437 #define IXGBE_RMCS              0x03D00
438 #define IXGBE_DPMCS             0x07F40
439 #define IXGBE_PDPMCS            0x0CD00
440 #define IXGBE_RUPPBMR           0x050A0
441 #define IXGBE_RT2CR(_i)         (0x03C20 + ((_i) * 4)) /* 8 of these (0-7) */
442 #define IXGBE_RT2SR(_i)         (0x03C40 + ((_i) * 4)) /* 8 of these (0-7) */
443 #define IXGBE_TDTQ2TCCR(_i)     (0x0602C + ((_i) * 0x40)) /* 8 of these (0-7) */
444 #define IXGBE_TDTQ2TCSR(_i)     (0x0622C + ((_i) * 0x40)) /* 8 of these (0-7) */
445 #define IXGBE_TDPT2TCCR(_i)     (0x0CD20 + ((_i) * 4)) /* 8 of these (0-7) */
446 #define IXGBE_TDPT2TCSR(_i)     (0x0CD40 + ((_i) * 4)) /* 8 of these (0-7) */


449 /* Security Control Registers */
```



```
450 #define IXGBE_SECTXCTRL         0x08800
451 #define IXGBE_SECTXSTAT         0x08804
452 #define IXGBE_SECTXBUFFAF       0x08808
453 #define IXGBE_SECTXMINIFG       0x08810
454 #define IXGBE_SECRXCTRL         0x08D00
455 #define IXGBE_SECRXSTAT         0x08D04

457 /* Security Bit Fields and Masks */
458 #define IXGBE_SECTXCTRL_SECTX_DIS       0x00000001
459 #define IXGBE_SECTXCTRL_TX_DIS          0x00000002
460 #define IXGBE_SECTXCTRL_STORE_FORWARD   0x00000004

462 #define IXGBE_SECTXSTAT_SECTX_RDY       0x00000001
463 #define IXGBE_SECTXSTAT_ECC_TXERR       0x00000002

465 #define IXGBE_SECRXCTRL_SECRX_DIS       0x00000001
466 #define IXGBE_SECRXCTRL_RX_DIS          0x00000002

468 #define IXGBE_SECRXSTAT_SECRX_RDY       0x00000001
469 #define IXGBE_SECRXSTAT_ECC_RXERR       0x00000002

471 /* LinkSec (MacSec) Registers */
472 #define IXGBE_LSECTXCAP         0x08A00
473 #define IXGBE_LSECRXCAP         0x08F00
474 #define IXGBE_LSECTXCTRL        0x08A04
475 #define IXGBE_LSECTXSCL         0x08A08 /* SCI Low */
476 #define IXGBE_LSECTXSCH         0x08A0C /* SCI High */
477 #define IXGBE_LSECTXSA          0x08A10
478 #define IXGBE_LSECTXPN0         0x08A14
479 #define IXGBE_LSECTXPN1         0x08A18
480 #define IXGBE_LSECTXKEY0(_n)    (0x08A1C + (4 * (_n))) /* 4 of these (0-3) */
481 #define IXGBE_LSECTXKEY1(_n)    (0x08A2C + (4 * (_n))) /* 4 of these (0-3) */
482 #define IXGBE_LSECRXCTRL        0x08F04
483 #define IXGBE_LSECRXSCL         0x08F08
484 #define IXGBE_LSECRXSCH         0x08F0C
485 #define IXGBE_LSECRXSA(_i)      (0x08F10 + (4 * (_i))) /* 2 of these (0-1) */
486 #define IXGBE_LSECRXPN(_i)      (0x08F18 + (4 * (_i))) /* 2 of these (0-1) */
487 #define IXGBE_LSECRXKEY(_n, _m) (0x08F20 + ((0x10 * (_n)) + (4 * (_m))))
488 #define IXGBE_LSECTXUT          0x08A3C /* OutPktsUntagged */
489 #define IXGBE_LSECTXPKTE        0x08A40 /* OutPktsEncrypted */
490 #define IXGBE_LSECTXPKTP        0x08A44 /* OutPktsProtected */
491 #define IXGBE_LSECTXOCTE        0x08A48 /* OutOctetsEncrypted */
492 #define IXGBE_LSECTXOCTP        0x08A4C /* OutOctetsProtected */
493 #define IXGBE_LSECRXUT          0x08F40 /* InPktsUntagged/InPktsNoTag */
494 #define IXGBE_LSECRXOCTD        0x08F44 /* InOctetsDecrypted */
495 #define IXGBE_LSECRXOCTV        0x08F48 /* InOctetsValidated */
496 #define IXGBE_LSECRXBAD         0x08F4C /* InPktsBadTag */
497 #define IXGBE_LSECRXNOSCI       0x08F50 /* InPktsNoSci */
498 #define IXGBE_LSECRXUNSCI       0x08F54 /* InPktsUnknownSci */
499 #define IXGBE_LSECRXUNCH        0x08F58 /* InPktsUnchecked */
500 #define IXGBE_LSECRXDELAY       0x08F5C /* InPktsDelayed */
501 #define IXGBE_LSECRXLATE        0x08F60 /* InPktsLate */
502 #define IXGBE_LSECRXOK(_n)      (0x08F64 + (0x04 * (_n))) /* InPktsOk */
503 #define IXGBE_LSECRXINV(_n)     (0x08F6C + (0x04 * (_n))) /* InPktsInvalid */
504 #define IXGBE_LSECRXNV(_n)      (0x08F74 + (0x04 * (_n))) /* InPktsNotValid */
505 #define IXGBE_LSECRXUNSA        0x08F7C /* InPktsUnusedSa */
506 #define IXGBE_LSECRXNUSA        0x08F80 /* InPktsNotUsingSa */

508 /* LinkSec (MacSec) Bit Fields and Masks */
509 #define IXGBE_LSECTXCAP_SUM_MASK        0x00FF0000
510 #define IXGBE_LSECTXCAP_SUM_SHIFT       16
511 #define IXGBE_LSECRXCAP_SUM_MASK        0x00FF0000
512 #define IXGBE_LSECRXCAP_SUM_SHIFT       16

514 #define IXGBE_LSECTXCTRL_EN_MASK        0x00000003
515 #define IXGBE_LSECTXCTRL_DISABLE        0x0
```

```
#define IXGBE_LSECTXCTRL_AUTH           0x1
#define IXGBE_LSECTXCTRL_AUTH_ENCRYPT   0x2
#define IXGBE_LSECTXCTRL_AISCI          0x00000020
#define IXGBE_LSECTXCTRL_PNTHRSH_MASK   0xFFFFFF00
#define IXGBE_LSECTXCTRL_RSV_MASK       0x000000D8

#define IXGBE_LSECRXCTRL_EN_MASK        0x0000000C
#define IXGBE_LSECRXCTRL_EN_SHIFT       2
#define IXGBE_LSECRXCTRL_DISABLE        0x0
#define IXGBE_LSECRXCTRL_CHECK          0x1
#define IXGBE_LSECRXCTRL_STRICT         0x2
#define IXGBE_LSECRXCTRL_DROP           0x3
#define IXGBE_LSECRXCTRL_PLSH           0x00000040
#define IXGBE_LSECRXCTRL_RP             0x00000080
#define IXGBE_LSECRXCTRL_RSV_MASK       0xFFFFFF33

/* IpSec Registers */
#define IXGBE_IPSTXIDX          0x08900
#define IXGBE_IPSTXSALT         0x08904
#define IXGBE_IPSTXKEY(_i)      (0x08908 + (4 * (_i))) /* 4 of these (0-3) */
#define IXGBE_IPSRXIDX          0x08E00
#define IXGBE_IPSRXIPADDR(_i)   (0x08E04 + (4 * (_i))) /* 4 of these (0-3) */
#define IXGBE_IPSRXSPI          0x08E14
#define IXGBE_IPSRXIPIDX        0x08E18
#define IXGBE_IPSRXKEY(_i)      (0x08E1C + (4 * (_i))) /* 4 of these (0-3) */
#define IXGBE_IPSRXSALT         0x08E2C
#define IXGBE_IPSRXMOD          0x08E30

#define IXGBE_SECTXCTRL_STORE_FORWARD_ENABLE    0x4

/* DCB registers */
#define IXGBE_RTRPCS            0x02430
#define IXGBE_RTTDCS            0x04900
#define IXGBE_RTTDCS_ARBDIS     0x00000040 /* DCB arbiter disable */
#define IXGBE_RTTPCS            0x0CD00
#define IXGBE_RTRUP2TC          0x03020
#define IXGBE_RTTUP2TC          0x0C800
#define IXGBE_RTRPT4C(_i)       (0x02140 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_TXLLQ(_i)         (0x082E0 + ((_i) * 4)) /* 4 of these (0-3) */
#define IXGBE_RTRPT4S(_i)       (0x02160 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_RTTDT2C(_i)       (0x04910 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_RTTDT2S(_i)       (0x04930 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_RTTPT2C(_i)       (0x0CD20 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_RTTPT2S(_i)       (0x0CD40 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_RTTDQSEL          0x04904
#define IXGBE_RTTDT1C           0x04908
#define IXGBE_RTTDT1S           0x0490C
#define IXGBE_RTTDTECC          0x04990
#define IXGBE_RTTDTECC_NO_BCN   0x00000100

#define IXGBE_RTTBCNRC                  0x04984
#define IXGBE_RTTBCNRC_RS_ENA           0x80000000
#define IXGBE_RTTBCNRC_RF_DEC_MASK      0x00003FFF
#define IXGBE_RTTBCNRC_RF_INT_SHIFT     14
#define IXGBE_RTTBCNRC_RF_INT_MASK \
        (IXGBE_RTTBCNRC_RF_DEC_MASK << IXGBE_RTTBCNRC_RF_INT_SHIFT)
#define IXGBE_RTTBCNRM  0x04980

/* BCN (for DCB) Registers */
#define IXGBE_RTTBCNRS  0x04988
#define IXGBE_RTTBCNCR  0x08B00
#define IXGBE_RTTBCNACH 0x08B04
#define IXGBE_RTTBCNACL 0x08B08
#define IXGBE_RTTBCNTG  0x04A90
#define IXGBE_RTTBCNIDX 0x08B0C
#define IXGBE_RTTBCNCP  0x08B10
```



```
#define IXGBE_RTFRTIMER 0x08B14
#define IXGBE_RTTBCNRTT 0x05150
#define IXGBE_RTTBCNRD  0x0498C

/* FCoE DMA Context Registers */
#define IXGBE_FCPTRL            0x02410 /* FC User Desc. PTR Low */
#define IXGBE_FCPTRH            0x02414 /* FC USer Desc. PTR High */
#define IXGBE_FCBUFF            0x02418 /* FC Buffer Control */
#define IXGBE_FCDMARW           0x02420 /* FC Receive DMA RW */
#define IXGBE_FCINVST0          0x03FC0 /* FC Invalid DMA Context Status Reg 0*/
#define IXGBE_FCINVST(_i)       (IXGBE_FCINVST0 + ((_i) * 4))
#define IXGBE_FCBUFF_VALID      (1 << 0)   /* DMA Context Valid */
#define IXGBE_FCBUFF_BUFFSIZE   (3 << 3)   /* User Buffer Size */
#define IXGBE_FCBUFF_WRCONTX    (1 << 7)   /* 0: Initiator, 1: Target */
#define IXGBE_FCBUFF_BUFFCNT    0x0000ff00 /* Number of User Buffers */
#define IXGBE_FCBUFF_OFFSET     0xffff0000 /* User Buffer Offset */
#define IXGBE_FCBUFF_BUFFSIZE_SHIFT     3
#define IXGBE_FCBUFF_BUFFCNT_SHIFT      8
#define IXGBE_FCBUFF_OFFSET_SHIFT       16
#define IXGBE_FCDMARW_WE                (1 << 14)   /* Write enable */
#define IXGBE_FCDMARW_RE                (1 << 15)   /* Read enable */
#define IXGBE_FCDMARW_FCOESEL           0x000001ff  /* FC X_ID: 11 bits */
#define IXGBE_FCDMARW_LASTSIZE          0xffff0000  /* Last User Buffer Size */
#define IXGBE_FCDMARW_LASTSIZE_SHIFT    16
/* FCoE SOF/EOF */
#define IXGBE_TEOFF             0x04A94 /* Tx FC EOF */
#define IXGBE_TSOFF             0x04A98 /* Tx FC SOF */
#define IXGBE_REOFF             0x05158 /* Rx FC EOF */
#define IXGBE_RSOFF             0x051F8 /* Rx FC SOF */
/* FCoE Filter Context Registers */
#define IXGBE_FCFLT             0x05108 /* FC FLT Context */
#define IXGBE_FCFLTRW           0x05110 /* FC Filter RW Control */
#define IXGBE_FCPARAM           0x051d8 /* FC Offset Parameter */
#define IXGBE_FCFLT_VALID       (1 << 0)   /* Filter Context Valid */
#define IXGBE_FCFLT_FIRST       (1 << 1)   /* Filter First */
#define IXGBE_FCFLT_SEQID       0x00ff0000 /* Sequence ID */
#define IXGBE_FCFLT_SEQCNT      0xff000000 /* Sequence Count */
#define IXGBE_FCFLTRW_RVALDT    (1 << 13)  /* Fast Re-Validation */
#define IXGBE_FCFLTRW_WE        (1 << 14)  /* Write Enable */
#define IXGBE_FCFLTRW_RE        (1 << 15)  /* Read Enable */
/* FCoE Receive Control */
#define IXGBE_FCRXCTRL          0x05100 /* FC Receive Control */
#define IXGBE_FCRXCTRL_FCOELLI  (1 << 0)   /* Low latency interrupt */
#define IXGBE_FCRXCTRL_SAVBAD   (1 << 1)   /* Save Bad Frames */
#define IXGBE_FCRXCTRL_FRSTRDH  (1 << 2)   /* EN 1st Read Header */
#define IXGBE_FCRXCTRL_LASTSEQH (1 << 3)   /* EN Last Header in Seq */
#define IXGBE_FCRXCTRL_ALLH     (1 << 4)   /* EN All Headers */
#define IXGBE_FCRXCTRL_FRSTSEQH (1 << 5)   /* EN 1st Seq. Header */
#define IXGBE_FCRXCTRL_ICRC     (1 << 6)   /* Ignore Bad FC CRC */
#define IXGBE_FCRXCTRL_FCCRCBO  (1 << 7)   /* FC CRC Byte Ordering */
#define IXGBE_FCRXCTRL_FCOEVER  0x00000f00 /* FCoE Version: 4 bits */
#define IXGBE_FCRXCTRL_FCOEVER_SHIFT    8
/* FCoE Redirection */
#define IXGBE_FCRECTL           0x0ED00 /* FC Redirection Control */
#define IXGBE_FCRETA0           0x0ED10 /* FC Redirection Table 0 */
#define IXGBE_FCRETA(_i)        (IXGBE_FCRETA0 + ((_i) * 4)) /* FCoE Redir */
#define IXGBE_FCRECTL_ENA       0x1 /* FCoE Redir Table Enable */
#define IXGBE_FCRETASEL_ENA     0x2 /* FCoE FCRETASEL bit */
#define IXGBE_FCRETA_SIZE       8 /* Max entries in FCRETA */
#define IXGBE_FCRETA_ENTRY_MASK 0x0000007f /* 7 bits for the queue index */

/* Stats registers */
#define IXGBE_CRCERRS   0x04000
#define IXGBE_ILLERRC   0x04004
#define IXGBE_ERRBC     0x04008
```

```
#define IXGBE_MSPDC     0x04010
#define IXGBE_MPC(_i)   (0x03FA0 + ((_i) * 4)) /* 8 of these 3FA0-3FBC*/
#define IXGBE_MLFC      0x04034
#define IXGBE_MRFC      0x04038
#define IXGBE_RLEC      0x04040
#define IXGBE_LXONTXC   0x03F60
#define IXGBE_LXONRXC   0x0CF60
#define IXGBE_LXOFFTXC  0x03F68
#define IXGBE_LXOFFRXC  0x0CF68
#define IXGBE_LXONRXCNT         0x041A4
#define IXGBE_LXOFFRXCNT        0x041A8
#define IXGBE_PXONRXCNT(_i)     (0x04140 + ((_i) * 4)) /* 8 of these */
#define IXGBE_PXOFFRXCNT(_i)    (0x04160 + ((_i) * 4)) /* 8 of these */
#define IXGBE_PXON2OFFCNT(_i)   (0x03240 + ((_i) * 4)) /* 8 of these */
#define IXGBE_PXONTXC(_i)       (0x03F00 + ((_i) * 4)) /* 8 of these 3F00-3F1C*/
#define IXGBE_PXONRXC(_i)       (0x0CF00 + ((_i) * 4)) /* 8 of these CF00-CF1C*/
#define IXGBE_PXOFFTXC(_i)      (0x03F20 + ((_i) * 4)) /* 8 of these 3F20-3F3C*/
#define IXGBE_PXOFFRXC(_i)      (0x0CF20 + ((_i) * 4)) /* 8 of these CF20-CF3C*/
#define IXGBE_PRC64             0x0405C
#define IXGBE_PRC127            0x04060
#define IXGBE_PRC255            0x04064
#define IXGBE_PRC511            0x04068
#define IXGBE_PRC1023           0x0406C
#define IXGBE_PRC1522           0x04070
#define IXGBE_GPRC              0x04074
#define IXGBE_BPRC              0x04078
#define IXGBE_MPRC              0x0407C
#define IXGBE_GPTC              0x04080
#define IXGBE_GORCL             0x04088
#define IXGBE_GORCH             0x0408C
#define IXGBE_GOTCL             0x04090
#define IXGBE_GOTCH             0x04094
#define IXGBE_RNBC(_i)          (0x03FC0 + ((_i) * 4)) /* 8 of these 3FC0-3FDC*/
#define IXGBE_RUC               0x040A4
#define IXGBE_RFC               0x040A8
#define IXGBE_ROC               0x040AC
#define IXGBE_RJC               0x040B0
#define IXGBE_MNGPRC            0x040B4
#define IXGBE_MNGPDC            0x040B8
#define IXGBE_MNGPTC            0x0CF90
#define IXGBE_TORL              0x040C0
#define IXGBE_TORH              0x040C4
#define IXGBE_TPR               0x040D0
#define IXGBE_TPT               0x040D4
#define IXGBE_PTC64             0x040D8
#define IXGBE_PTC127            0x040DC
#define IXGBE_PTC255            0x040E0
#define IXGBE_PTC511            0x040E4
#define IXGBE_PTC1023           0x040E8
#define IXGBE_PTC1522           0x040EC
#define IXGBE_MPTC              0x040F0
#define IXGBE_BPTC              0x040F4
#define IXGBE_XEC               0x04120
#define IXGBE_SSVPC             0x08780

#define IXGBE_RQSMR(_i) (0x02300 + ((_i) * 4))
#define IXGBE_TQSMR(_i) (((_i) <= 7) ? (0x07300 + ((_i) * 4)) : \
                         (0x08600 + ((_i) * 4)))
#define IXGBE_TQSM(_i)  (0x08600 + ((_i) * 4))

#define IXGBE_QPRC(_i)  (0x01030 + ((_i) * 0x40)) /* 16 of these */
#define IXGBE_QPTC(_i)  (0x06030 + ((_i) * 0x40)) /* 16 of these */
#define IXGBE_QBRC(_i)  (0x01034 + ((_i) * 0x40)) /* 16 of these */
#define IXGBE_QBTC(_i)  (0x06034 + ((_i) * 0x40)) /* 16 of these */
#define IXGBE_QBRC_L(_i)        (0x01034 + ((_i) * 0x40)) /* 16 of these */
#define IXGBE_QBRC_H(_i)        (0x01038 + ((_i) * 0x40)) /* 16 of these */
```



```
#define IXGBE_QPRDC(_i)         (0x01430 + ((_i) * 0x40)) /* 16 of these */
#define IXGBE_QBTC_L(_i)        (0x08700 + ((_i) * 0x8)) /* 16 of these */
#define IXGBE_QBTC_H(_i)        (0x08704 + ((_i) * 0x8)) /* 16 of these */
#define IXGBE_FCCRC             0x05118 /* Num of Good Eth CRC w/ Bad FC CRC */
#define IXGBE_FCOERPDC          0x0241C /* FCoE Rx Packets Dropped Count */
#define IXGBE_FCLAST            0x02424 /* FCoE Last Error Count */
#define IXGBE_FCOEPRC           0x02428 /* Number of FCoE Packets Received */
#define IXGBE_FCOEDWRC          0x0242C /* Number of FCoE DWords Received */
#define IXGBE_FCOEPTC           0x08784 /* Number of FCoE Packets Transmitted */
#define IXGBE_FCOEDWTC          0x08788 /* Number of FCoE DWords Transmitted */
#define IXGBE_FCCRC_CNT_MASK    0x0000FFFF /* CRC_CNT: bit 0 - 15 */
#define IXGBE_FCLAST_CNT_MASK   0x0000FFFF /* Last_CNT: bit 0 - 15 */
#define IXGBE_O2BGPTC           0x041C4
#define IXGBE_O2BSPC            0x087B0
#define IXGBE_B2OSPC            0x041C0
#define IXGBE_B2OGPRC           0x02F90
#define IXGBE_BUPRC             0x04180
#define IXGBE_BMPRC             0x04184
#define IXGBE_BBPRC             0x04188
#define IXGBE_BUPTC             0x0418C
#define IXGBE_BMPTC             0x04190
#define IXGBE_BBPTC             0x04194
#define IXGBE_BCRCERRS          0x04198
#define IXGBE_BXONRXC           0x0419C
#define IXGBE_BXOFFRXC          0x041E0
#define IXGBE_BXONTXC           0x041E4
#define IXGBE_BXOFFTXC          0x041E8
#define IXGBE_PCRC8ECL          0x0E810
#define IXGBE_PCRC8ECH          0x0E811
#define IXGBE_PCRC8ECH_MASK     0x1F
#define IXGBE_LDPCECL           0x0E820
#define IXGBE_LDPCECH           0x0E821

/* Management */
#define IXGBE_MAVTV(_i)         (0x05010 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_MFUTP(_i)         (0x05030 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_MANC              0x05820
#define IXGBE_MFVAL             0x05824
#define IXGBE_MANC2H            0x05860
#define IXGBE_MDEF(_i)          (0x05890 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_MIPAF             0x058B0
#define IXGBE_MMAL(_i)          (0x05910 + ((_i) * 8)) /* 4 of these (0-3) */
#define IXGBE_MMAH(_i)          (0x05914 + ((_i) * 8)) /* 4 of these (0-3) */
#define IXGBE_FTFT              0x09400 /* 0x9400-0x97FC */
#define IXGBE_METF(_i)          (0x05190 + ((_i) * 4)) /* 4 of these (0-3) */
#define IXGBE_MDEF_EXT(_i)      (0x05160 + ((_i) * 4)) /* 8 of these (0-7) */
#define IXGBE_LSWFW             0x15014
#define IXGBE_BMCIP(_i)         (0x05050 + ((_i) * 4)) /* 0x5050-0x505C */
#define IXGBE_BMCIPVAL          0x05060
#define IXGBE_BMCIP_IPADDR_TYPE 0x00000001
#define IXGBE_BMCIP_IPADDR_VALID        0x00000002

/* Management Bit Fields and Masks */
#define IXGBE_MANC_RCV_TCO_EN   0x00020000 /* Rcv TCO packet enable */
#define IXGBE_MANC_EN_BMC2OS    0x10000000 /* Ena BMC2OS and OS2BMC traffic */
#define IXGBE_MANC_EN_BMC2OS_SHIFT      28

/* Firmware Semaphore Register */
#define IXGBE_FWSM_MODE_MASK    0xE
#define IXGBE_FWSM_TS_ENABLED   0x1
#define IXGBE_FWSM_FW_MODE_PT   0x4

/* ARC Subsystem registers */
#define IXGBE_HICR              0x15F00
#define IXGBE_FWSTS             0x15F0C
#define IXGBE_HSMC0R            0x15F04
```

```
#define IXGBE_HSMC1R            0x15F08
#define IXGBE_SWSR              0x15F10
#define IXGBE_HFDR              0x15FE8
#define IXGBE_FLEX_MNG          0x15800 /* 0x15800 - 0x15EFC */

#define IXGBE_HICR_EN           0x01  /* Enable bit - RO */
/* Driver sets this bit when done to put command in RAM */
#define IXGBE_HICR_C            0x02
#define IXGBE_HICR_SV           0x04  /* Status Validity */
#define IXGBE_HICR_FW_RESET_ENABLE      0x40
#define IXGBE_HICR_FW_RESET     0x80

/* PCI-E registers */
#define IXGBE_GCR               0x11000
#define IXGBE_GTV               0x11004
#define IXGBE_FUNCTAG           0x11008
#define IXGBE_GLT               0x1100C
#define IXGBE_PCIEPIPEADR       0x11004
#define IXGBE_PCIEPIPEDAT       0x11008
#define IXGBE_GSCL_1            0x11010
#define IXGBE_GSCL_2            0x11014
#define IXGBE_GSCL_3            0x11018
#define IXGBE_GSCL_4            0x1101C
#define IXGBE_GSCN_0            0x11020
#define IXGBE_GSCN_1            0x11024
#define IXGBE_GSCN_2            0x11028
#define IXGBE_GSCN_3            0x1102C
#define IXGBE_FACTPS            0x10150
#define IXGBE_PCIEANACTL        0x11040
#define IXGBE_SWSM              0x10140
#define IXGBE_FWSM              0x10148
#define IXGBE_GSSR              0x10160
#define IXGBE_MREVID            0x11064
#define IXGBE_DCA_ID            0x11070
#define IXGBE_DCA_CTRL          0x11074
#define IXGBE_SWFW_SYNC         IXGBE_GSSR

/* PCI-E registers 82599-Specific */
#define IXGBE_GCR_EXT           0x11050
#define IXGBE_GSCL_5_82599      0x11030
#define IXGBE_GSCL_6_82599      0x11034
#define IXGBE_GSCL_7_82599      0x11038
#define IXGBE_GSCL_8_82599      0x1103C
#define IXGBE_PHYADR_82599      0x11040
#define IXGBE_PHYDAT_82599      0x11044
#define IXGBE_PHYCTL_82599      0x11048
#define IXGBE_PBACLR_82599      0x11068
#define IXGBE_CIAA_82599        0x11088
#define IXGBE_CIAD_82599        0x1108C
#define IXGBE_PICAUSE           0x110B0
#define IXGBE_PIENA             0x110B8
#define IXGBE_CDQ_MBR_82599     0x110B4
#define IXGBE_PCIESPARE         0x110BC
#define IXGBE_MISC_REG_82599    0x110F0
#define IXGBE_ECC_CTRL_0_82599  0x11100
#define IXGBE_ECC_CTRL_1_82599  0x11104
#define IXGBE_ECC_STATUS_82599  0x110E0
#define IXGBE_BAR_CTRL_82599    0x110F4

/* PCI Express Control */
#define IXGBE_GCR_CMPL_TMOUT_MASK       0x0000F000
#define IXGBE_GCR_CMPL_TMOUT_10ms       0x00001000
#define IXGBE_GCR_CMPL_TMOUT_RESEND     0x00010000
#define IXGBE_GCR_CAP_VER2              0x00040000

#define IXGBE_GCR_EXT_MSIX_EN           0x80000000
```



```
#define IXGBE_GCR_EXT_BUFFERS_CLEAR     0x40000000
#define IXGBE_GCR_EXT_VT_MODE_16        0x00000001
#define IXGBE_GCR_EXT_VT_MODE_32        0x00000002
#define IXGBE_GCR_EXT_VT_MODE_64        0x00000003
#define IXGBE_GCR_EXT_SRIOV             (IXGBE_GCR_EXT_MSIX_EN | \
                                         IXGBE_GCR_EXT_VT_MODE_64)
#define IXGBE_GCR_EXT_VT_MODE_MASK      0x00000003
/* Time Sync Registers */
#define IXGBE_TSYNCRXCTL        0x05188 /* Rx Time Sync Control register - RW */
#define IXGBE_TSYNCTXCTL        0x08C00 /* Tx Time Sync Control register - RW */
#define IXGBE_RXSTMPL   0x051E8 /* Rx timestamp Low - RO */
#define IXGBE_RXSTMPH   0x051A4 /* Rx timestamp High - RO */
#define IXGBE_RXSATRL   0x051A0 /* Rx timestamp attribute low - RO */
#define IXGBE_RXSATRH   0x051A8 /* Rx timestamp attribute high - RO */
#define IXGBE_RXMTRL    0x05120 /* RX message type register low - RW */
#define IXGBE_TXSTMPL   0x08C04 /* Tx timestamp value Low - RO */
#define IXGBE_TXSTMPH   0x08C08 /* Tx timestamp value High - RO */
#define IXGBE_SYSTIML   0x08C0C /* System time register Low - RO */
#define IXGBE_SYSTIMH   0x08C10 /* System time register High - RO */
#define IXGBE_TIMINCA   0x08C14 /* Increment attributes register - RW */
#define IXGBE_TIMADJL   0x08C18 /* Time Adjustment Offset register Low - RW */
#define IXGBE_TIMADJH   0x08C1C /* Time Adjustment Offset register High - RW */
#define IXGBE_TSAUXC    0x08C20 /* TimeSync Auxiliary Control register - RW */
#define IXGBE_TRGTTIML0 0x08C24 /* Target Time Register 0 Low - RW */
#define IXGBE_TRGTTIMH0 0x08C28 /* Target Time Register 0 High - RW */
#define IXGBE_TRGTTIML1 0x08C2C /* Target Time Register 1 Low - RW */
#define IXGBE_TRGTTIMH1 0x08C30 /* Target Time Register 1 High - RW */
#define IXGBE_CLKTIML   0x08C34 /* Clock Out Time Register Low - RW */
#define IXGBE_CLKTIMH   0x08C38 /* Clock Out Time Register High - RW */
#define IXGBE_FREQOUT0  0x08C34 /* Frequency Out 0 Control register - RW */
#define IXGBE_FREQOUT1  0x08C38 /* Frequency Out 1 Control register - RW */
#define IXGBE_AUXSTMPL0 0x08C3C /* Auxiliary Time Stamp 0 register Low - RO */
#define IXGBE_AUXSTMPH0 0x08C40 /* Auxiliary Time Stamp 0 register High - RO */
#define IXGBE_AUXSTMPL1 0x08C44 /* Auxiliary Time Stamp 1 register Low - RO */
#define IXGBE_AUXSTMPH1 0x08C48 /* Auxiliary Time Stamp 1 register High - RO */

/* Diagnostic Registers */
#define IXGBE_RDSTATCTL         0x02C20
#define IXGBE_RDSTAT(_i)        (0x02C00 + ((_i) * 4)) /* 0x02C00-0x02C1C */
#define IXGBE_RDHMPN            0x02F08
#define IXGBE_RIC_DW(_i)        (0x02F10 + ((_i) * 4))
#define IXGBE_RDPROBE           0x02F20
#define IXGBE_RDMAM             0x02F30
#define IXGBE_RDMAD             0x02F34
#define IXGBE_TDSTATCTL         0x07C20
#define IXGBE_TDSTAT(_i)        (0x07C00 + ((_i) * 4)) /* 0x07C00 - 0x07C1C */
#define IXGBE_TDHMPN            0x07F08
#define IXGBE_TDHMPN2           0x082FC
#define IXGBE_TXDESCIC          0x082CC
#define IXGBE_TIC_DW(_i)        (0x07F10 + ((_i) * 4))
#define IXGBE_TIC_DW2(_i)       (0x082B0 + ((_i) * 4))
#define IXGBE_TDPROBE           0x07F20
#define IXGBE_TXBUFCTRL         0x0C600
#define IXGBE_TXBUFDATA0        0x0C610
#define IXGBE_TXBUFDATA1        0x0C614
#define IXGBE_TXBUFDATA2        0x0C618
#define IXGBE_TXBUFDATA3        0x0C61C
#define IXGBE_RXBUFCTRL         0x03600
#define IXGBE_RXBUFDATA0        0x03610
#define IXGBE_RXBUFDATA1        0x03614
#define IXGBE_RXBUFDATA2        0x03618
#define IXGBE_RXBUFDATA3        0x0361C
#define IXGBE_PCIE_DIAG(_i)     (0x11090 + ((_i) * 4)) /* 8 of these */
#define IXGBE_RFVAL             0x050A4
#define IXGBE_MDFTC1            0x042B8
#define IXGBE_MDFTC2            0x042C0
```

```
 912 #define IXGBE_MDFTFIFO1         0x042C4
 913 #define IXGBE_MDFTFIFO2         0x042C8
 914 #define IXGBE_MDFTS             0x042CC
 915 #define IXGBE_RXDATAWRPTR(_i)   (0x03700 + ((_i) * 4)) /* 8 of these 3700-370C*/
 916 #define IXGBE_RXDESCWRPTR(_i)   (0x03710 + ((_i) * 4)) /* 8 of these 3710-371C*/
 917 #define IXGBE_RXDATARDPTR(_i)   (0x03720 + ((_i) * 4)) /* 8 of these 3720-372C*/
 918 #define IXGBE_RXDESCRDPTR(_i)   (0x03730 + ((_i) * 4)) /* 8 of these 3730-373C*/
 919 #define IXGBE_TXDATAWRPTR(_i)   (0x0C700 + ((_i) * 4)) /* 8 of these C700-C70C*/
 920 #define IXGBE_TXDESCWRPTR(_i)   (0x0C710 + ((_i) * 4)) /* 8 of these C710-C71C*/
 921 #define IXGBE_TXDATARDPTR(_i)   (0x0C720 + ((_i) * 4)) /* 8 of these C720-C72C*/
 922 #define IXGBE_TXDESCRDPTR(_i)   (0x0C730 + ((_i) * 4)) /* 8 of these C730-C73C*/
 923 #define IXGBE_PCIEECCCTL        0x1106C
 924 #define IXGBE_RXWRPTR(_i)       (0x03100 + ((_i) * 4)) /* 8 of these 3100-310C*/
 925 #define IXGBE_RXUSED(_i)        (0x03120 + ((_i) * 4)) /* 8 of these 3120-312C*/
 926 #define IXGBE_RXRDPTR(_i)       (0x03140 + ((_i) * 4)) /* 8 of these 3140-314C*/
 927 #define IXGBE_RXRDWRPTR(_i)     (0x03160 + ((_i) * 4)) /* 8 of these 3160-310C*/
 928 #define IXGBE_TXWRPTR(_i)       (0x0C100 + ((_i) * 4)) /* 8 of these C100-C10C*/
 929 #define IXGBE_TXUSED(_i)        (0x0C120 + ((_i) * 4)) /* 8 of these C120-C12C*/
 930 #define IXGBE_TXRDPTR(_i)       (0x0C140 + ((_i) * 4)) /* 8 of these C140-C14C*/
 931 #define IXGBE_TXRDWRPTR(_i)     (0x0C160 + ((_i) * 4)) /* 8 of these C160-C10C*/
 932 #define IXGBE_PCIEECCCTL0       0x11100
 933 #define IXGBE_PCIEECCCTL1       0x11104
 934 #define IXGBE_RXDBUECC          0x03F70
 935 #define IXGBE_TXDBUECC          0x0CF70
 936 #define IXGBE_RXDBUEST          0x03F74
 937 #define IXGBE_TXDBUEST          0x0CF74
 938 #define IXGBE_PBTXECC           0x0C300
 939 #define IXGBE_PBRXECC           0x03300
 940 #define IXGBE_GHECCR            0x110B0

 942 /* MAC Registers */
 943 #define IXGBE_PCS1GCFIG         0x04200
 944 #define IXGBE_PCS1GLCTL         0x04208
 945 #define IXGBE_PCS1GLSTA         0x0420C
 946 #define IXGBE_PCS1GDBG0         0x04210
 947 #define IXGBE_PCS1GDBG1         0x04214
 948 #define IXGBE_PCS1GANA          0x04218
 949 #define IXGBE_PCS1GANLP         0x0421C
 950 #define IXGBE_PCS1GANNP         0x04220
 951 #define IXGBE_PCS1GANLPNP       0x04224
 952 #define IXGBE_HLREG0            0x04240
 953 #define IXGBE_HLREG1            0x04244
 954 #define IXGBE_PAP               0x04248
 955 #define IXGBE_MACA              0x0424C
 956 #define IXGBE_APAE              0x04250
 957 #define IXGBE_ARD               0x04254
 958 #define IXGBE_AIS               0x04258
 959 #define IXGBE_MSCA              0x0425C
 960 #define IXGBE_MSRWD             0x04260
 961 #define IXGBE_MLADD             0x04264
 962 #define IXGBE_MHADD             0x04268
 963 #define IXGBE_MAXFRS            0x04268
 964 #define IXGBE_TREG              0x0426C
 965 #define IXGBE_PCSS1             0x04288
 966 #define IXGBE_PCSS2             0x0428C
 967 #define IXGBE_XPCSS             0x04290
 968 #define IXGBE_MFLCN             0x04294
 969 #define IXGBE_SERDESC           0x04298
 970 #define IXGBE_MACS              0x0429C
 971 #define IXGBE_AUTOC             0x042A0
 972 #define IXGBE_LINKS             0x042A4
 973 #define IXGBE_LINKS2            0x04324
 974 #define IXGBE_AUTOC2            0x042A8
 975 #define IXGBE_AUTOC3            0x042AC
 976 #define IXGBE_ANLP1             0x042B0
 977 #define IXGBE_ANLP2             0x042B4
```



```
 978 #define IXGBE_MACC              0x04330
 979 #define IXGBE_ATLASCTL          0x04800
 980 #define IXGBE_MMNGC             0x042D0
 981 #define IXGBE_ANLPNP1           0x042D4
 982 #define IXGBE_ANLPNP2           0x042D8
 983 #define IXGBE_KRPCSFC           0x042E0
 984 #define IXGBE_KRPCSS            0x042E4
 985 #define IXGBE_FECS1             0x042E8
 986 #define IXGBE_FECS2             0x042EC
 987 #define IXGBE_SMADARCTL         0x14F10
 988 #define IXGBE_MPVC              0x04318
 989 #define IXGBE_SGMIIC            0x04314

 991 /* Statistics Registers */
 992 #define IXGBE_RXNFGPC           0x041B0
 993 #define IXGBE_RXNFGBCL          0x041B4
 994 #define IXGBE_RXNFGBCH          0x041B8
 995 #define IXGBE_RXDGPC            0x02F50
 996 #define IXGBE_RXDGBCL           0x02F54
 997 #define IXGBE_RXDGBCH           0x02F58
 998 #define IXGBE_RXDDGPC           0x02F5C
 999 #define IXGBE_RXDDGBCL          0x02F60
1000 #define IXGBE_RXDDGBCH          0x02F64
1001 #define IXGBE_RXLPBKGPC         0x02F68
1002 #define IXGBE_RXLPBKGBCL        0x02F6C
1003 #define IXGBE_RXLPBKGBCH        0x02F70
1004 #define IXGBE_RXDLPBKGPC        0x02F74
1005 #define IXGBE_RXDLPBKGBCL       0x02F78
1006 #define IXGBE_RXDLPBKGBCH       0x02F7C
1007 #define IXGBE_TXDGPC            0x087A0
1008 #define IXGBE_TXDGBCL           0x087A4
1009 #define IXGBE_TXDGBCH           0x087A8

1011 #define IXGBE_RXDSTATCTRL       0x02F40

1013 /* Copper Pond 2 link timeout */
1014 #define IXGBE_VALIDATE_LINK_READY_TIMEOUT 50

1016 /* Omer CORECTL */
1017 #define IXGBE_CORECTL                   0x014F00
1018 /* BARCTRL */
1019 #define IXGBE_BARCTRL                   0x110F4
1020 #define IXGBE_BARCTRL_FLSIZE            0x0700
1021 #define IXGBE_BARCTRL_FLSIZE_SHIFT      8
1022 #define IXGBE_BARCTRL_CSRSIZE           0x2000

1024 /* RSCCTL Bit Masks */
1025 #define IXGBE_RSCCTL_RSCEN      0x01
1026 #define IXGBE_RSCCTL_MAXDESC_1  0x00
1027 #define IXGBE_RSCCTL_MAXDESC_4  0x04
1028 #define IXGBE_RSCCTL_MAXDESC_8  0x08
1029 #define IXGBE_RSCCTL_MAXDESC_16 0x0C
1030 #define IXGBE_RSCCTL_TS_DIS     0x02

1032 /* RSCDBU Bit Masks */
1033 #define IXGBE_RSCDBU_RSCSMALDIS_MASK    0x0000007F
1034 #define IXGBE_RSCDBU_RSCACKDIS          0x00000080

1036 /* RDRXCTL Bit Masks */
1037 #define IXGBE_RDRXCTL_RDMTS_1_2         0x00000000 /* Rx Desc Min THLD Size */
1038 #define IXGBE_RDRXCTL_CRCSTRIP          0x00000002 /* CRC Strip */
1039 #define IXGBE_RDRXCTL_MVMEN             0x00000020
1040 #define IXGBE_RDRXCTL_DMAIDONE          0x00000008 /* DMA init cycle done */
1041 #define IXGBE_RDRXCTL_AGGDIS            0x00010000 /* Aggregation disable */
1042 #define IXGBE_RDRXCTL_RSCFRSTSIZE       0x003E0000 /* RSC First packet size */
1043 #define IXGBE_RDRXCTL_RSCLLIDIS         0x00800000 /* Disable RSC compl on LLI*/
```

```
#define IXGBE_RDRXCTL_RSCLLIDIS         0x00800000 /* Disabl RSC compl on LLI */
#define IXGBE_RDRXCTL_RSCACKC           0x02000000 /* must set 1 when RSC ena */
#define IXGBE_RDRXCTL_FCOE_WRFIX        0x04000000 /* must set 1 when RSC ena */

/* RQTC Bit Masks and Shifts */
#define IXGBE_RQTC_SHIFT_TC(_i) ((_i) * 4)
#define IXGBE_RQTC_TC0_MASK     (0x7 << 0)
#define IXGBE_RQTC_TC1_MASK     (0x7 << 4)
#define IXGBE_RQTC_TC2_MASK     (0x7 << 8)
#define IXGBE_RQTC_TC3_MASK     (0x7 << 12)
#define IXGBE_RQTC_TC4_MASK     (0x7 << 16)
#define IXGBE_RQTC_TC5_MASK     (0x7 << 20)
#define IXGBE_RQTC_TC6_MASK     (0x7 << 24)
#define IXGBE_RQTC_TC7_MASK     (0x7 << 28)

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

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

/* FACTPS */
#define IXGBE_FACTPS_MNGCG      0x20000000 /* Manageblility Clock Gated */
#define IXGBE_FACTPS_LFS        0x40000000 /* LAN Function Select */

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

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

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

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

#define IXGBE_DCA_RXCTRL_CPUID_MASK     0x0000001F /* Rx CPUID Mask */
#define IXGBE_DCA_RXCTRL_CPUID_MASK_82599       0xFF000000 /* Rx CPUID Mask */
#define IXGBE_DCA_RXCTRL_CPUID_SHIFT_82599      24 /* Rx CPUID Shift */
#define IXGBE_DCA_RXCTRL_DESC_DCA_EN    (1 << 5) /* Rx Desc enable */
#define IXGBE_DCA_RXCTRL_HEAD_DCA_EN    (1 << 6) /* Rx Desc header ena */
#define IXGBE_DCA_RXCTRL_DATA_DCA_EN    (1 << 7) /* Rx Desc payload ena */
#define IXGBE_DCA_RXCTRL_DESC_RRO_EN    (1 << 9) /* Rx rd Desc Relax Order */
#define IXGBE_DCA_RXCTRL_DATA_WRO_EN    (1 << 13) /* Rx wr data Relax Order */
#define IXGBE_DCA_RXCTRL_HEAD_WRO_EN    (1 << 15) /* Rx wr header RO */

#define IXGBE_DCA_TXCTRL_CPUID_MASK     0x0000001F /* Tx CPUID Mask */
#define IXGBE_DCA_TXCTRL_CPUID_MASK_82599       0xFF000000 /* Tx CPUID Mask */
#define IXGBE_DCA_TXCTRL_CPUID_SHIFT_82599      24 /* Tx CPUID Shift */
#define IXGBE_DCA_TXCTRL_DESC_DCA_EN    (1 << 5) /* DCA Tx Desc enable */
#define IXGBE_DCA_TXCTRL_DESC_RRO_EN    (1 << 9) /* Tx rd Desc Relax Order */
#define IXGBE_DCA_TXCTRL_DESC_WRO_EN    (1 << 11) /* Tx Desc writeback RO bit */
#define IXGBE_DCA_TXCTRL_DATA_RRO_EN    (1 << 13) /* Tx rd data Relax Order */
#define IXGBE_DCA_MAX_QUEUES_82598      16 /* DCA regs only on 16 queues */

/* MSCA Bit Masks */
```



```
#define IXGBE_MSCA_NP_ADDR_MASK         0x0000FFFF /* MDI Addr (new prot) */
#define IXGBE_MSCA_NP_ADDR_SHIFT        0
#define IXGBE_MSCA_DEV_TYPE_MASK        0x001F0000 /* Dev Type (new prot) */
#define IXGBE_MSCA_DEV_TYPE_SHIFT       16 /* Register Address (old prot */
#define IXGBE_MSCA_PHY_ADDR_MASK        0x03E00000 /* PHY Address mask */
#define IXGBE_MSCA_PHY_ADDR_SHIFT       21 /* PHY Address shift*/
#define IXGBE_MSCA_OP_CODE_MASK         0x0C000000 /* OP CODE mask */
#define IXGBE_MSCA_OP_CODE_SHIFT        26 /* OP CODE shift */
#define IXGBE_MSCA_ADDR_CYCLE           0x00000000 /* OP CODE 00 (addr cycle) */
#define IXGBE_MSCA_WRITE                0x04000000 /* OP CODE 01 (wr) */
#define IXGBE_MSCA_READ                 0x0C000000 /* OP CODE 11 (rd) */
#define IXGBE_MSCA_READ_AUTOINC         0x08000000 /* OP CODE 10 (rd auto inc)*/
#define IXGBE_MSCA_ST_CODE_MASK         0x30000000 /* ST Code mask */
#define IXGBE_MSCA_ST_CODE_SHIFT        28 /* ST Code shift */
#define IXGBE_MSCA_NEW_PROTOCOL         0x00000000 /* ST CODE 00 (new prot) */
#define IXGBE_MSCA_OLD_PROTOCOL         0x10000000 /* ST CODE 01 (old prot) */
#define IXGBE_MSCA_MDI_COMMAND          0x40000000 /* Initiate MDI command */
#define IXGBE_MSCA_MDI_IN_PROG_EN       0x80000000 /* MDI in progress ena */

/* MSRWD bit masks */
#define IXGBE_MSRWD_WRITE_DATA_MASK     0x0000FFFF
#define IXGBE_MSRWD_WRITE_DATA_SHIFT    0
#define IXGBE_MSRWD_READ_DATA_MASK      0xFFFF0000
#define IXGBE_MSRWD_READ_DATA_SHIFT     16

/* Atlas registers */
#define IXGBE_ATLAS_PDN_LPBK            0x24
#define IXGBE_ATLAS_PDN_10G             0xB
#define IXGBE_ATLAS_PDN_1G              0xC
#define IXGBE_ATLAS_PDN_AN              0xD

/* Atlas bit masks */
#define IXGBE_ATLASCTL_WRITE_CMD        0x00010000
#define IXGBE_ATLAS_PDN_TX_REG_EN       0x10
#define IXGBE_ATLAS_PDN_TX_10G_QL_ALL   0xF0
#define IXGBE_ATLAS_PDN_TX_1G_QL_ALL    0xF0
#define IXGBE_ATLAS_PDN_TX_AN_QL_ALL    0xF0

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

/* Device Type definitions for new protocol MDIO commands */
#define IXGBE_MDIO_PMA_PMD_DEV_TYPE             0x1
#define IXGBE_MDIO_PCS_DEV_TYPE                 0x3
#define IXGBE_MDIO_PHY_XS_DEV_TYPE              0x4
#define IXGBE_MDIO_AUTO_NEG_DEV_TYPE            0x7
#define IXGBE_MDIO_VENDOR_SPECIFIC_1_DEV_TYPE   0x1E   /* Device 30 */
#define IXGBE_TWINAX_DEV                        1

#define IXGBE_MDIO_COMMAND_TIMEOUT      100 /* PHY Timeout for 1 GB mode */

#define IXGBE_MDIO_VENDOR_SPECIFIC_1_CONTROL            0x0 /* VS1 Ctrl Reg */
#define IXGBE_MDIO_VENDOR_SPECIFIC_1_STATUS             0x1 /* VS1 Status Reg */
#define IXGBE_MDIO_VENDOR_SPECIFIC_1_LINK_STATUS        0x0008 /* 1 = Link Up */
#define IXGBE_MDIO_VENDOR_SPECIFIC_1_SPEED_STATUS       0x0010 /* 0-10G, 1-1G */
#define IXGBE_MDIO_VENDOR_SPECIFIC_1_10G_SPEED          0x0018
#define IXGBE_MDIO_VENDOR_SPECIFIC_1_1G_SPEED           0x0010

#define IXGBE_MDIO_AUTO_NEG_CONTROL     0x0 /* AUTO_NEG Control Reg */
#define IXGBE_MDIO_AUTO_NEG_STATUS      0x1 /* AUTO_NEG Status Reg */
#define IXGBE_MDIO_AUTO_NEG_ADVT        0x10 /* AUTO_NEG Advt Reg */
#define IXGBE_MDIO_AUTO_NEG_LP          0x13 /* AUTO_NEG LP Status Reg */
#define IXGBE_MDIO_PHY_XS_CONTROL       0x0 /* PHY_XS Control Reg */
#define IXGBE_MDIO_PHY_XS_RESET         0x8000 /* PHY_XS Reset */
#define IXGBE_MDIO_PHY_ID_HIGH          0x2 /* PHY ID High Reg*/
#define IXGBE_MDIO_PHY_ID_LOW           0x3 /* PHY ID Low Reg*/
```

```
1175 #define IXGBE_MDIO_PHY_SPEED_ABILITY    0x4 /* Speed Ability Reg */
1176 #define IXGBE_MDIO_PHY_SPEED_10G        0x0001 /* 10G capable */
1177 #define IXGBE_MDIO_PHY_SPEED_1G         0x0010 /* 1G capable */
1178 #define IXGBE_MDIO_PHY_SPEED_100M       0x0020 /* 100M capable */
1179 #define IXGBE_MDIO_PHY_EXT_ABILITY      0xB /* Ext Ability Reg */
1180 #define IXGBE_MDIO_PHY_10GBASET_ABILITY         0x0004 /* 10GBaseT capable */
1181 #define IXGBE_MDIO_PHY_1000BASET_ABILITY        0x0020 /* 1000BaseT capable */
1182 #define IXGBE_MDIO_PHY_100BASETX_ABILITY        0x0080 /* 100BaseTX capable */
1183 #define IXGBE_MDIO_PHY_SET_LOW_POWER_MODE       0x0800 /* Set low power mode */

1185 #define IXGBE_MDIO_PMA_PMD_CONTROL_ADDR 0x0000 /* PMA/PMD Control Reg */
1186 #define IXGBE_MDIO_PMA_PMD_SDA_SCL_ADDR 0xC30A /* PHY_XS SDA/SCL Addr Reg */
1187 #define IXGBE_MDIO_PMA_PMD_SDA_SCL_DATA 0xC30B /* PHY_XS SDA/SCL Data Reg */
1188 #define IXGBE_MDIO_PMA_PMD_SDA_SCL_STAT 0xC30C /* PHY_XS SDA/SCL Status Reg */

1190 /* MII clause 22/28 definitions */
1191 #define IXGBE_MDIO_PHY_LOW_POWER_MODE   0x0800

1193 #define IXGBE_MII_10GBASE_T_AUTONEG_CTRL_REG    0x20   /* 10G Control Reg */
1194 #define IXGBE_MII_AUTONEG_VENDOR_PROVISION_1_REG 0xC400 /* 1G Provisioning 1 */
1195 #define IXGBE_MII_AUTONEG_XNP_TX_REG            0x17   /* 1G XNP Transmit */
1196 #define IXGBE_MII_AUTONEG_ADVERTISE_REG         0x10   /* 100M Advertisement */
1197 #define IXGBE_MII_10GBASE_T_ADVERTISE           0x1000 /* full duplex, bit:12*/
1198 #define IXGBE_MII_1GBASE_T_ADVERTISE_XNP_TX     0x4000 /* full duplex, bit:14*/
1199 #define IXGBE_MII_1GBASE_T_ADVERTISE            0x8000 /* full duplex, bit:15*/
1200 #define IXGBE_MII_100BASE_T_ADVERTISE           0x0100 /* full duplex, bit:8 */
1201 #define IXGBE_MII_100BASE_T_ADVERTISE_HALF      0x0080 /* half duplex, bit:7 */
1202 #define IXGBE_MII_RESTART                       0x200
1203 #define IXGBE_MII_AUTONEG_COMPLETE              0x20
1204 #define IXGBE_MII_AUTONEG_LINK_UP               0x04
1205 #define IXGBE_MII_AUTONEG_REG                   0x0

1207 #define IXGBE_PHY_REVISION_MASK         0xFFFFFFF0
1208 #define IXGBE_MAX_PHY_ADDR              32

1210 /* PHY IDs*/
1211 #define TN1010_PHY_ID   0x00A19410
1212 #define TNX_FW_REV      0xB
1213 #define X540_PHY_ID     0x01540200
1214 #define AQ_FW_REV       0x20
1215 #define QT2022_PHY_ID   0x0043A400
1216 #define ATH_PHY_ID      0x03429050

1218 /* PHY Types */
1219 #define IXGBE_M88E1145_E_PHY_ID 0x01410CD0

1221 /* Special PHY Init Routine */
1222 #define IXGBE_PHY_INIT_OFFSET_NL        0x002B
1223 #define IXGBE_PHY_INIT_END_NL           0xFFFF
1224 #define IXGBE_CONTROL_MASK_NL           0xF000
1225 #define IXGBE_DATA_MASK_NL              0x0FFF
1226 #define IXGBE_CONTROL_SHIFT_NL          12
1227 #define IXGBE_DELAY_NL                  0
1228 #define IXGBE_DATA_NL                   1
1229 #define IXGBE_CONTROL_NL                0x000F
1230 #define IXGBE_CONTROL_EOL_NL            0x0FFF
1231 #define IXGBE_CONTROL_SOL_NL            0x0000

1233 /* General purpose Interrupt Enable */
1234 #define IXGBE_SDP0_GPIEN        0x00000001 /* SDP0 */
1235 #define IXGBE_SDP1_GPIEN        0x00000002 /* SDP1 */
1236 #define IXGBE_SDP2_GPIEN        0x00000004 /* SDP2 */
1237 #define IXGBE_GPIE_MSIX_MODE    0x00000010 /* MSI-X mode */
1238 #define IXGBE_GPIE_OCD          0x00000020 /* Other Clear Disable */
1239 #define IXGBE_GPIE_EIMEN        0x00000040 /* Immediate Interrupt Enable */
1240 #define IXGBE_GPIE_EIAME        0x40000000
```



```
1241 #define IXGBE_GPIE_PBA_SUPPORT  0x80000000
1242 #define IXGBE_GPIE_RSC_DELAY_SHIFT      11
1243 #define IXGBE_GPIE_VTMODE_MASK  0x0000C000 /* VT Mode Mask */
1244 #define IXGBE_GPIE_VTMODE_16    0x00004000 /* 16 VFs 8 queues per VF */
1245 #define IXGBE_GPIE_VTMODE_32    0x00008000 /* 32 VFs 4 queues per VF */
1246 #define IXGBE_GPIE_VTMODE_64    0x0000C000 /* 64 VFs 2 queues per VF */

1248 /* Packet Buffer Initialization */
1249 #define IXGBE_MAX_PACKET_BUFFERS        8

1251 #define IXGBE_TXPBSIZE_20KB     0x00005000 /* 20KB Packet Buffer */
1252 #define IXGBE_TXPBSIZE_40KB     0x0000A000 /* 40KB Packet Buffer */
1253 #define IXGBE_RXPBSIZE_48KB     0x0000C000 /* 48KB Packet Buffer */
1254 #define IXGBE_RXPBSIZE_64KB     0x00010000 /* 64KB Packet Buffer */
1255 #define IXGBE_RXPBSIZE_80KB     0x00014000 /* 80KB Packet Buffer */
1256 #define IXGBE_RXPBSIZE_128KB    0x00020000 /* 128KB Packet Buffer */
1257 #define IXGBE_RXPBSIZE_MAX      0x00080000 /* 512KB Packet Buffer */
1258 #define IXGBE_TXPBSIZE_MAX      0x00028000 /* 160KB Packet Buffer */

1260 #define IXGBE_TXPKT_SIZE_MAX    0xA /* Max Tx Packet size */
1261 #define IXGBE_MAX_PB            8

1263 /* Packet buffer allocation strategies */
1264 enum {
1265         PBA_STRATEGY_EQUAL      = 0, /* Distribute PB space equally */
1266 #define PBA_STRATEGY_EQUAL      PBA_STRATEGY_EQUAL
1267         PBA_STRATEGY_WEIGHTED   = 1, /* Weight front half of TCs */
1268 #define PBA_STRATEGY_WEIGHTED   PBA_STRATEGY_WEIGHTED
1269 };

1271 /* Transmit Flow Control status */
1272 #define IXGBE_TFCS_TXOFF        0x00000001
1273 #define IXGBE_TFCS_TXOFF0       0x00000100
1274 #define IXGBE_TFCS_TXOFF1       0x00000200
1275 #define IXGBE_TFCS_TXOFF2       0x00000400
1276 #define IXGBE_TFCS_TXOFF3       0x00000800
1277 #define IXGBE_TFCS_TXOFF4       0x00001000
1278 #define IXGBE_TFCS_TXOFF5       0x00002000
1279 #define IXGBE_TFCS_TXOFF6       0x00004000
1280 #define IXGBE_TFCS_TXOFF7       0x00008000

1282 /* TCP Timer */
1283 #define IXGBE_TCPTIMER_KS               0x00000100
1284 #define IXGBE_TCPTIMER_COUNT_ENABLE     0x00000200
1285 #define IXGBE_TCPTIMER_COUNT_FINISH     0x00000400
1286 #define IXGBE_TCPTIMER_LOOP             0x00000800
1287 #define IXGBE_TCPTIMER_DURATION_MASK    0x000000FF

1289 /* HLREG0 Bit Masks */
1290 #define IXGBE_HLREG0_TXCRCEN            0x00000001 /* bit  0 */
1291 #define IXGBE_HLREG0_RXCRCSTRP          0x00000002 /* bit  1 */
1292 #define IXGBE_HLREG0_JUMBOEN            0x00000004 /* bit  2 */
1293 #define IXGBE_HLREG0_TXPADEN            0x00000400 /* bit 10 */
1294 #define IXGBE_HLREG0_TXPAUSEEN          0x00001000 /* bit 12 */
1295 #define IXGBE_HLREG0_RXPAUSEEN          0x00004000 /* bit 14 */
1296 #define IXGBE_HLREG0_LPBK               0x00008000 /* bit 15 */
1297 #define IXGBE_HLREG0_MDCSPD             0x00010000 /* bit 16 */
1298 #define IXGBE_HLREG0_CONTMDC            0x00020000 /* bit 17 */
1299 #define IXGBE_HLREG0_CTRLFLTR           0x00040000 /* bit 18 */
1300 #define IXGBE_HLREG0_PREPEND            0x00F00000 /* bits 20-23 */
1301 #define IXGBE_HLREG0_PRIPAUSEEN         0x01000000 /* bit 24 */
1302 #define IXGBE_HLREG0_RXPAUSERECDA       0x06000000 /* bits 25-26 */
1303 #define IXGBE_HLREG0_RXLNGTHERREN       0x08000000 /* bit 27 */
1304 #define IXGBE_HLREG0_RXPADSTRIPEN       0x10000000 /* bit 28 */

1306 /* VMD_CTL bitmasks */
```

```
1307 #define IXGBE_VMD_CTL_VMDQ_EN           0x00000001
1308 #define IXGBE_VMD_CTL_VMDQ_FILTER       0x00000002

1310 /* VT_CTL bitmasks */
1311 #define IXGBE_VT_CTL_DIS_DEFPL          0x20000000 /* disable default pool */
1312 #define IXGBE_VT_CTL_REPLEN             0x40000000 /* replication enabled */
1313 #define IXGBE_VT_CTL_VT_ENABLE          0x00000001  /* Enable VT Mode */
1314 #define IXGBE_VT_CTL_POOL_SHIFT         7
1315 #define IXGBE_VT_CTL_POOL_MASK          (0x3F << IXGBE_VT_CTL_POOL_SHIFT)

1317 /* VMOLR bitmasks */
1318 #define IXGBE_VMOLR_AUPE        0x01000000 /* accept untagged packets */
1319 #define IXGBE_VMOLR_ROMPE       0x02000000 /* accept packets in MTA tbl */
1320 #define IXGBE_VMOLR_ROPE        0x04000000 /* accept packets in UC tbl */
1321 #define IXGBE_VMOLR_BAM         0x08000000 /* accept broadcast packets */
1322 #define IXGBE_VMOLR_MPE         0x10000000 /* multicast promiscuous */

1324 /* VFRE bitmask */
1325 #define IXGBE_VFRE_ENABLE_ALL   0xFFFFFFFF

1327 #define IXGBE_VF_INIT_TIMEOUT   200 /* Number of retries to clear RSTI */

1329 /* RDHMPN and TDHMPN bitmasks */
1330 #define IXGBE_RDHMPN_RDICADDR           0x007FF800
1331 #define IXGBE_RDHMPN_RDICRDREQ          0x00800000
1332 #define IXGBE_RDHMPN_RDICADDR_SHIFT     11
1333 #define IXGBE_TDHMPN_TDICADDR           0x003FF800
1334 #define IXGBE_TDHMPN_TDICRDREQ          0x00800000
1335 #define IXGBE_TDHMPN_TDICADDR_SHIFT     11

1337 #define IXGBE_RDMAM_MEM_SEL_SHIFT               13
1338 #define IXGBE_RDMAM_DWORD_SHIFT                 9
1339 #define IXGBE_RDMAM_DESC_COMP_FIFO              1
1340 #define IXGBE_RDMAM_DFC_CMD_FIFO                2
1341 #define IXGBE_RDMAM_RSC_HEADER_ADDR             3
1342 #define IXGBE_RDMAM_TCN_STATUS_RAM              4
1343 #define IXGBE_RDMAM_WB_COLL_FIFO                5
1344 #define IXGBE_RDMAM_QSC_CNT_RAM                 6
1345 #define IXGBE_RDMAM_QSC_FCOE_RAM                7
1346 #define IXGBE_RDMAM_QSC_QUEUE_CNT               8
1347 #define IXGBE_RDMAM_QSC_QUEUE_RAM               0xA
1348 #define IXGBE_RDMAM_QSC_RSC_RAM                 0xB
1349 #define IXGBE_RDMAM_DESC_COM_FIFO_RANGE         135
1350 #define IXGBE_RDMAM_DESC_COM_FIFO_COUNT         4
1351 #define IXGBE_RDMAM_DFC_CMD_FIFO_RANGE          48
1352 #define IXGBE_RDMAM_DFC_CMD_FIFO_COUNT          7
1353 #define IXGBE_RDMAM_RSC_HEADER_ADDR_RANGE       32
1354 #define IXGBE_RDMAM_RSC_HEADER_ADDR_COUNT       4
1355 #define IXGBE_RDMAM_TCN_STATUS_RAM_RANGE        256
1356 #define IXGBE_RDMAM_TCN_STATUS_RAM_COUNT        9
1357 #define IXGBE_RDMAM_WB_COLL_FIFO_RANGE          8
1358 #define IXGBE_RDMAM_WB_COLL_FIFO_COUNT          4
1359 #define IXGBE_RDMAM_QSC_CNT_RAM_RANGE           64
1360 #define IXGBE_RDMAM_QSC_CNT_RAM_COUNT           4
1361 #define IXGBE_RDMAM_QSC_FCOE_RAM_RANGE          512
1362 #define IXGBE_RDMAM_QSC_FCOE_RAM_COUNT          5
1363 #define IXGBE_RDMAM_QSC_QUEUE_CNT_RANGE         32
1364 #define IXGBE_RDMAM_QSC_QUEUE_CNT_COUNT         4
1365 #define IXGBE_RDMAM_QSC_QUEUE_RAM_RANGE         128
1366 #define IXGBE_RDMAM_QSC_QUEUE_RAM_COUNT         8
1367 #define IXGBE_RDMAM_QSC_RSC_RAM_RANGE           32
1368 #define IXGBE_RDMAM_QSC_RSC_RAM_COUNT           8

1370 #define IXGBE_TXDESCIC_READY    0x80000000

1372 /* Receive Checksum Control */
```



```
1373 #define IXGBE_RXCSUM_IPPCSE     0x00001000 /* IP payload checksum enable */
1374 #define IXGBE_RXCSUM_PCSD       0x00002000 /* packet checksum disabled */

1376 /* FCRTL Bit Masks */
1377 #define IXGBE_FCRTL_XONE        0x80000000 /* XON enable */
1378 #define IXGBE_FCRTH_FCEN        0x80000000 /* Packet buffer fc enable */

1380 /* PAP bit masks*/
1381 #define IXGBE_PAP_TXPAUSECNT_MASK       0x0000FFFF /* Pause counter mask */

1383 /* RMCS Bit Masks */
1384 #define IXGBE_RMCS_RRM                  0x00000002 /* Rx Recycle Mode enable */
1385 /* Receive Arbitration Control: 0 Round Robin, 1 DFP */
1386 #define IXGBE_RMCS_RAC                  0x00000004
1387 /* Deficit Fixed Prio ena */
1388 #define IXGBE_RMCS_DFP                  IXGBE_RMCS_RAC
1389 #define IXGBE_RMCS_TFCE_802_3X          0x00000008 /* Tx Priority FC ena */
1390 #define IXGBE_RMCS_TFCE_PRIORITY        0x00000010 /* Tx Priority FC ena */
1391 #define IXGBE_RMCS_ARBDIS               0x00000040 /* Arbitration disable bit */

1393 /* FCCFG Bit Masks */
1394 #define IXGBE_FCCFG_TFCE_802_3X         0x00000008 /* Tx link FC enable */
1395 #define IXGBE_FCCFG_TFCE_PRIORITY       0x00000010 /* Tx priority FC enable */

1397 /* Interrupt register bitmasks */

1399 /* Extended Interrupt Cause Read */
1400 #define IXGBE_EICR_RTX_QUEUE    0x0000FFFF /* RTx Queue Interrupt */
1401 #define IXGBE_EICR_FLOW_DIR     0x00010000 /* FDir Exception */
1402 #define IXGBE_EICR_RX_MISS      0x00020000 /* Packet Buffer Overrun */
1403 #define IXGBE_EICR_PCI          0x00040000 /* PCI Exception */
1404 #define IXGBE_EICR_MAILBOX      0x00080000 /* VF to PF Mailbox Interrupt */
1405 #define IXGBE_EICR_LSC          0x00100000 /* Link Status Change */
1406 #define IXGBE_EICR_LINKSEC      0x00200000 /* PN Threshold */
1407 #define IXGBE_EICR_MNG          0x00400000 /* Manageability Event Interrupt */
1408 #define IXGBE_EICR_TS           0x00800000 /* Thermal Sensor Event */
1409 #define IXGBE_EICR_TIMESYNC     0x01000000 /* Timesync Event */
1410 #define IXGBE_EICR_GPI_SDP0     0x01000000 /* Gen Purpose Interrupt on SDP0 */
1411 #define IXGBE_EICR_GPI_SDP1     0x02000000 /* Gen Purpose Interrupt on SDP1 */
1412 #define IXGBE_EICR_GPI_SDP2     0x04000000 /* Gen Purpose Interrupt on SDP2 */
1413 #define IXGBE_EICR_ECC          0x10000000 /* ECC Error */
1414 #define IXGBE_EICR_PBUR         0x10000000 /* Packet Buffer Handler Error */
1415 #define IXGBE_EICR_DHER         0x20000000 /* Descriptor Handler Error */
1416 #define IXGBE_EICR_TCP_TIMER    0x40000000 /* TCP Timer */
1417 #define IXGBE_EICR_OTHER        0x80000000 /* Interrupt Cause Active */

1419 /* Extended Interrupt Cause Set */
1420 #define IXGBE_EICS_RTX_QUEUE    IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
1421 #define IXGBE_EICS_FLOW_DIR     IXGBE_EICR_FLOW_DIR  /* FDir Exception */
1422 #define IXGBE_EICS_RX_MISS      IXGBE_EICR_RX_MISS   /* Pkt Buffer Overrun */
1423 #define IXGBE_EICS_PCI          IXGBE_EICR_PCI /* PCI Exception */
1424 #define IXGBE_EICS_MAILBOX      IXGBE_EICR_MAILBOX   /* VF to PF Mailbox Int */
1425 #define IXGBE_EICS_LSC          IXGBE_EICR_LSC /* Link Status Change */
1426 #define IXGBE_EICS_MNG          IXGBE_EICR_MNG /* MNG Event Interrupt */
1427 #define IXGBE_EICS_TIMESYNC     IXGBE_EICR_TIMESYNC /* Timesync Event */
1428 #define IXGBE_EICS_GPI_SDP0     IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
1429 #define IXGBE_EICS_GPI_SDP1     IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
1430 #define IXGBE_EICS_GPI_SDP2     IXGBE_EICR_GPI_SDP2 /* SDP2 Gen Purpose Int */
1431 #define IXGBE_EICS_ECC          IXGBE_EICR_ECC /* ECC Error */
1432 #define IXGBE_EICS_PBUR         IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
1433 #define IXGBE_EICS_DHER         IXGBE_EICR_DHER /* Desc Handler Error */
1434 #define IXGBE_EICS_TCP_TIMER    IXGBE_EICR_TCP_TIMER /* TCP Timer */
1435 #define IXGBE_EICS_OTHER        IXGBE_EICR_OTHER /* INT Cause Active */

1437 /* Extended Interrupt Mask Set */
1438 #define IXGBE_EIMS_RTX_QUEUE    IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
```

```
#define IXGBE_EIMS_FLOW_DIR     IXGBE_EICR_FLOW_DIR /* FDir Exception */
#define IXGBE_EIMS_RX_MISS      IXGBE_EICR_RX_MISS /* Packet Buffer Overrun */
#define IXGBE_EIMS_PCI          IXGBE_EICR_PCI /* PCI Exception */
#define IXGBE_EIMS_MAILBOX      IXGBE_EICR_MAILBOX   /* VF to PF Mailbox Int */
#define IXGBE_EIMS_LSC          IXGBE_EICR_LSC /* Link Status Change */
#define IXGBE_EIMS_MNG          IXGBE_EICR_MNG /* MNG Event Interrupt */
#define IXGBE_EIMS_TS           IXGBE_EICR_TS /* Thermal Sensor Event */
#define IXGBE_EIMS_TIMESYNC     IXGBE_EICR_TIMESYNC /* Timesync Event */
#define IXGBE_EIMS_GPI_SDP0     IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
#define IXGBE_EIMS_GPI_SDP1     IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
#define IXGBE_EIMS_GPI_SDP2     IXGBE_EICR_GPI_SDP2 /* SDP2 Gen Purpose Int */
#define IXGBE_EIMS_ECC          IXGBE_EICR_ECC /* ECC Error */
#define IXGBE_EIMS_PBUR         IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
#define IXGBE_EIMS_DHER         IXGBE_EICR_DHER /* Descr Handler Error */
#define IXGBE_EIMS_TCP_TIMER    IXGBE_EICR_TCP_TIMER /* TCP Timer */
#define IXGBE_EIMS_OTHER        IXGBE_EICR_OTHER /* INT Cause Active */

/* Extended Interrupt Mask Clear */
#define IXGBE_EIMC_RTX_QUEUE    IXGBE_EICR_RTX_QUEUE /* RTx Queue Interrupt */
#define IXGBE_EIMC_FLOW_DIR     IXGBE_EICR_FLOW_DIR /* FDir Exception */
#define IXGBE_EIMC_RX_MISS      IXGBE_EICR_RX_MISS /* Packet Buffer Overrun */
#define IXGBE_EIMC_PCI          IXGBE_EICR_PCI /* PCI Exception */
#define IXGBE_EIMC_MAILBOX      IXGBE_EICR_MAILBOX /* VF to PF Mailbox Int */
#define IXGBE_EIMC_LSC          IXGBE_EICR_LSC /* Link Status Change */
#define IXGBE_EIMC_MNG          IXGBE_EICR_MNG /* MNG Event Interrupt */
#define IXGBE_EIMC_TIMESYNC     IXGBE_EICR_TIMESYNC /* Timesync Event */
#define IXGBE_EIMC_GPI_SDP0     IXGBE_EICR_GPI_SDP0 /* SDP0 Gen Purpose Int */
#define IXGBE_EIMC_GPI_SDP1     IXGBE_EICR_GPI_SDP1 /* SDP1 Gen Purpose Int */
#define IXGBE_EIMC_GPI_SDP2     IXGBE_EICR_GPI_SDP2  /* SDP2 Gen Purpose Int */
#define IXGBE_EIMC_ECC          IXGBE_EICR_ECC /* ECC Error */
#define IXGBE_EIMC_PBUR         IXGBE_EICR_PBUR /* Pkt Buf Handler Err */
#define IXGBE_EIMC_DHER         IXGBE_EICR_DHER /* Desc Handler Err */
#define IXGBE_EIMC_TCP_TIMER    IXGBE_EICR_TCP_TIMER /* TCP Timer */
#define IXGBE_EIMC_OTHER        IXGBE_EICR_OTHER /* INT Cause Active */

#define IXGBE_EIMS_ENABLE_MASK ( \
                                IXGBE_EIMS_RTX_QUEUE    | \
                                IXGBE_EIMS_LSC          | \
                                IXGBE_EIMS_TCP_TIMER    | \
                                IXGBE_EIMS_OTHER)

/* Immediate Interrupt Rx (A.K.A. Low Latency Interrupt) */
#define IXGBE_IMIR_PORT_IM_EN   0x00010000  /* TCP port enable */
#define IXGBE_IMIR_PORT_BP      0x00020000  /* TCP port check bypass */
#define IXGBE_IMIREXT_SIZE_BP   0x00001000  /* Packet size bypass */
#define IXGBE_IMIREXT_CTRL_URG  0x00002000  /* Check URG bit in header */
#define IXGBE_IMIREXT_CTRL_ACK  0x00004000  /* Check ACK bit in header */
#define IXGBE_IMIREXT_CTRL_PSH  0x00008000  /* Check PSH bit in header */
#define IXGBE_IMIREXT_CTRL_RST  0x00010000  /* Check RST bit in header */
#define IXGBE_IMIREXT_CTRL_SYN  0x00020000  /* Check SYN bit in header */
#define IXGBE_IMIREXT_CTRL_FIN  0x00040000  /* Check FIN bit in header */
#define IXGBE_IMIREXT_CTRL_BP   0x00080000  /* Bypass check of control bits */
#define IXGBE_IMIR_SIZE_BP_82599        0x00001000 /* Packet size bypass */
#define IXGBE_IMIR_CTRL_URG_82599       0x00002000 /* Check URG bit in header */
#define IXGBE_IMIR_CTRL_ACK_82599       0x00004000 /* Check ACK bit in header */
#define IXGBE_IMIR_CTRL_PSH_82599       0x00008000 /* Check PSH bit in header */
#define IXGBE_IMIR_CTRL_RST_82599       0x00010000 /* Check RST bit in header */
#define IXGBE_IMIR_CTRL_SYN_82599       0x00020000 /* Check SYN bit in header */
#define IXGBE_IMIR_CTRL_FIN_82599       0x00040000 /* Check FIN bit in header */
#define IXGBE_IMIR_CTRL_BP_82599        0x00080000 /* Bypass chk of ctrl bits */
#define IXGBE_IMIR_LLI_EN_82599         0x00100000 /* Enables low latency Int */
#define IXGBE_IMIR_RX_QUEUE_MASK_82599  0x0000007F /* Rx Queue Mask */
#define IXGBE_IMIR_RX_QUEUE_SHIFT_82599 21 /* Rx Queue Shift */
#define IXGBE_IMIRVP_PRIORITY_MASK      0x00000007 /* VLAN priority mask */
#define IXGBE_IMIRVP_PRIORITY_EN        0x00000008 /* VLAN priority enable */
```



```
#define IXGBE_MAX_FTQF_FILTERS          128
#define IXGBE_FTQF_PROTOCOL_MASK        0x00000003
#define IXGBE_FTQF_PROTOCOL_TCP         0x00000000
#define IXGBE_FTQF_PROTOCOL_UDP         0x00000001
#define IXGBE_FTQF_PROTOCOL_SCTP        2
#define IXGBE_FTQF_PRIORITY_MASK        0x00000007
#define IXGBE_FTQF_PRIORITY_SHIFT       2
#define IXGBE_FTQF_POOL_MASK            0x0000003F
#define IXGBE_FTQF_POOL_SHIFT           8
#define IXGBE_FTQF_5TUPLE_MASK_MASK     0x0000001F
#define IXGBE_FTQF_5TUPLE_MASK_SHIFT    25
#define IXGBE_FTQF_SOURCE_ADDR_MASK     0x1E
#define IXGBE_FTQF_DEST_ADDR_MASK       0x1D
#define IXGBE_FTQF_SOURCE_PORT_MASK     0x1B
#define IXGBE_FTQF_DEST_PORT_MASK       0x17
#define IXGBE_FTQF_PROTOCOL_COMP_MASK   0x0F
#define IXGBE_FTQF_POOL_MASK_EN         0x40000000
#define IXGBE_FTQF_QUEUE_ENABLE         0x80000000

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

/* Interrupt Vector Allocation Registers */
#define IXGBE_IVAR_REG_NUM              25
#define IXGBE_IVAR_REG_NUM_82599        64
#define IXGBE_IVAR_TXRX_ENTRY           96
#define IXGBE_IVAR_RX_ENTRY             64
#define IXGBE_IVAR_RX_QUEUE(_i)         (0 + (_i))
#define IXGBE_IVAR_TX_QUEUE(_i)         (64 + (_i))
#define IXGBE_IVAR_TX_ENTRY             32

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

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

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

/* ETYPE Queue Filter/Select Bit Masks */
#define IXGBE_MAX_ETQF_FILTERS          8
#define IXGBE_ETQF_FCOE                 0x08000000 /* bit 27 */
#define IXGBE_ETQF_BCN                  0x10000000 /* bit 28 */
#define IXGBE_ETQF_1588                 0x40000000 /* bit 30 */
#define IXGBE_ETQF_FILTER_EN            0x80000000 /* bit 31 */
#define IXGBE_ETQF_POOL_ENABLE          (1 << 26) /* bit 26 */
#define IXGBE_ETQF_POOL_SHIFT           20

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

/*
 * ETQF filter list: one static filter per filter consumer. This is
 *                   to avoid filter collisions later. Add new filters
 *                   here!!
 *
 * Current filters:
 *      EAPOL 802.1x (0x888e): Filter 0
 *      FCoE (0x8906):   Filter 2
 *      1588 (0x88f7):   Filter 3
 *      FIP  (0x8914):   Filter 4
 */
#define IXGBE_ETQF_FILTER_EAPOL         0
#define IXGBE_ETQF_FILTER_FCOE          2
#define IXGBE_ETQF_FILTER_1588          3
```

```
1571 #define IXGBE_ETQF_FILTER_FIP           4
1572 /* VLAN Control Bit Masks */
1573 #define IXGBE_VLNCTRL_VET               0x0000FFFF  /* bits 0-15 */
1574 #define IXGBE_VLNCTRL_CFI               0x10000000  /* bit 28 */
1575 #define IXGBE_VLNCTRL_CFIEN             0x20000000  /* bit 29 */
1576 #define IXGBE_VLNCTRL_VFE               0x40000000  /* bit 30 */
1577 #define IXGBE_VLNCTRL_VME               0x80000000  /* bit 31 */

1579 /* VLAN pool filtering masks */
1580 #define IXGBE_VLVF_VIEN                 0x80000000  /* filter is valid */
1581 #define IXGBE_VLVF_ENTRIES              64
1582 #define IXGBE_VLVF_VLANID_MASK          0x00000FFF
1583 /* Per VF Port VLAN insertion rules */
1584 #define IXGBE_VMVIR_VLANA_DEFAULT       0x40000000 /* Always use default VLAN */
1585 #define IXGBE_VMVIR_VLANA_NEVER         0x80000000 /* Never insert VLAN tag */

1587 #define IXGBE_ETHERNET_IEEE_VLAN_TYPE   0x8100  /* 802.1q protocol */

1589 /* STATUS Bit Masks */
1590 #define IXGBE_STATUS_LAN_ID             0x0000000C /* LAN ID */
1591 #define IXGBE_STATUS_LAN_ID_SHIFT       2 /* LAN ID Shift*/
1592 #define IXGBE_STATUS_GIO                0x00080000 /* GIO Master Ena Status */

1594 #define IXGBE_STATUS_LAN_ID_0   0x00000000 /* LAN ID 0 */
1595 #define IXGBE_STATUS_LAN_ID_1   0x00000004 /* LAN ID 1 */

1597 /* ESDP Bit Masks */
1598 #define IXGBE_ESDP_SDP0         0x00000001 /* SDP0 Data Value */
1599 #define IXGBE_ESDP_SDP1         0x00000002 /* SDP1 Data Value */
1600 #define IXGBE_ESDP_SDP2         0x00000004 /* SDP2 Data Value */
1601 #define IXGBE_ESDP_SDP3         0x00000008 /* SDP3 Data Value */
1602 #define IXGBE_ESDP_SDP4         0x00000010 /* SDP4 Data Value */
1603 #define IXGBE_ESDP_SDP5         0x00000020 /* SDP5 Data Value */
1604 #define IXGBE_ESDP_SDP6         0x00000040 /* SDP6 Data Value */
1605 #define IXGBE_ESDP_SDP7         0x00000080 /* SDP7 Data Value */
1606 #define IXGBE_ESDP_SDP0_DIR     0x00000100 /* SDP0 IO direction */
1607 #define IXGBE_ESDP_SDP1_DIR     0x00000200 /* SDP1 IO direction */
1608 #define IXGBE_ESDP_SDP2_DIR     0x00000400 /* SDP1 IO direction */
1609 #define IXGBE_ESDP_SDP3_DIR     0x00000800 /* SDP3 IO direction */
1610 #define IXGBE_ESDP_SDP4_DIR     0x00001000 /* SDP4 IO direction */
1611 #define IXGBE_ESDP_SDP5_DIR     0x00002000 /* SDP5 IO direction */
1612 #define IXGBE_ESDP_SDP6_DIR     0x00004000 /* SDP6 IO direction */
1613 #define IXGBE_ESDP_SDP7_DIR     0x00008000 /* SDP7 IO direction */
1614 #define IXGBE_ESDP_SDP0_NATIVE  0x00010000 /* SDP0 IO mode */
1615 #define IXGBE_ESDP_SDP1_NATIVE  0x00020000 /* SDP1 IO mode */


1618 /* LEDCTL Bit Masks */
1619 #define IXGBE_LED_IVRT_BASE             0x00000040
1620 #define IXGBE_LED_BLINK_BASE            0x00000080
1621 #define IXGBE_LED_MODE_MASK_BASE        0x0000000F
1622 #define IXGBE_LED_OFFSET(_base, _i)     (_base << (8 * (_i)))
1623 #define IXGBE_LED_MODE_SHIFT(_i)        (8*(_i))
1624 #define IXGBE_LED_IVRT(_i)      IXGBE_LED_OFFSET(IXGBE_LED_IVRT_BASE, _i)
1625 #define IXGBE_LED_BLINK(_i)     IXGBE_LED_OFFSET(IXGBE_LED_BLINK_BASE, _i)
1626 #define IXGBE_LED_MODE_MASK(_i) IXGBE_LED_OFFSET(IXGBE_LED_MODE_MASK_BASE, _i)

1628 /* LED modes */
1629 #define IXGBE_LED_LINK_UP       0x0
1630 #define IXGBE_LED_LINK_10G      0x1
1631 #define IXGBE_LED_MAC           0x2
1632 #define IXGBE_LED_FILTER        0x3
1633 #define IXGBE_LED_LINK_ACTIVE   0x4
1634 #define IXGBE_LED_LINK_1G       0x5
1635 #define IXGBE_LED_ON            0xE
1636 #define IXGBE_LED_OFF           0xF
```



```
1638 /* AUTOC Bit Masks */
1639 #define IXGBE_AUTOC_KX4_KX_SUPP_MASK 0xC0000000
1640 #define IXGBE_AUTOC_KX4_SUPP    0x80000000
1641 #define IXGBE_AUTOC_KX_SUPP     0x40000000
1642 #define IXGBE_AUTOC_PAUSE       0x30000000
1643 #define IXGBE_AUTOC_ASM_PAUSE   0x20000000
1644 #define IXGBE_AUTOC_SYM_PAUSE   0x10000000
1645 #define IXGBE_AUTOC_RF          0x08000000
1646 #define IXGBE_AUTOC_PD_TMR      0x06000000
1647 #define IXGBE_AUTOC_AN_RX_LOOSE 0x01000000
1648 #define IXGBE_AUTOC_AN_RX_DRIFT 0x00800000
1649 #define IXGBE_AUTOC_AN_RX_ALIGN 0x007C0000
1650 #define IXGBE_AUTOC_FECA        0x00040000
1651 #define IXGBE_AUTOC_FECR        0x00020000
1652 #define IXGBE_AUTOC_KR_SUPP     0x00010000
1653 #define IXGBE_AUTOC_AN_RESTART  0x00001000
1654 #define IXGBE_AUTOC_FLU         0x00000001
1655 #define IXGBE_AUTOC_LMS_SHIFT   13
1656 #define IXGBE_AUTOC_LMS_10G_SERIAL      (0x3 << IXGBE_AUTOC_LMS_SHIFT)
1657 #define IXGBE_AUTOC_LMS_KX4_KX_KR       (0x4 << IXGBE_AUTOC_LMS_SHIFT)
1658 #define IXGBE_AUTOC_LMS_SGMII_1G_100M   (0x5 << IXGBE_AUTOC_LMS_SHIFT)
1659 #define IXGBE_AUTOC_LMS_KX4_KX_KR_1G_AN (0x6 << IXGBE_AUTOC_LMS_SHIFT)
1660 #define IXGBE_AUTOC_LMS_KX4_KX_KR_SGMII (0x7 << IXGBE_AUTOC_LMS_SHIFT)
1661 #define IXGBE_AUTOC_LMS_MASK            (0x7 << IXGBE_AUTOC_LMS_SHIFT)
1662 #define IXGBE_AUTOC_LMS_1G_LINK_NO_AN   (0x0 << IXGBE_AUTOC_LMS_SHIFT)
1663 #define IXGBE_AUTOC_LMS_10G_LINK_NO_AN  (0x1 << IXGBE_AUTOC_LMS_SHIFT)
1664 #define IXGBE_AUTOC_LMS_1G_AN           (0x2 << IXGBE_AUTOC_LMS_SHIFT)
1665 #define IXGBE_AUTOC_LMS_KX4_AN          (0x4 << IXGBE_AUTOC_LMS_SHIFT)
1666 #define IXGBE_AUTOC_LMS_KX4_AN_1G_AN    (0x6 << IXGBE_AUTOC_LMS_SHIFT)
1667 #define IXGBE_AUTOC_LMS_ATTACH_TYPE     (0x7 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)

1669 #define IXGBE_AUTOC_1G_PMA_PMD_MASK     0x00000200
1670 #define IXGBE_AUTOC_1G_PMA_PMD_SHIFT    9
1671 #define IXGBE_AUTOC_10G_PMA_PMD_MASK    0x00000180
1672 #define IXGBE_AUTOC_10G_PMA_PMD_SHIFT   7
1673 #define IXGBE_AUTOC_10G_XAUI    (0x0 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
1674 #define IXGBE_AUTOC_10G_KX4     (0x1 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
1675 #define IXGBE_AUTOC_10G_CX4     (0x2 << IXGBE_AUTOC_10G_PMA_PMD_SHIFT)
1676 #define IXGBE_AUTOC_1G_BX       (0x0 << IXGBE_AUTOC_1G_PMA_PMD_SHIFT)
1677 #define IXGBE_AUTOC_1G_KX       (0x1 << IXGBE_AUTOC_1G_PMA_PMD_SHIFT)
1678 #define IXGBE_AUTOC_1G_SFI      (0x0 << IXGBE_AUTOC_1G_PMA_PMD_SHIFT)
1679 #define IXGBE_AUTOC_1G_KX_BX    (0x1 << IXGBE_AUTOC_1G_PMA_PMD_SHIFT)

1681 #define IXGBE_AUTOC2_UPPER_MASK 0xFFFF0000
1682 #define IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_MASK    0x00030000
1683 #define IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_SHIFT   16
1684 #define IXGBE_AUTOC2_10G_KR     (0x0 << IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_SHIFT)
1685 #define IXGBE_AUTOC2_10G_XFI    (0x1 << IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_SHIFT)
1686 #define IXGBE_AUTOC2_10G_SFI    (0x2 << IXGBE_AUTOC2_10G_SERIAL_PMA_PMD_SHIFT)
1687 #define IXGBE_AUTOC2_LINK_DISABLE_MASK  0x70000000

1689 #define IXGBE_MACC_FLU          0x00000001
1690 #define IXGBE_MACC_FSV_10G      0x00030000
1691 #define IXGBE_MACC_FS           0x00040000
1692 #define IXGBE_MAC_RX2TX_LPBK    0x00000002

1694 /* LINKS Bit Masks */
1695 #define IXGBE_LINKS_KX_AN_COMP  0x80000000
1696 #define IXGBE_LINKS_UP          0x40000000
1697 #define IXGBE_LINKS_SPEED       0x20000000
1698 #define IXGBE_LINKS_MODE        0x18000000
1699 #define IXGBE_LINKS_RX_MODE     0x06000000
1700 #define IXGBE_LINKS_TX_MODE     0x01800000
1701 #define IXGBE_LINKS_XGXS_EN     0x00400000
1702 #define IXGBE_LINKS_SGMII_EN    0x02000000
```

```
1703 #define IXGBE_LINKS_PCS_1G_EN   0x00200000
1704 #define IXGBE_LINKS_1G_AN_EN    0x00100000
1705 #define IXGBE_LINKS_KX_AN_IDLE  0x00080000
1706 #define IXGBE_LINKS_1G_SYNC     0x00040000
1707 #define IXGBE_LINKS_10G_ALIGN   0x00020000
1708 #define IXGBE_LINKS_10G_LANE_SYNC       0x00017000
1709 #define IXGBE_LINKS_TL_FAULT            0x00001000
1710 #define IXGBE_LINKS_SIGNAL              0x00000F00

1712 #define IXGBE_LINKS_SPEED_82599         0x30000000
1713 #define IXGBE_LINKS_SPEED_10G_82599     0x30000000
1714 #define IXGBE_LINKS_SPEED_1G_82599      0x20000000
1715 #define IXGBE_LINKS_SPEED_100_82599     0x10000000
1716 #define IXGBE_LINK_UP_TIME              90 /* 9.0 Seconds */
1717 #define IXGBE_AUTO_NEG_TIME             45 /* 4.5 Seconds */

1719 #define IXGBE_LINKS2_AN_SUPPORTED       0x00000040

1721 /* PCS1GLSTA Bit Masks */
1722 #define IXGBE_PCS1GLSTA_LINK_OK         1
1723 #define IXGBE_PCS1GLSTA_SYNK_OK         0x10
1724 #define IXGBE_PCS1GLSTA_AN_COMPLETE     0x10000
1725 #define IXGBE_PCS1GLSTA_AN_PAGE_RX      0x20000
1726 #define IXGBE_PCS1GLSTA_AN_TIMED_OUT    0x40000
1727 #define IXGBE_PCS1GLSTA_AN_REMOTE_FAULT 0x80000
1728 #define IXGBE_PCS1GLSTA_AN_ERROR_RWS    0x100000

1730 #define IXGBE_PCS1GANA_SYM_PAUSE        0x80
1731 #define IXGBE_PCS1GANA_ASM_PAUSE        0x100

1733 /* PCS1GLCTL Bit Masks */
1734 #define IXGBE_PCS1GLCTL_AN_1G_TIMEOUT_EN 0x00040000 /* PCS 1G autoneg to en */
1735 #define IXGBE_PCS1GLCTL_FLV_LINK_UP     1
1736 #define IXGBE_PCS1GLCTL_FORCE_LINK      0x20
1737 #define IXGBE_PCS1GLCTL_LOW_LINK_LATCH  0x40
1738 #define IXGBE_PCS1GLCTL_AN_ENABLE       0x10000
1739 #define IXGBE_PCS1GLCTL_AN_RESTART      0x20000

1741 /* ANLP1 Bit Masks */
1742 #define IXGBE_ANLP1_PAUSE               0x0C00
1743 #define IXGBE_ANLP1_SYM_PAUSE           0x0400
1744 #define IXGBE_ANLP1_ASM_PAUSE           0x0800
1745 #define IXGBE_ANLP1_AN_STATE_MASK       0x000f0000

1747 /* SW Semaphore Register bitmasks */
1748 #define IXGBE_SWSM_SMBI         0x00000001 /* Driver Semaphore bit */
1749 #define IXGBE_SWSM_SWESMBI      0x00000002 /* FW Semaphore bit */
1750 #define IXGBE_SWSM_WMNG         0x00000004 /* Wake MNG Clock */
1751 #define IXGBE_SWFW_REGSMP       0x80000000 /* Register Semaphore bit 31 */

1753 /* SW_FW_SYNC/GSSR definitions */
1754 #define IXGBE_GSSR_EEP_SM       0x0001
1755 #define IXGBE_GSSR_PHY0_SM      0x0002
1756 #define IXGBE_GSSR_PHY1_SM      0x0004
1757 #define IXGBE_GSSR_MAC_CSR_SM   0x0008
1758 #define IXGBE_GSSR_FLASH_SM     0x0010
1759 #define IXGBE_GSSR_SW_MNG_SM    0x0400

1761 /* FW Status register bitmask */
1762 #define IXGBE_FWSTS_FWRI        0x00000200 /* Firmware Reset Indication */

1764 /* EEC Register */
1765 #define IXGBE_EEC_SK            0x00000001 /* EEPROM Clock */
1766 #define IXGBE_EEC_CS            0x00000002 /* EEPROM Chip Select */
1767 #define IXGBE_EEC_DI            0x00000004 /* EEPROM Data In */
1768 #define IXGBE_EEC_DO            0x00000008 /* EEPROM Data Out */
```



```
1769 #define IXGBE_EEC_FWE_MASK      0x00000030 /* FLASH Write Enable */
1770 #define IXGBE_EEC_FWE_DIS       0x00000010 /* Disable FLASH writes */
1771 #define IXGBE_EEC_FWE_EN        0x00000020 /* Enable FLASH writes */
1772 #define IXGBE_EEC_FWE_SHIFT     4
1773 #define IXGBE_EEC_REQ           0x00000040 /* EEPROM Access Request */
1774 #define IXGBE_EEC_GNT           0x00000080 /* EEPROM Access Grant */
1775 #define IXGBE_EEC_PRES          0x00000100 /* EEPROM Present */
1776 #define IXGBE_EEC_ARD           0x00000200 /* EEPROM Auto Read Done */
1777 #define IXGBE_EEC_FLUP          0x00800000 /* Flash update command */
1778 #define IXGBE_EEC_SEC1VAL       0x02000000 /* Sector 1 Valid */
1779 #define IXGBE_EEC_FLUDONE       0x04000000 /* Flash update done */
1780 /* EEPROM Addressing bits based on type (0-small, 1-large) */
1781 #define IXGBE_EEC_ADDR_SIZE     0x00000400
1782 #define IXGBE_EEC_SIZE          0x00007800 /* EEPROM Size */
1783 #define IXGBE_EERD_MAX_ADDR     0x00003FFF /* EERD alows 14 bits for addr. */

1785 #define IXGBE_EEC_SIZE_SHIFT            11
1786 #define IXGBE_EEPROM_WORD_SIZE_SHIFT    6
1787 #define IXGBE_EEPROM_OPCODE_BITS        8

1789 /* Part Number String Length */
1790 #define IXGBE_PBANUM_LENGTH     11

1792 /* Checksum and EEPROM pointers */
1793 #define IXGBE_PBANUM_PTR_GUARD  0xFAFA
1794 #define IXGBE_EEPROM_CHECKSUM   0x3F
1795 #define IXGBE_EEPROM_SUM        0xBABA
1796 #define IXGBE_PCIE_ANALOG_PTR   0x03
1797 #define IXGBE_ATLAS0_CONFIG_PTR 0x04
1798 #define IXGBE_PHY_PTR           0x04
1799 #define IXGBE_ATLAS1_CONFIG_PTR 0x05
1800 #define IXGBE_OPTION_ROM_PTR    0x05
1801 #define IXGBE_PCIE_GENERAL_PTR  0x06
1802 #define IXGBE_PCIE_CONFIG0_PTR  0x07
1803 #define IXGBE_PCIE_CONFIG1_PTR  0x08
1804 #define IXGBE_CORE0_PTR         0x09
1805 #define IXGBE_CORE1_PTR         0x0A
1806 #define IXGBE_MAC0_PTR          0x0B
1807 #define IXGBE_MAC1_PTR          0x0C
1808 #define IXGBE_CSR0_CONFIG_PTR   0x0D
1809 #define IXGBE_CSR1_CONFIG_PTR   0x0E
1810 #define IXGBE_FW_PTR            0x0F
1811 #define IXGBE_PBANUM0_PTR       0x15
1812 #define IXGBE_PBANUM1_PTR       0x16
1813 #define IXGBE_ALT_MAC_ADDR_PTR  0x37
1814 #define IXGBE_FREE_SPACE_PTR    0X3E

1816 #define IXGBE_SAN_MAC_ADDR_PTR          0x28
1817 #define IXGBE_DEVICE_CAPS               0x2C
1818 #define IXGBE_SERIAL_NUMBER_MAC_ADDR    0x11
1819 #define IXGBE_PCIE_MSIX_82599_CAPS      0x72
1820 #define IXGBE_MAX_MSIX_VECTORS_82599    0x40
1821 #define IXGBE_PCIE_MSIX_82598_CAPS      0x62
1822 #define IXGBE_MAX_MSIX_VECTORS_82598    0x13

1824 /* MSI-X capability fields masks */
1825 #define IXGBE_PCIE_MSIX_TBL_SZ_MASK     0x7FF

1827 /* Legacy EEPROM word offsets */
1828 #define IXGBE_ISCSI_BOOT_CAPS           0x0033
1829 #define IXGBE_ISCSI_SETUP_PORT_0        0x0030
1830 #define IXGBE_ISCSI_SETUP_PORT_1        0x0034

1832 /* EEPROM Commands - SPI */
1833 #define IXGBE_EEPROM_MAX_RETRY_SPI      5000 /* Max wait 5ms for RDY signal */
1834 #define IXGBE_EEPROM_STATUS_RDY_SPI     0x01
```

```
1835 #define IXGBE_EEPROM_READ_OPCODE_SPI    0x03  /* EEPROM read opcode */
1836 #define IXGBE_EEPROM_WRITE_OPCODE_SPI   0x02  /* EEPROM write opcode */
1837 #define IXGBE_EEPROM_A8_OPCODE_SPI      0x08  /* opcode bit-3 = addr bit-8 */
1838 #define IXGBE_EEPROM_WREN_OPCODE_SPI    0x06  /* EEPROM set Write Ena latch */
1839 /* EEPROM reset Write Enable latch */
1840 #define IXGBE_EEPROM_WRDI_OPCODE_SPI    0x04
1841 #define IXGBE_EEPROM_RDSR_OPCODE_SPI    0x05  /* EEPROM read Status reg */
1842 #define IXGBE_EEPROM_WRSR_OPCODE_SPI    0x01  /* EEPROM write Status reg */
1843 #define IXGBE_EEPROM_ERASE4K_OPCODE_SPI 0x20  /* EEPROM ERASE 4KB */
1844 #define IXGBE_EEPROM_ERASE64K_OPCODE_SPI        0xD8  /* EEPROM ERASE 64KB */
1845 #define IXGBE_EEPROM_ERASE256_OPCODE_SPI        0xDB  /* EEPROM ERASE 256B */

1847 /* EEPROM Read Register */
1848 #define IXGBE_EEPROM_RW_REG_DATA        16 /* data offset in EEPROM read reg */
1849 #define IXGBE_EEPROM_RW_REG_DONE        2 /* Offset to READ done bit */
1850 #define IXGBE_EEPROM_RW_REG_START       1 /* First bit to start operation */
1851 #define IXGBE_EEPROM_RW_ADDR_SHIFT      2 /* Shift to the address bits */
1852 #define IXGBE_NVM_POLL_WRITE            1 /* Flag for polling for wr complete */
1853 #define IXGBE_NVM_POLL_READ             0 /* Flag for polling for rd complete */

1855 #define IXGBE_ETH_LENGTH_OF_ADDRESS     6

1857 #define IXGBE_EEPROM_PAGE_SIZE_MAX      128
1858 #define IXGBE_EEPROM_RD_BUFFER_MAX_COUNT        256 /* words rd in burst */
1841 #define IXGBE_EEPROM_RD_BUFFER_MAX_COUNT        512 /* words rd in burst */
1859 #define IXGBE_EEPROM_WR_BUFFER_MAX_COUNT        256 /* words wr in burst */

1861 #ifndef IXGBE_EEPROM_GRANT_ATTEMPTS
1862 #define IXGBE_EEPROM_GRANT_ATTEMPTS     1000 /* EEPROM attempts to gain grant */
1863 #endif

1865 /* Number of 5 microseconds we wait for EERD read and
1866  * EERW write to complete */
1867 #define IXGBE_EERD_EEWR_ATTEMPTS        100000

1869 /* # attempts we wait for flush update to complete */
1870 #define IXGBE_FLUDONE_ATTEMPTS          20000

1872 #define IXGBE_PCIE_CTRL2                0x5   /* PCIe Control 2 Offset */
1873 #define IXGBE_PCIE_CTRL2_DUMMY_ENABLE   0x8   /* Dummy Function Enable */
1874 #define IXGBE_PCIE_CTRL2_LAN_DISABLE    0x2   /* LAN PCI Disable */
1875 #define IXGBE_PCIE_CTRL2_DISABLE_SELECT 0x1   /* LAN Disable Select */

1877 #define IXGBE_SAN_MAC_ADDR_PORT0_OFFSET         0x0
1878 #define IXGBE_SAN_MAC_ADDR_PORT1_OFFSET         0x3
1879 #define IXGBE_DEVICE_CAPS_ALLOW_ANY_SFP         0x1
1880 #define IXGBE_DEVICE_CAPS_FCOE_OFFLOADS         0x2
1881 #define IXGBE_FW_LESM_PARAMETERS_PTR            0x2
1882 #define IXGBE_FW_LESM_STATE_1                   0x1
1883 #define IXGBE_FW_LESM_STATE_ENABLED             0x8000 /* LESM Enable bit */
1884 #define IXGBE_FW_PASSTHROUGH_PATCH_CONFIG_PTR   0x4
1885 #define IXGBE_FW_PATCH_VERSION_4                0x7
1886 #define IXGBE_FCOE_IBA_CAPS_BLK_PTR             0x33 /* iSCSI/FCOE block */
1887 #define IXGBE_FCOE_IBA_CAPS_FCOE                0x20 /* FCOE flags */
1888 #define IXGBE_ISCSI_FCOE_BLK_PTR                0x17 /* iSCSI/FCOE block */
1889 #define IXGBE_ISCSI_FCOE_FLAGS_OFFSET           0x0 /* FCOE flags */
1890 #define IXGBE_ISCSI_FCOE_FLAGS_ENABLE           0x1 /* FCOE flags enable bit */
1891 #define IXGBE_ALT_SAN_MAC_ADDR_BLK_PTR          0x27 /* Alt. SAN MAC block */
1892 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_OFFSET      0x0 /* Alt SAN MAC capability */
1893 #define IXGBE_ALT_SAN_MAC_ADDR_PORT0_OFFSET     0x1 /* Alt SAN MAC 0 offset */
1894 #define IXGBE_ALT_SAN_MAC_ADDR_PORT1_OFFSET     0x4 /* Alt SAN MAC 1 offset */
1895 #define IXGBE_ALT_SAN_MAC_ADDR_WWNN_OFFSET      0x7 /* Alt WWNN prefix offset */
1896 #define IXGBE_ALT_SAN_MAC_ADDR_WWPN_OFFSET      0x8 /* Alt WWPN prefix offset */
1897 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_SANMAC      0x0 /* Alt SAN MAC exists */
1898 #define IXGBE_ALT_SAN_MAC_ADDR_CAPS_ALTWWN      0x1 /* Alt WWN base exists */
```



```
1900 #define IXGBE_DEVICE_CAPS_WOL_PORT0_1   0x4 /* WoL supported on ports 0 & 1 */
1901 #define IXGBE_DEVICE_CAPS_WOL_PORT0     0x8 /* WoL supported on port 0 */
1902 #define IXGBE_DEVICE_CAPS_WOL_MASK      0xC /* Mask for WoL capabilities */

1904 /* PCI Bus Info */
1905 #define IXGBE_PCI_DEVICE_STATUS         0xAA
1906 #define IXGBE_PCI_DEVICE_STATUS_TRANSACTION_PENDING     0x0020
1907 #define IXGBE_PCI_LINK_STATUS           0xB2
1908 #define IXGBE_PCI_DEVICE_CONTROL2       0xC8
1909 #define IXGBE_PCI_LINK_WIDTH            0x3F0
1910 #define IXGBE_PCI_LINK_WIDTH_1          0x10
1911 #define IXGBE_PCI_LINK_WIDTH_2          0x20
1912 #define IXGBE_PCI_LINK_WIDTH_4          0x40
1913 #define IXGBE_PCI_LINK_WIDTH_8          0x80
1914 #define IXGBE_PCI_LINK_SPEED            0xF
1915 #define IXGBE_PCI_LINK_SPEED_2500       0x1
1916 #define IXGBE_PCI_LINK_SPEED_5000       0x2
1917 #define IXGBE_PCI_LINK_SPEED_8000       0x3
1918 #define IXGBE_PCI_HEADER_TYPE_REGISTER  0x0E
1919 #define IXGBE_PCI_HEADER_TYPE_MULTIFUNC 0x80
1920 #define IXGBE_PCI_DEVICE_CONTROL2_16ms  0x0005

1922 /* Number of 100 microseconds we wait for PCI Express master disable */
1923 #define IXGBE_PCI_MASTER_DISABLE_TIMEOUT        800

1925 /* Check whether address is multicast. This is little-endian specific check.*/
1926 #define IXGBE_IS_MULTICAST(Address) \
1927                 (bool)(((u8 *)(Address))[0] & ((u8)0x01))

1929 /* Check whether an address is broadcast. */
1930 #define IXGBE_IS_BROADCAST(Address) \
1931                 ((((u8 *)(Address))[0] == ((u8)0xff)) && \
1932                 (((u8 *)(Address))[1] == ((u8)0xff)))

1934 /* RAH */
1935 #define IXGBE_RAH_VIND_MASK     0x003C0000
1936 #define IXGBE_RAH_VIND_SHIFT    18
1937 #define IXGBE_RAH_AV            0x80000000
1938 #define IXGBE_CLEAR_VMDQ_ALL    0xFFFFFFFF

1940 /* Header split receive */
1941 #define IXGBE_RFCTL_ISCSI_DIS           0x00000001
1942 #define IXGBE_RFCTL_ISCSI_DWC_MASK      0x0000003E
1943 #define IXGBE_RFCTL_ISCSI_DWC_SHIFT     1
1944 #define IXGBE_RFCTL_RSC_DIS             0x00000010
1945 #define IXGBE_RFCTL_NFSW_DIS            0x00000040
1946 #define IXGBE_RFCTL_NFSR_DIS            0x00000080
1947 #define IXGBE_RFCTL_NFS_VER_MASK        0x00000300
1948 #define IXGBE_RFCTL_NFS_VER_SHIFT       8
1949 #define IXGBE_RFCTL_NFS_VER_2           0
1950 #define IXGBE_RFCTL_NFS_VER_3           1
1951 #define IXGBE_RFCTL_NFS_VER_4           2
1952 #define IXGBE_RFCTL_IPV6_DIS            0x00000400
1953 #define IXGBE_RFCTL_IPV6_XSUM_DIS       0x00000800
1954 #define IXGBE_RFCTL_IPFRSP_DIS          0x00004000
1955 #define IXGBE_RFCTL_IPV6_EX_DIS         0x00010000
1956 #define IXGBE_RFCTL_NEW_IPV6_EXT_DIS    0x00020000

1958 /* Transmit Config masks */
1959 #define IXGBE_TXDCTL_ENABLE             0x02000000 /* Ena specific Tx Queue */
1960 #define IXGBE_TXDCTL_SWFLSH             0x04000000 /* Tx Desc. wr-bk flushing */
1961 #define IXGBE_TXDCTL_WTHRESH_SHIFT      16 /* shift to WTHRESH bits */
1962 /* Enable short packet padding to 64 bytes */
1963 #define IXGBE_TX_PAD_ENABLE             0x00000400
1964 #define IXGBE_JUMBO_FRAME_ENABLE        0x00000004  /* Allow jumbo frames */
1965 /* This allows for 16K packets + 4k for vlan */
```

```
1966 #define IXGBE_MAX_FRAME_SZ              0x40040000

1968 #define IXGBE_TDWBAL_HEAD_WB_ENABLE     0x1 /* Tx head write-back enable */
1969 #define IXGBE_TDWBAL_SEQNUM_WB_ENABLE   0x2 /* Tx seq# write-back enable */

1971 /* Receive Config masks */
1972 #define IXGBE_RXCTRL_RXEN               0x00000001 /* Enable Receiver */
1973 #define IXGBE_RXCTRL_DMBYPS             0x00000002 /* Desc Monitor Bypass */
1974 #define IXGBE_RXDCTL_ENABLE             0x02000000 /* Ena specific Rx Queue */
1975 #define IXGBE_RXDCTL_SWFLSH             0x04000000 /* Rx Desc wr-bk flushing */
1976 #define IXGBE_RXDCTL_RLPMLMASK          0x00003FFF /* X540 supported only */
1977 #define IXGBE_RXDCTL_RLPML_EN           0x00008000
1978 #define IXGBE_RXDCTL_VME                0x40000000 /* VLAN mode enable */

1980 #define IXGBE_TSAUXC_EN_CLK             0x00000004
1981 #define IXGBE_TSAUXC_SYNCLK             0x00000008
1982 #define IXGBE_TSAUXC_SDP0_INT           0x00000040

1984 #define IXGBE_TSYNCTXCTL_VALID          0x00000001 /* Tx timestamp valid */
1985 #define IXGBE_TSYNCTXCTL_ENABLED        0x00000010 /* Tx timestamping enabled */

1987 #define IXGBE_TSYNCRXCTL_VALID          0x00000001 /* Rx timestamp valid */
1988 #define IXGBE_TSYNCRXCTL_TYPE_MASK      0x0000000E /* Rx type mask */
1989 #define IXGBE_TSYNCRXCTL_TYPE_L2_V2     0x00
1990 #define IXGBE_TSYNCRXCTL_TYPE_L4_V1     0x02
1991 #define IXGBE_TSYNCRXCTL_TYPE_L2_L4_V2  0x04
1992 #define IXGBE_TSYNCRXCTL_TYPE_EVENT_V2  0x0A
1993 #define IXGBE_TSYNCRXCTL_ENABLED        0x00000010 /* Rx Timestamping enabled */

1995 #define IXGBE_RXMTRL_V1_CTRLT_MASK      0x000000FF
1996 #define IXGBE_RXMTRL_V1_SYNC_MSG        0x00
1997 #define IXGBE_RXMTRL_V1_DELAY_REQ_MSG   0x01
1998 #define IXGBE_RXMTRL_V1_FOLLOWUP_MSG    0x02
1999 #define IXGBE_RXMTRL_V1_DELAY_RESP_MSG  0x03
2000 #define IXGBE_RXMTRL_V1_MGMT_MSG        0x04

2002 #define IXGBE_RXMTRL_V2_MSGID_MASK      0x0000FF00
2003 #define IXGBE_RXMTRL_V2_SYNC_MSG        0x0000
2004 #define IXGBE_RXMTRL_V2_DELAY_REQ_MSG   0x0100
2005 #define IXGBE_RXMTRL_V2_PDELAY_REQ_MSG  0x0200
2006 #define IXGBE_RXMTRL_V2_PDELAY_RESP_MSG 0x0300
2007 #define IXGBE_RXMTRL_V2_FOLLOWUP_MSG    0x0800
2008 #define IXGBE_RXMTRL_V2_DELAY_RESP_MSG  0x0900
2009 #define IXGBE_RXMTRL_V2_PDELAY_FOLLOWUP_MSG 0x0A00
2010 #define IXGBE_RXMTRL_V2_ANNOUNCE_MSG    0x0B00
2011 #define IXGBE_RXMTRL_V2_SIGNALLING_MSG  0x0C00
2012 #define IXGBE_RXMTRL_V2_MGMT_MSG        0x0D00

2014 #define IXGBE_FCTRL_SBP         0x00000002 /* Store Bad Packet */
2015 #define IXGBE_FCTRL_MPE         0x00000100 /* Multicast Promiscuous Ena*/
2016 #define IXGBE_FCTRL_UPE         0x00000200 /* Unicast Promiscuous Ena */
2017 #define IXGBE_FCTRL_BAM         0x00000400 /* Broadcast Accept Mode */
2018 #define IXGBE_FCTRL_PMCF        0x00001000 /* Pass MAC Control Frames */
2019 #define IXGBE_FCTRL_DPF         0x00002000 /* Discard Pause Frame */
2020 /* Receive Priority Flow Control Enable */
2021 #define IXGBE_FCTRL_RPFCE       0x00004000
2022 #define IXGBE_FCTRL_RFCE        0x00008000 /* Receive Flow Control Ena */
2023 #define IXGBE_MFLCN_PMCF        0x00000001 /* Pass MAC Control Frames */
2024 #define IXGBE_MFLCN_DPF         0x00000002 /* Discard Pause Frame */
2025 #define IXGBE_MFLCN_RPFCE       0x00000004 /* Receive Priority FC Enable */
2026 #define IXGBE_MFLCN_RFCE        0x00000008 /* Receive FC Enable */
2027 #define IXGBE_MFLCN_RPFCE_MASK  0x00000FF4 /* Rx Priority FC bitmap mask */
2028 #define IXGBE_MFLCN_RPFCE_SHIFT 4 /* Rx Priority FC bitmap shift */

2030 /* Multiple Receive Queue Control */
2031 #define IXGBE_MRQC_RSSEN        0x00000001  /* RSS Enable */
```



```
2032 #define IXGBE_MRQC_MRQE_MASK    0xF /* Bits 3:0 */
2033 #define IXGBE_MRQC_RT8TCEN      0x00000002 /* 8 TC no RSS */
2034 #define IXGBE_MRQC_RT4TCEN      0x00000003 /* 4 TC no RSS */
2035 #define IXGBE_MRQC_RTRSS8TCEN   0x00000004 /* 8 TC w/ RSS */
2036 #define IXGBE_MRQC_RTRSS4TCEN   0x00000005 /* 4 TC w/ RSS */
2037 #define IXGBE_MRQC_VMDQEN       0x00000008 /* VMDq2 64 pools no RSS */
2038 #define IXGBE_MRQC_VMDQRSS32EN  0x0000000A /* VMDq2 32 pools w/ RSS */
2039 #define IXGBE_MRQC_VMDQRSS64EN  0x0000000B /* VMDq2 64 pools w/ RSS */
2040 #define IXGBE_MRQC_VMDQRT8TCEN  0x0000000C /* VMDq2/RT 16 pool 8 TC */
2041 #define IXGBE_MRQC_VMDQRT4TCEN  0x0000000D /* VMDq2/RT 32 pool 4 TC */
2042 #define IXGBE_MRQC_RSS_FIELD_MASK       0xFFFF0000
2043 #define IXGBE_MRQC_RSS_FIELD_IPV4_TCP   0x00010000
2044 #define IXGBE_MRQC_RSS_FIELD_IPV4       0x00020000
2045 #define IXGBE_MRQC_RSS_FIELD_IPV6_EX_TCP 0x00040000
2046 #define IXGBE_MRQC_RSS_FIELD_IPV6_EX    0x00080000
2047 #define IXGBE_MRQC_RSS_FIELD_IPV6       0x00100000
2048 #define IXGBE_MRQC_RSS_FIELD_IPV6_TCP   0x00200000
2049 #define IXGBE_MRQC_RSS_FIELD_IPV4_UDP   0x00400000
2050 #define IXGBE_MRQC_RSS_FIELD_IPV6_UDP   0x00800000
2051 #define IXGBE_MRQC_RSS_FIELD_IPV6_EX_UDP 0x01000000
2052 #define IXGBE_MRQC_L3L4TXSWEN           0x00008000

2054 /* Queue Drop Enable */
2055 #define IXGBE_QDE_ENABLE        0x00000001
2056 #define IXGBE_QDE_IDX_MASK      0x00007F00
2057 #define IXGBE_QDE_IDX_SHIFT     8
2058 #define IXGBE_QDE_WRITE         0x00010000
2059 #define IXGBE_QDE_READ          0x00020000

2061 #define IXGBE_TXD_POPTS_IXSM    0x01 /* Insert IP checksum */
2062 #define IXGBE_TXD_POPTS_TXSM    0x02 /* Insert TCP/UDP checksum */
2063 #define IXGBE_TXD_CMD_EOP       0x01000000 /* End of Packet */
2064 #define IXGBE_TXD_CMD_IFCS      0x02000000 /* Insert FCS (Ethernet CRC) */
2065 #define IXGBE_TXD_CMD_IC        0x04000000 /* Insert Checksum */
2066 #define IXGBE_TXD_CMD_RS        0x08000000 /* Report Status */
2067 #define IXGBE_TXD_CMD_DEXT      0x20000000 /* Desc extension (0 = legacy) */
2068 #define IXGBE_TXD_CMD_VLE       0x40000000 /* Add VLAN tag */
2069 #define IXGBE_TXD_STAT_DD       0x00000001 /* Descriptor Done */

2071 #define IXGBE_RXDADV_IPSEC_STATUS_SECP          0x00020000
2072 #define IXGBE_RXDADV_IPSEC_ERROR_INVALID_PROTOCOL 0x08000000
2073 #define IXGBE_RXDADV_IPSEC_ERROR_INVALID_LENGTH 0x10000000
2074 #define IXGBE_RXDADV_IPSEC_ERROR_AUTH_FAILED    0x18000000
2075 #define IXGBE_RXDADV_IPSEC_ERROR_BIT_MASK       0x18000000
2076 /* Multiple Transmit Queue Command Register */
2077 #define IXGBE_MTQC_RT_ENA       0x1 /* DCB Enable */
2078 #define IXGBE_MTQC_VT_ENA       0x2 /* VMDQ2 Enable */
2079 #define IXGBE_MTQC_64Q_1PB      0x0 /* 64 queues 1 pack buffer */
2080 #define IXGBE_MTQC_32VF         0x8 /* 4 TX Queues per pool w/32VF's */
2081 #define IXGBE_MTQC_64VF         0x4 /* 2 TX Queues per pool w/64VF's */
2082 #define IXGBE_MTQC_4TC_4TQ      0x8 /* 4 TC if RT_ENA and VT_ENA */
2083 #define IXGBE_MTQC_8TC_8TQ      0xC /* 8 TC if RT_ENA or 8 TQ if VT_ENA */

2085 /* Receive Descriptor bit definitions */
2086 #define IXGBE_RXD_STAT_DD       0x01 /* Descriptor Done */
2087 #define IXGBE_RXD_STAT_EOP      0x02 /* End of Packet */
2088 #define IXGBE_RXD_STAT_FLM      0x04 /* FDir Match */
2089 #define IXGBE_RXD_STAT_VP       0x08 /* IEEE VLAN Packet */
2090 #define IXGBE_RXDADV_NEXTP_MASK 0x000FFFF0 /* Next Descriptor Index */
2091 #define IXGBE_RXDADV_NEXTP_SHIFT        0x00000004
2092 #define IXGBE_RXD_STAT_UDPCS    0x10 /* UDP xsum calculated */
2093 #define IXGBE_RXD_STAT_L4CS     0x20 /* L4 xsum calculated */
2094 #define IXGBE_RXD_STAT_IPCS     0x40 /* IP xsum calculated */
2095 #define IXGBE_RXD_STAT_PIF      0x80 /* passed in-exact filter */
2096 #define IXGBE_RXD_STAT_CRCV     0x100 /* Speculative CRC Valid */
2097 #define IXGBE_RXD_STAT_VEXT     0x200 /* 1st VLAN found */
```

```
2098 #define IXGBE_RXD_STAT_UDPV     0x400 /* Valid UDP checksum */
2099 #define IXGBE_RXD_STAT_DYNINT   0x800 /* Pkt caused INT via DYNINT */
2100 #define IXGBE_RXD_STAT_LLINT    0x800 /* Pkt caused Low Latency Interrupt */
2101 #define IXGBE_RXD_STAT_TS       0x10000 /* Time Stamp */
2102 #define IXGBE_RXD_STAT_SECP     0x20000 /* Security Processing */
2103 #define IXGBE_RXD_STAT_LB       0x40000 /* Loopback Status */
2104 #define IXGBE_RXD_STAT_ACK      0x8000 /* ACK Packet indication */
2105 #define IXGBE_RXD_ERR_CE        0x01 /* CRC Error */
2106 #define IXGBE_RXD_ERR_LE        0x02 /* Length Error */
2107 #define IXGBE_RXD_ERR_PE        0x08 /* Packet Error */
2108 #define IXGBE_RXD_ERR_OSE       0x10 /* Oversize Error */
2109 #define IXGBE_RXD_ERR_USE       0x20 /* Undersize Error */
2110 #define IXGBE_RXD_ERR_TCPE      0x40 /* TCP/UDP Checksum Error */
2111 #define IXGBE_RXD_ERR_IPE       0x80 /* IP Checksum Error */
2112 #define IXGBE_RXDADV_ERR_MASK           0xfff00000 /* RDESC.ERRORS mask */
2113 #define IXGBE_RXDADV_ERR_SHIFT          20 /* RDESC.ERRORS shift */
2114 #define IXGBE_RXDADV_ERR_RXE            0x20000000 /* Any MAC Error */
2115 #define IXGBE_RXDADV_ERR_FCEOFE         0x80000000 /* FCoEFe/IPE */
2116 #define IXGBE_RXDADV_ERR_FCERR          0x00700000 /* FCERR/FDIRERR */
2117 #define IXGBE_RXDADV_ERR_FDIR_LEN       0x00100000 /* FDIR Length error */
2118 #define IXGBE_RXDADV_ERR_FDIR_DROP      0x00200000 /* FDIR Drop error */
2119 #define IXGBE_RXDADV_ERR_FDIR_COLL      0x00400000 /* FDIR Collision error */
2120 #define IXGBE_RXDADV_ERR_HBO    0x00800000 /*Header Buffer Overflow */
2121 #define IXGBE_RXDADV_ERR_CE     0x01000000 /* CRC Error */
2122 #define IXGBE_RXDADV_ERR_LE     0x02000000 /* Length Error */
2123 #define IXGBE_RXDADV_ERR_PE     0x08000000 /* Packet Error */
2124 #define IXGBE_RXDADV_ERR_OSE    0x10000000 /* Oversize Error */
2125 #define IXGBE_RXDADV_ERR_USE    0x20000000 /* Undersize Error */
2126 #define IXGBE_RXDADV_ERR_TCPE   0x40000000 /* TCP/UDP Checksum Error */
2127 #define IXGBE_RXDADV_ERR_IPE    0x80000000 /* IP Checksum Error */
2128 #define IXGBE_RXD_VLAN_ID_MASK  0x0FFF  /* VLAN ID is in lower 12 bits */
2129 #define IXGBE_RXD_PRI_MASK      0xE000  /* Priority is in upper 3 bits */
2130 #define IXGBE_RXD_PRI_SHIFT     13
2131 #define IXGBE_RXD_CFI_MASK      0x1000  /* CFI is bit 12 */
2132 #define IXGBE_RXD_CFI_SHIFT     12

2134 #define IXGBE_RXDADV_STAT_DD            IXGBE_RXD_STAT_DD  /* Done */
2135 #define IXGBE_RXDADV_STAT_EOP           IXGBE_RXD_STAT_EOP /* End of Packet */
2136 #define IXGBE_RXDADV_STAT_FLM           IXGBE_RXD_STAT_FLM /* FDir Match */
2137 #define IXGBE_RXDADV_STAT_VP            IXGBE_RXD_STAT_VP  /* IEEE VLAN Pkt */
2138 #define IXGBE_RXDADV_STAT_MASK          0x000fffff /* Stat/NEXTP: bit 0-19 */
2139 #define IXGBE_RXDADV_STAT_FCEOFS        0x00000040 /* FCoE EOF/SOF Stat */
2140 #define IXGBE_RXDADV_STAT_FCSTAT        0x00000030 /* FCoE Pkt Stat */
2141 #define IXGBE_RXDADV_STAT_FCSTAT_NOMTCH 0x00000000 /* 00: No Ctxt Match */
2142 #define IXGBE_RXDADV_STAT_FCSTAT_NODDP  0x00000010 /* 01: Ctxt w/o DDP */
2143 #define IXGBE_RXDADV_STAT_FCSTAT_FCPRSP 0x00000020 /* 10: Recv. FCP_RSP */
2144 #define IXGBE_RXDADV_STAT_FCSTAT_DDP    0x00000030 /* 11: Ctxt w/ DDP */
2145 #define IXGBE_RXDADV_STAT_TS            0x00010000 /* IEEE1588 Time Stamp */

2147 /* PSRTYPE bit definitions */
2148 #define IXGBE_PSRTYPE_TCPHDR    0x00000010
2149 #define IXGBE_PSRTYPE_UDPHDR    0x00000020
2150 #define IXGBE_PSRTYPE_IPV4HDR   0x00000100
2151 #define IXGBE_PSRTYPE_IPV6HDR   0x00000200
2152 #define IXGBE_PSRTYPE_L2HDR     0x00001000

2154 /* SRRCTL bit definitions */
2155 #define IXGBE_SRRCTL_BSIZEPKT_SHIFT     10 /* so many KBs */
2156 #define IXGBE_SRRCTL_RDMTS_SHIFT        22
2157 #define IXGBE_SRRCTL_RDMTS_MASK         0x01C00000
2158 #define IXGBE_SRRCTL_DROP_EN            0x10000000
2159 #define IXGBE_SRRCTL_BSIZEPKT_MASK      0x0000007F
2160 #define IXGBE_SRRCTL_BSIZEHDR_MASK      0x00003F00
2161 #define IXGBE_SRRCTL_DESCTYPE_LEGACY    0x00000000
2162 #define IXGBE_SRRCTL_DESCTYPE_ADV_ONEBUF 0x02000000
2163 #define IXGBE_SRRCTL_DESCTYPE_HDR_SPLIT 0x04000000
```



```
2164 #define IXGBE_SRRCTL_DESCTYPE_HDR_REPLICATION_LARGE_PKT 0x08000000
2165 #define IXGBE_SRRCTL_DESCTYPE_HDR_SPLIT_ALWAYS 0x0A000000
2166 #define IXGBE_SRRCTL_DESCTYPE_MASK      0x0E000000

2168 #define IXGBE_RXDPS_HDRSTAT_HDRSP       0x00008000
2169 #define IXGBE_RXDPS_HDRSTAT_HDRLEN_MASK 0x000003FF

2171 #define IXGBE_RXDADV_RSSTYPE_MASK       0x0000000F
2172 #define IXGBE_RXDADV_PKTTYPE_MASK       0x0000FFF0
2173 #define IXGBE_RXDADV_PKTTYPE_MASK_EX    0x0001FFF0
2174 #define IXGBE_RXDADV_HDRBUFLEN_MASK     0x00007FE0
2175 #define IXGBE_RXDADV_RSCCNT_MASK        0x001E0000
2176 #define IXGBE_RXDADV_RSCCNT_SHIFT       17
2177 #define IXGBE_RXDADV_HDRBUFLEN_SHIFT    5
2178 #define IXGBE_RXDADV_SPLITHEADER_EN     0x00001000
2179 #define IXGBE_RXDADV_SPH                0x8000

2181 /* RSS Hash results */
2182 #define IXGBE_RXDADV_RSSTYPE_NONE       0x00000000
2183 #define IXGBE_RXDADV_RSSTYPE_IPV4_TCP   0x00000001
2184 #define IXGBE_RXDADV_RSSTYPE_IPV4       0x00000002
2185 #define IXGBE_RXDADV_RSSTYPE_IPV6_TCP   0x00000003
2186 #define IXGBE_RXDADV_RSSTYPE_IPV6_EX    0x00000004
2187 #define IXGBE_RXDADV_RSSTYPE_IPV6       0x00000005
2188 #define IXGBE_RXDADV_RSSTYPE_IPV6_TCP_EX 0x00000006
2189 #define IXGBE_RXDADV_RSSTYPE_IPV4_UDP   0x00000007
2190 #define IXGBE_RXDADV_RSSTYPE_IPV6_UDP   0x00000008
2191 #define IXGBE_RXDADV_RSSTYPE_IPV6_UDP_EX 0x00000009

2193 /* RSS Packet Types as indicated in the receive descriptor. */
2194 #define IXGBE_RXDADV_PKTTYPE_NONE       0x00000000
2195 #define IXGBE_RXDADV_PKTTYPE_IPV4       0x00000010 /* IPv4 hdr present */
2196 #define IXGBE_RXDADV_PKTTYPE_IPV4_EX    0x00000020 /* IPv4 hdr + extensions */
2197 #define IXGBE_RXDADV_PKTTYPE_IPV6       0x00000040 /* IPv6 hdr present */
2198 #define IXGBE_RXDADV_PKTTYPE_IPV6_EX    0x00000080 /* IPv6 hdr + extensions */
2199 #define IXGBE_RXDADV_PKTTYPE_TCP        0x00000100 /* TCP hdr present */
2200 #define IXGBE_RXDADV_PKTTYPE_UDP        0x00000200 /* UDP hdr present */
2201 #define IXGBE_RXDADV_PKTTYPE_SCTP       0x00000400 /* SCTP hdr present */
2202 #define IXGBE_RXDADV_PKTTYPE_NFS        0x00000800 /* NFS hdr present */
2203 #define IXGBE_RXDADV_PKTTYPE_IPSEC_ESP  0x00001000 /* IPSec ESP */
2204 #define IXGBE_RXDADV_PKTTYPE_IPSEC_AH   0x00002000 /* IPSec AH */
2205 #define IXGBE_RXDADV_PKTTYPE_LINKSEC    0x00004000 /* LinkSec Encap */
2206 #define IXGBE_RXDADV_PKTTYPE_ETQF       0x00008000 /* PKTTYPE is ETQF index */
2207 #define IXGBE_RXDADV_PKTTYPE_ETQF_MASK  0x00000070 /* ETQF has 8 indices */
2208 #define IXGBE_RXDADV_PKTTYPE_ETQF_SHIFT 4 /* Right-shift 4 bits */

2210 /* Security Processing bit Indication */
2211 #define IXGBE_RXDADV_LNKSEC_STATUS_SECP         0x00020000
2212 #define IXGBE_RXDADV_LNKSEC_ERROR_NO_SA_MATCH   0x08000000
2213 #define IXGBE_RXDADV_LNKSEC_ERROR_REPLAY_ERROR  0x10000000
2214 #define IXGBE_RXDADV_LNKSEC_ERROR_BIT_MASK      0x18000000
2215 #define IXGBE_RXDADV_LNKSEC_ERROR_BAD_SIG       0x18000000

2217 /* Masks to determine if packets should be dropped due to frame errors */
2218 #define IXGBE_RXD_ERR_FRAME_ERR_MASK ( \
2219                                 IXGBE_RXD_ERR_CE | \
2220                                 IXGBE_RXD_ERR_LE | \
2221                                 IXGBE_RXD_ERR_PE | \
2222                                 IXGBE_RXD_ERR_OSE | \
2223                                 IXGBE_RXD_ERR_USE)

2225 #define IXGBE_RXDADV_ERR_FRAME_ERR_MASK ( \
2226                                 IXGBE_RXDADV_ERR_CE | \
2227                                 IXGBE_RXDADV_ERR_LE | \
2228                                 IXGBE_RXDADV_ERR_PE | \
2229                                 IXGBE_RXDADV_ERR_OSE | \
```

```
2230                                 IXGBE_RXDADV_ERR_USE)

2232 #define IXGBE_RXDADV_ERR_FRAME_ERR_MASK_82599   IXGBE_RXDADV_ERR_RXE

2234 /* Multicast bit mask */
2235 #define IXGBE_MCSTCTRL_MFE      0x4

2237 /* Number of Transmit and Receive Descriptors must be a multiple of 8 */
2238 #define IXGBE_REQ_TX_DESCRIPTOR_MULTIPLE        8
2239 #define IXGBE_REQ_RX_DESCRIPTOR_MULTIPLE        8
2240 #define IXGBE_REQ_TX_BUFFER_GRANULARITY         1024

2242 /* Vlan-specific macros */
2243 #define IXGBE_RX_DESC_SPECIAL_VLAN_MASK 0x0FFF /* VLAN ID in lower 12 bits */
2244 #define IXGBE_RX_DESC_SPECIAL_PRI_MASK  0xE000 /* Priority in upper 3 bits */
2245 #define IXGBE_RX_DESC_SPECIAL_PRI_SHIFT 0x000D /* Priority in upper 3 of 16 */
2246 #define IXGBE_TX_DESC_SPECIAL_PRI_SHIFT IXGBE_RX_DESC_SPECIAL_PRI_SHIFT

2248 /* SR-IOV specific macros */
2249 #define IXGBE_MBVFICR_INDEX(vf_number)  (vf_number >> 4)
2250 #define IXGBE_MBVFICR(_i)               (0x00710 + ((_i) * 4))
2251 #define IXGBE_VFLRE(_i)                 (((_i & 1) ? 0x001C0 : 0x00600))
2252 #define IXGBE_VFLREC(_i)                 (0x00700 + ((_i) * 4))

2254 /* Little Endian defines */
2255 #ifndef __le16
2256 #define __le16  u16
2257 #endif
2258 #ifndef __le32
2259 #define __le32  u32
2260 #endif
2261 #ifndef __le64
2262 #define __le64  u64

2264 #endif
2265 #ifndef __be16
2266 /* Big Endian defines */
2267 #define __be16  u16
2268 #define __be32  u32
2269 #define __be64  u64

2271 #endif
2272 enum ixgbe_fdir_pballoc_type {
2273         IXGBE_FDIR_PBALLOC_NONE = 0,
2274         IXGBE_FDIR_PBALLOC_64K  = 1,
2275         IXGBE_FDIR_PBALLOC_128K = 2,
2276         IXGBE_FDIR_PBALLOC_256K = 3,
2277 };
```

*unchanged_portion_omitted_*

```
2473 /* Adv Transmit Descriptor Config Masks */
2474 #define IXGBE_ADVTXD_DTALEN_MASK        0x0000FFFF /* Data buf length(bytes) */
2475 #define IXGBE_ADVTXD_MAC_LINKSEC        0x00040000 /* Insert LinkSec */
2476 #define IXGBE_ADVTXD_MAC_TSTAMP         0x00080000 /* IEEE1588 time stamp */
2477 #define IXGBE_ADVTXD_IPSEC_SA_INDEX_MASK 0x000003FF /* IPSec SA index */
2478 #define IXGBE_ADVTXD_IPSEC_ESP_LEN_MASK 0x000001FF /* IPSec ESP length */
2479 #define IXGBE_ADVTXD_DTYP_MASK          0x00F00000 /* DTYP mask */
2480 #define IXGBE_ADVTXD_DTYP_CTXT          0x00200000 /* Adv Context Desc */
2481 #define IXGBE_ADVTXD_DTYP_DATA          0x00300000 /* Adv Data Descriptor */
2482 #define IXGBE_ADVTXD_DCMD_EOP           IXGBE_TXD_CMD_EOP  /* End of Packet */
2483 #define IXGBE_ADVTXD_DCMD_IFCS          IXGBE_TXD_CMD_IFCS /* Insert FCS */
2484 #define IXGBE_ADVTXD_DCMD_RS            IXGBE_TXD_CMD_RS /* Report Status */
2485 #define IXGBE_ADVTXD_DCMD_DDTYP_ISCSI   0x10000000 /* DDP hdr type or iSCSI */
2486 #define IXGBE_ADVTXD_DCMD_DEXT          IXGBE_TXD_CMD_DEXT /* Desc ext 1=Adv */
2487 #define IXGBE_ADVTXD_DCMD_VLE           IXGBE_TXD_CMD_VLE  /* VLAN pkt enable */
2488 #define IXGBE_ADVTXD_DCMD_TSE           0x80000000 /* TCP Seg enable */
```



```
2489 #define IXGBE_ADVTXD_STAT_DD            IXGBE_TXD_STAT_DD  /* Descriptor Done */
2490 #define IXGBE_ADVTXD_STAT_SN_CRC        0x00000002 /* NXTSEQ/SEED pres in WB */
2491 #define IXGBE_ADVTXD_STAT_RSV           0x0000000C /* STA Reserved */
2492 #define IXGBE_ADVTXD_IDX_SHIFT          4 /* Adv desc Index shift */
2493 #define IXGBE_ADVTXD_CC                 0x00000080 /* Check Context */
2494 #define IXGBE_ADVTXD_POPTS_SHIFT        8  /* Adv desc POPTS shift */
2495 #define IXGBE_ADVTXD_POPTS_IXSM         (IXGBE_TXD_POPTS_IXSM << \
2496                                          IXGBE_ADVTXD_POPTS_SHIFT)
2497 #define IXGBE_ADVTXD_POPTS_TXSM         (IXGBE_TXD_POPTS_TXSM << \
2498                                          IXGBE_ADVTXD_POPTS_SHIFT)
2499 #define IXGBE_ADVTXD_POPTS_ISCO_1ST     0x00000000 /* 1st TSO of iSCSI PDU */
2500 #define IXGBE_ADVTXD_POPTS_ISCO_MDL     0x00000800 /* Middle TSO of iSCSI PDU */
2501 #define IXGBE_ADVTXD_POPTS_ISCO_LAST    0x00001000 /* Last TSO of iSCSI PDU */
2502 /* 1st&Last TSO-full iSCSI PDU */
2503 #define IXGBE_ADVTXD_POPTS_ISCO_FULL    0x00001800
2504 #define IXGBE_ADVTXD_POPTS_RSV          0x00002000 /* POPTS Reserved */
2505 #define IXGBE_ADVTXD_PAYLEN_SHIFT       14 /* Adv desc PAYLEN shift */
2506 #define IXGBE_ADVTXD_MACLEN_SHIFT       9  /* Adv ctxt desc mac len shift */
2507 #define IXGBE_ADVTXD_VLAN_SHIFT         16  /* Adv ctxt vlan tag shift */
2508 #define IXGBE_ADVTXD_TUCMD_IPV4         0x00000400 /* IP Packet Type: 1=IPv4 */
2509 #define IXGBE_ADVTXD_TUCMD_IPV6         0x00000000 /* IP Packet Type: 0=IPv6 */
2510 #define IXGBE_ADVTXD_TUCMD_L4T_UDP      0x00000000 /* L4 Packet TYPE of UDP */
2511 #define IXGBE_ADVTXD_TUCMD_L4T_TCP      0x00000800 /* L4 Packet TYPE of TCP */
2512 #define IXGBE_ADVTXD_TUCMD_L4T_SCTP     0x00001000 /* L4 Packet TYPE of SCTP */
2513 #define IXGBE_ADVTXD_TUCMD_MKRREQ       0x00002000 /* req Markers and CRC */
2514 #define IXGBE_ADVTXD_POPTS_IPSEC        0x00000400 /* IPSec offload request */
2515 #define IXGBE_ADVTXD_TUCMD_IPSEC_TYPE_ESP 0x00002000 /* IPSec Type ESP */
2516 #define IXGBE_ADVTXD_TUCMD_IPSEC_ENCRYPT_EN 0x00004000/* ESP Encrypt Enable */
2517 #define IXGBE_ADVTXT_TUCMD_FCOE         0x00008000 /* FCoE Frame Type */
2518 #define IXGBE_ADVTXD_FCOEF_EOF_MASK     (0x3 << 10) /* FC EOF index */
2519 #define IXGBE_ADVTXD_FCOEF_SOF          ((1 << 2) << 10) /* FC SOF index */
2520 #define IXGBE_ADVTXD_FCOEF_PARINC       ((1 << 3) << 10) /* Rel_Off in F_CTL */
2521 #define IXGBE_ADVTXD_FCOEF_ORIE         ((1 << 4) << 10) /* Orientation End */
2522 #define IXGBE_ADVTXD_FCOEF_ORIS         ((1 << 5) << 10) /* Orientation Start */
2523 #define IXGBE_ADVTXD_FCOEF_EOF_N        (0x0 << 10) /* 00: EOFn */
2524 #define IXGBE_ADVTXD_FCOEF_EOF_T        (0x1 << 10) /* 01: EOFt */
2525 #define IXGBE_ADVTXD_FCOEF_EOF_NI       (0x2 << 10) /* 10: EOFni */
2526 #define IXGBE_ADVTXD_FCOEF_EOF_A        (0x3 << 10) /* 11: EOFa */
2527 #define IXGBE_ADVTXD_L4LEN_SHIFT        8  /* Adv ctxt L4LEN shift */
2528 #define IXGBE_ADVTXD_MSS_SHIFT          16  /* Adv ctxt MSS shift */

2530 /* Autonegotiation advertised speeds */
2531 typedef u32 ixgbe_autoneg_advertised;
2532 /* Link speed */
2533 typedef u32 ixgbe_link_speed;
2534 #define IXGBE_LINK_SPEED_UNKNOWN        0
2535 #define IXGBE_LINK_SPEED_100_FULL       0x0008
2536 #define IXGBE_LINK_SPEED_1GB_FULL       0x0020
2537 #define IXGBE_LINK_SPEED_10GB_FULL      0x0080
2538 #define IXGBE_LINK_SPEED_82598_AUTONEG  (IXGBE_LINK_SPEED_1GB_FULL | \
2539                                          IXGBE_LINK_SPEED_10GB_FULL)
2540 #define IXGBE_LINK_SPEED_82599_AUTONEG  (IXGBE_LINK_SPEED_100_FULL | \
2541                                          IXGBE_LINK_SPEED_1GB_FULL | \
2542                                          IXGBE_LINK_SPEED_10GB_FULL)


2544 /* Physical layer type */
2545 typedef u32 ixgbe_physical_layer;
2546 #define IXGBE_PHYSICAL_LAYER_UNKNOWN            0
2547 #define IXGBE_PHYSICAL_LAYER_10GBASE_T          0x0001
2548 #define IXGBE_PHYSICAL_LAYER_1000BASE_T         0x0002
2549 #define IXGBE_PHYSICAL_LAYER_100BASE_TX         0x0004
2550 #define IXGBE_PHYSICAL_LAYER_SFP_PLUS_CU        0x0008
2551 #define IXGBE_PHYSICAL_LAYER_10GBASE_LR         0x0010
2552 #define IXGBE_PHYSICAL_LAYER_10GBASE_LRM        0x0020
2553 #define IXGBE_PHYSICAL_LAYER_10GBASE_SR         0x0040
```

```
#define IXGBE_PHYSICAL_LAYER_10GBASE_KX4        0x0080
#define IXGBE_PHYSICAL_LAYER_10GBASE_CX4        0x0100
#define IXGBE_PHYSICAL_LAYER_1000BASE_KX        0x0200
#define IXGBE_PHYSICAL_LAYER_1000BASE_BX        0x0400
#define IXGBE_PHYSICAL_LAYER_10GBASE_KR         0x0800
#define IXGBE_PHYSICAL_LAYER_10GBASE_XAUI       0x1000
#define IXGBE_PHYSICAL_LAYER_SFP_ACTIVE_DA      0x2000
#define IXGBE_PHYSICAL_LAYER_1000BASE_SX        0x4000

/* Flow Control Data Sheet defined values
 * Calculation and defines taken from 802.1bb Annex O
 */

/* BitTimes (BT) conversion */
#define IXGBE_BT2KB(BT)         ((BT + (8 * 1024 - 1)) / (8 * 1024))
#define IXGBE_B2BT(BT)          (BT * 8)

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

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

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

#define IXGBE_ID_X540   (IXGBE_MAC_DC + IXGBE_XAUI_DC + IXGBE_PHY_DC)

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

#define IXGBE_ID        (IXGBE_MAC_D + IXGBE_XAUI_D + IXGBE_PHY_D)

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

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

/* Calculate X540 delay value in bit times */
#define IXGBE_DV_X540(_max_frame_link, _max_frame_tc) \
                        ((36 * \
                          (IXGBE_B2BT(_max_frame_link) + \
                           IXGBE_PFC_D + \
                           (2 * IXGBE_CABLE_DC) + \
                           (2 * IXGBE_ID_X540) + \
                           IXGBE_HD) / 25 + 1) + \
                         2 * IXGBE_B2BT(_max_frame_tc))

/* Calculate 82599, 82598 delay value in bit times */
#define IXGBE_DV(_max_frame_link, _max_frame_tc) \
                        ((36 * \
                          (IXGBE_B2BT(_max_frame_link) + \
                           IXGBE_PFC_D + \
                           (2 * IXGBE_CABLE_DC) + \
                           (2 * IXGBE_ID) + \
                           IXGBE_HD) / 25 + 1) + \
                         2 * IXGBE_B2BT(_max_frame_tc))

/* Calculate low threshold delay values */
#define IXGBE_LOW_DV_X540(_max_frame_tc) \
```



```
                        (2 * IXGBE_B2BT(_max_frame_tc) + \
                        (36 * IXGBE_PCI_DELAY / 25) + 1)
#define IXGBE_LOW_DV(_max_frame_tc) \
                        (2 * IXGBE_LOW_DV_X540(_max_frame_tc))

/* Software ATR hash keys */
#define IXGBE_ATR_BUCKET_HASH_KEY       0x3DAD14E2
#define IXGBE_ATR_SIGNATURE_HASH_KEY    0x174D3614

/* Software ATR input stream values and masks */
#define IXGBE_ATR_HASH_MASK             0x7fff
#define IXGBE_ATR_L4TYPE_MASK           0x3
#define IXGBE_ATR_L4TYPE_UDP            0x1
#define IXGBE_ATR_L4TYPE_TCP            0x2
#define IXGBE_ATR_L4TYPE_SCTP           0x3
#define IXGBE_ATR_L4TYPE_IPV6_MASK      0x4
enum ixgbe_atr_flow_type {
        IXGBE_ATR_FLOW_TYPE_IPV4        = 0x0,
        IXGBE_ATR_FLOW_TYPE_UDPV4       = 0x1,
        IXGBE_ATR_FLOW_TYPE_TCPV4       = 0x2,
        IXGBE_ATR_FLOW_TYPE_SCTPV4      = 0x3,
        IXGBE_ATR_FLOW_TYPE_IPV6        = 0x4,
        IXGBE_ATR_FLOW_TYPE_UDPV6       = 0x5,
        IXGBE_ATR_FLOW_TYPE_TCPV6       = 0x6,
        IXGBE_ATR_FLOW_TYPE_SCTPV6      = 0x7,
};
______unchanged_portion_omitted_

enum ixgbe_media_type {
        ixgbe_media_type_unknown = 0,
        ixgbe_media_type_fiber,
        ixgbe_media_type_fiber_fixed,
        ixgbe_media_type_copper,
        ixgbe_media_type_backplane,
        ixgbe_media_type_cx4,
        ixgbe_media_type_virtual
};
______unchanged_portion_omitted_

struct ixgbe_mac_operations {
        s32 (*init_hw)(struct ixgbe_hw *);
        s32 (*reset_hw)(struct ixgbe_hw *);
        s32 (*start_hw)(struct ixgbe_hw *);
        s32 (*clear_hw_cntrs)(struct ixgbe_hw *);
        void (*enable_relaxed_ordering)(struct ixgbe_hw *);
        enum ixgbe_media_type (*get_media_type)(struct ixgbe_hw *);
        u32 (*get_supported_physical_layer)(struct ixgbe_hw *);
        s32 (*get_mac_addr)(struct ixgbe_hw *, u8 *);
        s32 (*get_san_mac_addr)(struct ixgbe_hw *, u8 *);
        s32 (*set_san_mac_addr)(struct ixgbe_hw *, u8 *);
        s32 (*get_device_caps)(struct ixgbe_hw *, u16 *);
        s32 (*get_wwn_prefix)(struct ixgbe_hw *, u16 *, u16 *);
        s32 (*get_fcoe_boot_status)(struct ixgbe_hw *, u16 *);
        s32 (*stop_adapter)(struct ixgbe_hw *);
        s32 (*get_bus_info)(struct ixgbe_hw *);
        void (*set_lan_id)(struct ixgbe_hw *);
        s32 (*read_analog_reg8)(struct ixgbe_hw*, u32, u8*);
        s32 (*write_analog_reg8)(struct ixgbe_hw*, u32, u8);
        s32 (*setup_sfp)(struct ixgbe_hw *);
        s32 (*enable_rx_dma)(struct ixgbe_hw *, u32);
        s32 (*disable_sec_rx_path)(struct ixgbe_hw *);
        s32 (*enable_sec_rx_path)(struct ixgbe_hw *);
        s32 (*acquire_swfw_sync)(struct ixgbe_hw *, u16);
        void (*release_swfw_sync)(struct ixgbe_hw *, u16);

        /* Link */
```

```
2992         void (*disable_tx_laser)(struct ixgbe_hw *);
2993         void (*enable_tx_laser)(struct ixgbe_hw *);
2994         void (*flap_tx_laser)(struct ixgbe_hw *);
2995         s32 (*setup_link)(struct ixgbe_hw *, ixgbe_link_speed, bool);
2978         s32 (*setup_link)(struct ixgbe_hw *, ixgbe_link_speed, bool, bool);
2996         s32 (*check_link)(struct ixgbe_hw *, ixgbe_link_speed *, bool *, bool);
2997         s32 (*get_link_capabilities)(struct ixgbe_hw *, ixgbe_link_speed *,
2998                                      bool *);

3000         /* Packet Buffer manipulation */
3001         void (*setup_rxpba)(struct ixgbe_hw *, int, u32, int);

3003         /* LED */
3004         s32 (*led_on)(struct ixgbe_hw *, u32);
3005         s32 (*led_off)(struct ixgbe_hw *, u32);
3006         s32 (*blink_led_start)(struct ixgbe_hw *, u32);
3007         s32 (*blink_led_stop)(struct ixgbe_hw *, u32);

3009         /* RAR, Multicast, VLAN */
3010         s32 (*set_rar)(struct ixgbe_hw *, u32, u8 *, u32, u32);
3011         s32 (*set_uc_addr)(struct ixgbe_hw *, u32, u8 *);
3012         s32 (*clear_rar)(struct ixgbe_hw *, u32);
3013         s32 (*insert_mac_addr)(struct ixgbe_hw *, u8 *, u32);
3014         s32 (*set_vmdq)(struct ixgbe_hw *, u32, u32);
3015         s32 (*set_vmdq_san_mac)(struct ixgbe_hw *, u32);
3016         s32 (*clear_vmdq)(struct ixgbe_hw *, u32, u32);
3017         s32 (*init_rx_addrs)(struct ixgbe_hw *);
3018         s32 (*update_uc_addr_list)(struct ixgbe_hw *, u8 *, u32,
3019                                    ixgbe_mc_addr_itr);
3020         s32 (*update_mc_addr_list)(struct ixgbe_hw *, u8 *, u32,
3021                                    ixgbe_mc_addr_itr, bool clear);
3022         s32 (*enable_mc)(struct ixgbe_hw *);
3023         s32 (*disable_mc)(struct ixgbe_hw *);
3024         s32 (*clear_vfta)(struct ixgbe_hw *);
3025         s32 (*set_vfta)(struct ixgbe_hw *, u32, u32, bool);
3026         s32 (*set_vlvf)(struct ixgbe_hw *, u32, u32, bool, bool *);
3027         s32 (*init_uta_tables)(struct ixgbe_hw *);
3028         void (*set_mac_anti_spoofing)(struct ixgbe_hw *, bool, int);
3029         void (*set_vlan_anti_spoofing)(struct ixgbe_hw *, bool, int);

3031         /* Flow Control */
3032         s32 (*fc_enable)(struct ixgbe_hw *);

3034         /* Manageability interface */
3035         s32 (*set_fw_drv_ver)(struct ixgbe_hw *, u8, u8, u8, u8);
3036 };

3038 struct ixgbe_phy_operations {
3039         s32 (*identify)(struct ixgbe_hw *);
3040         s32 (*identify_sfp)(struct ixgbe_hw *);
3041         s32 (*init)(struct ixgbe_hw *);
3042         s32 (*reset)(struct ixgbe_hw *);
3043         s32 (*read_reg)(struct ixgbe_hw *, u32, u32, u16 *);
3044         s32 (*write_reg)(struct ixgbe_hw *, u32, u32, u16);
3045         s32 (*setup_link)(struct ixgbe_hw *);
3046         s32 (*setup_link_speed)(struct ixgbe_hw *, ixgbe_link_speed, bool);
3029         s32 (*setup_link_speed)(struct ixgbe_hw *, ixgbe_link_speed, bool,
3030                                 bool);
3047         s32 (*check_link)(struct ixgbe_hw *, ixgbe_link_speed *, bool *);
3048         s32 (*get_firmware_version)(struct ixgbe_hw *, u16 *);
3049         s32 (*read_i2c_byte)(struct ixgbe_hw *, u8, u8, u8 *);
3050         s32 (*write_i2c_byte)(struct ixgbe_hw *, u8, u8, u8);
3051         s32 (*read_i2c_sff8472)(struct ixgbe_hw *, u8 , u8 *);
3052         s32 (*read_i2c_eeprom)(struct ixgbe_hw *, u8 , u8 *);
3053         s32 (*write_i2c_eeprom)(struct ixgbe_hw *, u8, u8);
3054         void (*i2c_bus_clear)(struct ixgbe_hw *);
```



```
3055         s32 (*check_overtemp)(struct ixgbe_hw *);
3056 };
______unchanged_portion_omitted_

3067 #define IXGBE_FLAGS_DOUBLE_RESET_REQUIRED       0x01
3068 struct ixgbe_mac_info {
3069         struct ixgbe_mac_operations ops;
3070         enum ixgbe_mac_type type;
3071         u8 addr[IXGBE_ETH_LENGTH_OF_ADDRESS];
3072         u8 perm_addr[IXGBE_ETH_LENGTH_OF_ADDRESS];
3073         u8 san_addr[IXGBE_ETH_LENGTH_OF_ADDRESS];
3074         /* prefix for World Wide Node Name (WWNN) */
3075         u16 wwnn_prefix;
3076         /* prefix for World Wide Port Name (WWPN) */
3077         u16 wwpn_prefix;
3078 #define IXGBE_MAX_MTA                   128
3079         u32 mta_shadow[IXGBE_MAX_MTA];
3080         s32 mc_filter_type;
3081         u32 mcft_size;
3082         u32 vft_size;
3083         u32 num_rar_entries;
3084         u32 rar_highwater;
3085         u32 rx_pb_size;
3086         u32 max_tx_queues;
3087         u32 max_rx_queues;
3088         u32 orig_autoc;
3089         u32 cached_autoc;
3090         u8  san_mac_rar_index;
3091         bool get_link_status;
3092         u32 orig_autoc2;
3093         u16 max_msix_vectors;
3094         bool arc_subsystem_valid;
3095         bool orig_link_settings_stored;
3096         bool autotry_restart;
3097         u8 flags;
3098 };
______unchanged_portion_omitted_

3148 struct ixgbe_hw {
3149         u8 *hw_addr;
3150         void *back;
3151         struct ixgbe_mac_info mac;
3152         struct ixgbe_addr_filter_info addr_ctrl;
3153         struct ixgbe_fc_info fc;
3154         struct ixgbe_phy_info phy;
3155         struct ixgbe_eeprom_info eeprom;
3156         struct ixgbe_bus_info bus;
3157         struct ixgbe_mbx_info mbx;
3158         u16 device_id;
3159         u16 vendor_id;
3160         u16 subsystem_device_id;
3161         u16 subsystem_vendor_id;
3162         u8 revision_id;
3163         bool adapter_stopped;
3164         int api_version;
3165         bool force_full_reset;
3166         bool allow_unsupported_sfp;
3167 };

3169 #define ixgbe_call_func(hw, func, params, error) \
3170                 (func != NULL) ? func params : error


3173 /* Error Codes */
3174 #define IXGBE_SUCCESS                           0
3175 #define IXGBE_ERR_EEPROM                        -1
```

```
3176 #define IXGBE_ERR_EEPROM_CHECKSUM               -2
3177 #define IXGBE_ERR_PHY                          -3
3178 #define IXGBE_ERR_CONFIG                       -4
3179 #define IXGBE_ERR_PARAM                        -5
3180 #define IXGBE_ERR_MAC_TYPE                     -6
3181 #define IXGBE_ERR_UNKNOWN_PHY                  -7
3182 #define IXGBE_ERR_LINK_SETUP                   -8
3183 #define IXGBE_ERR_ADAPTER_STOPPED              -9
3184 #define IXGBE_ERR_INVALID_MAC_ADDR             -10
3185 #define IXGBE_ERR_DEVICE_NOT_SUPPORTED         -11
3186 #define IXGBE_ERR_MASTER_REQUESTS_PENDING      -12
3187 #define IXGBE_ERR_INVALID_LINK_SETTINGS        -13
3188 #define IXGBE_ERR_AUTONEG_NOT_COMPLETE         -14
3189 #define IXGBE_ERR_RESET_FAILED                 -15
3190 #define IXGBE_ERR_SWFW_SYNC                    -16
3191 #define IXGBE_ERR_PHY_ADDR_INVALID             -17
3192 #define IXGBE_ERR_I2C                          -18
3193 #define IXGBE_ERR_SFP_NOT_SUPPORTED            -19
3194 #define IXGBE_ERR_SFP_NOT_PRESENT              -20
3195 #define IXGBE_ERR_SFP_NO_INIT_SEQ_PRESENT      -21
3196 #define IXGBE_ERR_NO_SAN_ADDR_PTR              -22
3197 #define IXGBE_ERR_FDIR_REINIT_FAILED           -23
3198 #define IXGBE_ERR_EEPROM_VERSION               -24
3199 #define IXGBE_ERR_NO_SPACE                     -25
3200 #define IXGBE_ERR_OVERTEMP                     -26
3201 #define IXGBE_ERR_FC_NOT_NEGOTIATED            -27
3202 #define IXGBE_ERR_FC_NOT_SUPPORTED             -28
3203 #define IXGBE_ERR_SFP_SETUP_NOT_COMPLETE       -30
3204 #define IXGBE_ERR_PBA_SECTION                  -31
3205 #define IXGBE_ERR_INVALID_ARGUMENT             -32
3206 #define IXGBE_ERR_HOST_INTERFACE_COMMAND       -33
3207 #define IXGBE_ERR_OUT_OF_MEM                   -34
3208 #define IXGBE_ERR_FEATURE_NOT_SUPPORTED        -36

3210 #define IXGBE_NOT_IMPLEMENTED                  0x7FFFFFFF


3213 #endif /* _IXGBE_TYPE_H_ */
```

```
**********************************************************
   27878 Tue Apr 30 09:54:24 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_x540.c
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
```

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

```
  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

******************************************************************************/
/*$FreeBSD: src/sys/dev/ixgbe/ixgbe_x540.c,v 1.2 2012/07/05 20:51:44 jfv Exp $*/

#include "ixgbe_x540.h"
#include "ixgbe_type.h"
#include "ixgbe_api.h"
#include "ixgbe_common.h"
#include "ixgbe_phy.h"

static s32 ixgbe_update_flash_X540(struct ixgbe_hw *hw);
static s32 ixgbe_poll_flash_update_done_X540(struct ixgbe_hw *hw);
static s32 ixgbe_get_swfw_sync_semaphore(struct ixgbe_hw *hw);
static void ixgbe_release_swfw_sync_semaphore(struct ixgbe_hw *hw);

/**
 *  ixgbe_init_ops_X540 - Inits func ptrs and MAC type
 *  @hw: pointer to hardware structure
 *
 *  Initialize the function pointers and assign the MAC type for X540.
 *  Does not touch the hardware.
 **/
s32 ixgbe_init_ops_X540(struct ixgbe_hw *hw)
{
        struct ixgbe_mac_info *mac = &hw->mac;
        struct ixgbe_phy_info *phy = &hw->phy;
        struct ixgbe_eeprom_info *eeprom = &hw->eeprom;
        s32 ret_val;

        DEBUGFUNC("ixgbe_init_ops_X540");
```



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


        /* EEPROM */
        eeprom->ops.init_params = &ixgbe_init_eeprom_params_X540;
        eeprom->ops.read = &ixgbe_read_eerd_X540;
        eeprom->ops.read_buffer = &ixgbe_read_eerd_buffer_X540;
        eeprom->ops.write = &ixgbe_write_eewr_X540;
        eeprom->ops.write_buffer = &ixgbe_write_eewr_buffer_X540;
        eeprom->ops.update_checksum = &ixgbe_update_eeprom_checksum_X540;
        eeprom->ops.validate_checksum = &ixgbe_validate_eeprom_checksum_X540;
        eeprom->ops.calc_checksum = &ixgbe_calc_eeprom_checksum_X540;

        /* PHY */
        phy->ops.init = &ixgbe_init_phy_ops_generic;
        phy->ops.reset = NULL;

        /* MAC */
        mac->ops.reset_hw = &ixgbe_reset_hw_X540;
        mac->ops.enable_relaxed_ordering = &ixgbe_enable_relaxed_ordering_gen2;
        mac->ops.get_media_type = &ixgbe_get_media_type_X540;
        mac->ops.get_supported_physical_layer =
                                    &ixgbe_get_supported_physical_layer_X540;
        mac->ops.read_analog_reg8 = NULL;
        mac->ops.write_analog_reg8 = NULL;
        mac->ops.start_hw = &ixgbe_start_hw_X540;
        mac->ops.get_san_mac_addr = &ixgbe_get_san_mac_addr_generic;
        mac->ops.set_san_mac_addr = &ixgbe_set_san_mac_addr_generic;
        mac->ops.get_device_caps = &ixgbe_get_device_caps_generic;
        mac->ops.get_wwn_prefix = &ixgbe_get_wwn_prefix_generic;
        mac->ops.get_fcoe_boot_status = &ixgbe_get_fcoe_boot_status_generic;
        mac->ops.acquire_swfw_sync = &ixgbe_acquire_swfw_sync_X540;
        mac->ops.release_swfw_sync = &ixgbe_release_swfw_sync_X540;
        mac->ops.disable_sec_rx_path = &ixgbe_disable_sec_rx_path_generic;
        mac->ops.enable_sec_rx_path = &ixgbe_enable_sec_rx_path_generic;

        /* RAR, Multicast, VLAN */
        mac->ops.set_vmdq = &ixgbe_set_vmdq_generic;
        mac->ops.set_vmdq_san_mac = &ixgbe_set_vmdq_san_mac_generic;
        mac->ops.clear_vmdq = &ixgbe_clear_vmdq_generic;
        mac->ops.insert_mac_addr = &ixgbe_insert_mac_addr_generic;
        mac->rar_highwater = 1;
        mac->ops.set_vfta = &ixgbe_set_vfta_generic;
        mac->ops.set_vlvf = &ixgbe_set_vlvf_generic;
        mac->ops.clear_vfta = &ixgbe_clear_vfta_generic;
        mac->ops.init_uta_tables = &ixgbe_init_uta_tables_generic;
        mac->ops.set_mac_anti_spoofing = &ixgbe_set_mac_anti_spoofing;
        mac->ops.set_vlan_anti_spoofing = &ixgbe_set_vlan_anti_spoofing;

        /* Link */
        mac->ops.get_link_capabilities =
                                    &ixgbe_get_copper_link_capabilities_generic;
        mac->ops.setup_link = &ixgbe_setup_mac_link_X540;
        mac->ops.setup_rxpba = &ixgbe_set_rxpba_generic;
        mac->ops.check_link = &ixgbe_check_mac_link_generic;


        mac->mcft_size          = 128;
        mac->vft_size           = 128;
        mac->num_rar_entries    = 128;
        mac->rx_pb_size         = 384;
        mac->max_tx_queues      = 128;
        mac->max_rx_queues      = 128;
        mac->max_msix_vectors   = ixgbe_get_pcie_msix_count_generic(hw);
```

```
128         /*
129          * FWSM register
130          * ARC supported; valid only if manageability features are
131          * enabled.
132          */
133         mac->arc_subsystem_valid = (IXGBE_READ_REG(hw, IXGBE_FWSM) &
134                                     IXGBE_FWSM_MODE_MASK) ? TRUE : FALSE;

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

138         /* LEDs */
139         mac->ops.blink_led_start = ixgbe_blink_led_start_X540;
140         mac->ops.blink_led_stop = ixgbe_blink_led_stop_X540;

142         /* Manageability interface */
143         mac->ops.set_fw_drv_ver = &ixgbe_set_fw_drv_ver_generic;

145         return ret_val;
146 }
_____unchanged_portion_omitted_

175 /**
176  *  ixgbe_setup_mac_link_X540 - Sets the auto advertised capabilities
177  *  @hw: pointer to hardware structure
178  *  @speed: new link speed
178  *  @autoneg: TRUE if autonegotiation enabled
179  *  @autoneg_wait_to_complete: TRUE when waiting for completion is needed
180  **/
181 s32 ixgbe_setup_mac_link_X540(struct ixgbe_hw *hw,
182                               ixgbe_link_speed speed,
182                               ixgbe_link_speed speed, bool autoneg,
183                               bool autoneg_wait_to_complete)
184 {
185         DEBUGFUNC("ixgbe_setup_mac_link_X540");
186         return hw->phy.ops.setup_link_speed(hw, speed, autoneg_wait_to_complete)
186         return hw->phy.ops.setup_link_speed(hw, speed, autoneg,
187                                             autoneg_wait_to_complete);
187 }
_____unchanged_portion_omitted_
```

```
**********************************************************
    3158 Tue Apr 30 09:54:24 2013
new/usr/src/uts/common/io/ixgbe/ixgbe_x540.h
Import some changes from FreeBSD (details later, this is quick-n-dirty for now).
**********************************************************
/******************************************************************************

  Copyright (c) 2001-2013, Intel Corporation
  Copyright (c) 2001-2012, Intel Corporation
  All rights reserved.

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

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

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

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

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

******************************************************************************/
/*$FreeBSD$*/

#ifndef _IXGBE_X540_H_
#define _IXGBE_X540_H_

#include "ixgbe_type.h"

s32 ixgbe_get_link_capabilities_X540(struct ixgbe_hw *hw,
                                     ixgbe_link_speed *speed, bool *autoneg);
enum ixgbe_media_type ixgbe_get_media_type_X540(struct ixgbe_hw *hw);
s32 ixgbe_setup_mac_link_X540(struct ixgbe_hw *hw, ixgbe_link_speed speed,
                              bool link_up_wait_to_complete);
                              bool autoneg, bool link_up_wait_to_complete);
s32 ixgbe_reset_hw_X540(struct ixgbe_hw *hw);
s32 ixgbe_start_hw_X540(struct ixgbe_hw *hw);
u32 ixgbe_get_supported_physical_layer_X540(struct ixgbe_hw *hw);

s32 ixgbe_init_eeprom_params_X540(struct ixgbe_hw *hw);
s32 ixgbe_read_eerd_X540(struct ixgbe_hw *hw, u16 offset, u16 *data);
s32 ixgbe_read_eerd_buffer_X540(struct ixgbe_hw *hw, u16 offset, u16 words,
                                u16 *data);
s32 ixgbe_write_eewr_X540(struct ixgbe_hw *hw, u16 offset, u16 data);
s32 ixgbe_write_eewr_buffer_X540(struct ixgbe_hw *hw, u16 offset, u16 words,
                                 u16 *data);
s32 ixgbe_update_eeprom_checksum_X540(struct ixgbe_hw *hw);
s32 ixgbe_validate_eeprom_checksum_X540(struct ixgbe_hw *hw, u16 *checksum_val);
u16 ixgbe_calc_eeprom_checksum_X540(struct ixgbe_hw *hw);
```



```
s32 ixgbe_acquire_swfw_sync_X540(struct ixgbe_hw *hw, u16 mask);
void ixgbe_release_swfw_sync_X540(struct ixgbe_hw *hw, u16 mask);

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