```
**********************************************************
    3041 Wed Feb 26 09:49:36 2014
new/usr/src/pkg/manifests/driver-network-igb.mf
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2012, Nexenta Systems, Inc. All rights reserved.
# Copyright 2014 Pluribus Networks Inc.
#

#
# The default for payload-bearing actions in this package is to appear in the
# global zone only.  See the include file for greater detail, as well as
# information about overriding the defaults.
#
<include global_zone_only_component>
set name=pkg.fmri value=pkg:/driver/network/igb@$(PKGVERS)
set name=pkg.description value="Intel 82575 1Gb PCI Express NIC Driver"
set name=pkg.summary value="Intel 82575 1Gb PCI Express NIC Driver"
set name=info.classification \
    value=org.opensolaris.category.2008:Drivers/Networking
set name=variant.arch value=$(ARCH)
dir path=kernel group=sys
dir path=kernel/drv group=sys
dir path=kernel/drv/$(ARCH64) group=sys
dir path=usr/share/man
dir path=usr/share/man/man7d
driver name=igb clone_perms="igb 0666 root sys" perms="* 0666 root sys" \
    alias=pciex8086,10a7 \
    alias=pciex8086,10a9 \
    alias=pciex8086,10c9 \
    alias=pciex8086,10d6 \
    alias=pciex8086,10e6 \
    alias=pciex8086,10e7 \
    alias=pciex8086,10e8 \
    alias=pciex8086,150a \
    alias=pciex8086,150d \
    alias=pciex8086,150e \
    alias=pciex8086,150f \
    alias=pciex8086,1510 \
    alias=pciex8086,1511 \
    alias=pciex8086,1516 \
    alias=pciex8086,1518 \
```



```
    alias=pciex8086,1521 \
    alias=pciex8086,1522 \
    alias=pciex8086,1523 \
    alias=pciex8086,1524 \
    alias=pciex8086,1526 \
    alias=pciex8086,1533 \
    alias=pciex8086,1534 \
    alias=pciex8086,1535 \
    alias=pciex8086,1536 \
    alias=pciex8086,1537 \
    alias=pciex8086,1538 \
    alias=pciex8086,1539 \
    alias=pciex8086,1546 \
    alias=pciex8086,1f40 \
    alias=pciex8086,1f41 \
    alias=pciex8086,1f45 \
    alias=pciex8086,438
file path=kernel/drv/$(ARCH64)/igb group=sys
$(i386_ONLY)file path=kernel/drv/igb group=sys
file path=kernel/drv/igb.conf group=sys \
    original_name=SUNWigb:kernel/drv/igb.conf preserve=renamenew
file path=usr/share/man/man7d/igb.7d
legacy pkg=SUNWigb desc="Intel 82575 1Gb PCI Express NIC Driver" \
    name="Intel 82575 1Gb PCI Express NIC Driver"
license cr_Sun license=cr_Sun
license lic_CDDL license=lic_CDDL
```

```
**********************************************************
   94862 Wed Feb 26 09:49:36 2014
new/usr/src/uts/common/io/e1000api/e1000_82575.c
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
```

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

  Copyright (c) 2001-2013, 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.

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

/*
 * 82575EB Gigabit Network Connection
 * 82575EB Gigabit Backplane Connection
 * 82575GB Gigabit Network Connection
 * 82576 Gigabit Network Connection
 * 82576 Quad Port Gigabit Mezzanine Adapter
 * 82580 Gigabit Network Connection
 * I350 Gigabit Network Connection
 */

#include "e1000_api.h"
#include "e1000_i210.h"

static s32  e1000_init_phy_params_82575(struct e1000_hw *hw);
static s32  e1000_init_mac_params_82575(struct e1000_hw *hw);
static s32  e1000_acquire_phy_82575(struct e1000_hw *hw);
static void e1000_release_phy_82575(struct e1000_hw *hw);
static s32  e1000_acquire_nvm_82575(struct e1000_hw *hw);
static void e1000_release_nvm_82575(struct e1000_hw *hw);
static s32  e1000_check_for_link_82575(struct e1000_hw *hw);
static s32  e1000_get_cfg_done_82575(struct e1000_hw *hw);
static s32  e1000_get_link_up_info_82575(struct e1000_hw *hw, u16 *speed,
                                         u16 *duplex);
static s32  e1000_init_hw_82575(struct e1000_hw *hw);
static s32  e1000_phy_hw_reset_sgmii_82575(struct e1000_hw *hw);
static s32  e1000_read_phy_reg_sgmii_82575(struct e1000_hw *hw, u32 offset,
```



```
                                           u16 *data);
static s32  e1000_reset_hw_82575(struct e1000_hw *hw);
static s32  e1000_reset_hw_82580(struct e1000_hw *hw);
static s32  e1000_read_phy_reg_82580(struct e1000_hw *hw,
                                     u32 offset, u16 *data);
static s32  e1000_write_phy_reg_82580(struct e1000_hw *hw,
                                      u32 offset, u16 data);
static s32  e1000_set_d0_lplu_state_82580(struct e1000_hw *hw,
                                          bool active);
static s32  e1000_set_d3_lplu_state_82580(struct e1000_hw *hw,
                                          bool active);
static s32  e1000_set_d0_lplu_state_82575(struct e1000_hw *hw,
                                          bool active);
static s32  e1000_setup_copper_link_82575(struct e1000_hw *hw);
static s32  e1000_setup_serdes_link_82575(struct e1000_hw *hw);
static s32  e1000_get_media_type_82575(struct e1000_hw *hw);
static s32  e1000_set_sfp_media_type_82575(struct e1000_hw *hw);
static s32  e1000_valid_led_default_82575(struct e1000_hw *hw, u16 *data);
static s32  e1000_write_phy_reg_sgmii_82575(struct e1000_hw *hw,
                                            u32 offset, u16 data);
static void e1000_clear_hw_cntrs_82575(struct e1000_hw *hw);
static s32  e1000_acquire_swfw_sync_82575(struct e1000_hw *hw, u16 mask);
static s32  e1000_get_pcs_speed_and_duplex_82575(struct e1000_hw *hw,
                                                 u16 *speed, u16 *duplex);
static s32  e1000_get_phy_id_82575(struct e1000_hw *hw);
static void e1000_release_swfw_sync_82575(struct e1000_hw *hw, u16 mask);
static bool e1000_sgmii_active_82575(struct e1000_hw *hw);
static s32  e1000_reset_init_script_82575(struct e1000_hw *hw);
static s32  e1000_read_mac_addr_82575(struct e1000_hw *hw);
static void e1000_config_collision_dist_82575(struct e1000_hw *hw);
static void e1000_power_down_phy_copper_82575(struct e1000_hw *hw);
static void e1000_shutdown_serdes_link_82575(struct e1000_hw *hw);
static void e1000_power_up_serdes_link_82575(struct e1000_hw *hw);
static s32 e1000_set_pcie_completion_timeout(struct e1000_hw *hw);
static s32 e1000_reset_mdicnfg_82580(struct e1000_hw *hw);
static s32 e1000_validate_nvm_checksum_82580(struct e1000_hw *hw);
static s32 e1000_update_nvm_checksum_82580(struct e1000_hw *hw);
static s32 e1000_update_nvm_checksum_with_offset(struct e1000_hw *hw,
                                                 u16 offset);
static s32 e1000_validate_nvm_checksum_with_offset(struct e1000_hw *hw,
                                                   u16 offset);
static s32 e1000_validate_nvm_checksum_i350(struct e1000_hw *hw);
static s32 e1000_update_nvm_checksum_i350(struct e1000_hw *hw);
static void e1000_write_vfta_i350(struct e1000_hw *hw, u32 offset, u32 value);
static void e1000_clear_vfta_i350(struct e1000_hw *hw);

static void e1000_i2c_start(struct e1000_hw *hw);
static void e1000_i2c_stop(struct e1000_hw *hw);
static s32 e1000_clock_in_i2c_byte(struct e1000_hw *hw, u8 *data);
static s32 e1000_clock_out_i2c_byte(struct e1000_hw *hw, u8 data);
static s32 e1000_get_i2c_ack(struct e1000_hw *hw);
static s32 e1000_clock_in_i2c_bit(struct e1000_hw *hw, bool *data);
static s32 e1000_clock_out_i2c_bit(struct e1000_hw *hw, bool data);
static void e1000_raise_i2c_clk(struct e1000_hw *hw, u32 *i2cctl);
static void e1000_lower_i2c_clk(struct e1000_hw *hw, u32 *i2cctl);
static s32 e1000_set_i2c_data(struct e1000_hw *hw, u32 *i2cctl, bool data);
static bool e1000_get_i2c_data(u32 *i2cctl);

static const u16 e1000_82580_rxpbs_table[] = {
        36, 72, 144, 1, 2, 4, 8, 16, 35, 70, 140 };
#define E1000_82580_RXPBS_TABLE_SIZE \
        (sizeof(e1000_82580_rxpbs_table)/sizeof(u16))


/**
 *  e1000_sgmii_uses_mdio_82575 - Determine if I2C pins are for external MDIO
```

```
 *  @hw: pointer to the HW structure
 *
 *  Called to determine if the I2C pins are being used for I2C or as an
 *  external MDIO interface since the two options are mutually exclusive.
 **/
static bool e1000_sgmii_uses_mdio_82575(struct e1000_hw *hw)
{
        u32 reg = 0;
        bool ext_mdio = FALSE;

        DEBUGFUNC("e1000_sgmii_uses_mdio_82575");

        switch (hw->mac.type) {
        case e1000_82575:
        case e1000_82576:
                reg = E1000_READ_REG(hw, E1000_MDIC);
                ext_mdio = !!(reg & E1000_MDIC_DEST);
                break;
        case e1000_82580:
        case e1000_i350:
        case e1000_i354:
        case e1000_i210:
        case e1000_i211:
                reg = E1000_READ_REG(hw, E1000_MDICNFG);
                ext_mdio = !!(reg & E1000_MDICNFG_EXT_MDIO);
                break;
        default:
                break;
        }
        return ext_mdio;
}

/**
 *  e1000_init_phy_params_82575 - Init PHY func ptrs.
 *  @hw: pointer to the HW structure
 **/
static s32 e1000_init_phy_params_82575(struct e1000_hw *hw)
{
        struct e1000_phy_info *phy = &hw->phy;
        s32 ret_val = E1000_SUCCESS;
        u32 ctrl_ext;

        DEBUGFUNC("e1000_init_phy_params_82575");

        phy->ops.read_i2c_byte = e1000_read_i2c_byte_generic;
        phy->ops.write_i2c_byte = e1000_write_i2c_byte_generic;

        if (hw->phy.media_type != e1000_media_type_copper) {
                phy->type = e1000_phy_none;
                goto out;
        }

        phy->ops.power_up   = e1000_power_up_phy_copper;
        phy->ops.power_down = e1000_power_down_phy_copper_82575;

        phy->autoneg_mask       = AUTONEG_ADVERTISE_SPEED_DEFAULT;
        phy->reset_delay_us     = 100;

        phy->ops.acquire        = e1000_acquire_phy_82575;
        phy->ops.check_reset_block = e1000_check_reset_block_generic;
        phy->ops.commit         = e1000_phy_sw_reset_generic;
        phy->ops.get_cfg_done   = e1000_get_cfg_done_82575;
        phy->ops.release        = e1000_release_phy_82575;

        ctrl_ext = E1000_READ_REG(hw, E1000_CTRL_EXT);
```



```
        if (e1000_sgmii_active_82575(hw)) {
                phy->ops.reset = e1000_phy_hw_reset_sgmii_82575;
                ctrl_ext |= E1000_CTRL_I2C_ENA;
        } else {
                phy->ops.reset = e1000_phy_hw_reset_generic;
                ctrl_ext &= ~E1000_CTRL_I2C_ENA;
        }

        E1000_WRITE_REG(hw, E1000_CTRL_EXT, ctrl_ext);
        e1000_reset_mdicnfg_82580(hw);

        if (e1000_sgmii_active_82575(hw) && !e1000_sgmii_uses_mdio_82575(hw)) {
                phy->ops.read_reg = e1000_read_phy_reg_sgmii_82575;
                phy->ops.write_reg = e1000_write_phy_reg_sgmii_82575;
        } else {
                switch (hw->mac.type) {
                case e1000_82580:
                case e1000_i350:
                case e1000_i354:
                        phy->ops.read_reg = e1000_read_phy_reg_82580;
                        phy->ops.write_reg = e1000_write_phy_reg_82580;
                        break;
                case e1000_i210:
                case e1000_i211:
                        phy->ops.read_reg = e1000_read_phy_reg_gs40g;
                        phy->ops.write_reg = e1000_write_phy_reg_gs40g;
                        break;
                default:
                        phy->ops.read_reg = e1000_read_phy_reg_igp;
                        phy->ops.write_reg = e1000_write_phy_reg_igp;
                }
        }

        /* Set phy->phy_addr and phy->id. */
        ret_val = e1000_get_phy_id_82575(hw);

        /* Verify phy id and set remaining function pointers */
        switch (phy->id) {
        case M88E1543_E_PHY_ID:
        case M88E1512_E_PHY_ID:
        case I347AT4_E_PHY_ID:
        case M88E1112_E_PHY_ID:
        case M88E1340M_E_PHY_ID:
        case M88E1111_I_PHY_ID:
                phy->type                   = e1000_phy_m88;
                phy->ops.check_polarity = e1000_check_polarity_m88;
                phy->ops.get_info       = e1000_get_phy_info_m88;
                switch (phy->id) {
                case I347AT4_E_PHY_ID:
                case M88E1112_E_PHY_ID:
                case M88E1340M_E_PHY_ID:
                case M88E1543_E_PHY_ID:
                case M88E1512_E_PHY_ID:
                if (phy->id == I347AT4_E_PHY_ID ||
                    phy->id == M88E1112_E_PHY_ID ||
                    phy->id == M88E1340M_E_PHY_ID)
                        phy->ops.get_cable_length =
                                        e1000_get_cable_length_m88_gen2;
                default:
                else
                        phy->ops.get_cable_length = e1000_get_cable_length_m88;
                }
                phy->ops.force_speed_duplex = e1000_phy_force_speed_duplex_m88;
                break;
        case IGP03E1000_E_PHY_ID:
        case IGP04E1000_E_PHY_ID:
```

```
255                 phy->type = e1000_phy_igp_3;
256                 phy->ops.check_polarity = e1000_check_polarity_igp;
257                 phy->ops.get_info = e1000_get_phy_info_igp;
258                 phy->ops.get_cable_length = e1000_get_cable_length_igp_2;
259                 phy->ops.force_speed_duplex = e1000_phy_force_speed_duplex_igp;
260                 phy->ops.set_d0_lplu_state = e1000_set_d0_lplu_state_82575;
261                 phy->ops.set_d3_lplu_state = e1000_set_d3_lplu_state_generic;
262                 break;
263         case I82580_I_PHY_ID:
264         case I350_I_PHY_ID:
265                 phy->type = e1000_phy_82580;
266                 phy->ops.check_polarity = e1000_check_polarity_82577;
267                 phy->ops.force_speed_duplex =
268                                         e1000_phy_force_speed_duplex_82577;
269                 phy->ops.get_cable_length = e1000_get_cable_length_82577;
270                 phy->ops.get_info = e1000_get_phy_info_82577;
271                 phy->ops.set_d0_lplu_state = e1000_set_d0_lplu_state_82580;
272                 phy->ops.set_d3_lplu_state = e1000_set_d3_lplu_state_82580;
273                 break;
274         case I210_I_PHY_ID:
275                 phy->type               = e1000_phy_i210;
276                 phy->ops.check_polarity = e1000_check_polarity_m88;
277                 phy->ops.get_info       = e1000_get_phy_info_m88;
278                 phy->ops.get_cable_length = e1000_get_cable_length_m88_gen2;
279                 phy->ops.set_d0_lplu_state = e1000_set_d0_lplu_state_82580;
280                 phy->ops.set_d3_lplu_state = e1000_set_d3_lplu_state_82580;
281                 phy->ops.force_speed_duplex = e1000_phy_force_speed_duplex_m88;
282                 break;
283         default:
284                 ret_val = -E1000_ERR_PHY;
285                 goto out;
286         }

288 out:
289         return ret_val;
290 }

292 /**
293  *  e1000_init_nvm_params_82575 - Init NVM func ptrs.
294  *  @hw: pointer to the HW structure
295  **/
296 s32 e1000_init_nvm_params_82575(struct e1000_hw *hw)
297 {
298         struct e1000_nvm_info *nvm = &hw->nvm;
299         u32 eecd = E1000_READ_REG(hw, E1000_EECD);
300         u16 size;

302         DEBUGFUNC("e1000_init_nvm_params_82575");

304         size = (u16)((eecd & E1000_EECD_SIZE_EX_MASK) >>
305                      E1000_EECD_SIZE_EX_SHIFT);
306         /*
307          * Added to a constant, "size" becomes the left-shift value
308          * for setting word_size.
309          */
310         size += NVM_WORD_SIZE_BASE_SHIFT;

312         /* Just in case size is out of range, cap it to the largest
313          * EEPROM size supported
314          */
315         if (size > 15)
316                 size = 15;

318         nvm->word_size = 1 << size;
319         if (hw->mac.type < e1000_i210) {
320                 nvm->opcode_bits = 8;
```



```
321                 nvm->delay_usec = 1;

323                 switch (nvm->override) {
324                 case e1000_nvm_override_spi_large:
325                         nvm->page_size = 32;
326                         nvm->address_bits = 16;
327                         break;
328                 case e1000_nvm_override_spi_small:
329                         nvm->page_size = 8;
330                         nvm->address_bits = 8;
331                         break;
332                 default:
333                         nvm->page_size = eecd & E1000_EECD_ADDR_BITS ? 32 : 8;
334                         nvm->address_bits = eecd & E1000_EECD_ADDR_BITS ?
335                                             16 : 8;
336                         break;
337                 }
338                 if (nvm->word_size == (1 << 15))
339                         nvm->page_size = 128;

341                 nvm->type = e1000_nvm_eeprom_spi;
342         } else {
343                 nvm->type = e1000_nvm_flash_hw;
344         }

346         /* Function Pointers */
347         nvm->ops.acquire = e1000_acquire_nvm_82575;
348         nvm->ops.release = e1000_release_nvm_82575;
349         if (nvm->word_size < (1 << 15))
350                 nvm->ops.read = e1000_read_nvm_eerd;
351         else
352                 nvm->ops.read = e1000_read_nvm_spi;

354         nvm->ops.write = e1000_write_nvm_spi;
355         nvm->ops.validate = e1000_validate_nvm_checksum_generic;
356         nvm->ops.update = e1000_update_nvm_checksum_generic;
357         nvm->ops.valid_led_default = e1000_valid_led_default_82575;

359         /* override generic family function pointers for specific descendants */
360         switch (hw->mac.type) {
361         case e1000_82580:
362                 nvm->ops.validate = e1000_validate_nvm_checksum_82580;
363                 nvm->ops.update = e1000_update_nvm_checksum_82580;
364                 break;
365         case e1000_i350:
366         case e1000_i354:
367                 nvm->ops.validate = e1000_validate_nvm_checksum_i350;
368                 nvm->ops.update = e1000_update_nvm_checksum_i350;
369                 break;
370         default:
371                 break;
372         }

374         return E1000_SUCCESS;
375 }

377 /**
378  *  e1000_init_mac_params_82575 - Init MAC func ptrs.
379  *  @hw: pointer to the HW structure
380  **/
381 static s32 e1000_init_mac_params_82575(struct e1000_hw *hw)
382 {
383         struct e1000_mac_info *mac = &hw->mac;
384         struct e1000_dev_spec_82575 *dev_spec = &hw->dev_spec._82575;

386         DEBUGFUNC("e1000_init_mac_params_82575");
```

```
388         /* Derives media type */
389         e1000_get_media_type_82575(hw);
390         /* Set mta register count */
391         mac->mta_reg_count = 128;
392         /* Set uta register count */
393         mac->uta_reg_count = (hw->mac.type == e1000_82575) ? 0 : 128;
394         /* Set rar entry count */
395         mac->rar_entry_count = E1000_RAR_ENTRIES_82575;
396         if (mac->type == e1000_82576)
397                 mac->rar_entry_count = E1000_RAR_ENTRIES_82576;
398         if (mac->type == e1000_82580)
399                 mac->rar_entry_count = E1000_RAR_ENTRIES_82580;
400         if (mac->type == e1000_i350 || mac->type == e1000_i354)
401         if (mac->type == e1000_i350)
402                 mac->rar_entry_count = E1000_RAR_ENTRIES_I350;

403         /* Disable EEE default settings for EEE supported devices */
404         /* Enable EEE default settings for EEE supported devices */
405         if (mac->type >= e1000_i350)
406                 dev_spec->eee_disable = TRUE;

407         /* Allow a single clear of the SW semaphore on I210 and newer */
408         if (mac->type >= e1000_i210)
409                 dev_spec->clear_semaphore_once = TRUE;

410         /* Set if part includes ASF firmware */
411         mac->asf_firmware_present = TRUE;
412         /* FWSM register */
413         mac->has_fwsm = TRUE;
414         /* ARC supported; valid only if manageability features are enabled. */
415         mac->arc_subsystem_valid =
416                 !!(E1000_READ_REG(hw, E1000_FWSM) & E1000_FWSM_MODE_MASK);

417         /* Function pointers */

418         /* bus type/speed/width */
419         mac->ops.get_bus_info = e1000_get_bus_info_pcie_generic;
420         /* reset */
421         if (mac->type >= e1000_82580)
422                 mac->ops.reset_hw = e1000_reset_hw_82580;
423         else
424         mac->ops.reset_hw = e1000_reset_hw_82575;
425         /* hw initialization */
426         mac->ops.init_hw = e1000_init_hw_82575;
427         /* link setup */
428         mac->ops.setup_link = e1000_setup_link_generic;
429         /* physical interface link setup */
430         mac->ops.setup_physical_interface =
431                 (hw->phy.media_type == e1000_media_type_copper)
432                 ? e1000_setup_copper_link_82575 : e1000_setup_serdes_link_82575;
433         /* physical interface shutdown */
434         mac->ops.shutdown_serdes = e1000_shutdown_serdes_link_82575;
435         /* physical interface power up */
436         mac->ops.power_up_serdes = e1000_power_up_serdes_link_82575;
437         /* check for link */
438         mac->ops.check_for_link = e1000_check_for_link_82575;
439         /* read mac address */
440         mac->ops.read_mac_addr = e1000_read_mac_addr_82575;
441         /* configure collision distance */
442         mac->ops.config_collision_dist = e1000_config_collision_dist_82575;
443         /* multicast address update */
444         mac->ops.update_mc_addr_list = e1000_update_mc_addr_list_generic;
445         if (mac->type == e1000_i350 || mac->type == e1000_i354) {
446         if (mac->type == e1000_i350) {
447                 /* writing VFTA */
```



```
448                 mac->ops.write_vfta = e1000_write_vfta_i350;
449                 /* clearing VFTA */
450                 mac->ops.clear_vfta = e1000_clear_vfta_i350;
451         } else {
452                 /* writing VFTA */
453                 mac->ops.write_vfta = e1000_write_vfta_generic;
454                 /* clearing VFTA */
455                 mac->ops.clear_vfta = e1000_clear_vfta_generic;
456         }
457         if (hw->mac.type >= e1000_82580)
458                 mac->ops.validate_mdi_setting =
459                                 e1000_validate_mdi_setting_crossover_generic;
460         /* ID LED init */
461         mac->ops.id_led_init = e1000_id_led_init_generic;
462         /* blink LED */
463         mac->ops.blink_led = e1000_blink_led_generic;
464         /* setup LED */
465         mac->ops.setup_led = e1000_setup_led_generic;
466         /* cleanup LED */
467         mac->ops.cleanup_led = e1000_cleanup_led_generic;
468         /* turn on/off LED */
469         mac->ops.led_on = e1000_led_on_generic;
470         mac->ops.led_off = e1000_led_off_generic;
471         /* clear hardware counters */
472         mac->ops.clear_hw_cntrs = e1000_clear_hw_cntrs_82575;
473         /* link info */
474         mac->ops.get_link_up_info = e1000_get_link_up_info_82575;
475         /* acquire SW_FW sync */
476         mac->ops.acquire_swfw_sync = e1000_acquire_swfw_sync_82575;
477         mac->ops.release_swfw_sync = e1000_release_swfw_sync_82575;
478         if (mac->type >= e1000_i210) {
479                 mac->ops.acquire_swfw_sync = e1000_acquire_swfw_sync_i210;
480                 mac->ops.release_swfw_sync = e1000_release_swfw_sync_i210;
481         }

482         /* set lan id for port to determine which phy lock to use */
483         hw->mac.ops.set_lan_id(hw);

484         return E1000_SUCCESS;
485 }
_____unchanged_portion_omitted_

614 /**
615  *  e1000_get_phy_id_82575 - Retrieve PHY addr and id
616  *  @hw: pointer to the HW structure
617  *
618  *  Retrieves the PHY address and ID for both PHY's which do and do not use
619  *  sgmi interface.
620  **/
621 static s32 e1000_get_phy_id_82575(struct e1000_hw *hw)
622 {
623         struct e1000_phy_info *phy = &hw->phy;
624         s32  ret_val = E1000_SUCCESS;
625         u16 phy_id;
626         u32 ctrl_ext;
627         u32 mdic;

628         DEBUGFUNC("e1000_get_phy_id_82575");

629         /* some i354 devices need an extra read for phy id */
630         if (hw->mac.type == e1000_i354)
631                 e1000_get_phy_id(hw);

632         /*
633          * For SGMII PHYs, we try the list of possible addresses until
634          * we find one that works.  For non-SGMII PHYs
```

```
641          * (e.g. integrated copper PHYs), an address of 1 should
642          * work.  The result of this function should mean phy->phy_addr
643          * and phy->id are set correctly.
644          */
645         if (!e1000_sgmii_active_82575(hw)) {
646                 phy->addr = 1;
647                 ret_val = e1000_get_phy_id(hw);
648                 goto out;
649         }
650
651         if (e1000_sgmii_uses_mdio_82575(hw)) {
652                 switch (hw->mac.type) {
653                 case e1000_82575:
654                 case e1000_82576:
655                         mdic = E1000_READ_REG(hw, E1000_MDIC);
656                         mdic &= E1000_MDIC_PHY_MASK;
657                         phy->addr = mdic >> E1000_MDIC_PHY_SHIFT;
658                         break;
659                 case e1000_82580:
660                 case e1000_i350:
661                 case e1000_i354:
662                 case e1000_i210:
663                 case e1000_i211:
664                         mdic = E1000_READ_REG(hw, E1000_MDICNFG);
665                         mdic &= E1000_MDICNFG_PHY_MASK;
666                         phy->addr = mdic >> E1000_MDICNFG_PHY_SHIFT;
667                         break;
668                 default:
669                         ret_val = -E1000_ERR_PHY;
670                         goto out;
671                         break;
672                 }
673                 ret_val = e1000_get_phy_id(hw);
674                 goto out;
675         }
676
677         /* Power on sgmii phy if it is disabled */
678         ctrl_ext = E1000_READ_REG(hw, E1000_CTRL_EXT);
679         E1000_WRITE_REG(hw, E1000_CTRL_EXT,
680                         ctrl_ext & ~E1000_CTRL_EXT_SDP3_DATA);
681         E1000_WRITE_FLUSH(hw);
682         msec_delay(300);
683
684         /*
685          * The address field in the I2CCMD register is 3 bits and 0 is invalid.
686          * Therefore, we need to test 1-7
687          */
688         for (phy->addr = 1; phy->addr < 8; phy->addr++) {
689                 ret_val = e1000_read_phy_reg_sgmii_82575(hw, PHY_ID1, &phy_id);
690                 if (ret_val == E1000_SUCCESS) {
691                         DEBUGOUT2("Vendor ID 0x%08X read at address %u\n",
692                                   phy_id, phy->addr);
693                         /*
694                          * At the time of this writing, The M88 part is
695                          * the only supported SGMII PHY product.
696                          */
697                         if (phy_id == M88_VENDOR)
698                                 break;
699                 } else {
700                         DEBUGOUT1("PHY address %u was unreadable\n",
701                                   phy->addr);
702                 }
703         }
704
705         /* A valid PHY type couldn't be found. */
706         if (phy->addr == 8) {
```



```
707                 phy->addr = 0;
708                 ret_val = -E1000_ERR_PHY;
709         } else {
710                 ret_val = e1000_get_phy_id(hw);
711         }
712
713         /* restore previous sfp cage power state */
714         E1000_WRITE_REG(hw, E1000_CTRL_EXT, ctrl_ext);
715
716 out:
717         return ret_val;
718 }
______unchanged_portion_omitted_

1218 /**
1219  *  e1000_get_pcs_speed_and_duplex_82575 - Retrieve current speed/duplex
1220  *  @hw: pointer to the HW structure
1221  *  @speed: stores the current speed
1222  *  @duplex: stores the current duplex
1223  *
1224  *  Using the physical coding sub-layer (PCS), retrieve the current speed and
1225  *  duplex, then store the values in the pointers provided.
1226  **/
1227 static s32 e1000_get_pcs_speed_and_duplex_82575(struct e1000_hw *hw,
1228                                                 u16 *speed, u16 *duplex)
1229 {
1230         struct e1000_mac_info *mac = &hw->mac;
1231         u32 pcs;
1232         u32 status;
1233
1234         DEBUGFUNC("e1000_get_pcs_speed_and_duplex_82575");
1235
1236         /*
1237          * Read the PCS Status register for link state. For non-copper mode,
1238          * the status register is not accurate. The PCS status register is
1239          * used instead.
1240          */
1241         pcs = E1000_READ_REG(hw, E1000_PCS_LSTAT);
1242
1243         /*
1244          * The link up bit determines when link is up on autoneg.
1245          */
1246         if (pcs & E1000_PCS_LSTS_LINK_OK) {
1247                 mac->serdes_has_link = TRUE;
1248
1249                 /* Detect and store PCS speed */
1250                 if (pcs & E1000_PCS_LSTS_SPEED_1000)
1251                         *speed = SPEED_1000;
1252                 else if (pcs & E1000_PCS_LSTS_SPEED_100)
1253                         *speed = SPEED_100;
1254                 else
1255                         *speed = SPEED_10;
1256
1257                 /* Detect and store PCS duplex */
1258                 if (pcs & E1000_PCS_LSTS_DUPLEX_FULL)
1259                         *duplex = FULL_DUPLEX;
1260                 else
1261                         *duplex = HALF_DUPLEX;
1262
1263                 /* Check if it is an I354 2.5Gb backplane connection. */
1264                 if (mac->type == e1000_i354) {
1265                         status = E1000_READ_REG(hw, E1000_STATUS);
1266                         if ((status & E1000_STATUS_2P5_SKU) &&
1267                             !(status & E1000_STATUS_2P5_SKU_OVER)) {
1268                                 *speed = SPEED_2500;
1269                                 *duplex = FULL_DUPLEX;
```

```
1270                                DEBUGOUT("2500 Mbs, ");
1271                                DEBUGOUT("Full Duplex\n");
1272                        }
1273                }

1275        } else {
1276                mac->serdes_has_link = FALSE;
1277                *speed = 0;
1278                *duplex = 0;
1279        }

1281        return E1000_SUCCESS;
1282 }
```

*unchanged_portion_omitted_*

```
1439 /**
1440  *  e1000_setup_copper_link_82575 - Configure copper link settings
1441  *  @hw: pointer to the HW structure
1442  *
1443  *  Configures the link for auto-neg or forced speed and duplex.  Then we check
1444  *  for link, once link is established calls to configure collision distance
1445  *  and flow control are called.
1446  **/
1447 static s32 e1000_setup_copper_link_82575(struct e1000_hw *hw)
1448 {
1449         u32 ctrl;
1450         s32 ret_val;
1451         u32 phpm_reg;

1453         DEBUGFUNC("e1000_setup_copper_link_82575");

1455         ctrl = E1000_READ_REG(hw, E1000_CTRL);
1456         ctrl |= E1000_CTRL_SLU;
1457         ctrl &= ~(E1000_CTRL_FRCSPD | E1000_CTRL_FRCDPX);
1458         E1000_WRITE_REG(hw, E1000_CTRL, ctrl);

1460         /* Clear Go Link Disconnect bit on supported devices */
1461         switch (hw->mac.type) {
1462         case e1000_82580:
1463         case e1000_i350:
1464         case e1000_i210:
1465         case e1000_i211:
1433         /* Clear Go Link Disconnect bit */
1434         if (hw->mac.type >= e1000_82580) {
1466                 phpm_reg = E1000_READ_REG(hw, E1000_82580_PHY_POWER_MGMT);
1467                 phpm_reg &= ~E1000_82580_PM_GO_LINKD;
1468                 E1000_WRITE_REG(hw, E1000_82580_PHY_POWER_MGMT, phpm_reg);
1469                 break;
1470         default:
1471                 break;
1472         }

1474         ret_val = e1000_setup_serdes_link_82575(hw);
1475         if (ret_val)
1476                 goto out;

1478         if (e1000_sgmii_active_82575(hw)) {
1479                 /* allow time for SFP cage time to power up phy */
1480                 msec_delay(300);

1482                 ret_val = hw->phy.ops.reset(hw);
1483                 if (ret_val) {
1484                         DEBUGOUT("Error resetting the PHY.\n");
1485                         goto out;
1486                 }
1487         }
```



```
1488         switch (hw->phy.type) {
1489         case e1000_phy_i210:
1490         case e1000_phy_m88:
1491                 switch (hw->phy.id) {
1492                 case I347AT4_E_PHY_ID:
1493                 case M88E1112_E_PHY_ID:
1494                 case M88E1340M_E_PHY_ID:
1495                 case I210_I_PHY_ID:
1496                         ret_val = e1000_copper_link_setup_m88_gen2(hw);
1497                         break;
1498                 default:
1499                         ret_val = e1000_copper_link_setup_m88(hw);
1500                         break;
1501                 }
1502                 break;
1503         case e1000_phy_igp_3:
1504                 ret_val = e1000_copper_link_setup_igp(hw);
1505                 break;
1506         case e1000_phy_82580:
1507                 ret_val = e1000_copper_link_setup_82577(hw);
1508                 break;
1509         default:
1510                 ret_val = -E1000_ERR_PHY;
1511                 break;
1512         }

1514         if (ret_val)
1515                 goto out;

1517         ret_val = e1000_setup_copper_link_generic(hw);
1518 out:
1519         return ret_val;
1520 }
```

*unchanged_portion_omitted_*

```
2159 /**
2160  *  e1000_vmdq_set_anti_spoofing_pf - enable or disable anti-spoofing
2161  *  @hw: pointer to the hardware struct
2162  *  @enable: state to enter, either enabled or disabled
2163  *  @pf: Physical Function pool - do not set anti-spoofing for the PF
2164  *
2165  *  enables/disables L2 switch anti-spoofing functionality.
2166  **/
2167 void e1000_vmdq_set_anti_spoofing_pf(struct e1000_hw *hw, bool enable, int pf)
2168 {
2169         u32 reg_val, reg_offset;
2135         u32 dtxswc;

2171         switch (hw->mac.type) {
2172         case e1000_82576:
2173                 reg_offset = E1000_DTXSWC;
2139                 dtxswc = E1000_READ_REG(hw, E1000_DTXSWC);
2140                 if (enable) {
2141                         dtxswc |= (E1000_DTXSWC_MAC_SPOOF_MASK |
2142                                    E1000_DTXSWC_VLAN_SPOOF_MASK);
2143                         /* The PF can spoof - it has to in order to
2144                          * support emulation mode NICs */
2145                         dtxswc ^= (1 << pf | 1 << (pf +
2146                                    E1000_DTXSWC_VLAN_SPOOF_SHIFT));
2147                 } else {
2148                         dtxswc &= ~(E1000_DTXSWC_MAC_SPOOF_MASK |
2149                                     E1000_DTXSWC_VLAN_SPOOF_MASK);
2150                 }
2151                 E1000_WRITE_REG(hw, E1000_DTXSWC, dtxswc);
2174                 break;
2175         case e1000_i350:
```

```
2176         case e1000_i354:
2177                 reg_offset = E1000_TXSWC;
2178                 break;
2179         default:
2180                 return;
2181         }
2182         reg_val = E1000_READ_REG(hw, reg_offset);
2154                 dtxswc = E1000_READ_REG(hw, E1000_TXSWC);
2183         if (enable) {
2184                 reg_val |= (E1000_DTXSWC_MAC_SPOOF_MASK |
2156                         dtxswc |= (E1000_DTXSWC_MAC_SPOOF_MASK |
2185                             E1000_DTXSWC_VLAN_SPOOF_MASK);
2186                 /* The PF can spoof - it has to in order to
2187                  * support emulation mode NICs
2188                  */
2189                 reg_val ^= (1 << pf | 1 << (pf + MAX_NUM_VFS));
2161                         dtxswc ^= (1 << pf | 1 << (pf +
2162                                    E1000_DTXSWC_VLAN_SPOOF_SHIFT));
2190         } else {
2191                 reg_val &= ~(E1000_DTXSWC_MAC_SPOOF_MASK |
2164                         dtxswc &= ~(E1000_DTXSWC_MAC_SPOOF_MASK |
2192                             E1000_DTXSWC_VLAN_SPOOF_MASK);
2193         }
2194         E1000_WRITE_REG(hw, reg_offset, reg_val);
2167                 E1000_WRITE_REG(hw, E1000_TXSWC, dtxswc);
2168         default:
2169                 break;
2170         }
2195 }

2197 /**
2198  *  e1000_vmdq_set_loopback_pf - enable or disable vmdq loopback
2199  *  @hw: pointer to the hardware struct
2200  *  @enable: state to enter, either enabled or disabled
2201  *
2202  *  enables/disables L2 switch loopback functionality.
2203  **/
2204 void e1000_vmdq_set_loopback_pf(struct e1000_hw *hw, bool enable)
2205 {
2206         u32 dtxswc;

2208         switch (hw->mac.type) {
2209         case e1000_82576:
2210                 dtxswc = E1000_READ_REG(hw, E1000_DTXSWC);
2211                 if (enable)
2212                         dtxswc |= E1000_DTXSWC_VMDQ_LOOPBACK_EN;
2213                 else
2214                         dtxswc &= ~E1000_DTXSWC_VMDQ_LOOPBACK_EN;
2215                 E1000_WRITE_REG(hw, E1000_DTXSWC, dtxswc);
2216                 break;
2217         case e1000_i350:
2218         case e1000_i354:
2219                 dtxswc = E1000_READ_REG(hw, E1000_TXSWC);
2220                 if (enable)
2221                         dtxswc |= E1000_DTXSWC_VMDQ_LOOPBACK_EN;
2222                 else
2223                         dtxswc &= ~E1000_DTXSWC_VMDQ_LOOPBACK_EN;
2224                 E1000_WRITE_REG(hw, E1000_TXSWC, dtxswc);
2225                 break;
2226         default:
2227                 /* Currently no other hardware supports loopback */
2228                 break;
2229         }


2232 }
```

*unchanged_portion_omitted_*


```
2717 /**
2718  *  e1000_set_eee_i354 - Enable/disable EEE support
2719  *  @hw: pointer to the HW structure
2720  *
2721  *  Enable/disable EEE legacy mode based on setting in dev_spec structure.
2722  *
2723  **/
2724 s32 e1000_set_eee_i354(struct e1000_hw *hw)
2725 {
2726         struct e1000_phy_info *phy = &hw->phy;
2727         s32 ret_val = E1000_SUCCESS;
2728         u16 phy_data;

2730         DEBUGFUNC("e1000_set_eee_i354");

2732         if ((hw->phy.media_type != e1000_media_type_copper) ||
2733             ((phy->id != M88E1543_E_PHY_ID) &&
2734             (phy->id != M88E1512_E_PHY_ID)))
2735                 goto out;

2737         if (!hw->dev_spec._82575.eee_disable) {
2738                 /* Switch to PHY page 18. */
2739                 ret_val = phy->ops.write_reg(hw, E1000_M88E1543_PAGE_ADDR, 18);
2740                 if (ret_val)
2741                         goto out;

2743                 ret_val = phy->ops.read_reg(hw, E1000_M88E1543_EEE_CTRL_1,
2744                                             &phy_data);
2745                 if (ret_val)
2746                         goto out;

2748                 phy_data |= E1000_M88E1543_EEE_CTRL_1_MS;
2749                 ret_val = phy->ops.write_reg(hw, E1000_M88E1543_EEE_CTRL_1,
2750                                              phy_data);
2751                 if (ret_val)
2752                         goto out;

2754                 /* Return the PHY to page 0. */
2755                 ret_val = phy->ops.write_reg(hw, E1000_M88E1543_PAGE_ADDR, 0);
2756                 if (ret_val)
2757                         goto out;

2759                 /* Turn on EEE advertisement. */
2760                 ret_val = e1000_read_xmdio_reg(hw, E1000_EEE_ADV_ADDR_I354,
2761                                                E1000_EEE_ADV_DEV_I354,
2762                                                &phy_data);
2763                 if (ret_val)
2764                         goto out;

2766                 phy_data |= E1000_EEE_ADV_100_SUPPORTED |
2767                             E1000_EEE_ADV_1000_SUPPORTED;
2768                 ret_val = e1000_write_xmdio_reg(hw, E1000_EEE_ADV_ADDR_I354,
2769                                                 E1000_EEE_ADV_DEV_I354,
2770                                                 phy_data);
2771         } else {
2772                 /* Turn off EEE advertisement. */
2773                 ret_val = e1000_read_xmdio_reg(hw, E1000_EEE_ADV_ADDR_I354,
2774                                                E1000_EEE_ADV_DEV_I354,
2775                                                &phy_data);
2776                 if (ret_val)
2777                         goto out;

2779                 phy_data &= ~(E1000_EEE_ADV_100_SUPPORTED |
2780                               E1000_EEE_ADV_1000_SUPPORTED);
2781                 ret_val = e1000_write_xmdio_reg(hw, E1000_EEE_ADV_ADDR_I354,
```

```
2782                                            E1000_EEE_ADV_DEV_I354,
2783                                            phy_data);
2784         }

2786 out:
2787         return ret_val;
2788 }

2790 /**
2791  *  e1000_get_eee_status_i354 - Get EEE status
2792  *  @hw: pointer to the HW structure
2793  *  @status: EEE status
2794  *
2795  *  Get EEE status by guessing based on whether Tx or Rx LPI indications have
2796  *  been received.
2797  **/
2798 s32 e1000_get_eee_status_i354(struct e1000_hw *hw, bool *status)
2799 {
2800         struct e1000_phy_info *phy = &hw->phy;
2801         s32 ret_val = E1000_SUCCESS;
2802         u16 phy_data;

2804         DEBUGFUNC("e1000_get_eee_status_i354");

2806         /* Check if EEE is supported on this device. */
2807         if ((hw->phy.media_type != e1000_media_type_copper) ||
2808             ((phy->id != M88E1543_E_PHY_ID) &&
2809              (phy->id != M88E1512_E_PHY_ID)))
2810                 goto out;

2812         ret_val = e1000_read_xmdio_reg(hw, E1000_PCS_STATUS_ADDR_I354,
2813                                        E1000_PCS_STATUS_DEV_I354,
2814                                        &phy_data);
2815         if (ret_val)
2816                 goto out;

2818         *status = phy_data & (E1000_PCS_STATUS_TX_LPI_RCVD |
2819                               E1000_PCS_STATUS_RX_LPI_RCVD) ? TRUE : FALSE;

2821 out:
2822         return ret_val;
2823 }

2825 /* Due to a hw errata, if the host tries to  configure the VFTA register
2826  * while performing queries from the BMC or DMA, then the VFTA in some
2827  * cases won't be written.
2828  */

2830 /**
2831  *  e1000_clear_vfta_i350 - Clear VLAN filter table
2832  *  @hw: pointer to the HW structure
2833  *
2834  *  Clears the register array which contains the VLAN filter table by
2835  *  setting all the values to 0.
2836  **/
2837 void e1000_clear_vfta_i350(struct e1000_hw *hw)
2838 {
2839         u32 offset;
2840         int i;

2842         DEBUGFUNC("e1000_clear_vfta_350");

2844         for (offset = 0; offset < E1000_VLAN_FILTER_TBL_SIZE; offset++) {
2845                 for (i = 0; i < 10; i++)
2846                         E1000_WRITE_REG_ARRAY(hw, E1000_VFTA, offset, 0);
```



```
2848                 E1000_WRITE_FLUSH(hw);
2849         }
2850 }
```

*unchanged_portion_omitted_*

****
20929 Wed Feb 26 09:49:36 2014
new/usr/src/uts/common/io/e1000api/e1000_82575.h
4431 igb support for I354
4616 igb has uninitialized kstats
****
_____***unchanged_portion_omitted_***

```
 494 void e1000_vfta_set_vf(struct e1000_hw *, u16, bool);
 495 void e1000_rlpml_set_vf(struct e1000_hw *, u16);
 496 s32 e1000_promisc_set_vf(struct e1000_hw *, enum e1000_promisc_type type);
 497 u16 e1000_rxpbs_adjust_82580(u32 data);
 498 s32 e1000_set_eee_i350(struct e1000_hw *);
 499 s32 e1000_set_eee_i354(struct e1000_hw *);
 500 s32 e1000_get_eee_status_i354(struct e1000_hw *, bool *);

 502 /* I2C SDA and SCL timing parameters for standard mode */
 503 #define E1000_I2C_T_HD_STA      4
 504 #define E1000_I2C_T_LOW         5
 505 #define E1000_I2C_T_HIGH        4
 506 #define E1000_I2C_T_SU_STA      5
 507 #define E1000_I2C_T_HD_DATA     5
 508 #define E1000_I2C_T_SU_DATA     1
 509 #define E1000_I2C_T_RISE        1
 510 #define E1000_I2C_T_FALL        1
 511 #define E1000_I2C_T_SU_STO      4
 512 #define E1000_I2C_T_BUF         5

 514 s32 e1000_set_i2c_bb(struct e1000_hw *hw);
 515 s32 e1000_read_i2c_byte_generic(struct e1000_hw *hw, u8 byte_offset,
 516                                 u8 dev_addr, u8 *data);
 517 s32 e1000_write_i2c_byte_generic(struct e1000_hw *hw, u8 byte_offset,
 518                                  u8 dev_addr, u8 data);
 519 void e1000_i2c_bus_clear(struct e1000_hw *hw);

 521 #ifdef __cplusplus
 522 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   37802 Wed Feb 26 09:49:36 2014
new/usr/src/uts/common/io/e1000api/e1000_api.c
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
______unchanged_portion_omitted_

 141 /**
 142  *  e1000_set_mac_type - Sets MAC type
 143  *  @hw: pointer to the HW structure
 144  *
 145  *  This function sets the mac type of the adapter based on the
 146  *  device ID stored in the hw structure.
 147  *  MUST BE FIRST FUNCTION CALLED (explicitly or through
 148  *  e1000_setup_init_funcs()).
 149  **/
 150 s32 e1000_set_mac_type(struct e1000_hw *hw)
 151 {
 152         struct e1000_mac_info *mac = &hw->mac;
 153         s32 ret_val = E1000_SUCCESS;

 155         DEBUGFUNC("e1000_set_mac_type");

 157         switch (hw->device_id) {
 158         case E1000_DEV_ID_82542:
 159                 mac->type = e1000_82542;
 160                 break;
 161         case E1000_DEV_ID_82543GC_FIBER:
 162         case E1000_DEV_ID_82543GC_COPPER:
 163                 mac->type = e1000_82543;
 164                 break;
 165         case E1000_DEV_ID_82544EI_COPPER:
 166         case E1000_DEV_ID_82544EI_FIBER:
 167         case E1000_DEV_ID_82544GC_COPPER:
 168         case E1000_DEV_ID_82544GC_LOM:
 169                 mac->type = e1000_82544;
 170                 break;
 171         case E1000_DEV_ID_82540EM:
 172         case E1000_DEV_ID_82540EM_LOM:
 173         case E1000_DEV_ID_82540EP:
 174         case E1000_DEV_ID_82540EP_LOM:
 175         case E1000_DEV_ID_82540EP_LP:
 176                 mac->type = e1000_82540;
 177                 break;
 178         case E1000_DEV_ID_82545EM_COPPER:
 179         case E1000_DEV_ID_82545EM_FIBER:
 180                 mac->type = e1000_82545;
 181                 break;
 182         case E1000_DEV_ID_82545GM_COPPER:
 183         case E1000_DEV_ID_82545GM_FIBER:
 184         case E1000_DEV_ID_82545GM_SERDES:
 185                 mac->type = e1000_82545_rev_3;
 186                 break;
 187         case E1000_DEV_ID_82546EB_COPPER:
 188         case E1000_DEV_ID_82546EB_FIBER:
 189         case E1000_DEV_ID_82546EB_QUAD_COPPER:
 190                 mac->type = e1000_82546;
 191                 break;
 192         case E1000_DEV_ID_82546GB_COPPER:
 193         case E1000_DEV_ID_82546GB_FIBER:
 194         case E1000_DEV_ID_82546GB_SERDES:
 195         case E1000_DEV_ID_82546GB_PCIE:
 196         case E1000_DEV_ID_82546GB_QUAD_COPPER:
 197         case E1000_DEV_ID_82546GB_QUAD_COPPER_KSP3:
 198                 mac->type = e1000_82546_rev_3;
```



```
 199                 break;
 200         case E1000_DEV_ID_82541EI:
 201         case E1000_DEV_ID_82541EI_MOBILE:
 202         case E1000_DEV_ID_82541ER_LOM:
 203                 mac->type = e1000_82541;
 204                 break;
 205         case E1000_DEV_ID_82541ER:
 206         case E1000_DEV_ID_82541GI:
 207         case E1000_DEV_ID_82541GI_LF:
 208         case E1000_DEV_ID_82541GI_MOBILE:
 209                 mac->type = e1000_82541_rev_2;
 210                 break;
 211         case E1000_DEV_ID_82547EI:
 212         case E1000_DEV_ID_82547EI_MOBILE:
 213                 mac->type = e1000_82547;
 214                 break;
 215         case E1000_DEV_ID_82547GI:
 216                 mac->type = e1000_82547_rev_2;
 217                 break;
 218         case E1000_DEV_ID_82571EB_COPPER:
 219         case E1000_DEV_ID_82571EB_FIBER:
 220         case E1000_DEV_ID_82571EB_SERDES:
 221         case E1000_DEV_ID_82571EB_SERDES_DUAL:
 222         case E1000_DEV_ID_82571EB_SERDES_QUAD:
 223         case E1000_DEV_ID_82571EB_QUAD_COPPER:
 224         case E1000_DEV_ID_82571PT_QUAD_COPPER:
 225         case E1000_DEV_ID_82571EB_QUAD_FIBER:
 226         case E1000_DEV_ID_82571EB_QUAD_COPPER_LP:
 227                 mac->type = e1000_82571;
 228                 break;
 229         case E1000_DEV_ID_82572EI:
 230         case E1000_DEV_ID_82572EI_COPPER:
 231         case E1000_DEV_ID_82572EI_FIBER:
 232         case E1000_DEV_ID_82572EI_SERDES:
 233                 mac->type = e1000_82572;
 234                 break;
 235         case E1000_DEV_ID_82573E:
 236         case E1000_DEV_ID_82573E_IAMT:
 237         case E1000_DEV_ID_82573L:
 238                 mac->type = e1000_82573;
 239                 break;
 240         case E1000_DEV_ID_82574L:
 241         case E1000_DEV_ID_82574LA:
 242                 mac->type = e1000_82574;
 243                 break;
 244         case E1000_DEV_ID_82583V:
 245                 mac->type = e1000_82583;
 246                 break;
 247         case E1000_DEV_ID_80003ES2LAN_COPPER_DPT:
 248         case E1000_DEV_ID_80003ES2LAN_SERDES_DPT:
 249         case E1000_DEV_ID_80003ES2LAN_COPPER_SPT:
 250         case E1000_DEV_ID_80003ES2LAN_SERDES_SPT:
 251                 mac->type = e1000_80003es2lan;
 252                 break;
 253         case E1000_DEV_ID_ICH8_IFE:
 254         case E1000_DEV_ID_ICH8_IFE_GT:
 255         case E1000_DEV_ID_ICH8_IFE_G:
 256         case E1000_DEV_ID_ICH8_IGP_M:
 257         case E1000_DEV_ID_ICH8_IGP_M_AMT:
 258         case E1000_DEV_ID_ICH8_IGP_AMT:
 259         case E1000_DEV_ID_ICH8_IGP_C:
 260         case E1000_DEV_ID_ICH8_82567V_3:
 261                 mac->type = e1000_ich8lan;
 262                 break;
 263         case E1000_DEV_ID_ICH9_IFE:
 264         case E1000_DEV_ID_ICH9_IFE_GT:
```

```
265         case E1000_DEV_ID_ICH9_IFE_G:
266         case E1000_DEV_ID_ICH9_IGP_M:
267         case E1000_DEV_ID_ICH9_IGP_M_AMT:
268         case E1000_DEV_ID_ICH9_IGP_M_V:
269         case E1000_DEV_ID_ICH9_IGP_AMT:
270         case E1000_DEV_ID_ICH9_BM:
271         case E1000_DEV_ID_ICH9_IGP_C:
272         case E1000_DEV_ID_ICH10_R_BM_LM:
273         case E1000_DEV_ID_ICH10_R_BM_LF:
274         case E1000_DEV_ID_ICH10_R_BM_V:
275                 mac->type = e1000_ich9lan;
276                 break;
277         case E1000_DEV_ID_ICH10_D_BM_LM:
278         case E1000_DEV_ID_ICH10_D_BM_LF:
279         case E1000_DEV_ID_ICH10_D_BM_V:
280                 mac->type = e1000_ich10lan;
281                 break;
282         case E1000_DEV_ID_PCH_D_HV_DM:
283         case E1000_DEV_ID_PCH_D_HV_DC:
284         case E1000_DEV_ID_PCH_M_HV_LM:
285         case E1000_DEV_ID_PCH_M_HV_LC:
286                 mac->type = e1000_pchlan;
287                 break;
288         case E1000_DEV_ID_PCH2_LV_LM:
289         case E1000_DEV_ID_PCH2_LV_V:
290                 mac->type = e1000_pch2lan;
291                 break;
292         case E1000_DEV_ID_PCH_LPT_I217_LM:
293         case E1000_DEV_ID_PCH_LPT_I217_V:
294         case E1000_DEV_ID_PCH_LPTLP_I218_LM:
295         case E1000_DEV_ID_PCH_LPTLP_I218_V:
296                 mac->type = e1000_pch_lpt;
297                 break;
298         case E1000_DEV_ID_82575EB_COPPER:
299         case E1000_DEV_ID_82575EB_FIBER_SERDES:
300         case E1000_DEV_ID_82575GB_QUAD_COPPER:
301                 mac->type = e1000_82575;
302                 break;
303         case E1000_DEV_ID_82576:
304         case E1000_DEV_ID_82576_FIBER:
305         case E1000_DEV_ID_82576_SERDES:
306         case E1000_DEV_ID_82576_QUAD_COPPER:
307         case E1000_DEV_ID_82576_QUAD_COPPER_ET2:
308         case E1000_DEV_ID_82576_NS:
309         case E1000_DEV_ID_82576_NS_SERDES:
310         case E1000_DEV_ID_82576_SERDES_QUAD:
311                 mac->type = e1000_82576;
312                 break;
313         case E1000_DEV_ID_82580_COPPER:
314         case E1000_DEV_ID_82580_FIBER:
315         case E1000_DEV_ID_82580_SERDES:
316         case E1000_DEV_ID_82580_SGMII:
317         case E1000_DEV_ID_82580_COPPER_DUAL:
318         case E1000_DEV_ID_82580_QUAD_FIBER:
319         case E1000_DEV_ID_DH89XXCC_SGMII:
320         case E1000_DEV_ID_DH89XXCC_SERDES:
321         case E1000_DEV_ID_DH89XXCC_BACKPLANE:
322         case E1000_DEV_ID_DH89XXCC_SFP:
323                 mac->type = e1000_82580;
324                 break;
325         case E1000_DEV_ID_I350_COPPER:
326         case E1000_DEV_ID_I350_FIBER:
327         case E1000_DEV_ID_I350_SERDES:
328         case E1000_DEV_ID_I350_SGMII:
329         case E1000_DEV_ID_I350_DA4:
330                 mac->type = e1000_i350;
```



```
331                 break;
332 #if defined(QV_RELEASE) && defined(SPRINGVILLE_FLASHLESS_HW)
333         case E1000_DEV_ID_I210_NVMLESS:
334 #endif /* QV_RELEASE && SPRINGVILLE_FLASHLESS_HW */
335         case E1000_DEV_ID_I210_COPPER:
336         case E1000_DEV_ID_I210_COPPER_OEM1:
337         case E1000_DEV_ID_I210_COPPER_IT:
338         case E1000_DEV_ID_I210_FIBER:
339         case E1000_DEV_ID_I210_SERDES:
340         case E1000_DEV_ID_I210_SGMII:
341                 mac->type = e1000_i210;
342                 break;
343         case E1000_DEV_ID_I211_COPPER:
344                 mac->type = e1000_i211;
345                 break;
346         case E1000_DEV_ID_82576_VF:
347         case E1000_DEV_ID_82576_VF_HV:
348                 mac->type = e1000_vfadapt;
349                 break;
350         case E1000_DEV_ID_I350_VF:
351         case E1000_DEV_ID_I350_VF_HV:
352                 mac->type = e1000_vfadapt_i350;
353                 break;
354         case E1000_DEV_ID_I354_BACKPLANE_1GBPS:
355         case E1000_DEV_ID_I354_SGMII:
356         case E1000_DEV_ID_I354_BACKPLANE_2_5GBPS:
357                 mac->type = e1000_i354;
358                 break;

360         default:
361                 /* Should never have loaded on this device */
362                 ret_val = -E1000_ERR_MAC_INIT;
363                 break;
364         }

366         return ret_val;
367 }

369 /**
370  *  e1000_setup_init_funcs - Initializes function pointers
371  *  @hw: pointer to the HW structure
372  *  @init_device: TRUE will initialize the rest of the function pointers
373  *                getting the device ready for use.  FALSE will only set
374  *                MAC type and the function pointers for the other init
375  *                functions.  Passing FALSE will not generate any hardware
376  *                reads or writes.
377  *
378  *  This function must be called by a driver in order to use the rest
379  *  of the 'shared' code files. Called by drivers only.
380  **/
381 s32 e1000_setup_init_funcs(struct e1000_hw *hw, bool init_device)
382 {
383         s32 ret_val;

385         /* Can't do much good without knowing the MAC type. */
386         ret_val = e1000_set_mac_type(hw);
387         if (ret_val) {
388                 DEBUGOUT("ERROR: MAC type could not be set properly.\n");
389                 goto out;
390         }

392         if (!hw->hw_addr) {
393                 DEBUGOUT("ERROR: Registers not mapped\n");
394                 ret_val = -E1000_ERR_CONFIG;
395                 goto out;
396         }
```

```
398         /*
399          * Init function pointers to generic implementations. We do this first
400          * allowing a driver module to override it afterward.
401          */
402         e1000_init_mac_ops_generic(hw);
403         e1000_init_phy_ops_generic(hw);
404         e1000_init_nvm_ops_generic(hw);
405         e1000_init_mbx_ops_generic(hw);

407         /*
408          * Set up the init function pointers. These are functions within the
409          * adapter family file that sets up function pointers for the rest of
410          * the functions in that family.
411          */
412         switch (hw->mac.type) {
413         case e1000_82542:
414                 e1000_init_function_pointers_82542(hw);
415                 break;
416         case e1000_82543:
417         case e1000_82544:
418                 e1000_init_function_pointers_82543(hw);
419                 break;
420         case e1000_82540:
421         case e1000_82545:
422         case e1000_82545_rev_3:
423         case e1000_82546:
424         case e1000_82546_rev_3:
425                 e1000_init_function_pointers_82540(hw);
426                 break;
427         case e1000_82541:
428         case e1000_82541_rev_2:
429         case e1000_82547:
430         case e1000_82547_rev_2:
431                 e1000_init_function_pointers_82541(hw);
432                 break;
433         case e1000_82571:
434         case e1000_82572:
435         case e1000_82573:
436         case e1000_82574:
437         case e1000_82583:
438                 e1000_init_function_pointers_82571(hw);
439                 break;
440         case e1000_80003es2lan:
441                 e1000_init_function_pointers_80003es2lan(hw);
442                 break;
443         case e1000_ich8lan:
444         case e1000_ich9lan:
445         case e1000_ich10lan:
446         case e1000_pchlan:
447         case e1000_pch2lan:
448         case e1000_pch_lpt:
449                 e1000_init_function_pointers_ich8lan(hw);
450                 break;
451         case e1000_82575:
452         case e1000_82576:
453         case e1000_82580:
454         case e1000_i350:
455         case e1000_i354:
456                 e1000_init_function_pointers_82575(hw);
457                 break;
458         case e1000_i210:
459         case e1000_i211:
460                 e1000_init_function_pointers_i210(hw);
461                 break;
462         case e1000_vfadapt:
```



```
463                 e1000_init_function_pointers_vf(hw);
464                 break;
465         case e1000_vfadapt_i350:
466                 e1000_init_function_pointers_vf(hw);
467                 break;
468         default:
469                 DEBUGOUT("Hardware not supported\n");
470                 ret_val = -E1000_ERR_CONFIG;
471                 break;
472         }

474         /*
475          * Initialize the rest of the function pointers. These require some
476          * register reads/writes in some cases.
477          */
478         if (!(ret_val) && init_device) {
479                 ret_val = e1000_init_mac_params(hw);
480                 if (ret_val)
481                         goto out;

483                 ret_val = e1000_init_nvm_params(hw);
484                 if (ret_val)
485                         goto out;

487                 ret_val = e1000_init_phy_params(hw);
488                 if (ret_val)
489                         goto out;

491                 ret_val = e1000_init_mbx_params(hw);
492                 if (ret_val)
493                         goto out;
494         }

496 out:
497         return ret_val;
498 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   64295 Wed Feb 26 09:49:37 2014
new/usr/src/uts/common/io/e1000api/e1000_defines.h
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
```

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

  Copyright (c) 2001-2013, 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 _E1000_DEFINES_H_
#define _E1000_DEFINES_H_

/* Number of Transmit and Receive Descriptors must be a multiple of 8 */
#define REQ_TX_DESCRIPTOR_MULTIPLE  8
#define REQ_RX_DESCRIPTOR_MULTIPLE  8

/* Definitions for power management and wakeup registers */
/* Wake Up Control */
#define E1000_WUC_APME          0x00000001 /* APM Enable */
#define E1000_WUC_PME_EN        0x00000002 /* PME Enable */
#define E1000_WUC_PHY_WAKE      0x00000100 /* if PHY supports wakeup */

/* Wake Up Filter Control */
#define E1000_WUFC_LNKC 0x00000001 /* Link Status Change Wakeup Enable */
#define E1000_WUFC_MAG  0x00000002 /* Magic Packet Wakeup Enable */
#define E1000_WUFC_EX   0x00000004 /* Directed Exact Wakeup Enable */
#define E1000_WUFC_MC   0x00000008 /* Directed Multicast Wakeup Enable */
#define E1000_WUFC_BC   0x00000010 /* Broadcast Wakeup Enable */
#define E1000_WUFC_ARP  0x00000020 /* ARP Request Packet Wakeup Enable */
#define E1000_WUFC_IPV4 0x00000040 /* Directed IPv4 Packet Wakeup Enable */
#define E1000_WUFC_FLX0         0x00010000 /* Flexible Filter 0 Enable */

/* Wake Up Status */
#define E1000_WUS_LNKC          E1000_WUFC_LNKC
#define E1000_WUS_MAG           E1000_WUFC_MAG
```



```
#define E1000_WUS_EX            E1000_WUFC_EX
#define E1000_WUS_MC            E1000_WUFC_MC
#define E1000_WUS_BC            E1000_WUFC_BC

/* Extended Device Control */
#define E1000_CTRL_EXT_LPCD             0x00000004 /* LCD Power Cycle Done */
#define E1000_CTRL_EXT_SDP4_DATA        0x00000010 /* SW Definable Pin 4 data */
#define E1000_CTRL_EXT_SDP6_DATA        0x00000040 /* SW Definable Pin 6 data */
#define E1000_CTRL_EXT_SDP3_DATA        0x00000080 /* SW Definable Pin 3 data */
/* SDP 4/5 (bits 8,9) are reserved in >= 82575 */
#define E1000_CTRL_EXT_SDP4_DIR 0x00000100 /* Direction of SDP4 0=in 1=out */
#define E1000_CTRL_EXT_SDP6_DIR 0x00000400 /* Direction of SDP6 0=in 1=out */
#define E1000_CTRL_EXT_SDP3_DIR 0x00000800 /* Direction of SDP3 0=in 1=out */
#define E1000_CTRL_EXT_FORCE_SMBUS      0x00000800 /* Force SMBus mode */
#define E1000_CTRL_EXT_EE_RST   0x00002000 /* Reinitialize from EEPROM */
/* Physical Func Reset Done Indication */
#define E1000_CTRL_EXT_PFRSTD   0x00004000
#define E1000_CTRL_EXT_SPD_BYPS 0x00008000 /* Speed Select Bypass */
#define E1000_CTRL_EXT_RO_DIS   0x00020000 /* Relaxed Ordering disable */
#define E1000_CTRL_EXT_DMA_DYN_CLK_EN   0x00080000 /* DMA Dynamic Clk Gating */
#define E1000_CTRL_EXT_LINK_MODE_MASK   0x00C00000
/* Offset of the link mode field in Ctrl Ext register */
#define E1000_CTRL_EXT_LINK_MODE_OFFSET 22
#define E1000_CTRL_EXT_LINK_MODE_1000BASE_KX    0x00400000
#define E1000_CTRL_EXT_LINK_MODE_GMII   0x00000000
#define E1000_CTRL_EXT_LINK_MODE_PCIE_SERDES    0x00C00000
#define E1000_CTRL_EXT_LINK_MODE_SGMII  0x00800000
#define E1000_CTRL_EXT_EIAME            0x01000000
#define E1000_CTRL_EXT_IRCA             0x00000001
#define E1000_CTRL_EXT_DRV_LOAD         0x10000000 /* Drv loaded bit for FW */
#define E1000_CTRL_EXT_IAME             0x08000000 /* Int ACK Auto-mask */
#define E1000_CTRL_EXT_PBA_CLR          0x80000000 /* PBA Clear */
#define E1000_CTRL_EXT_LSECCK           0x00001000
#define E1000_CTRL_EXT_PHYPDEN          0x00100000
#define E1000_I2CCMD_REG_ADDR_SHIFT     16
#define E1000_I2CCMD_PHY_ADDR_SHIFT     24
#define E1000_I2CCMD_OPCODE_READ        0x08000000
#define E1000_I2CCMD_OPCODE_WRITE       0x00000000
#define E1000_I2CCMD_READY              0x20000000
#define E1000_I2CCMD_ERROR              0x80000000
#define E1000_I2CCMD_SFP_DATA_ADDR(a)   (0x0000 + (a))
#define E1000_I2CCMD_SFP_DIAG_ADDR(a)   (0x0100 + (a))
#define E1000_MAX_SGMII_PHY_REG_ADDR    255
#define E1000_I2CCMD_PHY_TIMEOUT        200
#define E1000_IVAR_VALID        0x80
#define E1000_GPIE_NSICR        0x00000001
#define E1000_GPIE_MSIX_MODE    0x00000010
#define E1000_GPIE_EIAME        0x40000000
#define E1000_GPIE_PBA          0x80000000

/* Receive Descriptor bit definitions */
#define E1000_RXD_STAT_DD       0x01    /* Descriptor Done */
#define E1000_RXD_STAT_EOP      0x02    /* End of Packet */
#define E1000_RXD_STAT_IXSM     0x04    /* Ignore checksum */
#define E1000_RXD_STAT_VP       0x08    /* IEEE VLAN Packet */
#define E1000_RXD_STAT_UDPCS    0x10    /* UDP xsum calculated */
#define E1000_RXD_STAT_TCPCS    0x20    /* TCP xsum calculated */
#define E1000_RXD_STAT_IPCS     0x40    /* IP xsum calculated */
#define E1000_RXD_STAT_PIF      0x80    /* passed in-exact filter */
#define E1000_RXD_STAT_IPIDV    0x200   /* IP identification valid */
#define E1000_RXD_STAT_UDPV     0x400   /* Valid UDP checksum */
#define E1000_RXD_STAT_DYNINT   0x800   /* Pkt caused INT via DYNINT */
#define E1000_RXD_ERR_CE        0x01    /* CRC Error */
#define E1000_RXD_ERR_SE        0x02    /* Symbol Error */
#define E1000_RXD_ERR_SEQ       0x04    /* Sequence Error */
#define E1000_RXD_ERR_CXE       0x10    /* Carrier Extension Error */
```

```
#define E1000_RXD_ERR_TCPE      0x20    /* TCP/UDP Checksum Error */
#define E1000_RXD_ERR_IPE       0x40    /* IP Checksum Error */
#define E1000_RXD_ERR_RXE       0x80    /* Rx Data Error */
#define E1000_RXD_SPC_VLAN_MASK 0x0FFF  /* VLAN ID is in lower 12 bits */

#define E1000_RXDEXT_STATERR_TST        0x00000100 /* Time Stamp taken */
#define E1000_RXDEXT_STATERR_LB         0x00040000
#define E1000_RXDEXT_STATERR_CE         0x01000000
#define E1000_RXDEXT_STATERR_SE         0x02000000
#define E1000_RXDEXT_STATERR_SEQ        0x04000000
#define E1000_RXDEXT_STATERR_CXE        0x10000000
#define E1000_RXDEXT_STATERR_TCPE       0x20000000
#define E1000_RXDEXT_STATERR_IPE        0x40000000
#define E1000_RXDEXT_STATERR_RXE        0x80000000

/* mask to determine if packets should be dropped due to frame errors */
#define E1000_RXD_ERR_FRAME_ERR_MASK ( \
        E1000_RXD_ERR_CE  |            \
        E1000_RXD_ERR_SE  |            \
        E1000_RXD_ERR_SEQ |            \
        E1000_RXD_ERR_CXE |            \
        E1000_RXD_ERR_RXE)

/* Same mask, but for extended and packet split descriptors */
#define E1000_RXDEXT_ERR_FRAME_ERR_MASK ( \
        E1000_RXDEXT_STATERR_CE  |        \
        E1000_RXDEXT_STATERR_SE  |        \
        E1000_RXDEXT_STATERR_SEQ |        \
        E1000_RXDEXT_STATERR_CXE |        \
        E1000_RXDEXT_STATERR_RXE)

#define E1000_MRQC_RSS_FIELD_MASK               0xFFFF0000
#define E1000_MRQC_RSS_FIELD_IPV4_TCP           0x00010000
#define E1000_MRQC_RSS_FIELD_IPV4               0x00020000
#define E1000_MRQC_RSS_FIELD_IPV6_TCP_EX        0x00040000
#define E1000_MRQC_RSS_FIELD_IPV6               0x00100000
#define E1000_MRQC_RSS_FIELD_IPV6_TCP           0x00200000

#define E1000_RXDPS_HDRSTAT_HDRSP               0x00008000

/* Management Control */
#define E1000_MANC_SMBUS_EN     0x00000001 /* SMBus Enabled - RO */
#define E1000_MANC_ASF_EN       0x00000002 /* ASF Enabled - RO */
#define E1000_MANC_ARP_EN       0x00002000 /* Enable ARP Request Filtering */
#define E1000_MANC_RCV_TCO_EN   0x00020000 /* Receive TCO Packets Enabled */
#define E1000_MANC_BLK_PHY_RST_ON_IDE   0x00040000 /* Block phy resets */
/* Enable MAC address filtering */
#define E1000_MANC_EN_MAC_ADDR_FILTER   0x00100000
/* Enable MNG packets to host memory */
#define E1000_MANC_EN_MNG2HOST          0x00200000

#define E1000_MANC2H_PORT_623           0x00000020 /* Port 0x26f */
#define E1000_MANC2H_PORT_664           0x00000040 /* Port 0x298 */
#define E1000_MDEF_PORT_623             0x00000800 /* Port 0x26f */
#define E1000_MDEF_PORT_664             0x00000400 /* Port 0x298 */

/* Receive Control */
#define E1000_RCTL_RST          0x00000001 /* Software reset */
#define E1000_RCTL_EN           0x00000002 /* enable */
#define E1000_RCTL_SBP          0x00000004 /* store bad packet */
#define E1000_RCTL_UPE          0x00000008 /* unicast promisc enable */
#define E1000_RCTL_MPE          0x00000010 /* multicast promisc enable */
#define E1000_RCTL_LPE          0x00000020 /* long packet enable */
#define E1000_RCTL_LBM_NO       0x00000000 /* no loopback mode */
#define E1000_RCTL_LBM_MAC      0x00000040 /* MAC loopback mode */
#define E1000_RCTL_LBM_TCVR     0x000000C0 /* tcvr loopback mode */
```



```
#define E1000_RCTL_DTYP_PS      0x00000400 /* Packet Split descriptor */
#define E1000_RCTL_RDMTS_HALF   0x00000000 /* Rx desc min thresh size */
#define E1000_RCTL_MO_SHIFT     12 /* multicast offset shift */
#define E1000_RCTL_MO_3         0x00003000 /* multicast offset 15:4 */
#define E1000_RCTL_BAM          0x00008000 /* broadcast enable */
/* these buffer sizes are valid if E1000_RCTL_BSEX is 0 */
#define E1000_RCTL_SZ_2048      0x00000000 /* Rx buffer size 2048 */
#define E1000_RCTL_SZ_1024      0x00010000 /* Rx buffer size 1024 */
#define E1000_RCTL_SZ_512       0x00020000 /* Rx buffer size 512 */
#define E1000_RCTL_SZ_256       0x00030000 /* Rx buffer size 256 */
/* these buffer sizes are valid if E1000_RCTL_BSEX is 1 */
#define E1000_RCTL_SZ_16384     0x00010000 /* Rx buffer size 16384 */
#define E1000_RCTL_SZ_8192      0x00020000 /* Rx buffer size 8192 */
#define E1000_RCTL_SZ_4096      0x00030000 /* Rx buffer size 4096 */
#define E1000_RCTL_VFE          0x00040000 /* vlan filter enable */
#define E1000_RCTL_CFIEN        0x00080000 /* canonical form enable */
#define E1000_RCTL_CFI          0x00100000 /* canonical form indicator */
#define E1000_RCTL_DPF          0x00400000 /* discard pause frames */
#define E1000_RCTL_PMCF         0x00800000 /* pass MAC control frames */
#define E1000_RCTL_BSEX         0x02000000 /* Buffer size extension */
#define E1000_RCTL_SECRC        0x04000000 /* Strip Ethernet CRC */

/* Use byte values for the following shift parameters
 * Usage:
 *     psrctl |= (((ROUNDUP(value0, 128) >> E1000_PSRCTL_BSIZE0_SHIFT) &
 *                 E1000_PSRCTL_BSIZE0_MASK) |
 *                ((ROUNDUP(value1, 1024) >> E1000_PSRCTL_BSIZE1_SHIFT) &
 *                 E1000_PSRCTL_BSIZE1_MASK) |
 *                ((ROUNDUP(value2, 1024) << E1000_PSRCTL_BSIZE2_SHIFT) &
 *                 E1000_PSRCTL_BSIZE2_MASK) |
 *                ((ROUNDUP(value3, 1024) << E1000_PSRCTL_BSIZE3_SHIFT) |;
 *                 E1000_PSRCTL_BSIZE3_MASK))
 * where value0 = [128..16256],  default=256
 *       value1 = [1024..64512], default=4096
 *       value2 = [0..64512],    default=4096
 *       value3 = [0..64512],    default=0
 */

#define E1000_PSRCTL_BSIZE0_MASK        0x0000007F
#define E1000_PSRCTL_BSIZE1_MASK        0x00003F00
#define E1000_PSRCTL_BSIZE2_MASK        0x003F0000
#define E1000_PSRCTL_BSIZE3_MASK        0x3F000000

#define E1000_PSRCTL_BSIZE0_SHIFT       7    /* Shift _right_ 7 */
#define E1000_PSRCTL_BSIZE1_SHIFT       2    /* Shift _right_ 2 */
#define E1000_PSRCTL_BSIZE2_SHIFT       6    /* Shift _left_ 6 */
#define E1000_PSRCTL_BSIZE3_SHIFT       14   /* Shift _left_ 14 */

/* SWFW_SYNC Definitions */
#define E1000_SWFW_EEP_SM       0x01
#define E1000_SWFW_PHY0_SM      0x02
#define E1000_SWFW_PHY1_SM      0x04
#define E1000_SWFW_CSR_SM       0x08
#define E1000_SWFW_PHY2_SM      0x20
#define E1000_SWFW_PHY3_SM      0x40
#define E1000_SWFW_SW_MNG_SM    0x400

/* Device Control */
#define E1000_CTRL_FD           0x00000001  /* Full duplex.0=half; 1=full */
#define E1000_CTRL_PRIOR        0x00000004  /* Priority on PCI. 0=rx,1=fair */
#define E1000_CTRL_GIO_MASTER_DISABLE 0x00000004 /*Blocks new Master reqs */
#define E1000_CTRL_LRST         0x00000008  /* Link reset. 0=normal,1=reset */
#define E1000_CTRL_ASDE         0x00000020  /* Auto-speed detect enable */
#define E1000_CTRL_SLU          0x00000040  /* Set link up (Force Link) */
#define E1000_CTRL_ILOS         0x00000080  /* Invert Loss-Of Signal */
#define E1000_CTRL_SPD_SEL      0x00000300  /* Speed Select Mask */
```

```
259 #define E1000_CTRL_SPD_10       0x00000000  /* Force 10Mb */
260 #define E1000_CTRL_SPD_100      0x00000100  /* Force 100Mb */
261 #define E1000_CTRL_SPD_1000     0x00000200  /* Force 1Gb */
262 #define E1000_CTRL_FRCSPD       0x00000800  /* Force Speed */
263 #define E1000_CTRL_FRCDPX       0x00001000  /* Force Duplex */
264 #define E1000_CTRL_LANPHYPC_OVERRIDE    0x00010000 /* SW control of LANPHYPC */
265 #define E1000_CTRL_LANPHYPC_VALUE       0x00020000 /* SW value of LANPHYPC */
266 #define E1000_CTRL_MEHE         0x00080000 /* Memory Error Handling Enable */
267 #define E1000_CTRL_SWDPIN0      0x00040000 /* SWDPIN 0 value */
268 #define E1000_CTRL_SWDPIN1      0x00080000 /* SWDPIN 1 value */
269 #define E1000_CTRL_SWDPIN2      0x00100000 /* SWDPIN 2 value */
270 #define E1000_CTRL_ADVD3WUC     0x00100000 /* D3 WUC */
271 #define E1000_CTRL_EN_PHY_PWR_MGMT      0x00200000 /* PHY PM enable */
272 #define E1000_CTRL_SWDPIN3      0x00200000 /* SWDPIN 3 value */
273 #define E1000_CTRL_SWDPIO0      0x00400000 /* SWDPIN 0 Input or output */
274 #define E1000_CTRL_SWDPIO2      0x01000000 /* SWDPIN 2 input or output */
275 #define E1000_CTRL_SWDPIO3      0x02000000 /* SWDPIN 3 input or output */
276 #define E1000_CTRL_RST          0x04000000 /* Global reset */
277 #define E1000_CTRL_RFCE         0x08000000 /* Receive Flow Control enable */
278 #define E1000_CTRL_TFCE         0x10000000 /* Transmit flow control enable */
279 #define E1000_CTRL_VME          0x40000000 /* IEEE VLAN mode enable */
280 #define E1000_CTRL_PHY_RST      0x80000000 /* PHY Reset */
281 #define E1000_CTRL_I2C_ENA      0x02000000 /* I2C enable */

283 #define E1000_CTRL_MDIO_DIR             E1000_CTRL_SWDPIO2
284 #define E1000_CTRL_MDIO                 E1000_CTRL_SWDPIN2
285 #define E1000_CTRL_MDC_DIR              E1000_CTRL_SWDPIO3
286 #define E1000_CTRL_MDC                  E1000_CTRL_SWDPIN3

288 #define E1000_CONNSW_ENRGSRC            0x4
289 #define E1000_CONNSW_PHYSD              0x400
290 #define E1000_CONNSW_PHY_PDN            0x800
291 #define E1000_CONNSW_SERDESD            0x200
292 #define E1000_CONNSW_AUTOSENSE_CONF     0x2
293 #define E1000_CONNSW_AUTOSENSE_EN       0x1
294 #define E1000_PCS_CFG_PCS_EN            8
295 #define E1000_PCS_LCTL_FLV_LINK_UP      1
296 #define E1000_PCS_LCTL_FSV_10           0
297 #define E1000_PCS_LCTL_FSV_100          2
298 #define E1000_PCS_LCTL_FSV_1000         4
299 #define E1000_PCS_LCTL_FDV_FULL         8
300 #define E1000_PCS_LCTL_FSD              0x10
301 #define E1000_PCS_LCTL_FORCE_LINK       0x20
302 #define E1000_PCS_LCTL_FORCE_FCTRL      0x80
303 #define E1000_PCS_LCTL_AN_ENABLE        0x10000
304 #define E1000_PCS_LCTL_AN_RESTART       0x20000
305 #define E1000_PCS_LCTL_AN_TIMEOUT       0x40000
306 #define E1000_ENABLE_SERDES_LOOPBACK    0x0410

308 #define E1000_PCS_LSTS_LINK_OK          1
309 #define E1000_PCS_LSTS_SPEED_100        2
310 #define E1000_PCS_LSTS_SPEED_1000       4
311 #define E1000_PCS_LSTS_DUPLEX_FULL      8
312 #define E1000_PCS_LSTS_SYNK_OK          0x10
313 #define E1000_PCS_LSTS_AN_COMPLETE      0x10000

315 /* Device Status */
316 #define E1000_STATUS_FD                 0x00000001 /* Duplex 0=half 1=full */
317 #define E1000_STATUS_LU                 0x00000002 /* Link up.0=no,1=link */
318 #define E1000_STATUS_FUNC_MASK          0x0000000C /* PCI Function Mask */
319 #define E1000_STATUS_FUNC_SHIFT         2
320 #define E1000_STATUS_FUNC_1             0x00000004 /* Function 1 */
321 #define E1000_STATUS_TXOFF              0x00000010 /* transmission paused */
322 #define E1000_STATUS_SPEED_MASK 0x000000C0
323 #define E1000_STATUS_SPEED_10           0x00000000 /* Speed 10Mb/s */
324 #define E1000_STATUS_SPEED_100          0x00000040 /* Speed 100Mb/s */
```



```
325 #define E1000_STATUS_SPEED_1000         0x00000080 /* Speed 1000Mb/s */
326 #define E1000_STATUS_LAN_INIT_DONE      0x00000200 /* Lan Init Compltn by NVM */
327 #define E1000_STATUS_PHYRA              0x00000400 /* PHY Reset Asserted */
328 #define E1000_STATUS_GIO_MASTER_ENABLE  0x00080000 /* Master request status */
329 #define E1000_STATUS_PCI66              0x00000800 /* In 66Mhz slot */
330 #define E1000_STATUS_BUS64              0x00001000 /* In 64 bit slot */
331 #define E1000_STATUS_2P5_SKU            0x00001000 /* Val of 2.5GBE SKU strap */
332 #define E1000_STATUS_2P5_SKU_OVER       0x00002000 /* Val of 2.5GBE SKU Over */
333 #define E1000_STATUS_PCIX_MODE          0x00002000 /* PCI-X mode */
334 #define E1000_STATUS_PCIX_SPEED         0x0000C000 /* PCI-X bus speed */

336 /* Constants used to interpret the masked PCI-X bus speed. */
337 #define E1000_STATUS_PCIX_SPEED_66      0x00000000 /* PCI-X bus spd 50-66MHz */
338 #define E1000_STATUS_PCIX_SPEED_100     0x00004000 /* PCI-X bus spd 66-100MHz */
339 #define E1000_STATUS_PCIX_SPEED_133     0x00008000 /* PCI-X bus spd 100-133MHz*/

341 #define SPEED_10        10
342 #define SPEED_100       100
343 #define SPEED_1000      1000
344 #define SPEED_2500      2500
345 #define HALF_DUPLEX     1
346 #define FULL_DUPLEX     2

348 #define PHY_FORCE_TIME  20

350 #define ADVERTISE_10_HALF               0x0001
351 #define ADVERTISE_10_FULL               0x0002
352 #define ADVERTISE_100_HALF              0x0004
353 #define ADVERTISE_100_FULL              0x0008
354 #define ADVERTISE_1000_HALF             0x0010 /* Not used, just FYI */
355 #define ADVERTISE_1000_FULL             0x0020

357 /* 1000/H is not supported, nor spec-compliant. */
358 #define E1000_ALL_SPEED_DUPLEX  ( \
359         ADVERTISE_10_HALF | ADVERTISE_10_FULL | ADVERTISE_100_HALF | \
360         ADVERTISE_100_FULL | ADVERTISE_1000_FULL)
361 #define E1000_ALL_NOT_GIG       ( \
362         ADVERTISE_10_HALF | ADVERTISE_10_FULL | ADVERTISE_100_HALF | \
363         ADVERTISE_100_FULL)
364 #define E1000_ALL_100_SPEED     (ADVERTISE_100_HALF | ADVERTISE_100_FULL)
365 #define E1000_ALL_10_SPEED      (ADVERTISE_10_HALF | ADVERTISE_10_FULL)
366 #define E1000_ALL_HALF_DUPLEX   (ADVERTISE_10_HALF | ADVERTISE_100_HALF)

368 #define AUTONEG_ADVERTISE_SPEED_DEFAULT         E1000_ALL_SPEED_DUPLEX

370 /* LED Control */
371 #define E1000_PHY_LED0_MODE_MASK        0x00000007
372 #define E1000_PHY_LED0_IVRT             0x00000008
373 #define E1000_PHY_LED0_MASK             0x0000001F

375 #define E1000_LEDCTL_LED0_MODE_MASK     0x0000000F
376 #define E1000_LEDCTL_LED0_MODE_SHIFT    0
377 #define E1000_LEDCTL_LED0_IVRT          0x00000040
378 #define E1000_LEDCTL_LED0_BLINK         0x00000080

380 #define E1000_LEDCTL_MODE_LINK_UP       0x2
381 #define E1000_LEDCTL_MODE_LED_ON        0xE
382 #define E1000_LEDCTL_MODE_LED_OFF       0xF

384 /* Transmit Descriptor bit definitions */
385 #define E1000_TXD_DTYP_D        0x00100000 /* Data Descriptor */
386 #define E1000_TXD_DTYP_C        0x00000000 /* Context Descriptor */
387 #define E1000_TXD_POPTS_IXSM    0x01       /* Insert IP checksum */
388 #define E1000_TXD_POPTS_TXSM    0x02       /* Insert TCP/UDP checksum */
389 #define E1000_TXD_CMD_EOP       0x01000000 /* End of Packet */
390 #define E1000_TXD_CMD_IFCS      0x02000000 /* Insert FCS (Ethernet CRC) */
```

```
391 #define E1000_TXD_CMD_IC        0x04000000 /* Insert Checksum */
392 #define E1000_TXD_CMD_RS        0x08000000 /* Report Status */
393 #define E1000_TXD_CMD_RPS       0x10000000 /* Report Packet Sent */
394 #define E1000_TXD_CMD_DEXT      0x20000000 /* Desc extension (0 = legacy) */
395 #define E1000_TXD_CMD_VLE       0x40000000 /* Add VLAN tag */
396 #define E1000_TXD_CMD_IDE       0x80000000 /* Enable Tidv register */
397 #define E1000_TXD_STAT_DD       0x00000001 /* Descriptor Done */
398 #define E1000_TXD_STAT_EC       0x00000002 /* Excess Collisions */
399 #define E1000_TXD_STAT_LC       0x00000004 /* Late Collisions */
400 #define E1000_TXD_STAT_TU       0x00000008 /* Transmit underrun */
401 #define E1000_TXD_CMD_TCP       0x01000000 /* TCP packet */
402 #define E1000_TXD_CMD_IP        0x02000000 /* IP packet */
403 #define E1000_TXD_CMD_TSE       0x04000000 /* TCP Seg enable */
404 #define E1000_TXD_STAT_TC       0x00000004 /* Tx Underrun */
405 #define E1000_TXD_EXTCMD_TSTAMP 0x00000010 /* IEEE1588 Timestamp packet */

407 /* Transmit Control */
408 #define E1000_TCTL_EN           0x00000002 /* enable Tx */
409 #define E1000_TCTL_PSP          0x00000008 /* pad short packets */
410 #define E1000_TCTL_CT           0x00000ff0 /* collision threshold */
411 #define E1000_TCTL_COLD         0x003ff000 /* collision distance */
412 #define E1000_TCTL_RTLC         0x01000000 /* Re-transmit on late collision */
413 #define E1000_TCTL_MULR         0x10000000 /* Multiple request support */

415 /* Transmit Arbitration Count */
416 #define E1000_TARC0_ENABLE      0x00000400 /* Enable Tx Queue 0 */

418 /* SerDes Control */
419 #define E1000_SCTL_DISABLE_SERDES_LOOPBACK      0x0400
420 #define E1000_SCTL_ENABLE_SERDES_LOOPBACK       0x0410

422 /* Receive Checksum Control */
423 #define E1000_RXCSUM_IPOFL      0x00000100 /* IPv4 checksum offload */
424 #define E1000_RXCSUM_TUOFL      0x00000200 /* TCP / UDP checksum offload */
425 #define E1000_RXCSUM_CRCOFL     0x00000800 /* CRC32 offload enable */
426 #define E1000_RXCSUM_IPPCSE     0x00001000 /* IP payload checksum enable */
427 #define E1000_RXCSUM_PCSD       0x00002000 /* packet checksum disabled */

429 /* Header split receive */
430 #define E1000_RFCTL_NFSW_DIS            0x00000040
431 #define E1000_RFCTL_NFSR_DIS            0x00000080
432 #define E1000_RFCTL_ACK_DIS             0x00001000
433 #define E1000_RFCTL_EXTEN               0x00008000
434 #define E1000_RFCTL_IPV6_EX_DIS         0x00010000
435 #define E1000_RFCTL_NEW_IPV6_EXT_DIS    0x00020000
436 #define E1000_RFCTL_LEF                 0x00040000

438 /* Collision related configuration parameters */
439 #define E1000_COLLISION_THRESHOLD       15
440 #define E1000_CT_SHIFT                  4
441 #define E1000_COLLISION_DISTANCE        63
442 #define E1000_COLD_SHIFT                12

444 /* Default values for the transmit IPG register */
445 #define DEFAULT_82542_TIPG_IPGT         10
446 #define DEFAULT_82543_TIPG_IPGT_FIBER   9
447 #define DEFAULT_82543_TIPG_IPGT_COPPER  8

449 #define E1000_TIPG_IPGT_MASK            0x000003FF

451 #define DEFAULT_82542_TIPG_IPGR1        2
452 #define DEFAULT_82543_TIPG_IPGR1        8
453 #define E1000_TIPG_IPGR1_SHIFT          10

455 #define DEFAULT_82542_TIPG_IPGR2        10
456 #define DEFAULT_82543_TIPG_IPGR2        6
```



```
457 #define DEFAULT_80003ES2LAN_TIPG_IPGR2  7
458 #define E1000_TIPG_IPGR2_SHIFT          20

460 /* Ethertype field values */
461 #define ETHERNET_IEEE_VLAN_TYPE         0x8100  /* 802.3ac packet */

463 #define ETHERNET_FCS_SIZE               4
464 #define MAX_JUMBO_FRAME_SIZE            0x3F00

466 /* Extended Configuration Control and Size */
467 #define E1000_EXTCNF_CTRL_MDIO_SW_OWNERSHIP     0x00000020
468 #define E1000_EXTCNF_CTRL_LCD_WRITE_ENABLE      0x00000001
469 #define E1000_EXTCNF_CTRL_OEM_WRITE_ENABLE      0x00000008
470 #define E1000_EXTCNF_CTRL_SWFLAG                0x00000020
471 #define E1000_EXTCNF_CTRL_GATE_PHY_CFG          0x00000080
472 #define E1000_EXTCNF_SIZE_EXT_PCIE_LENGTH_MASK  0x00FF0000
473 #define E1000_EXTCNF_SIZE_EXT_PCIE_LENGTH_SHIFT 16
474 #define E1000_EXTCNF_CTRL_EXT_CNF_POINTER_MASK  0x0FFF0000
475 #define E1000_EXTCNF_CTRL_EXT_CNF_POINTER_SHIFT 16

477 #define E1000_PHY_CTRL_D0A_LPLU                 0x00000002
478 #define E1000_PHY_CTRL_NOND0A_LPLU              0x00000004
479 #define E1000_PHY_CTRL_NOND0A_GBE_DISABLE       0x00000008
480 #define E1000_PHY_CTRL_GBE_DISABLE              0x00000040

482 #define E1000_KABGTXD_BGSQLBIAS                 0x00050000

484 /* Low Power IDLE Control */
485 #define E1000_LPIC_LPIET_SHIFT          24      /* Low Power Idle Entry Time */

487 /* PBA constants */
488 #define E1000_PBA_8K            0x0008    /* 8KB */
489 #define E1000_PBA_10K           0x000A    /* 10KB */
490 #define E1000_PBA_12K           0x000C    /* 12KB */
491 #define E1000_PBA_14K           0x000E    /* 14KB */
492 #define E1000_PBA_16K           0x0010    /* 16KB */
493 #define E1000_PBA_18K           0x0012
494 #define E1000_PBA_20K           0x0014
495 #define E1000_PBA_22K           0x0016
496 #define E1000_PBA_24K           0x0018
497 #define E1000_PBA_26K           0x001A
498 #define E1000_PBA_30K           0x001E
499 #define E1000_PBA_32K           0x0020
500 #define E1000_PBA_34K           0x0022
501 #define E1000_PBA_35K           0x0023
502 #define E1000_PBA_38K           0x0026
503 #define E1000_PBA_40K           0x0028
504 #define E1000_PBA_48K           0x0030    /* 48KB */
505 #define E1000_PBA_64K           0x0040    /* 64KB */

507 #define E1000_PBA_RXA_MASK      0xFFFF

509 #define E1000_PBS_16K           E1000_PBA_16K

511 /* Uncorrectable/correctable ECC Error counts and enable bits */
512 #define E1000_PBECCSTS_CORR_ERR_CNT_MASK        0x000000FF
513 #define E1000_PBECCSTS_UNCORR_ERR_CNT_MASK      0x0000FF00
514 #define E1000_PBECCSTS_UNCORR_ERR_CNT_SHIFT     8
515 #define E1000_PBECCSTS_ECC_ENABLE               0x00010000

517 #define IFS_MAX                 80
518 #define IFS_MIN                 40
519 #define IFS_RATIO               4
520 #define IFS_STEP                10
521 #define MIN_NUM_XMITS           1000
```

```
523 /* SW Semaphore Register */
524 #define E1000_SWSM_SMBI         0x00000001 /* Driver Semaphore bit */
525 #define E1000_SWSM_SWESMBI      0x00000002 /* FW Semaphore bit */
526 #define E1000_SWSM_DRV_LOAD     0x00000008 /* Driver Loaded Bit */

528 #define E1000_SWSM2_LOCK        0x00000002 /* Secondary driver semaphore bit */

530 /* Interrupt Cause Read */
531 #define E1000_ICR_TXDW          0x00000001 /* Transmit desc written back */
532 #define E1000_ICR_TXQE          0x00000002 /* Transmit Queue empty */
533 #define E1000_ICR_LSC           0x00000004 /* Link Status Change */
534 #define E1000_ICR_RXSEQ         0x00000008 /* Rx sequence error */
535 #define E1000_ICR_RXDMT0        0x00000010 /* Rx desc min. threshold (0) */
536 #define E1000_ICR_RXO           0x00000040 /* Rx overrun */
537 #define E1000_ICR_RXT0          0x00000080 /* Rx timer intr (ring 0) */
538 #define E1000_ICR_VMMB          0x00000100 /* VM MB event */
539 #define E1000_ICR_RXCFG         0x00000400 /* Rx /c/ ordered set */
540 #define E1000_ICR_GPI_EN0       0x00000800 /* GP Int 0 */
541 #define E1000_ICR_GPI_EN1       0x00001000 /* GP Int 1 */
542 #define E1000_ICR_GPI_EN2       0x00002000 /* GP Int 2 */
543 #define E1000_ICR_GPI_EN3       0x00004000 /* GP Int 3 */
544 #define E1000_ICR_TXD_LOW       0x00008000
545 #define E1000_ICR_MNG           0x00040000 /* Manageability event */
546 #define E1000_ICR_ECCER         0x00400000 /* Uncorrectable ECC Error */
547 #define E1000_ICR_TS            0x00080000 /* Time Sync Interrupt */
548 #define E1000_ICR_DRSTA         0x40000000 /* Device Reset Asserted */
549 /* If this bit asserted, the driver should claim the interrupt */
550 #define E1000_ICR_INT_ASSERTED  0x80000000
551 #define E1000_ICR_DOUTSYNC      0x10000000 /* NIC DMA out of sync */
552 #define E1000_ICR_RXQ0          0x00100000 /* Rx Queue 0 Interrupt */
553 #define E1000_ICR_RXQ1          0x00200000 /* Rx Queue 1 Interrupt */
554 #define E1000_ICR_TXQ0          0x00400000 /* Tx Queue 0 Interrupt */
555 #define E1000_ICR_TXQ1          0x00800000 /* Tx Queue 1 Interrupt */
556 #define E1000_ICR_OTHER         0x01000000 /* Other Interrupts */
557 #define E1000_ICR_FER           0x00400000 /* Fatal Error */

559 #define E1000_ICR_THS           0x00800000 /* ICR.THS: Thermal Sensor Event*/
560 #define E1000_ICR_MDDET         0x10000000 /* Malicious Driver Detect */

562 #define E1000_ITR_MASK          0x000FFFFF /* ITR value bitfield */
563 #define E1000_ITR_MULT          256 /* ITR mulitplier in nsec */

565 /* PBA ECC Register */
566 #define E1000_PBA_ECC_COUNTER_MASK      0xFFF00000 /* ECC counter mask */
567 #define E1000_PBA_ECC_COUNTER_SHIFT     20 /* ECC counter shift value */
568 #define E1000_PBA_ECC_CORR_EN   0x00000001 /* Enable ECC error correction */
569 #define E1000_PBA_ECC_STAT_CLR  0x00000002 /* Clear ECC error counter */
570 #define E1000_PBA_ECC_INT_EN    0x00000004 /* Enable ICR bit 5 on ECC error */

572 /* Extended Interrupt Cause Read */
573 #define E1000_EICR_RX_QUEUE0    0x00000001 /* Rx Queue 0 Interrupt */
574 #define E1000_EICR_RX_QUEUE1    0x00000002 /* Rx Queue 1 Interrupt */
575 #define E1000_EICR_RX_QUEUE2    0x00000004 /* Rx Queue 2 Interrupt */
576 #define E1000_EICR_RX_QUEUE3    0x00000008 /* Rx Queue 3 Interrupt */
577 #define E1000_EICR_TX_QUEUE0    0x00000100 /* Tx Queue 0 Interrupt */
578 #define E1000_EICR_TX_QUEUE1    0x00000200 /* Tx Queue 1 Interrupt */
579 #define E1000_EICR_TX_QUEUE2    0x00000400 /* Tx Queue 2 Interrupt */
580 #define E1000_EICR_TX_QUEUE3    0x00000800 /* Tx Queue 3 Interrupt */
581 #define E1000_EICR_TCP_TIMER    0x40000000 /* TCP Timer */
582 #define E1000_EICR_OTHER        0x80000000 /* Interrupt Cause Active */
583 /* TCP Timer */
584 #define E1000_TCPTIMER_KS       0x00000100 /* KickStart */
585 #define E1000_TCPTIMER_COUNT_ENABLE     0x00000200 /* Count Enable */
586 #define E1000_TCPTIMER_COUNT_FINISH     0x00000400 /* Count finish */
587 #define E1000_TCPTIMER_LOOP     0x00000800 /* Loop */
```



```
589 /* This defines the bits that are set in the Interrupt Mask
590  * Set/Read Register.  Each bit is documented below:
591  *   o RXT0   = Receiver Timer Interrupt (ring 0)
592  *   o TXDW   = Transmit Descriptor Written Back
593  *   o RXDMT0 = Receive Descriptor Minimum Threshold hit (ring 0)
594  *   o RXSEQ  = Receive Sequence Error
595  *   o LSC    = Link Status Change
596  */
597 #define IMS_ENABLE_MASK ( \
598         E1000_IMS_RXT0   |    \
599         E1000_IMS_TXDW   |    \
600         E1000_IMS_RXDMT0 |    \
601         E1000_IMS_RXSEQ  |    \
602         E1000_IMS_LSC)

604 /* Interrupt Mask Set */
605 #define E1000_IMS_TXDW          E1000_ICR_TXDW    /* Tx desc written back */
606 #define E1000_IMS_TXQE          E1000_ICR_TXQE    /* Transmit Queue empty */
607 #define E1000_IMS_LSC           E1000_ICR_LSC     /* Link Status Change */
608 #define E1000_IMS_VMMB          E1000_ICR_VMMB    /* Mail box activity */
609 #define E1000_IMS_RXSEQ         E1000_ICR_RXSEQ   /* Rx sequence error */
610 #define E1000_IMS_RXDMT0        E1000_ICR_RXDMT0  /* Rx desc min. threshold */
611 #define E1000_IMS_RXO           E1000_ICR_RXO     /* Rx overrun */
612 #define E1000_IMS_RXT0          E1000_ICR_RXT0    /* Rx timer intr */
613 #define E1000_IMS_TXD_LOW       E1000_ICR_TXD_LOW
614 #define E1000_IMS_ECCER         E1000_ICR_ECCER   /* Uncorrectable ECC Error */
615 #define E1000_IMS_TS            E1000_ICR_TS      /* Time Sync Interrupt */
616 #define E1000_IMS_DRSTA         E1000_ICR_DRSTA   /* Device Reset Asserted */
617 #define E1000_IMS_DOUTSYNC      E1000_ICR_DOUTSYNC /* NIC DMA out of sync */
618 #define E1000_IMS_RXQ0          E1000_ICR_RXQ0 /* Rx Queue 0 Interrupt */
619 #define E1000_IMS_RXQ1          E1000_ICR_RXQ1 /* Rx Queue 1 Interrupt */
620 #define E1000_IMS_TXQ0          E1000_ICR_TXQ0 /* Tx Queue 0 Interrupt */
621 #define E1000_IMS_TXQ1          E1000_ICR_TXQ1 /* Tx Queue 1 Interrupt */
622 #define E1000_IMS_OTHER         E1000_ICR_OTHER /* Other Interrupts */
623 #define E1000_IMS_FER           E1000_ICR_FER /* Fatal Error */

625 #define E1000_IMS_THS           E1000_ICR_THS /* ICR.TS: Thermal Sensor Event*/
626 #define E1000_IMS_MDDET         E1000_ICR_MDDET /* Malicious Driver Detect */
627 /* Extended Interrupt Mask Set */
628 #define E1000_EIMS_RX_QUEUE0    E1000_EICR_RX_QUEUE0 /* Rx Queue 0 Interrupt */
629 #define E1000_EIMS_RX_QUEUE1    E1000_EICR_RX_QUEUE1 /* Rx Queue 1 Interrupt */
630 #define E1000_EIMS_RX_QUEUE2    E1000_EICR_RX_QUEUE2 /* Rx Queue 2 Interrupt */
631 #define E1000_EIMS_RX_QUEUE3    E1000_EICR_RX_QUEUE3 /* Rx Queue 3 Interrupt */
632 #define E1000_EIMS_TX_QUEUE0    E1000_EICR_TX_QUEUE0 /* Tx Queue 0 Interrupt */
633 #define E1000_EIMS_TX_QUEUE1    E1000_EICR_TX_QUEUE1 /* Tx Queue 1 Interrupt */
634 #define E1000_EIMS_TX_QUEUE2    E1000_EICR_TX_QUEUE2 /* Tx Queue 2 Interrupt */
635 #define E1000_EIMS_TX_QUEUE3    E1000_EICR_TX_QUEUE3 /* Tx Queue 3 Interrupt */
636 #define E1000_EIMS_TCP_TIMER    E1000_EICR_TCP_TIMER /* TCP Timer */
637 #define E1000_EIMS_OTHER        E1000_EICR_OTHER   /* Interrupt Cause Active */

639 /* Interrupt Cause Set */
640 #define E1000_ICS_LSC           E1000_ICR_LSC       /* Link Status Change */
641 #define E1000_ICS_RXSEQ         E1000_ICR_RXSEQ     /* Rx sequence error */
642 #define E1000_ICS_RXDMT0        E1000_ICR_RXDMT0    /* Rx desc min. threshold */

644 /* Extended Interrupt Cause Set */
645 #define E1000_EICS_RX_QUEUE0    E1000_EICR_RX_QUEUE0 /* Rx Queue 0 Interrupt */
646 #define E1000_EICS_RX_QUEUE1    E1000_EICR_RX_QUEUE1 /* Rx Queue 1 Interrupt */
647 #define E1000_EICS_RX_QUEUE2    E1000_EICR_RX_QUEUE2 /* Rx Queue 2 Interrupt */
648 #define E1000_EICS_RX_QUEUE3    E1000_EICR_RX_QUEUE3 /* Rx Queue 3 Interrupt */
649 #define E1000_EICS_TX_QUEUE0    E1000_EICR_TX_QUEUE0 /* Tx Queue 0 Interrupt */
650 #define E1000_EICS_TX_QUEUE1    E1000_EICR_TX_QUEUE1 /* Tx Queue 1 Interrupt */
651 #define E1000_EICS_TX_QUEUE2    E1000_EICR_TX_QUEUE2 /* Tx Queue 2 Interrupt */
652 #define E1000_EICS_TX_QUEUE3    E1000_EICR_TX_QUEUE3 /* Tx Queue 3 Interrupt */
653 #define E1000_EICS_TCP_TIMER    E1000_EICR_TCP_TIMER /* TCP Timer */
654 #define E1000_EICS_OTHER        E1000_EICR_OTHER   /* Interrupt Cause Active */
```

```
656 #define E1000_EITR_ITR_INT_MASK 0x0000FFFF
657 /* E1000_EITR_CNT_IGNR is only for 82576 and newer */
658 #define E1000_EITR_CNT_IGNR     0x80000000 /* Don't reset counters on write */
659 #define E1000_EITR_INTERVAL     0x00007FFC

661 /* Transmit Descriptor Control */
662 #define E1000_TXDCTL_PTHRESH    0x0000003F /* TXDCTL Prefetch Threshold */
663 #define E1000_TXDCTL_HTHRESH    0x00003F00 /* TXDCTL Host Threshold */
664 #define E1000_TXDCTL_WTHRESH    0x003F0000 /* TXDCTL Writeback Threshold */
665 #define E1000_TXDCTL_GRAN       0x01000000 /* TXDCTL Granularity */
666 #define E1000_TXDCTL_FULL_TX_DESC_WB    0x01010000 /* GRAN=1, WTHRESH=1 */
667 #define E1000_TXDCTL_MAX_TX_DESC_PREFETCH 0x0100001F /* GRAN=1, PTHRESH=31 */
668 /* Enable the counting of descriptors still to be processed. */
669 #define E1000_TXDCTL_COUNT_DESC 0x00400000

671 /* Flow Control Constants */
672 #define FLOW_CONTROL_ADDRESS_LOW        0x00C28001
673 #define FLOW_CONTROL_ADDRESS_HIGH       0x00000100
674 #define FLOW_CONTROL_TYPE               0x8808

676 /* 802.1q VLAN Packet Size */
677 #define VLAN_TAG_SIZE                   4    /* 802.3ac tag (not DMA'd) */
678 #define E1000_VLAN_FILTER_TBL_SIZE      128  /* VLAN Filter Table (4096 bits) */

680 /* Receive Address
681  * Number of high/low register pairs in the RAR. The RAR (Receive Address
682  * Registers) holds the directed and multicast addresses that we monitor.
683  * Technically, we have 16 spots.  However, we reserve one of these spots
684  * (RAR[15]) for our directed address used by controllers with
685  * manageability enabled, allowing us room for 15 multicast addresses.
686  */
687 #define E1000_RAR_ENTRIES       15
688 #define E1000_RAH_AV            0x80000000 /* Receive descriptor valid */
689 #define E1000_RAL_MAC_ADDR_LEN  4
690 #define E1000_RAH_MAC_ADDR_LEN  2
691 #define E1000_RAH_QUEUE_MASK_82575      0x000C0000
692 #define E1000_RAH_POOL_1        0x00040000

694 /* Error Codes */
695 #define E1000_SUCCESS                   0
696 #define E1000_ERR_NVM                   1
697 #define E1000_ERR_PHY                   2
698 #define E1000_ERR_CONFIG                3
699 #define E1000_ERR_PARAM                 4
700 #define E1000_ERR_MAC_INIT              5
701 #define E1000_ERR_PHY_TYPE              6
702 #define E1000_ERR_RESET                 9
703 #define E1000_ERR_MASTER_REQUESTS_PENDING       10
704 #define E1000_ERR_HOST_INTERFACE_COMMAND        11
705 #define E1000_BLK_PHY_RESET             12
706 #define E1000_ERR_SWFW_SYNC             13
707 #define E1000_NOT_IMPLEMENTED           14
708 #define E1000_ERR_MBX                   15
709 #define E1000_ERR_INVALID_ARGUMENT      16
710 #define E1000_ERR_NO_SPACE              17
711 #define E1000_ERR_NVM_PBA_SECTION       18
712 #define E1000_ERR_I2C                   19
713 #define E1000_ERR_INVM_VALUE_NOT_FOUND  20

715 /* Loop limit on how long we wait for auto-negotiation to complete */
716 #define FIBER_LINK_UP_LIMIT             50
717 #define COPPER_LINK_UP_LIMIT            10
718 #define PHY_AUTO_NEG_LIMIT              45
719 #define PHY_FORCE_LIMIT                 20
720 /* Number of 100 microseconds we wait for PCI Express master disable */
```



```
721 #define MASTER_DISABLE_TIMEOUT          800
722 /* Number of milliseconds we wait for PHY configuration done after MAC reset */
723 #define PHY_CFG_TIMEOUT                 100
724 /* Number of 2 milliseconds we wait for acquiring MDIO ownership. */
725 #define MDIO_OWNERSHIP_TIMEOUT          10
726 /* Number of milliseconds for NVM auto read done after MAC reset. */
727 #define AUTO_READ_DONE_TIMEOUT          10

729 /* Flow Control */
730 #define E1000_FCRTH_RTH         0x0000FFF8 /* Mask Bits[15:3] for RTH */
731 #define E1000_FCRTL_RTL         0x0000FFF8 /* Mask Bits[15:3] for RTL */
732 #define E1000_FCRTL_XONE        0x80000000 /* Enable XON frame transmission */

734 /* Transmit Configuration Word */
735 #define E1000_TXCW_FD           0x00000020 /* TXCW full duplex */
736 #define E1000_TXCW_PAUSE        0x00000080 /* TXCW sym pause request */
737 #define E1000_TXCW_ASM_DIR      0x00000100 /* TXCW astm pause direction */
738 #define E1000_TXCW_PAUSE_MASK   0x00000180 /* TXCW pause request mask */
739 #define E1000_TXCW_ANE          0x80000000 /* Auto-neg enable */

741 /* Receive Configuration Word */
742 #define E1000_RXCW_CW           0x0000ffff /* RxConfigWord mask */
743 #define E1000_RXCW_IV           0x08000000 /* Receive config invalid */
744 #define E1000_RXCW_C            0x20000000 /* Receive config */
745 #define E1000_RXCW_SYNCH        0x40000000 /* Receive config synch */

747 #define E1000_TSYNCTXCTL_VALID          0x00000001 /* Tx timestamp valid */
748 #define E1000_TSYNCTXCTL_ENABLED        0x00000010 /* enable Tx timestamping */

750 #define E1000_TSYNCRXCTL_VALID          0x00000001 /* Rx timestamp valid */
751 #define E1000_TSYNCRXCTL_TYPE_MASK      0x0000000E /* Rx type mask */
752 #define E1000_TSYNCRXCTL_TYPE_L2_V2     0x00
753 #define E1000_TSYNCRXCTL_TYPE_L4_V1     0x02
754 #define E1000_TSYNCRXCTL_TYPE_L2_L4_V2  0x04
755 #define E1000_TSYNCRXCTL_TYPE_ALL       0x08
756 #define E1000_TSYNCRXCTL_TYPE_EVENT_V2  0x0A
757 #define E1000_TSYNCRXCTL_ENABLED        0x00000010 /* enable Rx timestamping */
758 #define E1000_TSYNCRXCTL_SYSCFI         0x00000020 /* Sys clock frequency */

760 #define E1000_RXMTRL_PTP_V1_SYNC_MESSAGE        0x00000000
761 #define E1000_RXMTRL_PTP_V1_DELAY_REQ_MESSAGE   0x00010000

763 #define E1000_RXMTRL_PTP_V2_SYNC_MESSAGE        0x00000000
764 #define E1000_RXMTRL_PTP_V2_DELAY_REQ_MESSAGE   0x01000000

766 #define E1000_TSYNCRXCFG_PTP_V1_CTRLT_MASK              0x000000FF
767 #define E1000_TSYNCRXCFG_PTP_V1_SYNC_MESSAGE            0x00
768 #define E1000_TSYNCRXCFG_PTP_V1_DELAY_REQ_MESSAGE       0x01
769 #define E1000_TSYNCRXCFG_PTP_V1_FOLLOWUP_MESSAGE        0x02
770 #define E1000_TSYNCRXCFG_PTP_V1_DELAY_RESP_MESSAGE      0x03
771 #define E1000_TSYNCRXCFG_PTP_V1_MANAGEMENT_MESSAGE      0x04

773 #define E1000_TSYNCRXCFG_PTP_V2_MSGID_MASK              0x00000F00
774 #define E1000_TSYNCRXCFG_PTP_V2_SYNC_MESSAGE            0x0000
775 #define E1000_TSYNCRXCFG_PTP_V2_DELAY_REQ_MESSAGE       0x0100
776 #define E1000_TSYNCRXCFG_PTP_V2_PATH_DELAY_REQ_MESSAGE  0x0200
777 #define E1000_TSYNCRXCFG_PTP_V2_PATH_DELAY_RESP_MESSAGE 0x0300
778 #define E1000_TSYNCRXCFG_PTP_V2_FOLLOWUP_MESSAGE        0x0800
779 #define E1000_TSYNCRXCFG_PTP_V2_DELAY_RESP_MESSAGE      0x0900
780 #define E1000_TSYNCRXCFG_PTP_V2_PATH_DELAY_FOLLOWUP_MESSAGE 0x0A00
781 #define E1000_TSYNCRXCFG_PTP_V2_ANNOUNCE_MESSAGE        0x0B00
782 #define E1000_TSYNCRXCFG_PTP_V2_SIGNALLING_MESSAGE      0x0C00
783 #define E1000_TSYNCRXCFG_PTP_V2_MANAGEMENT_MESSAGE      0x0D00

785 #define E1000_TIMINCA_16NS_SHIFT        24
786 #define E1000_TIMINCA_INCPERIOD_SHIFT   24
```

```
787 #define E1000_TIMINCA_INCVALUE_MASK     0x00FFFFFF

789 #define E1000_TSICR_TXTS                0x00000002
790 #define E1000_TSIM_TXTS                 0x00000002
791 /* TUPLE Filtering Configuration */
792 #define E1000_TTQF_DISABLE_MASK         0xF0008000 /* TTQF Disable Mask */
793 #define E1000_TTQF_QUEUE_ENABLE         0x100   /* TTQF Queue Enable Bit */
794 #define E1000_TTQF_PROTOCOL_MASK        0xFF    /* TTQF Protocol Mask */
795 /* TTQF TCP Bit, shift with E1000_TTQF_PROTOCOL SHIFT */
796 #define E1000_TTQF_PROTOCOL_TCP         0x0
797 /* TTQF UDP Bit, shift with E1000_TTQF_PROTOCOL_SHIFT */
798 #define E1000_TTQF_PROTOCOL_UDP         0x1
799 /* TTQF SCTP Bit, shift with E1000_TTQF_PROTOCOL_SHIFT */
800 #define E1000_TTQF_PROTOCOL_SCTP        0x2
801 #define E1000_TTQF_PROTOCOL_SHIFT       5       /* TTQF Protocol Shift */
802 #define E1000_TTQF_QUEUE_SHIFT          16      /* TTQF Queue Shfit */
803 #define E1000_TTQF_RX_QUEUE_MASK        0x70000 /* TTQF Queue Mask */
804 #define E1000_TTQF_MASK_ENABLE          0x10000000 /* TTQF Mask Enable Bit */
805 #define E1000_IMIR_CLEAR_MASK           0xF001FFFF /* IMIR Reg Clear Mask */
806 #define E1000_IMIR_PORT_BYPASS          0x20000 /* IMIR Port Bypass Bit */
807 #define E1000_IMIR_PRIORITY_SHIFT       29 /* IMIR Priority Shift */
808 #define E1000_IMIREXT_CLEAR_MASK        0x7FFFF /* IMIREXT Reg Clear Mask */

810 #define E1000_MDICNFG_EXT_MDIO          0x80000000 /* MDI ext/int destination */
811 #define E1000_MDICNFG_COM_MDIO          0x40000000 /* MDI shared w/ lan 0 */
812 #define E1000_MDICNFG_PHY_MASK          0x03E00000
813 #define E1000_MDICNFG_PHY_SHIFT         21

815 #define E1000_THSTAT_LOW_EVENT          0x20000000 /* Low thermal threshold */
816 #define E1000_THSTAT_MID_EVENT          0x00200000 /* Mid thermal threshold */
817 #define E1000_THSTAT_HIGH_EVENT         0x00002000 /* High thermal threshold */
818 #define E1000_THSTAT_PWR_DOWN           0x00000001 /* Power Down Event */
819 #define E1000_THSTAT_LINK_THROTTLE      0x00000002 /* Link Spd Throttle Event */

821 /* I350 EEE defines */
822 #define E1000_IPCNFG_EEE_1G_AN          0x00000008 /* IPCNFG EEE Ena 1G AN */
823 #define E1000_IPCNFG_EEE_100M_AN        0x00000004 /* IPCNFG EEE Ena 100M AN */
824 #define E1000_EEER_TX_LPI_EN            0x00010000 /* EEER Tx LPI Enable */
825 #define E1000_EEER_RX_LPI_EN            0x00020000 /* EEER Rx LPI Enable */
826 #define E1000_EEER_LPI_FC               0x00040000 /* EEER Ena on Flow Cntrl */
827 /* EEE status */
828 #define E1000_EEER_EEE_NEG              0x20000000 /* EEE capability nego */
829 #define E1000_EEER_RX_LPI_STATUS        0x40000000 /* Rx in LPI state */
830 #define E1000_EEER_TX_LPI_STATUS        0x80000000 /* Tx in LPI state */
831 #define E1000_EEE_LP_ADV_ADDR_I350      0x040F     /* EEE LP Advertisement */
832 #define E1000_M88E1543_PAGE_ADDR        0x16       /* Page Offset Register */
833 #define E1000_M88E1543_EEE_CTRL_1       0x0
834 #define E1000_M88E1543_EEE_CTRL_1_MS    0x0001     /* EEE Master/Slave */
835 #define E1000_EEE_ADV_DEV_I354          7
836 #define E1000_EEE_ADV_ADDR_I354         60
837 #define E1000_EEE_ADV_100_SUPPORTED     (1 << 1)   /* 100BaseTx EEE Supported */
838 #define E1000_EEE_ADV_1000_SUPPORTED    (1 << 2)   /* 1000BaseT EEE Supported */
839 #define E1000_PCS_STATUS_DEV_I354       3
840 #define E1000_PCS_STATUS_ADDR_I354      1
841 #define E1000_PCS_STATUS_RX_LPI_RCVD    0x0400
842 #define E1000_PCS_STATUS_TX_LPI_RCVD    0x0800

844 #define E1000_EEE_SU_LPI_CLK_STP        0x00800000 /* EEE LPI Clock Stop */
845 /* PCI Express Control */
846 #define E1000_GCR_RXD_NO_SNOOP          0x00000001
847 #define E1000_GCR_RXDSCW_NO_SNOOP       0x00000002
848 #define E1000_GCR_RXDSCR_NO_SNOOP       0x00000004
849 #define E1000_GCR_TXD_NO_SNOOP          0x00000008
850 #define E1000_GCR_TXDSCW_NO_SNOOP       0x00000010
851 #define E1000_GCR_TXDSCR_NO_SNOOP       0x00000020
852 #define E1000_GCR_CMPL_TMOUT_MASK       0x0000F000
```



```
853 #define E1000_GCR_CMPL_TMOUT_10ms       0x00001000
854 #define E1000_GCR_CMPL_TMOUT_RESEND     0x00010000
855 #define E1000_GCR_CAP_VER2              0x00040000

858 #define PCIE_NO_SNOOP_ALL       (E1000_GCR_RXD_NO_SNOOP | \
859                                  E1000_GCR_RXDSCW_NO_SNOOP | \
860                                  E1000_GCR_RXDSCR_NO_SNOOP | \
861                                  E1000_GCR_TXD_NO_SNOOP    | \
862                                  E1000_GCR_TXDSCW_NO_SNOOP | \
863                                  E1000_GCR_TXDSCR_NO_SNOOP)

865 #define E1000_MMDAC_FUNC_DATA   0x4000 /* Data, no post increment */

867 /* mPHY address control and data registers */
868 #define E1000_MPHY_ADDR_CTL             0x0024 /* Address Control Reg */
869 #define E1000_MPHY_ADDR_CTL_OFFSET_MASK 0xFFFF0000
870 #define E1000_MPHY_DATA                 0x0E10 /* Data Register */

872 /* AFE CSR Offset for PCS CLK */
873 #define E1000_MPHY_PCS_CLK_REG_OFFSET   0x0004
874 /* Override for near end digital loopback. */
875 #define E1000_MPHY_PCS_CLK_REG_DIGINELBEN       0x10

877 /* PHY Control Register */
878 #define MII_CR_SPEED_SELECT_MSB 0x0040  /* bits 6,13: 10=1000, 01=100, 00=10 */
879 #define MII_CR_COLL_TEST_ENABLE 0x0080  /* Collision test enable */
880 #define MII_CR_FULL_DUPLEX      0x0100  /* FDX =1, half duplex =0 */
881 #define MII_CR_RESTART_AUTO_NEG 0x0200  /* Restart auto negotiation */
882 #define MII_CR_ISOLATE          0x0400  /* Isolate PHY from MII */
883 #define MII_CR_POWER_DOWN       0x0800  /* Power down */
884 #define MII_CR_AUTO_NEG_EN      0x1000  /* Auto Neg Enable */
885 #define MII_CR_SPEED_SELECT_LSB 0x2000  /* bits 6,13: 10=1000, 01=100, 00=10 */
886 #define MII_CR_LOOPBACK         0x4000  /* 0 = normal, 1 = loopback */
887 #define MII_CR_RESET            0x8000  /* 0 = normal, 1 = PHY reset */
888 #define MII_CR_SPEED_1000       0x0040
889 #define MII_CR_SPEED_100        0x2000
890 #define MII_CR_SPEED_10         0x0000

892 /* PHY Status Register */
893 #define MII_SR_EXTENDED_CAPS    0x0001 /* Extended register capabilities */
894 #define MII_SR_JABBER_DETECT    0x0002 /* Jabber Detected */
895 #define MII_SR_LINK_STATUS      0x0004 /* Link Status 1 = link */
896 #define MII_SR_AUTONEG_CAPS     0x0008 /* Auto Neg Capable */
897 #define MII_SR_REMOTE_FAULT     0x0010 /* Remote Fault Detect */
898 #define MII_SR_AUTONEG_COMPLETE 0x0020 /* Auto Neg Complete */
899 #define MII_SR_PREAMBLE_SUPPRESS 0x0040 /* Preamble may be suppressed */
900 #define MII_SR_EXTENDED_STATUS  0x0100 /* Ext. status info in Reg 0x0F */
901 #define MII_SR_100T2_HD_CAPS    0x0200 /* 100T2 Half Duplex Capable */
902 #define MII_SR_100T2_FD_CAPS    0x0400 /* 100T2 Full Duplex Capable */
903 #define MII_SR_10T_HD_CAPS      0x0800 /* 10T   Half Duplex Capable */
904 #define MII_SR_10T_FD_CAPS      0x1000 /* 10T   Full Duplex Capable */
905 #define MII_SR_100X_HD_CAPS     0x2000 /* 100X  Half Duplex Capable */
906 #define MII_SR_100X_FD_CAPS     0x4000 /* 100X  Full Duplex Capable */
907 #define MII_SR_100T4_CAPS       0x8000 /* 100T4 Capable */

909 /* Autoneg Advertisement Register */
910 #define NWAY_AR_SELECTOR_FIELD  0x0001   /* indicates IEEE 802.3 CSMA/CD */
911 #define NWAY_AR_10T_HD_CAPS     0x0020   /* 10T   Half Duplex Capable */
912 #define NWAY_AR_10T_FD_CAPS     0x0040   /* 10T   Full Duplex Capable */
913 #define NWAY_AR_100TX_HD_CAPS   0x0080   /* 100TX Half Duplex Capable */
914 #define NWAY_AR_100TX_FD_CAPS   0x0100   /* 100TX Full Duplex Capable */
915 #define NWAY_AR_100T4_CAPS      0x0200   /* 100T4 Capable */
916 #define NWAY_AR_PAUSE           0x0400   /* Pause operation desired */
917 #define NWAY_AR_ASM_DIR         0x0800   /* Asymmetric Pause Direction bit */
918 #define NWAY_AR_REMOTE_FAULT    0x2000   /* Remote Fault detected */
```

```
 919 #define NWAY_AR_NEXT_PAGE       0x8000   /* Next Page ability supported */

 921 /* Link Partner Ability Register (Base Page) */
 922 #define NWAY_LPAR_SELECTOR_FIELD        0x0000 /* LP protocol selector field */
 923 #define NWAY_LPAR_10T_HD_CAPS           0x0020 /* LP 10T Half Dplx Capable */
 924 #define NWAY_LPAR_10T_FD_CAPS           0x0040 /* LP 10T Full Dplx Capable */
 925 #define NWAY_LPAR_100TX_HD_CAPS         0x0080 /* LP 100TX Half Dplx Capable */
 926 #define NWAY_LPAR_100TX_FD_CAPS         0x0100 /* LP 100TX Full Dplx Capable */
 927 #define NWAY_LPAR_100T4_CAPS            0x0200 /* LP is 100T4 Capable */
 928 #define NWAY_LPAR_PAUSE                 0x0400 /* LP Pause operation desired */
 929 #define NWAY_LPAR_ASM_DIR               0x0800 /* LP Asym Pause Direction bit */
 930 #define NWAY_LPAR_REMOTE_FAULT          0x2000 /* LP detected Remote Fault */
 931 #define NWAY_LPAR_ACKNOWLEDGE           0x4000 /* LP rx'd link code word */
 932 #define NWAY_LPAR_NEXT_PAGE             0x8000 /* Next Page ability supported */

 934 /* Autoneg Expansion Register */
 935 #define NWAY_ER_LP_NWAY_CAPS            0x0001 /* LP has Auto Neg Capability */
 936 #define NWAY_ER_PAGE_RXD                0x0002 /* LP 10T Half Dplx Capable */
 937 #define NWAY_ER_NEXT_PAGE_CAPS          0x0004 /* LP 10T Full Dplx Capable */
 938 #define NWAY_ER_LP_NEXT_PAGE_CAPS       0x0008 /* LP 100TX Half Dplx Capable */
 939 #define NWAY_ER_PAR_DETECT_FAULT        0x0010 /* LP 100TX Full Dplx Capable */

 941 /* 1000BASE-T Control Register */
 942 #define CR_1000T_ASYM_PAUSE     0x0080 /* Advertise asymmetric pause bit */
 943 #define CR_1000T_HD_CAPS        0x0100 /* Advertise 1000T HD capability */
 944 #define CR_1000T_FD_CAPS        0x0200 /* Advertise 1000T FD capability  */
 945 /* 1=Repeater/switch device port 0=DTE device */
 946 #define CR_1000T_REPEATER_DTE   0x0400
 947 /* 1=Configure PHY as Master 0=Configure PHY as Slave */
 948 #define CR_1000T_MS_VALUE       0x0800
 949 /* 1=Master/Slave manual config value 0=Automatic Master/Slave config */
 950 #define CR_1000T_MS_ENABLE      0x1000
 951 #define CR_1000T_TEST_MODE_NORMAL 0x0000 /* Normal Operation */
 952 #define CR_1000T_TEST_MODE_1    0x2000 /* Transmit Waveform test */
 953 #define CR_1000T_TEST_MODE_2    0x4000 /* Master Transmit Jitter test */
 954 #define CR_1000T_TEST_MODE_3    0x6000 /* Slave Transmit Jitter test */
 955 #define CR_1000T_TEST_MODE_4    0x8000 /* Transmitter Distortion test */

 957 /* 1000BASE-T Status Register */
 958 #define SR_1000T_IDLE_ERROR_CNT         0x00FF /* Num idle err since last rd */
 959 #define SR_1000T_ASYM_PAUSE_DIR         0x0100 /* LP asym pause direction bit */
 960 #define SR_1000T_LP_HD_CAPS             0x0400 /* LP is 1000T HD capable */
 961 #define SR_1000T_LP_FD_CAPS             0x0800 /* LP is 1000T FD capable */
 962 #define SR_1000T_REMOTE_RX_STATUS       0x1000 /* Remote receiver OK */
 963 #define SR_1000T_LOCAL_RX_STATUS        0x2000 /* Local receiver OK */
 964 #define SR_1000T_MS_CONFIG_RES          0x4000 /* 1=Local Tx Master, 0=Slave */
 965 #define SR_1000T_MS_CONFIG_FAULT        0x8000 /* Master/Slave config fault */

 967 #define SR_1000T_PHY_EXCESSIVE_IDLE_ERR_COUNT   5

 969 /* PHY 1000 MII Register/Bit Definitions */
 970 /* PHY Registers defined by IEEE */
 971 #define PHY_CONTROL             0x00 /* Control Register */
 972 #define PHY_STATUS              0x01 /* Status Register */
 973 #define PHY_ID1                 0x02 /* Phy Id Reg (word 1) */
 974 #define PHY_ID2                 0x03 /* Phy Id Reg (word 2) */
 975 #define PHY_AUTONEG_ADV         0x04 /* Autoneg Advertisement */
 976 #define PHY_LP_ABILITY          0x05 /* Link Partner Ability (Base Page) */
 977 #define PHY_AUTONEG_EXP         0x06 /* Autoneg Expansion Reg */
 978 #define PHY_NEXT_PAGE_TX        0x07 /* Next Page Tx */
 979 #define PHY_LP_NEXT_PAGE        0x08 /* Link Partner Next Page */
 980 #define PHY_1000T_CTRL          0x09 /* 1000Base-T Control Reg */
 981 #define PHY_1000T_STATUS        0x0A /* 1000Base-T Status Reg */
 982 #define PHY_EXT_STATUS          0x0F /* Extended Status Reg */

 984 #define PHY_CONTROL_LB          0x4000 /* PHY Loopback bit */
```



```
 986 /* NVM Control */
 987 #define E1000_EECD_SK           0x00000001 /* NVM Clock */
 988 #define E1000_EECD_CS           0x00000002 /* NVM Chip Select */
 989 #define E1000_EECD_DI           0x00000004 /* NVM Data In */
 990 #define E1000_EECD_DO           0x00000008 /* NVM Data Out */
 991 #define E1000_EECD_REQ          0x00000040 /* NVM Access Request */
 992 #define E1000_EECD_GNT          0x00000080 /* NVM Access Grant */
 993 #define E1000_EECD_PRES         0x00000100 /* NVM Present */
 994 #define E1000_EECD_SIZE         0x00000200 /* NVM Size (0=64 word 1=256 word) */
 995 #define E1000_EECD_BLOCKED      0x00008000 /* Bit banging access blocked flag */
 996 #define E1000_EECD_ABORT        0x00010000 /* NVM operation aborted flag */
 997 #define E1000_EECD_TIMEOUT      0x00020000 /* NVM read operation timeout flag */
 998 #define E1000_EECD_ERROR_CLR    0x00040000 /* NVM error status clear bit */
 999 /* NVM Addressing bits based on type 0=small, 1=large */
1000 #define E1000_EECD_ADDR_BITS    0x00000400
1001 #define E1000_EECD_TYPE         0x00002000 /* NVM Type (1-SPI, 0-Microwire) */
1002 #ifndef E1000_NVM_GRANT_ATTEMPTS
1003 #define E1000_NVM_GRANT_ATTEMPTS        1000 /* NVM # attempts to gain grant */
1004 #endif
1005 #define E1000_EECD_AUTO_RD              0x00000200  /* NVM Auto Read done */
1006 #define E1000_EECD_SIZE_EX_MASK         0x00007800  /* NVM Size */
1007 #define E1000_EECD_SIZE_EX_SHIFT        11
1008 #define E1000_EECD_FLUPD                0x00080000 /* Update FLASH */
1009 #define E1000_EECD_AUPDEN               0x00100000 /* Ena Auto FLASH update */
1010 #define E1000_EECD_SEC1VAL              0x00400000 /* Sector One Valid */
1011 #define E1000_EECD_SEC1VAL_VALID_MASK   (E1000_EECD_AUTO_RD | E1000_EECD_PRES)
1012 #define E1000_EECD_FLUPD_I210           0x00800000 /* Update FLASH */
1013 #define E1000_EECD_FLUDONE_I210         0x04000000 /* Update FLASH done */
1014 #define E1000_EECD_FLASH_DETECTED_I210  0x00080000 /* FLASH detected */
1015 #define E1000_EECD_SEC1VAL_I210         0x02000000 /* Sector One Valid */
1016 #define E1000_FLUDONE_ATTEMPTS          20000
1017 #define E1000_EERD_EEWR_MAX_COUNT       512 /* buffered EEPROM words rw */
1018 #define E1000_I210_FIFO_SEL_RX          0x00
1019 #define E1000_I210_FIFO_SEL_TX_QAV(_i)  (0x02 + (_i))
1020 #define E1000_I210_FIFO_SEL_TX_LEGACY   E1000_I210_FIFO_SEL_TX_QAV(0)
1021 #define E1000_I210_FIFO_SEL_BMC2OS_TX   0x06
1022 #define E1000_I210_FIFO_SEL_BMC2OS_RX   0x01

1024 #define E1000_I210_FLASH_SECTOR_SIZE    0x1000 /* 4KB FLASH sector unit size */
1025 /* Secure FLASH mode requires removing MSb */
1026 #define E1000_I210_FW_PTR_MASK          0x7FFF
1027 /* Firmware code revision field word offset*/
1028 #define E1000_I210_FW_VER_OFFSET        328

1030 #define E1000_NVM_RW_REG_DATA   16  /* Offset to data in NVM read/write regs */
1031 #define E1000_NVM_RW_REG_DONE   2   /* Offset to READ/WRITE done bit */
1032 #define E1000_NVM_RW_REG_START  1   /* Start operation */
1033 #define E1000_NVM_RW_ADDR_SHIFT 2   /* Shift to the address bits */
1034 #define E1000_NVM_POLL_WRITE    1   /* Flag for polling for write complete */
1035 #define E1000_NVM_POLL_READ     0   /* Flag for polling for read complete */
1036 #define E1000_FLASH_UPDATES     2000

1038 /* NVM Word Offsets */
1039 #define NVM_COMPAT                      0x0003
1040 #define NVM_ID_LED_SETTINGS             0x0004
1041 #define NVM_VERSION                     0x0005
1042 #define NVM_SERDES_AMPLITUDE            0x0006 /* SERDES output amplitude */
1043 #define NVM_PHY_CLASS_WORD              0x0007
1044 #define E1000_I210_NVM_FW_MODULE_PTR    0x0010
1045 #define E1000_I350_NVM_FW_MODULE_PTR    0x0051
1046 #define NVM_FUTURE_INIT_WORD1           0x0019
1047 #define NVM_MAC_ADDR                    0x0000
1048 #define NVM_SUB_DEV_ID                  0x000B
1049 #define NVM_SUB_VEN_ID                  0x000C
1050 #define NVM_DEV_ID                      0x000D
```

```
1051 #define NVM_VEN_ID                      0x000E
1052 #define NVM_INIT_CTRL_2                 0x000F
1053 #define NVM_INIT_CTRL_4                 0x0013
1054 #define NVM_LED_1_CFG                   0x001C
1055 #define NVM_LED_0_2_CFG                 0x001F

1057 #define NVM_COMPAT_VALID_CSUM           0x0001
1058 #define NVM_FUTURE_INIT_WORD1_VALID_CSUM        0x0040

1060 #define NVM_INIT_CONTROL2_REG           0x000F
1061 #define NVM_INIT_CONTROL3_PORT_B        0x0014
1062 #define NVM_INIT_3GIO_3                 0x001A
1063 #define NVM_SWDEF_PINS_CTRL_PORT_0      0x0020
1064 #define NVM_INIT_CONTROL3_PORT_A        0x0024
1065 #define NVM_CFG                         0x0012
1066 #define NVM_ALT_MAC_ADDR_PTR            0x0037
1067 #define NVM_CHECKSUM_REG                0x003F
1068 #define NVM_COMPATIBILITY_REG_3         0x0003
1069 #define NVM_COMPATIBILITY_BIT_MASK      0x8000

1071 #define E1000_NVM_CFG_DONE_PORT_0       0x040000 /* MNG config cycle done */
1072 #define E1000_NVM_CFG_DONE_PORT_1       0x080000 /* ...for second port */
1073 #define E1000_NVM_CFG_DONE_PORT_2       0x100000 /* ...for third port */
1074 #define E1000_NVM_CFG_DONE_PORT_3       0x200000 /* ...for fourth port */

1076 #define NVM_82580_LAN_FUNC_OFFSET(a)    ((a) ? (0x40 + (0x40 * (a))) : 0)

1078 /* Mask bits for fields in Word 0x24 of the NVM */
1079 #define NVM_WORD24_COM_MDIO             0x0008 /* MDIO interface shared */
1080 #define NVM_WORD24_EXT_MDIO             0x0004 /* MDIO accesses routed extrnl */
1081 /* Offset of Link Mode bits for 82575/82576 */
1082 #define NVM_WORD24_LNK_MODE_OFFSET      8
1083 /* Offset of Link Mode bits for 82580 up */
1084 #define NVM_WORD24_82580_LNK_MODE_OFFSET        4


1087 /* Mask bits for fields in Word 0x0f of the NVM */
1088 #define NVM_WORD0F_PAUSE_MASK           0x3000
1089 #define NVM_WORD0F_PAUSE                0x1000
1090 #define NVM_WORD0F_ASM_DIR              0x2000
1091 #define NVM_WORD0F_SWPDIO_EXT_MASK      0x00F0

1093 /* Mask bits for fields in Word 0x1a of the NVM */
1094 #define NVM_WORD1A_ASPM_MASK            0x000C

1096 /* Mask bits for fields in Word 0x03 of the EEPROM */
1097 #define NVM_COMPAT_LOM                  0x0800

1099 /* length of string needed to store PBA number */
1100 #define E1000_PBANUM_LENGTH             11

1102 /* For checksumming, the sum of all words in the NVM should equal 0xBABA. */
1103 #define NVM_SUM                         0xBABA

1105 /* OEM NVM Offsets */
1106 #define NVM_OEM_OFFSET_0                6
1107 #define NVM_OEM_OFFSET_1                7

1109 /* PBA (printed board assembly) number words */
1110 #define NVM_PBA_OFFSET_0                8
1111 #define NVM_PBA_OFFSET_1                9
1112 #define NVM_PBA_PTR_GUARD               0xFAFA
1113 #define NVM_RESERVED_WORD               0xFFFF
1114 #define NVM_PHY_CLASS_A                 0x8000
1115 #define NVM_SERDES_AMPLITUDE_MASK       0x000F
1116 #define NVM_SIZE_MASK                   0x1C00
```



```
1117 #define NVM_SIZE_SHIFT                  10
1118 #define NVM_WORD_SIZE_BASE_SHIFT        6
1119 #define NVM_SWDPIO_EXT_SHIFT            4

1121 /* NVM Commands - Microwire */
1122 #define NVM_READ_OPCODE_MICROWIRE       0x6  /* NVM read opcode */
1123 #define NVM_WRITE_OPCODE_MICROWIRE      0x5  /* NVM write opcode */
1124 #define NVM_ERASE_OPCODE_MICROWIRE      0x7  /* NVM erase opcode */
1125 #define NVM_EWEN_OPCODE_MICROWIRE       0x13 /* NVM erase/write enable */
1126 #define NVM_EWDS_OPCODE_MICROWIRE       0x10 /* NVM erase/write disable */

1128 /* NVM Commands - SPI */
1129 #define NVM_MAX_RETRY_SPI       5000 /* Max wait of 5ms, for RDY signal */
1130 #define NVM_READ_OPCODE_SPI     0x03 /* NVM read opcode */
1131 #define NVM_WRITE_OPCODE_SPI    0x02 /* NVM write opcode */
1132 #define NVM_A8_OPCODE_SPI       0x08 /* opcode bit-3 = address bit-8 */
1133 #define NVM_WREN_OPCODE_SPI     0x06 /* NVM set Write Enable latch */
1134 #define NVM_RDSR_OPCODE_SPI     0x05 /* NVM read Status register */

1136 /* SPI NVM Status Register */
1137 #define NVM_STATUS_RDY_SPI      0x01

1139 /* Word definitions for ID LED Settings */
1140 #define ID_LED_RESERVED_0000    0x0000
1141 #define ID_LED_RESERVED_FFFF    0xFFFF
1142 #define ID_LED_DEFAULT          ((ID_LED_OFF1_ON2  << 12) | \
1143                                  (ID_LED_OFF1_OFF2 <<  8) | \
1144                                  (ID_LED_DEF1_DEF2 <<  4) | \
1145                                  (ID_LED_DEF1_DEF2))
1146 #define ID_LED_DEF1_DEF2        0x1
1147 #define ID_LED_DEF1_ON2         0x2
1148 #define ID_LED_DEF1_OFF2        0x3
1149 #define ID_LED_ON1_DEF2         0x4
1150 #define ID_LED_ON1_ON2          0x5
1151 #define ID_LED_ON1_OFF2         0x6
1152 #define ID_LED_OFF1_DEF2        0x7
1153 #define ID_LED_OFF1_ON2         0x8
1154 #define ID_LED_OFF1_OFF2        0x9

1156 #define IGP_ACTIVITY_LED_MASK   0xFFFFF0FF
1157 #define IGP_ACTIVITY_LED_ENABLE 0x0300
1158 #define IGP_LED3_MODE           0x07000000

1160 /* PCI/PCI-X/PCI-EX Config space */
1161 #define PCIX_COMMAND_REGISTER           0xE6
1162 #define PCIX_STATUS_REGISTER_LO         0xE8
1163 #define PCIX_STATUS_REGISTER_HI         0xEA
1164 #define PCI_HEADER_TYPE_REGISTER        0x0E
1165 #define PCIE_LINK_STATUS                0x12
1166 #define PCIE_DEVICE_CONTROL2            0x28

1168 #define PCIX_COMMAND_MMRBC_MASK         0x000C
1169 #define PCIX_COMMAND_MMRBC_SHIFT        0x2
1170 #define PCIX_STATUS_HI_MMRBC_MASK       0x0060
1171 #define PCIX_STATUS_HI_MMRBC_SHIFT      0x5
1172 #define PCIX_STATUS_HI_MMRBC_4K         0x3
1173 #define PCIX_STATUS_HI_MMRBC_2K         0x2
1174 #define PCIX_STATUS_LO_FUNC_MASK        0x7
1175 #define PCI_HEADER_TYPE_MULTIFUNC       0x80
1176 #define PCIE_LINK_WIDTH_MASK            0x3F0
1177 #define PCIE_LINK_WIDTH_SHIFT           4
1178 #define PCIE_LINK_SPEED_MASK            0x0F
1179 #define PCIE_LINK_SPEED_2500            0x01
1180 #define PCIE_LINK_SPEED_5000            0x02
1181 #define PCIE_DEVICE_CONTROL2_16ms       0x0005
```

```
#ifndef ETH_ADDR_LEN
#define ETH_ADDR_LEN                    6
#endif

#define PHY_REVISION_MASK               0xFFFFFFF0
#define MAX_PHY_REG_ADDRESS             0x1F  /* 5 bit address bus (0-0x1F) */
#define MAX_PHY_MULTI_PAGE_REG          0xF

/* Bit definitions for valid PHY IDs.
 * I = Integrated
 * E = External
 */
#define M88E1000_E_PHY_ID       0x01410C50
#define M88E1000_I_PHY_ID       0x01410C30
#define M88E1011_I_PHY_ID       0x01410C20
#define IGP01E1000_I_PHY_ID     0x02A80380
#define M88E1111_I_PHY_ID       0x01410CC0
#define M88E1543_E_PHY_ID       0x01410EA0
#define M88E1512_E_PHY_ID       0x01410DD0
#define M88E1112_E_PHY_ID       0x01410C90
#define I347AT4_E_PHY_ID        0x01410DC0
#define M88E1340M_E_PHY_ID      0x01410DF0
#define GG82563_E_PHY_ID        0x01410CA0
#define IGP03E1000_E_PHY_ID     0x02A80390
#define IFE_E_PHY_ID            0x02A80330
#define IFE_PLUS_E_PHY_ID       0x02A80320
#define IFE_C_E_PHY_ID          0x02A80310
#define BME1000_E_PHY_ID        0x01410CB0
#define BME1000_E_PHY_ID_R2     0x01410CB1
#define I82577_E_PHY_ID         0x01540050
#define I82578_E_PHY_ID         0x004DD040
#define I82579_E_PHY_ID         0x01540090
#define I217_E_PHY_ID           0x015400A0
#define I82580_I_PHY_ID         0x015403A0
#define I350_I_PHY_ID           0x015403B0
#define I210_I_PHY_ID           0x01410C00
#define IGP04E1000_E_PHY_ID     0x02A80391
#define M88_VENDOR              0x0141

/* M88E1000 Specific Registers */
#define M88E1000_PHY_SPEC_CTRL          0x10  /* PHY Specific Control Reg */
#define M88E1000_PHY_SPEC_STATUS        0x11  /* PHY Specific Status Reg */
#define M88E1000_EXT_PHY_SPEC_CTRL      0x14  /* Extended PHY Specific Cntrl */
#define M88E1000_RX_ERR_CNTR            0x15  /* Receive Error Counter */

#define M88E1000_PHY_EXT_CTRL           0x1A  /* PHY extend control register */
#define M88E1000_PHY_PAGE_SELECT        0x1D  /* Reg 29 for pg number setting */
#define M88E1000_PHY_GEN_CONTROL        0x1E  /* meaning depends on reg 29 */
#define M88E1000_PHY_VCO_REG_BIT8       0x100 /* Bits 8 & 11 are adjusted for */
#define M88E1000_PHY_VCO_REG_BIT11      0x800 /* improved BER performance */

/* M88E1000 PHY Specific Control Register */
#define M88E1000_PSCR_POLARITY_REVERSAL 0x0002 /* 1=Polarity Reverse enabled */
/* MDI Crossover Mode bits 6:5 Manual MDI configuration */
#define M88E1000_PSCR_MDI_MANUAL_MODE   0x0000
#define M88E1000_PSCR_MDIX_MANUAL_MODE  0x0020  /* Manual MDIX configuration */
/* 1000BASE-T: Auto crossover, 100BASE-TX/10BASE-T: MDI Mode */
#define M88E1000_PSCR_AUTO_X_1000T      0x0040
/* Auto crossover enabled all speeds */
#define M88E1000_PSCR_AUTO_X_MODE       0x0060
#define M88E1000_PSCR_ASSERT_CRS_ON_TX  0x0800 /* 1=Assert CRS on Tx */

/* M88E1000 PHY Specific Status Register */
#define M88E1000_PSSR_REV_POLARITY      0x0002 /* 1=Polarity reversed */
#define M88E1000_PSSR_DOWNSHIFT         0x0020 /* 1=Downshifted */
#define M88E1000_PSSR_MDIX              0x0040 /* 1=MDIX; 0=MDI */
```



```
/* 0 = <50M
 * 1 = 50-80M
 * 2 = 80-110M
 * 3 = 110-140M
 * 4 = >140M
 */
#define M88E1000_PSSR_CABLE_LENGTH      0x0380
#define M88E1000_PSSR_LINK              0x0400 /* 1=Link up, 0=Link down */
#define M88E1000_PSSR_SPD_DPLX_RESOLVED 0x0800 /* 1=Speed & Duplex resolved */
#define M88E1000_PSSR_DPLX              0x2000 /* 1=Duplex 0=Half Duplex */
#define M88E1000_PSSR_SPEED             0xC000 /* Speed, bits 14:15 */
#define M88E1000_PSSR_100MBS            0x4000 /* 01=100Mbs */
#define M88E1000_PSSR_1000MBS           0x8000 /* 10=1000Mbs */

#define M88E1000_PSSR_CABLE_LENGTH_SHIFT        7

/* Number of times we will attempt to autonegotiate before downshifting if we
 * are the master
 */
#define M88E1000_EPSCR_MASTER_DOWNSHIFT_MASK    0x0C00
#define M88E1000_EPSCR_MASTER_DOWNSHIFT_1X      0x0000
/* Number of times we will attempt to autonegotiate before downshifting if we
 * are the slave
 */
#define M88E1000_EPSCR_SLAVE_DOWNSHIFT_MASK     0x0300
#define M88E1000_EPSCR_SLAVE_DOWNSHIFT_1X       0x0100
#define M88E1000_EPSCR_TX_CLK_25        0x0070 /* 25  MHz TX_CLK */

/* Intel I347AT4 Registers */
#define I347AT4_PCDL            0x10 /* PHY Cable Diagnostics Length */
#define I347AT4_PCDC            0x15 /* PHY Cable Diagnostics Control */
#define I347AT4_PAGE_SELECT     0x16

/* I347AT4 Extended PHY Specific Control Register */

/* Number of times we will attempt to autonegotiate before downshifting if we
 * are the master
 */
#define I347AT4_PSCR_DOWNSHIFT_ENABLE   0x0800
#define I347AT4_PSCR_DOWNSHIFT_MASK     0x7000
#define I347AT4_PSCR_DOWNSHIFT_1X       0x0000
#define I347AT4_PSCR_DOWNSHIFT_2X       0x1000
#define I347AT4_PSCR_DOWNSHIFT_3X       0x2000
#define I347AT4_PSCR_DOWNSHIFT_4X       0x3000
#define I347AT4_PSCR_DOWNSHIFT_5X       0x4000
#define I347AT4_PSCR_DOWNSHIFT_6X       0x5000
#define I347AT4_PSCR_DOWNSHIFT_7X       0x6000
#define I347AT4_PSCR_DOWNSHIFT_8X       0x7000

/* I347AT4 PHY Cable Diagnostics Control */
#define I347AT4_PCDC_CABLE_LENGTH_UNIT  0x0400 /* 0=cm 1=meters */

/* M88E1112 only registers */
#define M88E1112_VCT_DSP_DISTANCE       0x001A

/* M88EC018 Rev 2 specific DownShift settings */
#define M88EC018_EPSCR_DOWNSHIFT_COUNTER_MASK   0x0E00
#define M88EC018_EPSCR_DOWNSHIFT_COUNTER_5X     0x0800

#define I82578_EPSCR_DOWNSHIFT_ENABLE           0x0020
#define I82578_EPSCR_DOWNSHIFT_COUNTER_MASK     0x001C

/* BME1000 PHY Specific Control Register */
#define BME1000_PSCR_ENABLE_DOWNSHIFT   0x0800 /* 1 = enable downshift */

/* Bits...
```

```
1315  * 15-5: page
1316  * 4-0: register offset
1317  */
1318 #define GG82563_PAGE_SHIFT      5
1319 #define GG82563_REG(page, reg)  \
1320         (((page) << GG82563_PAGE_SHIFT) | ((reg) & MAX_PHY_REG_ADDRESS))
1321 #define GG82563_MIN_ALT_REG     30

1323 /* GG82563 Specific Registers */
1324 #define GG82563_PHY_SPEC_CTRL           GG82563_REG(0, 16) /* PHY Spec Cntrl */
1325 #define GG82563_PHY_PAGE_SELECT         GG82563_REG(0, 22) /* Page Select */
1326 #define GG82563_PHY_SPEC_CTRL_2         GG82563_REG(0, 26) /* PHY Spec Cntrl2 */
1327 #define GG82563_PHY_PAGE_SELECT_ALT     GG82563_REG(0, 29) /* Alt Page Select */

1329 /* MAC Specific Control Register */
1330 #define GG82563_PHY_MAC_SPEC_CTRL       GG82563_REG(2, 21)

1332 #define GG82563_PHY_DSP_DISTANCE        GG82563_REG(5, 26) /* DSP Distance */

1334 /* Page 193 - Port Control Registers */
1335 /* Kumeran Mode Control */
1336 #define GG82563_PHY_KMRN_MODE_CTRL      GG82563_REG(193, 16)
1337 #define GG82563_PHY_PWR_MGMT_CTRL       GG82563_REG(193, 20) /* Pwr Mgt Ctrl */

1339 /* Page 194 - KMRN Registers */
1340 #define GG82563_PHY_INBAND_CTRL         GG82563_REG(194, 18) /* Inband Ctrl */

1342 /* MDI Control */
1343 #define E1000_MDIC_REG_MASK     0x001F0000
1344 #define E1000_MDIC_REG_SHIFT    16
1345 #define E1000_MDIC_PHY_MASK     0x03E00000
1346 #define E1000_MDIC_PHY_SHIFT    21
1347 #define E1000_MDIC_OP_WRITE     0x04000000
1348 #define E1000_MDIC_OP_READ      0x08000000
1349 #define E1000_MDIC_READY        0x10000000
1350 #define E1000_MDIC_ERROR        0x40000000
1351 #define E1000_MDIC_DEST         0x80000000

1353 /* SerDes Control */
1354 #define E1000_GEN_CTL_READY             0x80000000
1355 #define E1000_GEN_CTL_ADDRESS_SHIFT     8
1356 #define E1000_GEN_POLL_TIMEOUT          640

1358 /* LinkSec register fields */
1359 #define E1000_LSECTXCAP_SUM_MASK        0x00FF0000
1360 #define E1000_LSECTXCAP_SUM_SHIFT       16
1361 #define E1000_LSECRXCAP_SUM_MASK        0x00FF0000
1362 #define E1000_LSECRXCAP_SUM_SHIFT       16

1364 #define E1000_LSECTXCTRL_EN_MASK        0x00000003
1365 #define E1000_LSECTXCTRL_DISABLE        0x0
1366 #define E1000_LSECTXCTRL_AUTH           0x1
1367 #define E1000_LSECTXCTRL_AUTH_ENCRYPT   0x2
1368 #define E1000_LSECTXCTRL_AISCI          0x00000020
1369 #define E1000_LSECTXCTRL_PNTHRSH_MASK   0xFFFFFF00
1370 #define E1000_LSECTXCTRL_RSV_MASK       0x000000D8

1372 #define E1000_LSECRXCTRL_EN_MASK        0x0000000C
1373 #define E1000_LSECRXCTRL_EN_SHIFT       2
1374 #define E1000_LSECRXCTRL_DISABLE        0x0
1375 #define E1000_LSECRXCTRL_CHECK          0x1
1376 #define E1000_LSECRXCTRL_STRICT         0x2
1377 #define E1000_LSECRXCTRL_DROP           0x3
1378 #define E1000_LSECRXCTRL_PLSH           0x00000040
1379 #define E1000_LSECRXCTRL_RP             0x00000080
1380 #define E1000_LSECRXCTRL_RSV_MASK       0xFFFFFF33
```



```
1382 /* Tx Rate-Scheduler Config fields */
1383 #define E1000_RTTBCNRC_RS_ENA           0x80000000
1384 #define E1000_RTTBCNRC_RF_DEC_MASK      0x00003FFF
1385 #define E1000_RTTBCNRC_RF_INT_SHIFT     14
1386 #define E1000_RTTBCNRC_RF_INT_MASK      \
1387         (E1000_RTTBCNRC_RF_DEC_MASK << E1000_RTTBCNRC_RF_INT_SHIFT)

1389 /* DMA Coalescing register fields */
1390 /* DMA Coalescing Watchdog Timer */
1391 #define E1000_DMACR_DMACWT_MASK         0x00003FFF
1392 /* DMA Coalescing Rx Threshold */
1393 #define E1000_DMACR_DMACTHR_MASK        0x00FF0000
1394 #define E1000_DMACR_DMACTHR_SHIFT       16
1395 /* Lx when no PCIe transactions */
1396 #define E1000_DMACR_DMAC_LX_MASK        0x30000000
1397 #define E1000_DMACR_DMAC_LX_SHIFT       28
1398 #define E1000_DMACR_DMAC_EN             0x80000000 /* Enable DMA Coalescing */
1399 /* DMA Coalescing BMC-to-OS Watchdog Enable */
1400 #define E1000_DMACR_DC_BMC2OSW_EN       0x00008000

1402 /* DMA Coalescing Transmit Threshold */
1403 #define E1000_DMCTXTH_DMCTTHR_MASK      0x00000FFF

1405 #define E1000_DMCTLX_TTLX_MASK          0x00000FFF /* Time to LX request */

1407 /* Rx Traffic Rate Threshold */
1408 #define E1000_DMCRTRH_UTRESH_MASK       0x0007FFFF
1409 /* Rx packet rate in current window */
1410 #define E1000_DMCRTRH_LRPRCW            0x80000000

1412 /* DMA Coal Rx Traffic Current Count */
1413 #define E1000_DMCCNT_CCOUNT_MASK        0x01FFFFFF

1415 /* Flow ctrl Rx Threshold High val */
1416 #define E1000_FCRTC_RTH_COAL_MASK       0x0003FFF0
1417 #define E1000_FCRTC_RTH_COAL_SHIFT      4
1418 /* Lx power decision based on DMA coal */
1419 #define E1000_PCIEMISC_LX_DECISION      0x00000080

1421 #define E1000_RXPBS_CFG_TS_EN           0x80000000 /* Timestamp in Rx buffer */
1422 #define E1000_RXPBS_SIZE_I210_MASK      0x0000003F /* Rx packet buffer size */
1423 #define E1000_TXPB0S_SIZE_I210_MASK     0x0000003F /* Tx packet buffer 0 size */
1424 #define E1000_DOBFFCTL_OBFFTHR_MASK     0x000000FF /* OBFF threshold */
1425 #define E1000_DOBFFCTL_EXIT_ACT_MASK    0x01000000 /* Exit active CB */

1427 /* Proxy Filter Control */
1428 #define E1000_PROXYFC_D0                0x00000001 /* Enable offload in D0 */
1429 #define E1000_PROXYFC_EX                0x00000004 /* Directed exact proxy */
1430 #define E1000_PROXYFC_MC                0x00000008 /* Directed MC Proxy */
1431 #define E1000_PROXYFC_BC                0x00000010 /* Broadcast Proxy Enable */
1432 #define E1000_PROXYFC_ARP_DIRECTED      0x00000020 /* Directed ARP Proxy Ena */
1433 #define E1000_PROXYFC_IPV4              0x00000040 /* Directed IPv4 Enable */
1434 #define E1000_PROXYFC_IPV6              0x00000080 /* Directed IPv6 Enable */
1435 #define E1000_PROXYFC_NS                0x00000200 /* IPv6 Neighbor Solicitation
1436 #define E1000_PROXYFC_ARP               0x00000800 /* ARP Request Proxy Ena */
1437 /* Proxy Status */
1438 #define E1000_PROXYS_CLEAR              0xFFFFFFFF /* Clear */

1440 /* Firmware Status */
1441 #define E1000_FWSTS_FWRI                0x80000000 /* FW Reset Indication */
1442 /* VF Control */
1443 #define E1000_VTCTRL_RST                0x04000000 /* Reset VF */

1445 #define E1000_STATUS_LAN_ID_MASK        0x00000000C /* Mask for Lan ID field */
1446 /* Lan ID bit field offset in status register */
```

```
1447 #define E1000_STATUS_LAN_ID_OFFSET      2
1448 #define E1000_VFTA_ENTRIES              128
1449 #endif /* _E1000_DEFINES_H_ */
```

```
**********************************************************
   26002 Wed Feb 26 09:49:37 2014
new/usr/src/uts/common/io/e1000api/e1000_hw.h
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
```

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

  Copyright (c) 2001-2013, 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 _E1000_HW_H_
#define _E1000_HW_H_

#ifdef __cplusplus
extern "C" {
#endif

#include "e1000_osdep.h"
#include "e1000_regs.h"
#include "e1000_defines.h"

struct e1000_hw;

#define E1000_DEV_ID_82542                      0x1000
#define E1000_DEV_ID_82543GC_FIBER              0x1001
#define E1000_DEV_ID_82543GC_COPPER             0x1004
#define E1000_DEV_ID_82544EI_COPPER             0x1008
#define E1000_DEV_ID_82544EI_FIBER              0x1009
#define E1000_DEV_ID_82544GC_COPPER             0x100C
#define E1000_DEV_ID_82544GC_LOM                0x100D
#define E1000_DEV_ID_82540EM                    0x100E
#define E1000_DEV_ID_82540EM_LOM                0x1015
#define E1000_DEV_ID_82540EP_LOM                0x1016
#define E1000_DEV_ID_82540EP                    0x1017
#define E1000_DEV_ID_82540EP_LP                 0x101E
#define E1000_DEV_ID_82545EM_COPPER             0x100F
```



```
#define E1000_DEV_ID_82545EM_FIBER              0x1011
#define E1000_DEV_ID_82545GM_COPPER             0x1026
#define E1000_DEV_ID_82545GM_FIBER              0x1027
#define E1000_DEV_ID_82545GM_SERDES             0x1028
#define E1000_DEV_ID_82546EB_COPPER             0x1010
#define E1000_DEV_ID_82546EB_FIBER              0x1012
#define E1000_DEV_ID_82546EB_QUAD_COPPER        0x101D
#define E1000_DEV_ID_82546GB_COPPER             0x1079
#define E1000_DEV_ID_82546GB_FIBER              0x107A
#define E1000_DEV_ID_82546GB_SERDES             0x107B
#define E1000_DEV_ID_82546GB_PCIE               0x108A
#define E1000_DEV_ID_82546GB_QUAD_COPPER        0x1099
#define E1000_DEV_ID_82546GB_QUAD_COPPER_KSP3   0x10B5
#define E1000_DEV_ID_82541EI                    0x1013
#define E1000_DEV_ID_82541EI_MOBILE             0x1018
#define E1000_DEV_ID_82541ER_LOM                0x1014
#define E1000_DEV_ID_82541ER                    0x1078
#define E1000_DEV_ID_82541GI                    0x1076
#define E1000_DEV_ID_82541GI_LF                 0x107C
#define E1000_DEV_ID_82541GI_MOBILE             0x1077
#define E1000_DEV_ID_82547EI                    0x1019
#define E1000_DEV_ID_82547EI_MOBILE             0x101A
#define E1000_DEV_ID_82547GI                    0x1075
#define E1000_DEV_ID_82571EB_COPPER             0x105E
#define E1000_DEV_ID_82571EB_FIBER              0x105F
#define E1000_DEV_ID_82571EB_SERDES             0x1060
#define E1000_DEV_ID_82571EB_SERDES_DUAL        0x10D9
#define E1000_DEV_ID_82571EB_SERDES_QUAD        0x10DA
#define E1000_DEV_ID_82571EB_QUAD_COPPER        0x10A4
#define E1000_DEV_ID_82571PT_QUAD_COPPER        0x10D5
#define E1000_DEV_ID_82571EB_QUAD_FIBER         0x10A5
#define E1000_DEV_ID_82571EB_QUAD_COPPER_LP     0x10BC
#define E1000_DEV_ID_82572EI_COPPER             0x107D
#define E1000_DEV_ID_82572EI_FIBER              0x107E
#define E1000_DEV_ID_82572EI_SERDES             0x107F
#define E1000_DEV_ID_82572EI                    0x10B9
#define E1000_DEV_ID_82573E                     0x108B
#define E1000_DEV_ID_82573E_IAMT                0x108C
#define E1000_DEV_ID_82573L                     0x109A
#define E1000_DEV_ID_82574L                     0x10D3
#define E1000_DEV_ID_82574LA                    0x10F6
#define E1000_DEV_ID_82583V                     0x150C
#define E1000_DEV_ID_80003ES2LAN_COPPER_DPT     0x1096
#define E1000_DEV_ID_80003ES2LAN_SERDES_DPT     0x1098
#define E1000_DEV_ID_80003ES2LAN_COPPER_SPT     0x10BA
#define E1000_DEV_ID_80003ES2LAN_SERDES_SPT     0x10BB
#define E1000_DEV_ID_ICH8_82567V_3              0x1501
#define E1000_DEV_ID_ICH8_IGP_M_AMT             0x1049
#define E1000_DEV_ID_ICH8_IGP_AMT               0x104A
#define E1000_DEV_ID_ICH8_IGP_C                 0x104B
#define E1000_DEV_ID_ICH8_IFE                   0x104C
#define E1000_DEV_ID_ICH8_IFE_GT                0x10C4
#define E1000_DEV_ID_ICH8_IFE_G                 0x10C5
#define E1000_DEV_ID_ICH8_IGP_M                 0x104D
#define E1000_DEV_ID_ICH9_IGP_M                 0x10BF
#define E1000_DEV_ID_ICH9_IGP_M_AMT             0x10F5
#define E1000_DEV_ID_ICH9_IGP_M_V               0x10CB
#define E1000_DEV_ID_ICH9_IGP_AMT               0x10BD
#define E1000_DEV_ID_ICH9_BM                    0x10E5
#define E1000_DEV_ID_ICH9_IGP_C                 0x294C
#define E1000_DEV_ID_ICH9_IFE                   0x10C0
#define E1000_DEV_ID_ICH9_IFE_GT                0x10C3
#define E1000_DEV_ID_ICH9_IFE_G                 0x10C2
#define E1000_DEV_ID_ICH10_R_BM_LM              0x10CC
#define E1000_DEV_ID_ICH10_R_BM_LF              0x10CD
#define E1000_DEV_ID_ICH10_R_BM_V               0x10CE
```

```
 127 #define E1000_DEV_ID_ICH10_D_BM_LM              0x10DE
 128 #define E1000_DEV_ID_ICH10_D_BM_LF              0x10DF
 129 #define E1000_DEV_ID_ICH10_D_BM_V               0x1525
 130 #define E1000_DEV_ID_PCH_M_HV_LM                0x10EA
 131 #define E1000_DEV_ID_PCH_M_HV_LC                0x10EB
 132 #define E1000_DEV_ID_PCH_D_HV_DM                0x10EF
 133 #define E1000_DEV_ID_PCH_D_HV_DC                0x10F0
 134 #define E1000_DEV_ID_PCH2_LV_LM                 0x1502
 135 #define E1000_DEV_ID_PCH2_LV_V                  0x1503
 136 #define E1000_DEV_ID_PCH_LPT_I217_LM            0x153A
 137 #define E1000_DEV_ID_PCH_LPT_I217_V             0x153B
 138 #define E1000_DEV_ID_PCH_LPTLP_I218_LM          0x155A
 139 #define E1000_DEV_ID_PCH_LPTLP_I218_V           0x1559
 140 #define E1000_DEV_ID_82576                      0x10C9
 141 #define E1000_DEV_ID_82576_FIBER                0x10E6
 142 #define E1000_DEV_ID_82576_SERDES               0x10E7
 143 #define E1000_DEV_ID_82576_QUAD_COPPER          0x10E8
 144 #define E1000_DEV_ID_82576_QUAD_COPPER_ET2      0x1526
 145 #define E1000_DEV_ID_82576_NS                   0x150A
 146 #define E1000_DEV_ID_82576_NS_SERDES            0x1518
 147 #define E1000_DEV_ID_82576_SERDES_QUAD          0x150D
 148 #define E1000_DEV_ID_82576_VF                   0x10CA
 149 #define E1000_DEV_ID_82576_VF_HV                0x152D
 150 #define E1000_DEV_ID_I350_VF                    0x1520
 151 #define E1000_DEV_ID_I350_VF_HV                 0x152F
 152 #define E1000_DEV_ID_82575EB_COPPER             0x10A7
 153 #define E1000_DEV_ID_82575EB_FIBER_SERDES       0x10A9
 154 #define E1000_DEV_ID_82575GB_QUAD_COPPER        0x10D6
 155 #define E1000_DEV_ID_82580_COPPER               0x150E
 156 #define E1000_DEV_ID_82580_FIBER                0x150F
 157 #define E1000_DEV_ID_82580_SERDES               0x1510
 158 #define E1000_DEV_ID_82580_SGMII                0x1511
 159 #define E1000_DEV_ID_82580_COPPER_DUAL          0x1516
 160 #define E1000_DEV_ID_82580_QUAD_FIBER           0x1527
 161 #define E1000_DEV_ID_I350_COPPER                0x1521
 162 #define E1000_DEV_ID_I350_FIBER                 0x1522
 163 #define E1000_DEV_ID_I350_SERDES                0x1523
 164 #define E1000_DEV_ID_I350_SGMII                 0x1524
 165 #define E1000_DEV_ID_I350_DA4                   0x1546
 166 #define E1000_DEV_ID_I210_COPPER                0x1533
 167 #define E1000_DEV_ID_I210_COPPER_OEM1           0x1534
 168 #define E1000_DEV_ID_I210_COPPER_IT             0x1535
 169 #define E1000_DEV_ID_I210_FIBER                 0x1536
 170 #define E1000_DEV_ID_I210_SERDES                0x1537
 171 #define E1000_DEV_ID_I210_SGMII                 0x1538
 172 #define E1000_DEV_ID_I211_COPPER                0x1539
 173 #define E1000_DEV_ID_I354_BACKPLANE_1GBPS       0x1F40
 174 #define E1000_DEV_ID_I354_SGMII                 0x1F41
 175 #define E1000_DEV_ID_I354_BACKPLANE_2_5GBPS     0x1F45
 176 #define E1000_DEV_ID_DH89XXCC_SGMII             0x0438
 177 #define E1000_DEV_ID_DH89XXCC_SERDES            0x043A
 178 #define E1000_DEV_ID_DH89XXCC_BACKPLANE         0x043C
 179 #define E1000_DEV_ID_DH89XXCC_SFP               0x0440

 181 #define E1000_REVISION_0        0
 182 #define E1000_REVISION_1        1
 183 #define E1000_REVISION_2        2
 184 #define E1000_REVISION_3        3
 185 #define E1000_REVISION_4        4

 187 #define E1000_FUNC_0            0
 188 #define E1000_FUNC_1            1
 189 #define E1000_FUNC_2            2
 190 #define E1000_FUNC_3            3

 192 #define E1000_ALT_MAC_ADDRESS_OFFSET_LAN0       0
```



```
 193 #define E1000_ALT_MAC_ADDRESS_OFFSET_LAN1       3
 194 #define E1000_ALT_MAC_ADDRESS_OFFSET_LAN2       6
 195 #define E1000_ALT_MAC_ADDRESS_OFFSET_LAN3       9

 197 enum e1000_mac_type {
 198         e1000_undefined = 0,
 199         e1000_82542,
 200         e1000_82543,
 201         e1000_82544,
 202         e1000_82540,
 203         e1000_82545,
 204         e1000_82545_rev_3,
 205         e1000_82546,
 206         e1000_82546_rev_3,
 207         e1000_82541,
 208         e1000_82541_rev_2,
 209         e1000_82547,
 210         e1000_82547_rev_2,
 211         e1000_82571,
 212         e1000_82572,
 213         e1000_82573,
 214         e1000_82574,
 215         e1000_82583,
 216         e1000_80003es2lan,
 217         e1000_ich8lan,
 218         e1000_ich9lan,
 219         e1000_ich10lan,
 220         e1000_pchlan,
 221         e1000_pch2lan,
 222         e1000_pch_lpt,
 223         e1000_82575,
 224         e1000_82576,
 225         e1000_82580,
 226         e1000_i350,
 227         e1000_i354,
 228         e1000_i210,
 229         e1000_i211,
 230         e1000_vfadapt,
 231         e1000_vfadapt_i350,
 232         e1000_num_macs  /* List is 1-based, so subtract 1 for TRUE count. */
 233 };
```

*unchanged_portion_omitted_*

```
**********************************************************
   22930 Wed Feb 26 09:49:37 2014
new/usr/src/uts/common/io/e1000api/e1000_i210.c
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
______unchanged_portion_omitted_
```

```
 /**
  *  __e1000_access_xmdio_reg - Read/write XMDIO register
  *  @hw: pointer to the HW structure
  *  @address: XMDIO address to program
  *  @dev_addr: device address to program
  *  @data: pointer to value to read/write from/to the XMDIO address
  *  @read: boolean flag to indicate read or write
  **/
 static s32 __e1000_access_xmdio_reg(struct e1000_hw *hw, u16 address,
                                     u8 dev_addr, u16 *data, bool read)
 {
         s32 ret_val = E1000_SUCCESS;

         DEBUGFUNC("__e1000_access_xmdio_reg");

         ret_val = hw->phy.ops.write_reg(hw, E1000_MMDAC, dev_addr);
         if (ret_val)
                 return ret_val;

         ret_val = hw->phy.ops.write_reg(hw, E1000_MMDAAD, address);
         if (ret_val)
                 return ret_val;

         ret_val = hw->phy.ops.write_reg(hw, E1000_MMDAC, E1000_MMDAC_FUNC_DATA |
                                                          dev_addr);
         if (ret_val)
                 return ret_val;

         if (read)
                 ret_val = hw->phy.ops.read_reg(hw, E1000_MMDAAD, data);
         else
                 ret_val = hw->phy.ops.write_reg(hw, E1000_MMDAAD, *data);
         if (ret_val)
                 return ret_val;

         /* Recalibrate the device back to 0 */
         ret_val = hw->phy.ops.write_reg(hw, E1000_MMDAC, 0);
         if (ret_val)
                 return ret_val;

         return ret_val;
 }

 /**
  *  e1000_read_xmdio_reg - Read XMDIO register
  *  @hw: pointer to the HW structure
  *  @addr: XMDIO address to program
  *  @dev_addr: device address to program
  *  @data: value to be read from the EMI address
  **/
 s32 e1000_read_xmdio_reg(struct e1000_hw *hw, u16 addr, u8 dev_addr, u16 *data)
 {
         DEBUGFUNC("e1000_read_xmdio_reg");

         return __e1000_access_xmdio_reg(hw, addr, dev_addr, data, TRUE);
 }

 /**
```



```
  *  e1000_write_xmdio_reg - Write XMDIO register
  *  @hw: pointer to the HW structure
  *  @addr: XMDIO address to program
  *  @dev_addr: device address to program
  *  @data: value to be written to the XMDIO address
  **/
 s32 e1000_write_xmdio_reg(struct e1000_hw *hw, u16 addr, u8 dev_addr, u16 data)
 {
         DEBUGFUNC("e1000_read_xmdio_reg");

         return __e1000_access_xmdio_reg(hw, addr, dev_addr, &data, FALSE);
 }
```

****
4157 Wed Feb 26 09:49:37 2014
new/usr/src/uts/common/io/e1000api/e1000_i210.h
4431 igb support for I354
4616 igb has uninitialized kstats
****

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

  Copyright (c) 2001-2013, 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 _E1000_I210_H_
#define _E1000_I210_H_

#ifdef __cplusplus
extern "C" {
#endif

s32 e1000_update_flash_i210(struct e1000_hw *hw);
s32 e1000_update_nvm_checksum_i210(struct e1000_hw *hw);
s32 e1000_validate_nvm_checksum_i210(struct e1000_hw *hw);
s32 e1000_write_nvm_srwr_i210(struct e1000_hw *hw, u16 offset,
                              u16 words, u16 *data);
s32 e1000_read_nvm_srrd_i210(struct e1000_hw *hw, u16 offset,
                             u16 words, u16 *data);
s32 e1000_read_invm_i211(struct e1000_hw *hw, u8 address, u16 *data);
s32 e1000_acquire_swfw_sync_i210(struct e1000_hw *hw, u16 mask);
void e1000_release_swfw_sync_i210(struct e1000_hw *hw, u16 mask);
s32 e1000_read_xmdio_reg(struct e1000_hw *hw, u16 addr, u8 dev_addr,
                         u16 *data);
s32 e1000_write_xmdio_reg(struct e1000_hw *hw, u16 addr, u8 dev_addr,
                          u16 data);

#define E1000_STM_OPCODE                0xDB00
#define E1000_EEPROM_FLASH_SIZE_WORD    0x11

#define INVM_DWORD_TO_RECORD_TYPE(invm_dword) \
```



```
        (u8)((invm_dword) & 0x7)
#define INVM_DWORD_TO_WORD_ADDRESS(invm_dword) \
        (u8)(((invm_dword) & 0x0000FE00) >> 9)
#define INVM_DWORD_TO_WORD_DATA(invm_dword) \
        (u16)(((invm_dword) & 0xFFFF0000) >> 16)

enum E1000_INVM_STRUCTURE_TYPE {
        E1000_INVM_UNINITIALIZED_STRUCTURE              = 0x00,
        E1000_INVM_WORD_AUTOLOAD_STRUCTURE              = 0x01,
        E1000_INVM_CSR_AUTOLOAD_STRUCTURE               = 0x02,
        E1000_INVM_PHY_REGISTER_AUTOLOAD_STRUCTURE      = 0x03,
        E1000_INVM_RSA_KEY_SHA256_STRUCTURE             = 0x04,
        E1000_INVM_INVALIDATED_STRUCTURE                = 0x0F,
};
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   37850 Wed Feb 26 09:49:37 2014
new/usr/src/uts/common/io/e1000api/e1000_regs.h
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
```

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

  Copyright (c) 2001-2013, 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 _E1000_REGS_H_
#define _E1000_REGS_H_

#ifdef __cplusplus
extern "C" {
#endif

#define E1000_CTRL      0x00000  /* Device Control - RW */
#define E1000_CTRL_DUP  0x00004  /* Device Control Duplicate (Shadow) - RW */
#define E1000_STATUS    0x00008  /* Device Status - RO */
#define E1000_EECD      0x00010  /* EEPROM/Flash Control - RW */
#define E1000_EERD      0x00014  /* EEPROM Read - RW */
#define E1000_CTRL_EXT  0x00018  /* Extended Device Control - RW */
#define E1000_FLA       0x0001C  /* Flash Access - RW */
#define E1000_MDIC      0x00020  /* MDI Control - RW */
#define E1000_MDICNFG   0x00E04  /* MDI Config - RW */
#define E1000_REGISTER_SET_SIZE         0x20000 /* CSR Size */
#define E1000_EEPROM_INIT_CTRL_WORD_2   0x0F /* EEPROM Init Ctrl Word 2 */
#define E1000_EEPROM_PCIE_CTRL_WORD_2   0x28 /* EEPROM PCIe Ctrl Word 2 */
#define E1000_BARCTRL                   0x5BBC /* BAR ctrl reg */
#define E1000_BARCTRL_FLSIZE            0x0700 /* BAR ctrl Flsize */
#define E1000_BARCTRL_CSRSIZE           0x2000 /* BAR ctrl CSR size */
#define E1000_MPHY_ADDR_CTRL            0x0024 /* GbE MPHY Address Control */
#define E1000_MPHY_DATA                 0x0E10 /* GBE MPHY Data */
#define E1000_MPHY_STAT                 0x0E0C /* GBE MPHY Statistics */
#define E1000_PPHY_CTRL                 0x5b48 /* PCIe PHY Control */
```



```
#define E1000_I350_BARCTRL              0x5BFC /* BAR ctrl reg */
#define E1000_I350_DTXMXPKTSZ           0x355C /* Maximum sent packet size reg*/
#define E1000_SCTL      0x00024  /* SerDes Control - RW */
#define E1000_FCAL      0x00028  /* Flow Control Address Low - RW */
#define E1000_FCAH      0x0002C  /* Flow Control Address High -RW */
#define E1000_FEXTNVM   0x00028  /* Future Extended NVM - RW */
#define E1000_FEXTNVM3  0x0003C  /* Future Extended NVM 3 - RW */
#define E1000_FEXTNVM4  0x00024  /* Future Extended NVM 4 - RW */
#define E1000_FEXTNVM6  0x00010  /* Future Extended NVM 6 - RW */
#define E1000_FEXTNVM7  0x000E4  /* Future Extended NVM 7 - RW */
#define E1000_FCT       0x00030  /* Flow Control Type - RW */
#define E1000_CONNSW    0x00034  /* Copper/Fiber switch control - RW */
#define E1000_VET       0x00038  /* VLAN Ether Type - RW */
#define E1000_ICR       0x000C0  /* Interrupt Cause Read - R/clr */
#define E1000_ITR       0x000C4  /* Interrupt Throttling Rate - RW */
#define E1000_ICS       0x000C8  /* Interrupt Cause Set - WO */
#define E1000_IMS       0x000D0  /* Interrupt Mask Set - RW */
#define E1000_IMC       0x000D8  /* Interrupt Mask Clear - WO */
#define E1000_IAM       0x000E0  /* Interrupt Acknowledge Auto Mask */
#define E1000_IVAR      0x000E4  /* Interrupt Vector Allocation Register - RW */
#define E1000_SVCR      0x000F0
#define E1000_SVT       0x000F4
#define E1000_LPIC      0x000FC  /* Low Power IDLE control */
#define E1000_RCTL      0x00100  /* Rx Control - RW */
#define E1000_FCTTV     0x00170  /* Flow Control Transmit Timer Value - RW */
#define E1000_TXCW      0x00178  /* Tx Configuration Word - RW */
#define E1000_RXCW      0x00180  /* Rx Configuration Word - RO */
#define E1000_PBA_ECC   0x01100  /* PBA ECC Register */
#define E1000_EICR      0x01580  /* Ext. Interrupt Cause Read - R/clr */
#define E1000_EITR(_n)  (0x01680 + (0x4 * (_n)))
#define E1000_EICS      0x01520  /* Ext. Interrupt Cause Set - W0 */
#define E1000_EIMS      0x01524  /* Ext. Interrupt Mask Set/Read - RW */
#define E1000_EIMC      0x01528  /* Ext. Interrupt Mask Clear - WO */
#define E1000_EIAC      0x0152C  /* Ext. Interrupt Auto Clear - RW */
#define E1000_EIAM      0x01530  /* Ext. Interrupt Ack Auto Clear Mask - RW */
#define E1000_GPIE      0x01514  /* General Purpose Interrupt Enable - RW */
#define E1000_IVAR0     0x01700  /* Interrupt Vector Allocation (array) - RW */
#define E1000_IVAR_MISC 0x01740 /* IVAR for "other" causes - RW */
#define E1000_TCTL      0x00400  /* Tx Control - RW */
#define E1000_TCTL_EXT  0x00404  /* Extended Tx Control - RW */
#define E1000_TIPG      0x00410  /* Tx Inter-packet gap -RW */
#define E1000_TBT       0x00448  /* Tx Burst Timer - RW */
#define E1000_AIT       0x00458  /* Adaptive Interframe Spacing Throttle - RW */
#define E1000_LEDCTL    0x00E00  /* LED Control - RW */
#define E1000_LEDMUX    0x08130 /* LED MUX Control */
#define E1000_EXTCNF_CTRL       0x00F00  /* Extended Configuration Control */
#define E1000_EXTCNF_SIZE       0x00F08  /* Extended Configuration Size */
#define E1000_PHY_CTRL  0x00F10  /* PHY Control Register in CSR */
#define E1000_POEMB     E1000_PHY_CTRL /* PHY OEM Bits */
#define E1000_PBA       0x01000  /* Packet Buffer Allocation - RW */
#define E1000_PBS       0x01008  /* Packet Buffer Size */
#define E1000_PBECCSTS  0x0100C  /* Packet Buffer ECC Status - RW */
#define E1000_EEMNGCTL  0x01010  /* MNG EEprom Control */
#define E1000_EEARBC    0x01024  /* EEPROM Auto Read Bus Control */
#define E1000_FLASHT    0x01028  /* FLASH Timer Register */
#define E1000_EEWR      0x0102C  /* EEPROM Write Register - RW */
#define E1000_FLSWCTL   0x01030  /* FLASH control register */
#define E1000_FLSWDATA  0x01034  /* FLASH data register */
#define E1000_FLSWCNT   0x01038  /* FLASH Access Counter */
#define E1000_FLOP      0x0103C  /* FLASH Opcode Register */
#define E1000_I2CCMD    0x01028  /* SFPI2C Command Register - RW */
#define E1000_I2CPARAMS 0x0102C /* SFPI2C Parameters Register - RW */
#define E1000_I2CBB_EN  0x00000100  /* I2C - Bit Bang Enable */
#define E1000_I2C_CLK_OUT       0x00000200  /* I2C- Clock */
#define E1000_I2C_DATA_OUT      0x00000400  /* I2C- Data Out */
#define E1000_I2C_DATA_OE_N     0x00000800  /* I2C- Data Output Enable */
```

```
#define E1000_I2C_DATA_IN       0x00001000  /* I2C- Data In */
#define E1000_I2C_CLK_OE_N      0x00002000  /* I2C- Clock Output Enable */
#define E1000_I2C_CLK_IN        0x00004000  /* I2C- Clock In */
#define E1000_I2C_CLK_STRETCH_DIS       0x00008000 /* I2C- Dis Clk Stretching */
#define E1000_WDSTP     0x01040  /* Watchdog Setup - RW */
#define E1000_SWDSTS    0x01044  /* SW Device Status - RW */
#define E1000_FRTIMER   0x01048  /* Free Running Timer - RW */
#define E1000_TCPTIMER  0x0104C  /* TCP Timer - RW */
#define E1000_VPDDIAG   0x01060  /* VPD Diagnostic - RO */
#define E1000_ICR_V2    0x01500  /* Intr Cause - new location - RC */
#define E1000_ICS_V2    0x01504  /* Intr Cause Set - new location - WO */
#define E1000_IMS_V2    0x01508  /* Intr Mask Set/Read - new location - RW */
#define E1000_IMC_V2    0x0150C  /* Intr Mask Clear - new location - WO */
#define E1000_IAM_V2    0x01510  /* Intr Ack Auto Mask - new location - RW */
#define E1000_ERT       0x02008  /* Early Rx Threshold - RW */
#define E1000_FCRTL     0x02160  /* Flow Control Receive Threshold Low - RW */
#define E1000_FCRTH     0x02168  /* Flow Control Receive Threshold High - RW */
#define E1000_PSRCTL    0x02170  /* Packet Split Receive Control - RW */
#define E1000_RDFH      0x02410  /* Rx Data FIFO Head - RW */
#define E1000_RDFT      0x02418  /* Rx Data FIFO Tail - RW */
#define E1000_RDFHS     0x02420  /* Rx Data FIFO Head Saved - RW */
#define E1000_RDFTS     0x02428  /* Rx Data FIFO Tail Saved - RW */
#define E1000_RDFPC     0x02430  /* Rx Data FIFO Packet Count - RW */
#define E1000_PBRTH     0x02458  /* PB Rx Arbitration Threshold - RW */
#define E1000_FCRTV     0x02460  /* Flow Control Refresh Timer Value - RW */
/* Split and Replication Rx Control - RW */
#define E1000_RDPUMB    0x025CC  /* DMA Rx Descriptor uC Mailbox - RW */
#define E1000_RDPUAD    0x025D0  /* DMA Rx Descriptor uC Addr Command - RW */
#define E1000_RDPUWD    0x025D4  /* DMA Rx Descriptor uC Data Write - RW */
#define E1000_RDPURD    0x025D8  /* DMA Rx Descriptor uC Data Read - RW */
#define E1000_RDPUCTL   0x025DC  /* DMA Rx Descriptor uC Control - RW */
#define E1000_PBDIAG    0x02458  /* Packet Buffer Diagnostic - RW */
#define E1000_RXPBS     0x02404  /* Rx Packet Buffer Size - RW */
#define E1000_IRPBS     0x02404 /* Same as RXPBS, renamed for newer Si - RW */
#define E1000_PBRWAC    0x024E8 /* Rx packet buffer wrap around counter - RO */
#define E1000_RDTR      0x02820  /* Rx Delay Timer - RW */
#define E1000_RADV      0x0282C  /* Rx Interrupt Absolute Delay Timer - RW */
#define E1000_EMIADD    0x10    /* Extended Memory Indirect Address */
#define E1000_EMIDATA   0x11    /* Extended Memory Indirect Data */
#define E1000_SRWR              0x12018  /* Shadow Ram Write Register - RW */
#define E1000_I210_FLMNGCTL     0x12038
#define E1000_I210_FLMNGDATA    0x1203C
#define E1000_I210_FLMNGCNT     0x12040

#define E1000_I210_FLSWCTL      0x12048
#define E1000_I210_FLSWDATA     0x1204C
#define E1000_I210_FLSWCNT      0x12050

#define E1000_I210_FLA          0x1201C

#define E1000_INVM_DATA_REG(_n) (0x12120 + 4*(_n))
#define E1000_INVM_SIZE         64 /* Number of INVM Data Registers */

/* QAV Tx mode control register */
#define E1000_I210_TQAVCTRL     0x3570

/* QAV Tx mode control register bitfields masks */
/* QAV enable */
#define E1000_TQAVCTRL_MODE                     (1 << 0)
/* Fetching arbitration type */
#define E1000_TQAVCTRL_FETCH_ARB                (1 << 4)
/* Fetching timer enable */
#define E1000_TQAVCTRL_FETCH_TIMER_ENABLE       (1 << 5)
/* Launch arbitration type */
#define E1000_TQAVCTRL_LAUNCH_ARB               (1 << 8)
/* Launch timer enable */
```



```
#define E1000_TQAVCTRL_LAUNCH_TIMER_ENABLE      (1 << 9)
/* SP waits for SR enable */
#define E1000_TQAVCTRL_SP_WAIT_SR               (1 << 10)
/* Fetching timer correction */
#define E1000_TQAVCTRL_FETCH_TIMER_DELTA_OFFSET 16
#define E1000_TQAVCTRL_FETCH_TIMER_DELTA        \
                        (0xFFFF << E1000_TQAVCTRL_FETCH_TIMER_DELTA_OFFSET)

/* High credit registers where _n can be 0 or 1. */
#define E1000_I210_TQAVHC(_n)                   (0x300C + 0x40 * (_n))

/* Queues fetch arbitration priority control register */
#define E1000_I210_TQAVARBCTRL                  0x3574
/* Queues priority masks where _n and _p can be 0-3. */
#define E1000_TQAVARBCTRL_QUEUE_PRI(_n, _p)     ((_p) << (2 * _n))
/* QAV Tx mode control registers where _n can be 0 or 1. */
#define E1000_I210_TQAVCC(_n)                   (0x3004 + 0x40 * (_n))

/* QAV Tx mode control register bitfields masks */
#define E1000_TQAVCC_IDLE_SLOPE         0xFFFF /* Idle slope */
#define E1000_TQAVCC_KEEP_CREDITS       (1 << 30) /* Keep credits opt enable */
#define E1000_TQAVCC_QUEUE_MODE         (1 << 31) /* SP vs. SR Tx mode */

/* Good transmitted packets counter registers */
#define E1000_PQGPTC(_n)                (0x010014 + (0x100 * (_n)))

/* Queues packet buffer size masks where _n can be 0-3 and _s 0-63 [kB] */
#define E1000_I210_TXPBS_SIZE(_n, _s)   ((_s) << (6 * _n))

#define E1000_MMDAC                     13 /* MMD Access Control */
#define E1000_MMDAAD                    14 /* MMD Access Address/Data */

/* Convenience macros
 *
 * Note: "_n" is the queue number of the register to be written to.
 *
 * Example usage:
 * E1000_RDBAL_REG(current_rx_queue)
 */
#define E1000_RDBAL(_n) ((_n) < 4 ? (0x02800 + ((_n) * 0x100)) : \
                         (0x0C000 + ((_n) * 0x40)))
#define E1000_RDBAH(_n) ((_n) < 4 ? (0x02804 + ((_n) * 0x100)) : \
                         (0x0C004 + ((_n) * 0x40)))
#define E1000_RDLEN(_n) ((_n) < 4 ? (0x02808 + ((_n) * 0x100)) : \
                         (0x0C008 + ((_n) * 0x40)))
#define E1000_SRRCTL(_n)        ((_n) < 4 ? (0x0280C + ((_n) * 0x100)) : \
                                 (0x0C00C + ((_n) * 0x40)))
#define E1000_RDH(_n)   ((_n) < 4 ? (0x02810 + ((_n) * 0x100)) : \
                         (0x0C010 + ((_n) * 0x40)))
#define E1000_RXCTL(_n) ((_n) < 4 ? (0x02814 + ((_n) * 0x100)) : \
                         (0x0C014 + ((_n) * 0x40)))
#define E1000_DCA_RXCTRL(_n)    E1000_RXCTL(_n)
#define E1000_RDT(_n)   ((_n) < 4 ? (0x02818 + ((_n) * 0x100)) : \
                         (0x0C018 + ((_n) * 0x40)))
#define E1000_RXDCTL(_n)        ((_n) < 4 ? (0x02828 + ((_n) * 0x100)) : \
                                 (0x0C028 + ((_n) * 0x40)))
#define E1000_RQDPC(_n) ((_n) < 4 ? (0x02830 + ((_n) * 0x100)) : \
                         (0x0C030 + ((_n) * 0x40)))
#define E1000_TDBAL(_n) ((_n) < 4 ? (0x03800 + ((_n) * 0x100)) : \
                         (0x0E000 + ((_n) * 0x40)))
#define E1000_TDBAH(_n) ((_n) < 4 ? (0x03804 + ((_n) * 0x100)) : \
                         (0x0E004 + ((_n) * 0x40)))
#define E1000_TDLEN(_n) ((_n) < 4 ? (0x03808 + ((_n) * 0x100)) : \
                         (0x0E008 + ((_n) * 0x40)))
#define E1000_TDH(_n)   ((_n) < 4 ? (0x03810 + ((_n) * 0x100)) : \
                         (0x0E010 + ((_n) * 0x40)))
```

```
259 #define E1000_TXCTL(_n) ((_n) < 4 ? (0x03814 + ((_n) * 0x100)) : \
260                          (0x0E014 + ((_n) * 0x40)))
261 #define E1000_DCA_TXCTRL(_n) E1000_TXCTL(_n)
262 #define E1000_TDT(_n)   ((_n) < 4 ? (0x03818 + ((_n) * 0x100)) : \
263                          (0x0E018 + ((_n) * 0x40)))
264 #define E1000_TXDCTL(_n)        ((_n) < 4 ? (0x03828 + ((_n) * 0x100)) : \
265                                  (0x0E028 + ((_n) * 0x40)))
266 #define E1000_TDWBAL(_n)        ((_n) < 4 ? (0x03838 + ((_n) * 0x100)) : \
267                                  (0x0E038 + ((_n) * 0x40)))
268 #define E1000_TDWBAH(_n)        ((_n) < 4 ? (0x0383C + ((_n) * 0x100)) : \
269                                  (0x0E03C + ((_n) * 0x40)))
270 #define E1000_TARC(_n)          (0x03840 + ((_n) * 0x100))
271 #define E1000_RSRPD             0x02C00  /* Rx Small Packet Detect - RW */
272 #define E1000_RAID              0x02C08  /* Receive Ack Interrupt Delay - RW */
273 #define E1000_TXDMAC            0x03000  /* Tx DMA Control - RW */
274 #define E1000_KABGTXD           0x03004  /* AFE Band Gap Transmit Ref Data */
275 #define E1000_PSRTYPE(_i)       (0x05480 + ((_i) * 4))
276 #define E1000_RAL(_i)           (((_i) <= 15) ? (0x05400 + ((_i) * 8)) : \
277                                  (0x054E0 + ((_i - 16) * 8)))
278 #define E1000_RAH(_i)           (((_i) <= 15) ? (0x05404 + ((_i) * 8)) : \
279                                  (0x054E4 + ((_i - 16) * 8)))
280 #define E1000_SHRAL(_i)         (0x05438 + ((_i) * 8))
281 #define E1000_SHRAH(_i)         (0x0543C + ((_i) * 8))
282 #define E1000_IP4AT_REG(_i)     (0x05840 + ((_i) * 8))
283 #define E1000_IP6AT_REG(_i)     (0x05880 + ((_i) * 4))
284 #define E1000_WUPM_REG(_i)      (0x05A00 + ((_i) * 4))
285 #define E1000_FFMT_REG(_i)      (0x09000 + ((_i) * 8))
286 #define E1000_FFVT_REG(_i)      (0x09800 + ((_i) * 8))
287 #define E1000_FFLT_REG(_i)      (0x05F00 + ((_i) * 8))
288 #define E1000_PBSLAC            0x03100  /* Pkt Buffer Slave Access Control */
289 #define E1000_PBSLAD(_n)        (0x03110 + (0x4 * (_n)))  /* Pkt Buffer DWORD */
290 #define E1000_TXPBS             0x03404  /* Tx Packet Buffer Size - RW */
291 /* Same as TXPBS, renamed for newer Si - RW */
292 #define E1000_ITPBS             0x03404
293 #define E1000_TDFH              0x03410  /* Tx Data FIFO Head - RW */
294 #define E1000_TDFT              0x03418  /* Tx Data FIFO Tail - RW */
295 #define E1000_TDFHS             0x03420  /* Tx Data FIFO Head Saved - RW */
296 #define E1000_TDFTS             0x03428  /* Tx Data FIFO Tail Saved - RW */
297 #define E1000_TDFPC             0x03430  /* Tx Data FIFO Packet Count - RW */
298 #define E1000_TDPUMB            0x0357C  /* DMA Tx Desc uC Mail Box - RW */
299 #define E1000_TDPUAD            0x03580  /* DMA Tx Desc uC Addr Command - RW */
300 #define E1000_TDPUWD            0x03584  /* DMA Tx Desc uC Data Write - RW */
301 #define E1000_TDPURD            0x03588  /* DMA Tx Desc uC Data  Read  - RW */
302 #define E1000_TDPUCTL           0x0358C  /* DMA Tx Desc uC Control - RW */
303 #define E1000_DTXCTL            0x03590  /* DMA Tx Control - RW */
304 #define E1000_DTXTCPFLGL        0x0359C /* DMA Tx Control flag low - RW */
305 #define E1000_DTXTCPFLGH        0x035A0 /* DMA Tx Control flag high - RW */
306 /* DMA Tx Max Total Allow Size Reqs - RW */
307 #define E1000_DTXMXSZRQ         0x03540
308 #define E1000_TIDV      0x03820  /* Tx Interrupt Delay Value - RW */
309 #define E1000_TADV      0x0382C  /* Tx Interrupt Absolute Delay Val - RW */
310 #define E1000_TSPMT     0x03830  /* TCP Segmentation PAD & Min Threshold - RW */
311 #define E1000_CRCERRS   0x04000  /* CRC Error Count - R/clr */
312 #define E1000_ALGNERRC  0x04004  /* Alignment Error Count - R/clr */
313 #define E1000_SYMERRS   0x04008  /* Symbol Error Count - R/clr */
314 #define E1000_RXERRC    0x0400C  /* Receive Error Count - R/clr */
315 #define E1000_MPC       0x04010  /* Missed Packet Count - R/clr */
316 #define E1000_SCC       0x04014  /* Single Collision Count - R/clr */
317 #define E1000_ECOL      0x04018  /* Excessive Collision Count - R/clr */
318 #define E1000_MCC       0x0401C  /* Multiple Collision Count - R/clr */
319 #define E1000_LATECOL   0x04020  /* Late Collision Count - R/clr */
320 #define E1000_COLC      0x04028  /* Collision Count - R/clr */
321 #define E1000_DC        0x04030  /* Defer Count - R/clr */
322 #define E1000_TNCRS     0x04034  /* Tx-No CRS - R/clr */
323 #define E1000_SEC       0x04038  /* Sequence Error Count - R/clr */
324 #define E1000_CEXTERR   0x0403C  /* Carrier Extension Error Count - R/clr */
```



```
325 #define E1000_RLEC      0x04040  /* Receive Length Error Count - R/clr */
326 #define E1000_XONRXC    0x04048  /* XON Rx Count - R/clr */
327 #define E1000_XONTXC    0x0404C  /* XON Tx Count - R/clr */
328 #define E1000_XOFFRXC   0x04050  /* XOFF Rx Count - R/clr */
329 #define E1000_XOFFTXC   0x04054  /* XOFF Tx Count - R/clr */
330 #define E1000_FCRUC     0x04058  /* Flow Control Rx Unsupported Count- R/clr */
331 #define E1000_PRC64     0x0405C  /* Packets Rx (64 bytes) - R/clr */
332 #define E1000_PRC127    0x04060  /* Packets Rx (65-127 bytes) - R/clr */
333 #define E1000_PRC255    0x04064  /* Packets Rx (128-255 bytes) - R/clr */
334 #define E1000_PRC511    0x04068  /* Packets Rx (255-511 bytes) - R/clr */
335 #define E1000_PRC1023   0x0406C  /* Packets Rx (512-1023 bytes) - R/clr */
336 #define E1000_PRC1522   0x04070  /* Packets Rx (1024-1522 bytes) - R/clr */
337 #define E1000_GPRC      0x04074  /* Good Packets Rx Count - R/clr */
338 #define E1000_BPRC      0x04078  /* Broadcast Packets Rx Count - R/clr */
339 #define E1000_MPRC      0x0407C  /* Multicast Packets Rx Count - R/clr */
340 #define E1000_GPTC      0x04080  /* Good Packets Tx Count - R/clr */
341 #define E1000_GORCL     0x04088  /* Good Octets Rx Count Low - R/clr */
342 #define E1000_GORCH     0x0408C  /* Good Octets Rx Count High - R/clr */
343 #define E1000_GOTCL     0x04090  /* Good Octets Tx Count Low - R/clr */
344 #define E1000_GOTCH     0x04094  /* Good Octets Tx Count High - R/clr */
345 #define E1000_RNBC      0x040A0  /* Rx No Buffers Count - R/clr */
346 #define E1000_RUC       0x040A4  /* Rx Undersize Count - R/clr */
347 #define E1000_RFC       0x040A8  /* Rx Fragment Count - R/clr */
348 #define E1000_ROC       0x040AC  /* Rx Oversize Count - R/clr */
349 #define E1000_RJC       0x040B0  /* Rx Jabber Count - R/clr */
350 #define E1000_MGTPRC    0x040B4  /* Management Packets Rx Count - R/clr */
351 #define E1000_MGTPDC    0x040B8  /* Management Packets Dropped Count - R/clr */
352 #define E1000_MGTPTC    0x040BC  /* Management Packets Tx Count - R/clr */
353 #define E1000_TORL      0x040C0  /* Total Octets Rx Low - R/clr */
354 #define E1000_TORH      0x040C4  /* Total Octets Rx High - R/clr */
355 #define E1000_TOTL      0x040C8  /* Total Octets Tx Low - R/clr */
356 #define E1000_TOTH      0x040CC  /* Total Octets Tx High - R/clr */
357 #define E1000_TPR       0x040D0  /* Total Packets Rx - R/clr */
358 #define E1000_TPT       0x040D4  /* Total Packets Tx - R/clr */
359 #define E1000_PTC64     0x040D8  /* Packets Tx (64 bytes) - R/clr */
360 #define E1000_PTC127    0x040DC  /* Packets Tx (65-127 bytes) - R/clr */
361 #define E1000_PTC255    0x040E0  /* Packets Tx (128-255 bytes) - R/clr */
362 #define E1000_PTC511    0x040E4  /* Packets Tx (256-511 bytes) - R/clr */
363 #define E1000_PTC1023   0x040E8  /* Packets Tx (512-1023 bytes) - R/clr */
364 #define E1000_PTC1522   0x040EC  /* Packets Tx (1024-1522 Bytes) - R/clr */
365 #define E1000_MPTC      0x040F0  /* Multicast Packets Tx Count - R/clr */
366 #define E1000_BPTC      0x040F4  /* Broadcast Packets Tx Count - R/clr */
367 #define E1000_TSCTC     0x040F8  /* TCP Segmentation Context Tx - R/clr */
368 #define E1000_TSCTFC    0x040FC  /* TCP Segmentation Context Tx Fail - R/clr */
369 #define E1000_IAC       0x04100  /* Interrupt Assertion Count */
370 #define E1000_ICRXPTC   0x04104  /* Interrupt Cause Rx Pkt Timer Expire Count */
371 #define E1000_ICRXATC   0x04108  /* Interrupt Cause Rx Abs Timer Expire Count */
372 #define E1000_ICTXPTC   0x0410C  /* Interrupt Cause Tx Pkt Timer Expire Count */
373 #define E1000_ICTXATC   0x04110  /* Interrupt Cause Tx Abs Timer Expire Count */
374 #define E1000_ICTXQEC   0x04118  /* Interrupt Cause Tx Queue Empty Count */
375 #define E1000_ICTXQMTC  0x0411C  /* Interrupt Cause Tx Queue Min Thresh Count */
376 #define E1000_ICRXDMTC  0x04120  /* Interrupt Cause Rx Desc Min Thresh Count */
377 #define E1000_ICRXOC    0x04124  /* Interrupt Cause Receiver Overrun Count */
378 #define E1000_CRC_OFFSET        0x05F50  /* CRC Offset register */

380 #define E1000_VFGPRC    0x00F10
381 #define E1000_VFGORC    0x00F18
382 #define E1000_VFMPRC    0x00F3C
383 #define E1000_VFGPTC    0x00F14
384 #define E1000_VFGOTC    0x00F34
385 #define E1000_VFGOTLBC  0x00F50
386 #define E1000_VFGPTLBC  0x00F44
387 #define E1000_VFGORLBC  0x00F48
388 #define E1000_VFGPRLBC  0x00F40
389 /* Virtualization statistical counters */
390 #define E1000_PFVFGPRC(_n)      (0x010010 + (0x100 * (_n)))
```

```
391 #define E1000_PFVFGPTC(_n)      (0x010014 + (0x100 * (_n)))
392 #define E1000_PFVFGORC(_n)      (0x010018 + (0x100 * (_n)))
393 #define E1000_PFVFGOTC(_n)      (0x010034 + (0x100 * (_n)))
394 #define E1000_PFVFMPRC(_n)      (0x010038 + (0x100 * (_n)))
395 #define E1000_PFVFGPRLBC(_n)    (0x010040 + (0x100 * (_n)))
396 #define E1000_PFVFGPTLBC(_n)    (0x010044 + (0x100 * (_n)))
397 #define E1000_PFVFGORLBC(_n)    (0x010048 + (0x100 * (_n)))
398 #define E1000_PFVFGOTLBC(_n)    (0x010050 + (0x100 * (_n)))

400 /* LinkSec */
401 #define E1000_LSECTXUT          0x04300  /* Tx Untagged Pkt Cnt */
402 #define E1000_LSECTXPKTE        0x04304  /* Encrypted Tx Pkts Cnt */
403 #define E1000_LSECTXPKTP        0x04308  /* Protected Tx Pkt Cnt */
404 #define E1000_LSECTXOCTE        0x0430C  /* Encrypted Tx Octets Cnt */
405 #define E1000_LSECTXOCTP        0x04310  /* Protected Tx Octets Cnt */
406 #define E1000_LSECRXUT          0x04314  /* Untagged non-Strict Rx Pkt Cnt */
407 #define E1000_LSECRXOCTD        0x0431C  /* Rx Octets Decrypted Count */
408 #define E1000_LSECRXOCTV        0x04320  /* Rx Octets Validated */
409 #define E1000_LSECRXBAD         0x04324  /* Rx Bad Tag */
410 #define E1000_LSECRXNOSCI       0x04328  /* Rx Packet No SCI Count */
411 #define E1000_LSECRXUNSCI       0x0432C  /* Rx Packet Unknown SCI Count */
412 #define E1000_LSECRXUNCH        0x04330  /* Rx Unchecked Packets Count */
413 #define E1000_LSECRXDELAY       0x04340  /* Rx Delayed Packet Count */
414 #define E1000_LSECRXLATE        0x04350  /* Rx Late Packets Count */
415 #define E1000_LSECRXOK(_n)      (0x04360 + (0x04 * (_n))) /* Rx Pkt OK Cnt */
416 #define E1000_LSECRXINV(_n)     (0x04380 + (0x04 * (_n))) /* Rx Invalid Cnt */
417 #define E1000_LSECRXNV(_n)      (0x043A0 + (0x04 * (_n))) /* Rx Not Valid Cnt */
418 #define E1000_LSECRXUNSA        0x043C0  /* Rx Unused SA Count */
419 #define E1000_LSECRXNUSA        0x043D0  /* Rx Not Using SA Count */
420 #define E1000_LSECTXCAP         0x0B000  /* Tx Capabilities Register - RO */
421 #define E1000_LSECRXCAP         0x0B300  /* Rx Capabilities Register - RO */
422 #define E1000_LSECTXCTRL        0x0B004  /* Tx Control - RW */
423 #define E1000_LSECRXCTRL        0x0B304  /* Rx Control - RW */
424 #define E1000_LSECTXSCL         0x0B008  /* Tx SCI Low - RW */
425 #define E1000_LSECTXSCH         0x0B00C  /* Tx SCI High - RW */
426 #define E1000_LSECTXSA          0x0B010  /* Tx SA0 - RW */
427 #define E1000_LSECTXPN0         0x0B018  /* Tx SA PN 0 - RW */
428 #define E1000_LSECTXPN1         0x0B01C  /* Tx SA PN 1 - RW */
429 #define E1000_LSECRXSCL         0x0B3D0  /* Rx SCI Low - RW */
430 #define E1000_LSECRXSCH         0x0B3E0  /* Rx SCI High - RW */
431 /* LinkSec Tx 128-bit Key 0 - WO */
432 #define E1000_LSECTXKEY0(_n)    (0x0B020 + (0x04 * (_n)))
433 /* LinkSec Tx 128-bit Key 1 - WO */
434 #define E1000_LSECTXKEY1(_n)    (0x0B030 + (0x04 * (_n)))
435 #define E1000_LSECRXSA(_n)      (0x0B310 + (0x04 * (_n))) /* Rx SAs - RW */
436 #define E1000_LSECRXPN(_n)      (0x0B330 + (0x04 * (_n))) /* Rx SAs - RW */
437 /* LinkSec Rx Keys  - where _n is the SA no. and _m the 4 dwords of the 128 bit
438  * key - RW.
439  */
440 #define E1000_LSECRXKEY(_n, _m) (0x0B350 + (0x10 * (_n)) + (0x04 * (_m)))

442 #define E1000_SSVPC             0x041A0 /* Switch Security Violation Pkt Cnt */
443 #define E1000_IPSCTRL           0xB430  /* IpSec Control Register */
444 #define E1000_IPSRXCMD          0x0B408 /* IPSec Rx Command Register - RW */
445 #define E1000_IPSRXIDX          0x0B400 /* IPSec Rx Index - RW */
446 /* IPSec Rx IPv4/v6 Address - RW */
447 #define E1000_IPSRXIPADDR(_n)   (0x0B420 + (0x04 * (_n)))
448 /* IPSec Rx 128-bit Key - RW */
449 #define E1000_IPSRXKEY(_n)      (0x0B410 + (0x04 * (_n)))
450 #define E1000_IPSRXSALT         0x0B404  /* IPSec Rx Salt - RW */
451 #define E1000_IPSRXSPI          0x0B40C  /* IPSec Rx SPI - RW */
452 /* IPSec Tx 128-bit Key - RW */
453 #define E1000_IPSTXKEY(_n)      (0x0B460 + (0x04 * (_n)))
454 #define E1000_IPSTXSALT         0x0B454  /* IPSec Tx Salt - RW */
455 #define E1000_IPSTXIDX          0x0B450  /* IPSec Tx SA IDX - RW */
456 #define E1000_PCS_CFG0  0x04200  /* PCS Configuration 0 - RW */
```



```
457 #define E1000_PCS_LCTL  0x04208  /* PCS Link Control - RW */
458 #define E1000_PCS_LSTAT 0x0420C  /* PCS Link Status - RO */
459 #define E1000_CBTMPC    0x0402C  /* Circuit Breaker Tx Packet Count */
460 #define E1000_HTDPMC    0x0403C  /* Host Transmit Discarded Packets */
461 #define E1000_CBRDPC    0x04044  /* Circuit Breaker Rx Dropped Count */
462 #define E1000_CBRMPC    0x040FC  /* Circuit Breaker Rx Packet Count */
463 #define E1000_RPTHC     0x04104  /* Rx Packets To Host */
464 #define E1000_HGPTC     0x04118  /* Host Good Packets Tx Count */
465 #define E1000_HTCBDPC   0x04124  /* Host Tx Circuit Breaker Dropped Count */
466 #define E1000_HGORCL    0x04128  /* Host Good Octets Received Count Low */
467 #define E1000_HGORCH    0x0412C  /* Host Good Octets Received Count High */
468 #define E1000_HGOTCL    0x04130  /* Host Good Octets Transmit Count Low */
469 #define E1000_HGOTCH    0x04134  /* Host Good Octets Transmit Count High */
470 #define E1000_LENERRS   0x04138  /* Length Errors Count */
471 #define E1000_SCVPC     0x04228  /* SerDes/SGMII Code Violation Pkt Count */
472 #define E1000_HRMPC     0x0A018  /* Header Redirection Missed Packet Count */
473 #define E1000_PCS_ANADV 0x04218  /* AN advertisement - RW */
474 #define E1000_PCS_LPAB  0x0421C  /* Link Partner Ability - RW */
475 #define E1000_PCS_NPTX  0x04220  /* AN Next Page Transmit - RW */
476 #define E1000_PCS_LPABNP        0x04224 /* Link Partner Ability Next Pg - RW */
477 #define E1000_RXCSUM    0x05000  /* Rx Checksum Control - RW */
478 #define E1000_RLPML     0x05004  /* Rx Long Packet Max Length */
479 #define E1000_RFCTL     0x05008  /* Receive Filter Control*/
480 #define E1000_MTA       0x05200  /* Multicast Table Array - RW Array */
481 #define E1000_RA        0x05400  /* Receive Address - RW Array */
482 #define E1000_RA2       0x054E0  /* 2nd half of Rx address array - RW Array */
483 #define E1000_VFTA      0x05600  /* VLAN Filter Table Array - RW Array */
484 #define E1000_VT_CTL    0x0581C  /* VMDq Control - RW */
485 #define E1000_CIAA      0x05B88  /* Config Indirect Access Address - RW */
486 #define E1000_CIAD      0x05B8C  /* Config Indirect Access Data - RW */
487 #define E1000_VFQA0     0x0B000  /* VLAN Filter Queue Array 0 - RW Array */
488 #define E1000_VFQA1     0x0B200  /* VLAN Filter Queue Array 1 - RW Array */
489 #define E1000_WUC       0x05800  /* Wakeup Control - RW */
490 #define E1000_WUFC      0x05808  /* Wakeup Filter Control - RW */
491 #define E1000_WUS       0x05810  /* Wakeup Status - RO */
492 #define E1000_MANC      0x05820  /* Management Control - RW */
493 #define E1000_IPAV      0x05838  /* IP Address Valid - RW */
494 #define E1000_IP4AT     0x05840  /* IPv4 Address Table - RW Array */
495 #define E1000_IP6AT     0x05880  /* IPv6 Address Table - RW Array */
496 #define E1000_WUPL      0x05900  /* Wakeup Packet Length - RW */
497 #define E1000_WUPM      0x05A00  /* Wakeup Packet Memory - RO A */
498 #define E1000_PBACL     0x05B68  /* MSIx PBA Clear - Read/Write 1's to clear */
499 #define E1000_FFLT      0x05F00  /* Flexible Filter Length Table - RW Array */
500 #define E1000_HOST_IF   0x08800  /* Host Interface */
501 #define E1000_FFMT      0x09000  /* Flexible Filter Mask Table - RW Array */
502 #define E1000_FFVT      0x09800  /* Flexible Filter Value Table - RW Array */
503 #define E1000_HIBBA     0x8F40   /* Host Interface Buffer Base Address */
504 /* Flexible Host Filter Table */
505 #define E1000_FHFT(_n)  (0x09000 + ((_n) * 0x100))
506 /* Ext Flexible Host Filter Table */
507 #define E1000_FHFT_EXT(_n)      (0x09A00 + ((_n) * 0x100))


510 #define E1000_KMRNCTRLSTA       0x00034 /* MAC-PHY interface - RW */
511 #define E1000_MANC2H            0x05860 /* Management Control To Host - RW */
512 /* Management Decision Filters */
513 #define E1000_MDEF(_n)          (0x05890 + (4 * (_n)))
514 #define E1000_SW_FW_SYNC        0x05B5C /* SW-FW Synchronization - RW */
515 #define E1000_CCMCTL    0x05B48 /* CCM Control Register */
516 #define E1000_GIOCTL    0x05B44 /* GIO Analog Control Register */
517 #define E1000_SCCTL     0x05B4C /* PCIc PLL Configuration Register */
518 #define E1000_GCR       0x05B00 /* PCI-Ex Control */
519 #define E1000_GCR2      0x05B64 /* PCI-Ex Control #2 */
520 #define E1000_GSCL_1    0x05B10 /* PCI-Ex Statistic Control #1 */
521 #define E1000_GSCL_2    0x05B14 /* PCI-Ex Statistic Control #2 */
522 #define E1000_GSCL_3    0x05B18 /* PCI-Ex Statistic Control #3 */
```

```
#define E1000_GSCL_4    0x05B1C /* PCI-Ex Statistic Control #4 */
#define E1000_FACTPS    0x05B30 /* Function Active and Power State to MNG */
#define E1000_SWSM      0x05B50 /* SW Semaphore */
#define E1000_FWSM      0x05B54 /* FW Semaphore */
/* Driver-only SW semaphore (not used by BOOT agents) */
#define E1000_SWSM2     0x05B58
#define E1000_DCA_ID    0x05B70 /* DCA Requester ID Information - RO */
#define E1000_DCA_CTRL  0x05B74 /* DCA Control - RW */
#define E1000_UFUSE     0x05B78 /* UFUSE - RO */
#define E1000_FFLT_DBG  0x05F04 /* Debug Register */
#define E1000_HICR      0x08F00 /* Host Interface Control */
#define E1000_FWSTS     0x08F0C /* FW Status */

/* RSS registers */
#define E1000_CPUVEC    0x02C10 /* CPU Vector Register - RW */
#define E1000_MRQC      0x05818 /* Multiple Receive Control - RW */
#define E1000_IMIR(_i)  (0x05A80 + ((_i) * 4))  /* Immediate Interrupt */
#define E1000_IMIREXT(_i)       (0x05AA0 + ((_i) * 4)) /* Immediate INTR Ext*/
#define E1000_IMIRVP            0x05AC0 /* Immediate INT Rx VLAN Priority -RW */
#define E1000_MSIXBM(_i)        (0x01600 + ((_i) * 4)) /* MSI-X Alloc Reg -RW */
#define E1000_RETA(_i)  (0x05C00 + ((_i) * 4)) /* Redirection Table - RW */
#define E1000_RSSRK(_i) (0x05C80 + ((_i) * 4)) /* RSS Random Key - RW */
#define E1000_RSSIM     0x05864 /* RSS Interrupt Mask */
#define E1000_RSSIR     0x05868 /* RSS Interrupt Request */
/* VT Registers */
#define E1000_SWPBS     0x03004 /* Switch Packet Buffer Size - RW */
#define E1000_MBVFICR   0x00C80 /* Mailbox VF Cause - RWC */
#define E1000_MBVFIMR   0x00C84 /* Mailbox VF int Mask - RW */
#define E1000_VFLRE     0x00C88 /* VF Register Events - RWC */
#define E1000_VFRE      0x00C8C /* VF Receive Enables */
#define E1000_VFTE      0x00C90 /* VF Transmit Enables */
#define E1000_QDE       0x02408 /* Queue Drop Enable - RW */
#define E1000_DTXSWC    0x03500 /* DMA Tx Switch Control - RW */
#define E1000_WVBR      0x03554 /* VM Wrong Behavior - RWS */
#define E1000_RPLOLR    0x05AF0 /* Replication Offload - RW */
#define E1000_UTA       0x0A000 /* Unicast Table Array - RW */
#define E1000_IOVTCL    0x05BBC /* IOV Control Register */
#define E1000_VMRCTL    0x05D80 /* Virtual Mirror Rule Control */
#define E1000_VMRVLAN   0x05D90 /* Virtual Mirror Rule VLAN */
#define E1000_VMRVM     0x05DA0 /* Virtual Mirror Rule VM */
#define E1000_MDFB      0x03558 /* Malicious Driver free block */
#define E1000_LVMMC     0x03548 /* Last VM Misbehavior cause */
#define E1000_TXSWC     0x05ACC /* Tx Switch Control */
#define E1000_SCCRL     0x05DB0 /* Storm Control Control */
#define E1000_BSCTRH    0x05DB8 /* Broadcast Storm Control Threshold */
#define E1000_MSCTRH    0x05DBC /* Multicast Storm Control Threshold */
/* These act per VF so an array friendly macro is used */
#define E1000_V2PMAILBOX(_n)    (0x00C40 + (4 * (_n)))
#define E1000_P2VMAILBOX(_n)    (0x00C00 + (4 * (_n)))
#define E1000_VMBMEM(_n)        (0x00800 + (64 * (_n)))
#define E1000_VFVMBMEM(_n)      (0x00800 + (_n))
#define E1000_VMOLR(_n)         (0x05AD0 + (4 * (_n)))
/* VLAN Virtual Machine Filter - RW */
#define E1000_VLVF(_n)          (0x05D00 + (4 * (_n)))
#define E1000_VMVIR(_n)         (0x03700 + (4 * (_n)))
#define E1000_DVMOLR(_n)        (0x0C038 + (0x40 * (_n))) /* DMA VM offload */
#define E1000_VTCTRL(_n)        (0x10000 + (0x100 * (_n))) /* VT Control */
#define E1000_TSYNCRXCTL        0x0B620 /* Rx Time Sync Control register - RW */
#define E1000_TSYNCTXCTL        0x0B614 /* Tx Time Sync Control register - RW */
#define E1000_TSYNCRXCFG        0x05F50 /* Time Sync Rx Configuration - RW */
#define E1000_RXSTMPL   0x0B624 /* Rx timestamp Low - RO */
#define E1000_RXSTMPH   0x0B628 /* Rx timestamp High - RO */
#define E1000_RXSATRL   0x0B62C /* Rx timestamp attribute low - RO */
#define E1000_RXSATRH   0x0B630 /* Rx timestamp attribute high - RO */
#define E1000_TXSTMPL   0x0B618 /* Tx timestamp value Low - RO */
#define E1000_TXSTMPH   0x0B61C /* Tx timestamp value High - RO */
```



```
#define E1000_SYSTIML   0x0B600 /* System time register Low - RO */
#define E1000_SYSTIMH   0x0B604 /* System time register High - RO */
#define E1000_TIMINCA   0x0B608 /* Increment attributes register - RW */
#define E1000_TIMADJL   0x0B60C /* Time sync time adjustment offset Low - RW */
#define E1000_TIMADJH   0x0B610 /* Time sync time adjustment offset High - RW */
#define E1000_TSAUXC    0x0B640 /* Timesync Auxiliary Control register */
#define E1000_SYSTIMR   0x0B6F8 /* System time register Residue */
#define E1000_TSICR     0x0B66C /* Interrupt Cause Register */
#define E1000_TSIM      0x0B674 /* Interrupt Mask Register */
#define E1000_RXMTRL    0x0B634 /* Time sync Rx EtherType and Msg Type - RW */
#define E1000_RXUDP     0x0B638 /* Time Sync Rx UDP Port - RW */

/* Filtering Registers */
#define E1000_SAQF(_n)  (0x05980 + (4 * (_n))) /* Source Address Queue Fltr */
#define E1000_DAQF(_n)  (0x059A0 + (4 * (_n))) /* Dest Address Queue Fltr */
#define E1000_SPQF(_n)  (0x059C0 + (4 * (_n))) /* Source Port Queue Fltr */
#define E1000_FTQF(_n)  (0x059E0 + (4 * (_n))) /* 5-tuple Queue Fltr */
#define E1000_TTQF(_n)  (0x059E0 + (4 * (_n))) /* 2-tuple Queue Fltr */
#define E1000_SYNQF(_n) (0x055FC + (4 * (_n))) /* SYN Packet Queue Fltr */
#define E1000_ETQF(_n)  (0x05CB0 + (4 * (_n))) /* EType Queue Fltr */

#define E1000_RTTDCS    0x3600 /* Reedtown Tx Desc plane control and status */
#define E1000_RTTPCS    0x3474 /* Reedtown Tx Packet Plane control and status */
#define E1000_RTRPCS    0x2474 /* Rx packet plane control and status */
#define E1000_RTRUP2TC  0x05AC4 /* Rx User Priority to Traffic Class */
#define E1000_RTTUP2TC  0x0418 /* Transmit User Priority to Traffic Class */
/* Tx Desc plane TC Rate-scheduler config */
#define E1000_RTTDTCRC(_n)      (0x3610 + ((_n) * 4))
/* Tx Packet plane TC Rate-Scheduler Config */
#define E1000_RTTPTCRC(_n)      (0x3480 + ((_n) * 4))
/* Rx Packet plane TC Rate-Scheduler Config */
#define E1000_RTRPTCRC(_n)      (0x2480 + ((_n) * 4))
/* Tx Desc Plane TC Rate-Scheduler Status */
#define E1000_RTTDTCRS(_n)      (0x3630 + ((_n) * 4))
/* Tx Desc Plane TC Rate-Scheduler MMW */
#define E1000_RTTDTCRM(_n)      (0x3650 + ((_n) * 4))
/* Tx Packet plane TC Rate-Scheduler Status */
#define E1000_RTTPTCRS(_n)      (0x34A0 + ((_n) * 4))
/* Tx Packet plane TC Rate-scheduler MMW */
#define E1000_RTTPTCRM(_n)      (0x34C0 + ((_n) * 4))
/* Rx Packet plane TC Rate-Scheduler Status */
#define E1000_RTRPTCRS(_n)      (0x24A0 + ((_n) * 4))
/* Rx Packet plane TC Rate-Scheduler MMW */
#define E1000_RTRPTCRM(_n)      (0x24C0 + ((_n) * 4))
/* Tx Desc plane VM Rate-Scheduler MMW*/
#define E1000_RTTDVMRM(_n)      (0x3670 + ((_n) * 4))
/* Tx BCN Rate-Scheduler MMW */
#define E1000_RTTBCNRM(_n)      (0x3690 + ((_n) * 4))
#define E1000_RTTDQSEL  0x3604  /* Tx Desc Plane Queue Select */
#define E1000_RTTDVMRC  0x3608  /* Tx Desc Plane VM Rate-Scheduler Config */
#define E1000_RTTDVMRS  0x360C  /* Tx Desc Plane VM Rate-Scheduler Status */
#define E1000_RTTBCNRC  0x36B0  /* Tx BCN Rate-Scheduler Config */
#define E1000_RTTBCNRS  0x36B4  /* Tx BCN Rate-Scheduler Status */
#define E1000_RTTBCNCR  0xB200  /* Tx BCN Control Register */
#define E1000_RTTBCNTG  0x35A4  /* Tx BCN Tagging */
#define E1000_RTTBCNCP  0xB208  /* Tx BCN Congestion point */
#define E1000_RTRBCNCR  0xB20C  /* Rx BCN Control Register */
#define E1000_RTTBCNRD  0x36B8  /* Tx BCN Rate Drift */
#define E1000_PFCTOP    0x1080  /* Priority Flow Control Type and Opcode */
#define E1000_RTTBCNIDX 0xB204  /* Tx BCN Congestion Point */
#define E1000_RTTBCNACH 0x0B214 /* Tx BCN Control High */
#define E1000_RTTBCNACL 0x0B210 /* Tx BCN Control Low */

/* DMA Coalescing registers */
#define E1000_DMACR     0x02508 /* Control Register */
#define E1000_DMCTXTH   0x03550 /* Transmit Threshold */
```

```
 655 #define E1000_DMCTLX    0x02514 /* Time to Lx Request */
 656 #define E1000_DMCRTRH   0x05DD0 /* Receive Packet Rate Threshold */
 657 #define E1000_DMCCNT    0x05DD4 /* Current Rx Count */
 658 #define E1000_FCRTC     0x02170 /* Flow Control Rx high watermark */
 659 #define E1000_PCIEMISC  0x05BB8 /* PCIE misc config register */

 661 /* PCIe Parity Status Register */
 662 #define E1000_PCIEERRSTS        0x05BA8

 664 #define E1000_PROXYS    0x5F64 /* Proxying Status */
 665 #define E1000_PROXYFC   0x5F60 /* Proxying Filter Control */
 666 /* Thermal sensor configuration and status registers */
 667 #define E1000_THMJT     0x08100 /* Junction Temperature */
 668 #define E1000_THLOWTC   0x08104 /* Low Threshold Control */
 669 #define E1000_THMIDTC   0x08108 /* Mid Threshold Control */
 670 #define E1000_THHIGHTC  0x0810C /* High Threshold Control */
 671 #define E1000_THSTAT    0x08110 /* Thermal Sensor Status */

 673 /* Energy Efficient Ethernet "EEE" registers */
 674 #define E1000_IPCNFG    0x0E38 /* Internal PHY Configuration */
 675 #define E1000_LTRC      0x01A0 /* Latency Tolerance Reporting Control */
 676 #define E1000_EEER      0x0E30 /* Energy Efficient Ethernet "EEE"*/
 677 #define E1000_EEE_SU    0x0E34 /* EEE Setup */
 678 #define E1000_TLPIC     0x4148 /* EEE Tx LPI Count - TLPIC */
 679 #define E1000_RLPIC     0x414C /* EEE Rx LPI Count - RLPIC */

 681 /* OS2BMC Registers */
 682 #define E1000_B2OSPC    0x08FE0 /* BMC2OS packets sent by BMC */
 683 #define E1000_B2OGPRC   0x04158 /* BMC2OS packets received by host */
 684 #define E1000_O2BGPTC   0x08FE4 /* OS2BMC packets received by BMC */
 685 #define E1000_O2BSPC    0x0415C /* OS2BMC packets transmitted by host */

 687 #define E1000_DOBFFCTL  0x3F24 /* DMA OBFF Control Register */


 690 #ifdef __cplusplus
 691 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   37464 Wed Feb 26 09:49:37 2014
new/usr/src/uts/common/io/igb/igb_gld.c
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright(c) 2007-2010 Intel Corporation. All rights reserved.
 */

/*
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2013, Nexenta Systems, Inc. All rights reserved.
 * Copyright 2014 Pluribus Networks Inc.
 */

#include "igb_sw.h"

int
igb_m_stat(void *arg, uint_t stat, uint64_t *val)
{
        igb_t *igb = (igb_t *)arg;
        struct e1000_hw *hw = &igb->hw;
        igb_stat_t *igb_ks;
        uint32_t low_val, high_val;

        igb_ks = (igb_stat_t *)igb->igb_ks->ks_data;

        mutex_enter(&igb->gen_lock);

        if (igb->igb_state & IGB_SUSPENDED) {
                mutex_exit(&igb->gen_lock);
                return (ECANCELED);
        }

        switch (stat) {
        case MAC_STAT_IFSPEED:
                *val = igb->link_speed * 1000000ull;
                break;

        case MAC_STAT_MULTIRCV:
                igb->stat_mprc += E1000_READ_REG(hw, E1000_MPRC);
                *val = igb->stat_mprc;
                igb_ks->mprc.value.ui64 +=
                    E1000_READ_REG(hw, E1000_MPRC);
```



```
                *val = igb_ks->mprc.value.ui64;
                break;

        case MAC_STAT_BRDCSTRCV:
                igb->stat_bprc += E1000_READ_REG(hw, E1000_BPRC);
                *val = igb->stat_bprc;
                igb_ks->bprc.value.ui64 +=
                    E1000_READ_REG(hw, E1000_BPRC);
                *val = igb_ks->bprc.value.ui64;
                break;

        case MAC_STAT_MULTIXMT:
                igb->stat_mptc += E1000_READ_REG(hw, E1000_MPTC);
                *val = igb->stat_mptc;
                igb_ks->mptc.value.ui64 +=
                    E1000_READ_REG(hw, E1000_MPTC);
                *val = igb_ks->mptc.value.ui64;
                break;

        case MAC_STAT_BRDCSTXMT:
                igb->stat_bptc += E1000_READ_REG(hw, E1000_BPTC);
                *val = igb->stat_bptc;
                igb_ks->bptc.value.ui64 +=
                    E1000_READ_REG(hw, E1000_BPTC);
                *val = igb_ks->bptc.value.ui64;
                break;

        case MAC_STAT_NORCVBUF:
                igb->stat_rnbc += E1000_READ_REG(hw, E1000_RNBC);
                *val = igb->stat_rnbc;
                igb_ks->rnbc.value.ui64 +=
                    E1000_READ_REG(hw, E1000_RNBC);
                *val = igb_ks->rnbc.value.ui64;
                break;

        case MAC_STAT_IERRORS:
                igb->stat_rxerrc += E1000_READ_REG(hw, E1000_RXERRC);
                igb->stat_algnerrc += E1000_READ_REG(hw, E1000_ALGNERRC);
                igb_ks->rxerrc.value.ui64 +=
                    E1000_READ_REG(hw, E1000_RXERRC);
                igb_ks->algnerrc.value.ui64 +=
                    E1000_READ_REG(hw, E1000_ALGNERRC);
                igb_ks->rlec.value.ui64 +=
                    E1000_READ_REG(hw, E1000_RLEC);
                igb->stat_crcerrs += E1000_READ_REG(hw, E1000_CRCERRS);
                igb->stat_cexterr += E1000_READ_REG(hw, E1000_CEXTERR);
                *val = igb->stat_rxerrc +
                    igb->stat_algnerrc +
                igb_ks->crcerrs.value.ui64 +=
                    E1000_READ_REG(hw, E1000_CRCERRS);
                igb_ks->cexterr.value.ui64 +=
                    E1000_READ_REG(hw, E1000_CEXTERR);
                *val = igb_ks->rxerrc.value.ui64 +
                    igb_ks->algnerrc.value.ui64 +
                    igb_ks->rlec.value.ui64 +
                    igb->stat_crcerrs +
                    igb->stat_cexterr;
                    igb_ks->crcerrs.value.ui64 +
                    igb_ks->cexterr.value.ui64;
                break;

        case MAC_STAT_NOXMTBUF:
                *val = 0;
                break;

        case MAC_STAT_OERRORS:
```

```
100                 igb->stat_ecol += E1000_READ_REG(hw, E1000_ECOL);
101                 *val = igb->stat_ecol;
108                 igb_ks->ecol.value.ui64 +=
109                     E1000_READ_REG(hw, E1000_ECOL);
110                 *val = igb_ks->ecol.value.ui64;
102                 break;

104         case MAC_STAT_COLLISIONS:
105                 igb->stat_colc += E1000_READ_REG(hw, E1000_COLC);
106                 *val = igb->stat_colc;
114                 igb_ks->colc.value.ui64 +=
115                     E1000_READ_REG(hw, E1000_COLC);
116                 *val = igb_ks->colc.value.ui64;
107                 break;

109         case MAC_STAT_RBYTES:
110                 /*
111                  * The 64-bit register will reset whenever the upper
112                  * 32 bits are read. So we need to read the lower
113                  * 32 bits first, then read the upper 32 bits.
114                  */
115                 low_val = E1000_READ_REG(hw, E1000_TORL);
116                 high_val = E1000_READ_REG(hw, E1000_TORH);
117                 igb->stat_tor += (uint64_t)high_val << 32 | (uint64_t)low_val;
118                 *val = igb->stat_tor;
127                 igb_ks->tor.value.ui64 +=
128                     (uint64_t)high_val << 32 | (uint64_t)low_val;
129                 *val = igb_ks->tor.value.ui64;
119                 break;

121         case MAC_STAT_IPACKETS:
122                 igb->stat_tpr += E1000_READ_REG(hw, E1000_TPR);
123                 *val = igb->stat_tpr;
133                 igb_ks->tpr.value.ui64 +=
134                     E1000_READ_REG(hw, E1000_TPR);
135                 *val = igb_ks->tpr.value.ui64;
124                 break;

126         case MAC_STAT_OBYTES:
127                 /*
128                  * The 64-bit register will reset whenever the upper
129                  * 32 bits are read. So we need to read the lower
130                  * 32 bits first, then read the upper 32 bits.
131                  */
132                 low_val = E1000_READ_REG(hw, E1000_TOTL);
133                 high_val = E1000_READ_REG(hw, E1000_TOTH);
134                 igb->stat_tot += (uint64_t)high_val << 32 | (uint64_t)low_val;
135                 *val = igb->stat_tot;
146                 igb_ks->tot.value.ui64 +=
147                     (uint64_t)high_val << 32 | (uint64_t)low_val;
148                 *val = igb_ks->tot.value.ui64;
136                 break;

138         case MAC_STAT_OPACKETS:
139                 igb->stat_tpt += E1000_READ_REG(hw, E1000_TPT);
140                 *val = igb->stat_tpt;
152                 igb_ks->tpt.value.ui64 +=
153                     E1000_READ_REG(hw, E1000_TPT);
154                 *val = igb_ks->tpt.value.ui64;
141                 break;

143         /* RFC 1643 stats */
144         case ETHER_STAT_ALIGN_ERRORS:
145                 igb->stat_algnerrc += E1000_READ_REG(hw, E1000_ALGNERRC);
146                 *val = igb->stat_algnerrc;
159                 igb_ks->algnerrc.value.ui64 +=
```



```
160                     E1000_READ_REG(hw, E1000_ALGNERRC);
161                 *val = igb_ks->algnerrc.value.ui64;
147                 break;

149         case ETHER_STAT_FCS_ERRORS:
150                 igb->stat_crcerrs += E1000_READ_REG(hw, E1000_CRCERRS);
151                 *val = igb->stat_crcerrs;
165                 igb_ks->crcerrs.value.ui64 +=
166                     E1000_READ_REG(hw, E1000_CRCERRS);
167                 *val = igb_ks->crcerrs.value.ui64;
152                 break;

154         case ETHER_STAT_FIRST_COLLISIONS:
155                 igb->stat_scc += E1000_READ_REG(hw, E1000_SCC);
156                 *val = igb->stat_scc;
171                 igb_ks->scc.value.ui64 +=
172                     E1000_READ_REG(hw, E1000_SCC);
173                 *val = igb_ks->scc.value.ui64;
157                 break;

159         case ETHER_STAT_MULTI_COLLISIONS:
160                 igb->stat_mcc += E1000_READ_REG(hw, E1000_MCC);
161                 *val = igb->stat_mcc;
177                 igb_ks->mcc.value.ui64 +=
178                     E1000_READ_REG(hw, E1000_MCC);
179                 *val = igb_ks->mcc.value.ui64;
162                 break;

164         case ETHER_STAT_SQE_ERRORS:
165                 igb->stat_sec += E1000_READ_REG(hw, E1000_SEC);
166                 *val = igb->stat_sec;
183                 igb_ks->sec.value.ui64 +=
184                     E1000_READ_REG(hw, E1000_SEC);
185                 *val = igb_ks->sec.value.ui64;
167                 break;

169         case ETHER_STAT_DEFER_XMTS:
170                 igb->stat_dc += E1000_READ_REG(hw, E1000_DC);
171                 *val = igb->stat_dc;
189                 igb_ks->dc.value.ui64 +=
190                     E1000_READ_REG(hw, E1000_DC);
191                 *val = igb_ks->dc.value.ui64;
172                 break;

174         case ETHER_STAT_TX_LATE_COLLISIONS:
175                 igb->stat_latecol += E1000_READ_REG(hw, E1000_LATECOL);
176                 *val = igb->stat_latecol;
195                 igb_ks->latecol.value.ui64 +=
196                     E1000_READ_REG(hw, E1000_LATECOL);
197                 *val = igb_ks->latecol.value.ui64;
177                 break;

179         case ETHER_STAT_EX_COLLISIONS:
180                 igb->stat_ecol += E1000_READ_REG(hw, E1000_ECOL);
181                 *val = igb->stat_ecol;
201                 igb_ks->ecol.value.ui64 +=
202                     E1000_READ_REG(hw, E1000_ECOL);
203                 *val = igb_ks->ecol.value.ui64;
182                 break;

184         case ETHER_STAT_MACXMT_ERRORS:
185                 igb->stat_ecol += E1000_READ_REG(hw, E1000_ECOL);
186                 *val = igb->stat_ecol;
207                 igb_ks->ecol.value.ui64 +=
208                     E1000_READ_REG(hw, E1000_ECOL);
209                 *val = igb_ks->ecol.value.ui64;
```

```
187                 break;

189         case ETHER_STAT_CARRIER_ERRORS:
190                 igb->stat_cexterr += E1000_READ_REG(hw, E1000_CEXTERR);
191                 *val = igb->stat_cexterr;
213                 igb_ks->cexterr.value.ui64 +=
214                     E1000_READ_REG(hw, E1000_CEXTERR);
215                 *val = igb_ks->cexterr.value.ui64;
192                 break;

194         case ETHER_STAT_TOOLONG_ERRORS:
195                 igb->stat_roc += E1000_READ_REG(hw, E1000_ROC);
196                 *val = igb->stat_roc;
219                 igb_ks->roc.value.ui64 +=
220                     E1000_READ_REG(hw, E1000_ROC);
221                 *val = igb_ks->roc.value.ui64;
197                 break;

199         case ETHER_STAT_MACRCV_ERRORS:
200                 igb->stat_rxerrc += E1000_READ_REG(hw, E1000_RXERRC);
201                 *val = igb->stat_rxerrc;
225                 igb_ks->rxerrc.value.ui64 +=
226                     E1000_READ_REG(hw, E1000_RXERRC);
227                 *val = igb_ks->rxerrc.value.ui64;
202                 break;

204         /* MII/GMII stats */
205         case ETHER_STAT_XCVR_ADDR:
206                 /* The Internal PHY's MDI address for each MAC is 1 */
207                 *val = 1;
208                 break;

210         case ETHER_STAT_XCVR_ID:
211                 *val = hw->phy.id | hw->phy.revision;
212                 break;

214         case ETHER_STAT_XCVR_INUSE:
215                 switch (igb->link_speed) {
216                 case SPEED_1000:
217                         *val =
218                             (hw->phy.media_type == e1000_media_type_copper) ?
219                             XCVR_1000T : XCVR_1000X;
220                         break;
221                 case SPEED_100:
222                         *val =
223                             (hw->phy.media_type == e1000_media_type_copper) ?
224                             (igb->param_100t4_cap == 1) ?
225                             XCVR_100T4 : XCVR_100T2 : XCVR_100X;
226                         break;
227                 case SPEED_10:
228                         *val = XCVR_10;
229                         break;
230                 default:
231                         *val = XCVR_NONE;
232                         break;
233                 }
234                 break;

236         case ETHER_STAT_CAP_1000FDX:
237                 *val = igb->param_1000fdx_cap;
238                 break;

240         case ETHER_STAT_CAP_1000HDX:
241                 *val = igb->param_1000hdx_cap;
242                 break;
```



```
244         case ETHER_STAT_CAP_100FDX:
245                 *val = igb->param_100fdx_cap;
246                 break;

248         case ETHER_STAT_CAP_100HDX:
249                 *val = igb->param_100hdx_cap;
250                 break;

252         case ETHER_STAT_CAP_10FDX:
253                 *val = igb->param_10fdx_cap;
254                 break;

256         case ETHER_STAT_CAP_10HDX:
257                 *val = igb->param_10hdx_cap;
258                 break;

260         case ETHER_STAT_CAP_ASMPAUSE:
261                 *val = igb->param_asym_pause_cap;
262                 break;

264         case ETHER_STAT_CAP_PAUSE:
265                 *val = igb->param_pause_cap;
266                 break;

268         case ETHER_STAT_CAP_AUTONEG:
269                 *val = igb->param_autoneg_cap;
270                 break;

272         case ETHER_STAT_ADV_CAP_1000FDX:
273                 *val = igb->param_adv_1000fdx_cap;
274                 break;

276         case ETHER_STAT_ADV_CAP_1000HDX:
277                 *val = igb->param_adv_1000hdx_cap;
278                 break;

280         case ETHER_STAT_ADV_CAP_100FDX:
281                 *val = igb->param_adv_100fdx_cap;
282                 break;

284         case ETHER_STAT_ADV_CAP_100HDX:
285                 *val = igb->param_adv_100hdx_cap;
286                 break;

288         case ETHER_STAT_ADV_CAP_10FDX:
289                 *val = igb->param_adv_10fdx_cap;
290                 break;

292         case ETHER_STAT_ADV_CAP_10HDX:
293                 *val = igb->param_adv_10hdx_cap;
294                 break;

296         case ETHER_STAT_ADV_CAP_ASMPAUSE:
297                 *val = igb->param_adv_asym_pause_cap;
298                 break;

300         case ETHER_STAT_ADV_CAP_PAUSE:
301                 *val = igb->param_adv_pause_cap;
302                 break;

304         case ETHER_STAT_ADV_CAP_AUTONEG:
305                 *val = hw->mac.autoneg;
306                 break;

308         case ETHER_STAT_LP_CAP_1000FDX:
309                 *val = igb->param_lp_1000fdx_cap;
```

```
310                 break;

312         case ETHER_STAT_LP_CAP_1000HDX:
313                 *val = igb->param_lp_1000hdx_cap;
314                 break;

316         case ETHER_STAT_LP_CAP_100FDX:
317                 *val = igb->param_lp_100fdx_cap;
318                 break;

320         case ETHER_STAT_LP_CAP_100HDX:
321                 *val = igb->param_lp_100hdx_cap;
322                 break;

324         case ETHER_STAT_LP_CAP_10FDX:
325                 *val = igb->param_lp_10fdx_cap;
326                 break;

328         case ETHER_STAT_LP_CAP_10HDX:
329                 *val = igb->param_lp_10hdx_cap;
330                 break;

332         case ETHER_STAT_LP_CAP_ASMPAUSE:
333                 *val = igb->param_lp_asym_pause_cap;
334                 break;

336         case ETHER_STAT_LP_CAP_PAUSE:
337                 *val = igb->param_lp_pause_cap;
338                 break;

340         case ETHER_STAT_LP_CAP_AUTONEG:
341                 *val = igb->param_lp_autoneg_cap;
342                 break;

344         case ETHER_STAT_LINK_ASMPAUSE:
345                 *val = igb->param_asym_pause_cap;
346                 break;

348         case ETHER_STAT_LINK_PAUSE:
349                 *val = igb->param_pause_cap;
350                 break;

352         case ETHER_STAT_LINK_AUTONEG:
353                 *val = hw->mac.autoneg;
354                 break;

356         case ETHER_STAT_LINK_DUPLEX:
357                 *val = (igb->link_duplex == FULL_DUPLEX) ?
358                     LINK_DUPLEX_FULL : LINK_DUPLEX_HALF;
359                 break;

361         case ETHER_STAT_TOOSHORT_ERRORS:
362                 igb->stat_ruc += E1000_READ_REG(hw, E1000_RUC);
363                 *val = igb->stat_ruc;
388                 igb_ks->ruc.value.ui64 +=
389                     E1000_READ_REG(hw, E1000_RUC);
390                 *val = igb_ks->ruc.value.ui64;
364                 break;

366         case ETHER_STAT_CAP_REMFAULT:
367                 *val = igb->param_rem_fault;
368                 break;

370         case ETHER_STAT_ADV_REMFAULT:
371                 *val = igb->param_adv_rem_fault;
372                 break;
```



```
374         case ETHER_STAT_LP_REMFAULT:
375                 *val = igb->param_lp_rem_fault;
376                 break;

378         case ETHER_STAT_JABBER_ERRORS:
379                 igb->stat_rjc += E1000_READ_REG(hw, E1000_RJC);
380                 *val = igb->stat_rjc;
406                 igb_ks->rjc.value.ui64 +=
407                     E1000_READ_REG(hw, E1000_RJC);
408                 *val = igb_ks->rjc.value.ui64;
381                 break;

383         case ETHER_STAT_CAP_100T4:
384                 *val = igb->param_100t4_cap;
385                 break;

387         case ETHER_STAT_ADV_CAP_100T4:
388                 *val = igb->param_adv_100t4_cap;
389                 break;

391         case ETHER_STAT_LP_CAP_100T4:
392                 *val = igb->param_lp_100t4_cap;
393                 break;

395         default:
396                 mutex_exit(&igb->gen_lock);
397                 return (ENOTSUP);
398         }

400         mutex_exit(&igb->gen_lock);

402         if (igb_check_acc_handle(igb->osdep.reg_handle) != DDI_FM_OK) {
403                 ddi_fm_service_impact(igb->dip, DDI_SERVICE_DEGRADED);
404                 return (EIO);
405         }

407         return (0);
408 }
```

*unchanged_portion_omitted_*

```
1351 /* ARGSUSED */
1352 int
1353 igb_set_priv_prop(igb_t *igb, const char *pr_name,
1354     uint_t pr_valsize, const void *pr_val)
1355 {
1356         int err = 0;
1357         long result;
1358         struct e1000_hw *hw = &igb->hw;
1359         int i;

1361         if (strcmp(pr_name, "_eee_support") == 0) {
1362                 if (pr_val == NULL)
1363                         return (EINVAL);
1364                 (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
1365                 switch (result) {
1366                 case 0:
1367                 case 1:
1396                         if (hw->mac.type != e1000_i350) {
1368                         /*
1369                          * For now, only supported on I350/I354.
1398                                 * For now, only supported on I350.
1370                          * Add new mac.type values (or use < instead)
1371                          * as new cards offer up EEE.
1372                          */
1373                         switch (hw->mac.type) {
```

```
                        case e1000_i350:
                                return (ENXIO);
                        }
                                /* Must set this prior to the set call. */
                                hw->dev_spec._82575.eee_disable = !result;
                                if (e1000_set_eee_i350(hw) != E1000_SUCCESS)
                                        err = EIO;
                                break;
                        case e1000_i354:
                                /* Must set this prior to the set call. */
                                hw->dev_spec._82575.eee_disable = !result;
                                if (e1000_set_eee_i354(hw) != E1000_SUCCESS)
                                        err = EIO;
                                break;
                        default:
                                return (ENXIO);
                        }
                        break;
                default:
                        err = EINVAL;
                        /* FALLTHRU */
                }
                return (err);
        }
        if (strcmp(pr_name, "_tx_copy_thresh") == 0) {
                if (pr_val == NULL) {
                        err = EINVAL;
                        return (err);
                }
                (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
                if (result < MIN_TX_COPY_THRESHOLD ||
                    result > MAX_TX_COPY_THRESHOLD)
                        err = EINVAL;
                else {
                        igb->tx_copy_thresh = (uint32_t)result;
                }
                return (err);
        }
        if (strcmp(pr_name, "_tx_recycle_thresh") == 0) {
                if (pr_val == NULL) {
                        err = EINVAL;
                        return (err);
                }
                (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
                if (result < MIN_TX_RECYCLE_THRESHOLD ||
                    result > MAX_TX_RECYCLE_THRESHOLD)
                        err = EINVAL;
                else {
                        igb->tx_recycle_thresh = (uint32_t)result;
                }
                return (err);
        }
        if (strcmp(pr_name, "_tx_overload_thresh") == 0) {
                if (pr_val == NULL) {
                        err = EINVAL;
                        return (err);
                }
                (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
                if (result < MIN_TX_OVERLOAD_THRESHOLD ||
                    result > MAX_TX_OVERLOAD_THRESHOLD)
                        err = EINVAL;
                else {
                        igb->tx_overload_thresh = (uint32_t)result;
                }
                return (err);
        }
```



```
        if (strcmp(pr_name, "_tx_resched_thresh") == 0) {
                if (pr_val == NULL) {
                        err = EINVAL;
                        return (err);
                }
                (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
                if (result < MIN_TX_RESCHED_THRESHOLD ||
                    result > MAX_TX_RESCHED_THRESHOLD ||
                    result > igb->tx_ring_size)
                        err = EINVAL;
                else {
                        igb->tx_resched_thresh = (uint32_t)result;
                }
                return (err);
        }
        if (strcmp(pr_name, "_rx_copy_thresh") == 0) {
                if (pr_val == NULL) {
                        err = EINVAL;
                        return (err);
                }
                (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
                if (result < MIN_RX_COPY_THRESHOLD ||
                    result > MAX_RX_COPY_THRESHOLD)
                        err = EINVAL;
                else {
                        igb->rx_copy_thresh = (uint32_t)result;
                }
                return (err);
        }
        if (strcmp(pr_name, "_rx_limit_per_intr") == 0) {
                if (pr_val == NULL) {
                        err = EINVAL;
                        return (err);
                }
                (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);
                if (result < MIN_RX_LIMIT_PER_INTR ||
                    result > MAX_RX_LIMIT_PER_INTR)
                        err = EINVAL;
                else {
                        igb->rx_limit_per_intr = (uint32_t)result;
                }
                return (err);
        }
        if (strcmp(pr_name, "_intr_throttling") == 0) {
                if (pr_val == NULL) {
                        err = EINVAL;
                        return (err);
                }
                (void) ddi_strtol(pr_val, (char **)NULL, 0, &result);

                if (result < igb->capab->min_intr_throttle ||
                    result > igb->capab->max_intr_throttle)
                        err = EINVAL;
                else {
                        igb->intr_throttling[0] = (uint32_t)result;

                        for (i = 0; i < MAX_NUM_EITR; i++)
                                igb->intr_throttling[i] =
                                    igb->intr_throttling[0];

                        /* Set interrupt throttling rate */
                        for (i = 0; i < igb->intr_cnt; i++)
                                E1000_WRITE_REG(hw, E1000_EITR(i),
                                    igb->intr_throttling[i]);
                }
                return (err);
```

```
1504         }
1505         return (ENOTSUP);
1506 }

1508 int
1509 igb_get_priv_prop(igb_t *igb, const char *pr_name, uint_t pr_valsize,
1510     void *pr_val)
1511 {
1512         int value;

1514         if (strcmp(pr_name, "_adv_pause_cap") == 0) {
1515                 value = igb->param_adv_pause_cap;
1516         } else if (strcmp(pr_name, "_adv_asym_pause_cap") == 0) {
1517                 value = igb->param_adv_asym_pause_cap;
1518         } else if (strcmp(pr_name, "_eee_support") == 0) {
1519                 /*
1520                  * For now, only supported on I350.  Add new mac.type values
1521                  * (or use < instead) as new cards offer up EEE.
1522                  */
1523                 switch (igb->hw.mac.type) {
1524                 case e1000_i350:
1525                 case e1000_i354:
1526                         value = !(igb->hw.dev_spec._82575.eee_disable);
1527                         break;
1528                 default:
1529                         value = 0;
1530                 }
1542                 value = (igb->hw.mac.type != e1000_i350) ? 0 :
1543                     !(igb->hw.dev_spec._82575.eee_disable);
1531         } else if (strcmp(pr_name, "_tx_copy_thresh") == 0) {
1532                 value = igb->tx_copy_thresh;
1533         } else if (strcmp(pr_name, "_tx_recycle_thresh") == 0) {
1534                 value = igb->tx_recycle_thresh;
1535         } else if (strcmp(pr_name, "_tx_overload_thresh") == 0) {
1536                 value = igb->tx_overload_thresh;
1537         } else if (strcmp(pr_name, "_tx_resched_thresh") == 0) {
1538                 value = igb->tx_resched_thresh;
1539         } else if (strcmp(pr_name, "_rx_copy_thresh") == 0) {
1540                 value = igb->rx_copy_thresh;
1541         } else if (strcmp(pr_name, "_rx_limit_per_intr") == 0) {
1542                 value = igb->rx_limit_per_intr;
1543         } else if (strcmp(pr_name, "_intr_throttling") == 0) {
1544                 value = igb->intr_throttling[0];
1545         } else {
1546                 return (ENOTSUP);
1547         }

1549         (void) snprintf(pr_val, pr_valsize, "%d", value);
1550         return (0);
1551 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
  130303 Wed Feb 26 09:49:37 2014
new/usr/src/uts/common/io/igb/igb_main.c
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
______unchanged_portion_omitted_

static adapter_info_t igb_i354_cap = {
        /* limits */
        8,              /* maximum number of rx queues */
        1,              /* minimum number of rx queues */
        4,              /* default number of rx queues */
        8,              /* maximum number of tx queues */
        1,              /* minimum number of tx queues */
        4,              /* default number of tx queues */
        65535,          /* maximum interrupt throttle rate */
        0,              /* minimum interrupt throttle rate */
        200,            /* default interrupt throttle rate */

        /* function pointers */
        igb_enable_adapter_interrupts_82580,
        igb_setup_msix_82580,

        /* capabilities */
        (IGB_FLAG_HAS_DCA |     /* capability flags */
        IGB_FLAG_VMDQ_POOL |
        IGB_FLAG_NEED_CTX_IDX),

        0xfff00000              /* mask for RXDCTL register */
};

/*
 * Module Initialization Functions
 */

int
_init(void)
{
        int status;

        mac_init_ops(&igb_dev_ops, MODULE_NAME);

        status = mod_install(&igb_modlinkage);

        if (status != DDI_SUCCESS) {
                mac_fini_ops(&igb_dev_ops);
        }

        return (status);
}
______unchanged_portion_omitted_

/*
 * igb_attach - driver attach
 *
 * This function is the device specific initialization entry
 * point. This entry point is required and must be written.
 * The DDI_ATTACH command must be provided in the attach entry
 * point. When attach() is called with cmd set to DDI_ATTACH,
 * all normal kernel services (such as kmem_alloc(9F)) are
 * available for use by the driver.
 *
 * The attach() function will be called once for each instance
 * of  the  device  on  the  system with cmd set to DDI_ATTACH.
 * Until attach() succeeds, the only driver entry points which
```



```
 * may be called are open(9E) and getinfo(9E).
 */
static int
igb_attach(dev_info_t *devinfo, ddi_attach_cmd_t cmd)
{
        igb_t *igb;
        struct igb_osdep *osdep;
        struct e1000_hw *hw;
        int instance;

        /*
         * Check the command and perform corresponding operations
         */
        switch (cmd) {
        default:
                return (DDI_FAILURE);

        case DDI_RESUME:
                return (igb_resume(devinfo));

        case DDI_ATTACH:
                break;
        }

        /* Get the device instance */
        instance = ddi_get_instance(devinfo);

        /* Allocate memory for the instance data structure */
        igb = kmem_zalloc(sizeof (igb_t), KM_SLEEP);

        igb->dip = devinfo;
        igb->instance = instance;

        hw = &igb->hw;
        osdep = &igb->osdep;
        hw->back = osdep;
        osdep->igb = igb;

        /* Attach the instance pointer to the dev_info data structure */
        ddi_set_driver_private(devinfo, igb);


        /* Initialize for fma support */
        igb->fm_capabilities = igb_get_prop(igb, "fm-capable",
            0, 0x0f,
            DDI_FM_EREPORT_CAPABLE | DDI_FM_ACCCHK_CAPABLE |
            DDI_FM_DMACHK_CAPABLE | DDI_FM_ERRCB_CAPABLE);
        igb_fm_init(igb);
        igb->attach_progress |= ATTACH_PROGRESS_FMINIT;

        /*
         * Map PCI config space registers
         */
        if (pci_config_setup(devinfo, &osdep->cfg_handle) != DDI_SUCCESS) {
                igb_error(igb, "Failed to map PCI configurations");
                goto attach_fail;
        }
        igb->attach_progress |= ATTACH_PROGRESS_PCI_CONFIG;

        /*
         * Identify the chipset family
         */
        if (igb_identify_hardware(igb) != IGB_SUCCESS) {
                igb_error(igb, "Failed to identify hardware");
                goto attach_fail;
        }
```

```
487         /*
488          * Map device registers
489          */
490         if (igb_regs_map(igb) != IGB_SUCCESS) {
491                 igb_error(igb, "Failed to map device registers");
492                 goto attach_fail;
493         }
494         igb->attach_progress |= ATTACH_PROGRESS_REGS_MAP;

496         /*
497          * Initialize driver parameters
498          */
499         igb_init_properties(igb);
500         igb->attach_progress |= ATTACH_PROGRESS_PROPS;

502         /*
503          * Allocate interrupts
504          */
505         if (igb_alloc_intrs(igb) != IGB_SUCCESS) {
506                 igb_error(igb, "Failed to allocate interrupts");
507                 goto attach_fail;
508         }
509         igb->attach_progress |= ATTACH_PROGRESS_ALLOC_INTR;

511         /*
512          * Allocate rx/tx rings based on the ring numbers.
513          * The actual numbers of rx/tx rings are decided by the number of
514          * allocated interrupt vectors, so we should allocate the rings after
515          * interrupts are allocated.
516          */
517         if (igb_alloc_rings(igb) != IGB_SUCCESS) {
518                 igb_error(igb, "Failed to allocate rx/tx rings or groups");
519                 goto attach_fail;
520         }
521         igb->attach_progress |= ATTACH_PROGRESS_ALLOC_RINGS;

523         /*
524          * Add interrupt handlers
525          */
526         if (igb_add_intr_handlers(igb) != IGB_SUCCESS) {
527                 igb_error(igb, "Failed to add interrupt handlers");
528                 goto attach_fail;
529         }
530         igb->attach_progress |= ATTACH_PROGRESS_ADD_INTR;

532         /*
533          * Initialize driver parameters
534          */
535         if (igb_init_driver_settings(igb) != IGB_SUCCESS) {
536                 igb_error(igb, "Failed to initialize driver settings");
537                 goto attach_fail;
538         }

540         if (igb_check_acc_handle(igb->osdep.cfg_handle) != DDI_FM_OK) {
541                 ddi_fm_service_impact(igb->dip, DDI_SERVICE_LOST);
542                 goto attach_fail;
543         }

545         /*
546          * Initialize mutexes for this device.
547          * Do this before enabling the interrupt handler and
548          * register the softint to avoid the condition where
549          * interrupt handler can try using uninitialized mutex
550          */
551         igb_init_locks(igb);
```



```
552         igb->attach_progress |= ATTACH_PROGRESS_LOCKS;

554         /*
555          * Initialize the adapter
556          */
557         if (igb_init(igb) != IGB_SUCCESS) {
558                 igb_error(igb, "Failed to initialize adapter");
559                 goto attach_fail;
560         }
561         igb->attach_progress |= ATTACH_PROGRESS_INIT_ADAPTER;

563         /*
564          * Initialize statistics
565          */
566         if (igb_init_stats(igb) != IGB_SUCCESS) {
567                 igb_error(igb, "Failed to initialize statistics");
568                 goto attach_fail;
569         }
570         igb->attach_progress |= ATTACH_PROGRESS_STATS;

572         /*
573          * Register the driver to the MAC
574          */
575         if (igb_register_mac(igb) != IGB_SUCCESS) {
576                 igb_error(igb, "Failed to register MAC");
577                 goto attach_fail;
578         }
579         igb->attach_progress |= ATTACH_PROGRESS_MAC;

581         /*
582          * Now that mutex locks are initialized, and the chip is also
583          * initialized, enable interrupts.
584          */
585         if (igb_enable_intrs(igb) != IGB_SUCCESS) {
586                 igb_error(igb, "Failed to enable DDI interrupts");
587                 goto attach_fail;
588         }
589         igb->attach_progress |= ATTACH_PROGRESS_ENABLE_INTR;

591         igb_log(igb, "%s", igb_version);
592         atomic_or_32(&igb->igb_state, IGB_INITIALIZED);

594         /*
595          * Newer models have Energy Efficient Ethernet, let's disable this by
596          * default.
597          */
598         if (igb->hw.mac.type == e1000_i350)
599                 (void) e1000_set_eee_i350(&igb->hw);
600         else if (igb->hw.mac.type == e1000_i354)
601                 (void) e1000_set_eee_i354(&igb->hw);

603         return (DDI_SUCCESS);

605 attach_fail:
606         igb_unconfigure(devinfo, igb);
607         return (DDI_FAILURE);
608 }
_____unchanged_portion_omitted_

879 /*
880  * igb_identify_hardware - Identify the type of the chipset
881  */
882 static int
883 igb_identify_hardware(igb_t *igb)
884 {
885         struct e1000_hw *hw = &igb->hw;
```

```
 886         struct igb_osdep *osdep = &igb->osdep;

 888         /*
 889          * Get the device id
 890          */
 891         hw->vendor_id =
 892             pci_config_get16(osdep->cfg_handle, PCI_CONF_VENID);
 893         hw->device_id =
 894             pci_config_get16(osdep->cfg_handle, PCI_CONF_DEVID);
 895         hw->revision_id =
 896             pci_config_get8(osdep->cfg_handle, PCI_CONF_REVID);
 897         hw->subsystem_device_id =
 898             pci_config_get16(osdep->cfg_handle, PCI_CONF_SUBSYSID);
 899         hw->subsystem_vendor_id =
 900             pci_config_get16(osdep->cfg_handle, PCI_CONF_SUBVENID);

 902         /*
 903          * Set the mac type of the adapter based on the device id
 904          */
 905         if (e1000_set_mac_type(hw) != E1000_SUCCESS) {
 906                 return (IGB_FAILURE);
 907         }

 909         /*
 910          * Install adapter capabilities based on mac type
 911          */
 912         switch (hw->mac.type) {
 913         case e1000_82575:
 914                 igb->capab = &igb_82575_cap;
 915                 break;
 916         case e1000_82576:
 917                 igb->capab = &igb_82576_cap;
 918                 break;
 919         case e1000_82580:
 920                 igb->capab = &igb_82580_cap;
 921                 break;
 922         case e1000_i350:
 923                 igb->capab = &igb_i350_cap;
 924                 break;
 925         case e1000_i210:
 926         case e1000_i211:
 927                 igb->capab = &igb_i210_cap;
 928                 break;
 929         case e1000_i354:
 930                 igb->capab = &igb_i354_cap;
 931                 break;
 932         default:
 933                 return (IGB_FAILURE);
 934         }

 936         return (IGB_SUCCESS);
 937 }
```

*unchanged_portion_omitted_*

```
1307 /*
1308  * igb_init_adapter - Initialize the adapter
1309  */
1310 static int
1311 igb_init_adapter(igb_t *igb)
1312 {
1313         struct e1000_hw *hw = &igb->hw;
1314         uint32_t pba;
1315         int oemid[2];
1316         uint16_t nvmword;
1317         uint32_t hwm;
1318         uint32_t default_mtu;
```



```
1319         u8 pbanum[E1000_PBANUM_LENGTH];
1320         char eepromver[5];      /* f.ff */
1321         int i;

1323         ASSERT(mutex_owned(&igb->gen_lock));

1325         /*
1326          * In order to obtain the default MAC address, this will reset the
1327          * adapter and validate the NVM that the address and many other
1328          * default settings come from.
1329          */
1330         if (igb_init_mac_address(igb) != IGB_SUCCESS) {
1331                 igb_error(igb, "Failed to initialize MAC address");
1332                 goto init_adapter_fail;
1333         }

1335         /*
1336          * Packet Buffer Allocation (PBA)
1337          * Writing PBA sets the receive portion of the buffer
1338          * the remainder is used for the transmit buffer.
1339          */
1340         switch (hw->mac.type) {
1341         case e1000_82575:
1342                 pba = E1000_PBA_32K;
1343                 break;
1344         case e1000_82576:
1345                 pba = E1000_READ_REG(hw, E1000_RXPBS);
1346                 pba &= E1000_RXPBS_SIZE_MASK_82576;
1347                 break;
1348         case e1000_82580:
1349         case e1000_i350:
1350         case e1000_i354:
1351                 pba = E1000_READ_REG(hw, E1000_RXPBS);
1352                 pba = e1000_rxpbs_adjust_82580(pba);
1353                 break;
1354         case e1000_i210:
1355         case e1000_i211:
1356                 pba = E1000_PBA_34K;
1357         default:
1358                 break;
1359         }

1361         /* Special needs in case of Jumbo frames */
1362         default_mtu = igb_get_prop(igb, PROP_DEFAULT_MTU,
1363             MIN_MTU, MAX_MTU, DEFAULT_MTU);
1364         if ((hw->mac.type == e1000_82575) && (default_mtu > ETHERMTU)) {
1365                 u32 tx_space, min_tx, min_rx;
1366                 pba = E1000_READ_REG(hw, E1000_PBA);
1367                 tx_space = pba >> 16;
1368                 pba &= 0xffff;
1369                 min_tx = (igb->max_frame_size +
1370                     sizeof (struct e1000_tx_desc) - ETHERNET_FCS_SIZE) * 2;
1371                 min_tx = roundup(min_tx, 1024);
1372                 min_tx >>= 10;
1373                 min_rx = igb->max_frame_size;
1374                 min_rx = roundup(min_rx, 1024);
1375                 min_rx >>= 10;
1376                 if (tx_space < min_tx &&
1377                     ((min_tx - tx_space) < pba)) {
1378                         pba = pba - (min_tx - tx_space);
1379                         /*
1380                          * if short on rx space, rx wins
1381                          * and must trump tx adjustment
1382                          */
1383                         if (pba < min_rx)
1384                                 pba = min_rx;
```

```
1385                }
1386                E1000_WRITE_REG(hw, E1000_PBA, pba);
1387        }

1389        DEBUGOUT1("igb_init: pba=%dK", pba);

1391        /*
1392         * These parameters control the automatic generation (Tx) and
1393         * response (Rx) to Ethernet PAUSE frames.
1394         * - High water mark should allow for at least two frames to be
1395         *   received after sending an XOFF.
1396         * - Low water mark works best when it is very near the high water mark.
1397         *   This allows the receiver to restart by sending XON when it has
1398         *   drained a bit.
1399         */
1400        hwm = min(((pba << 10) * 9 / 10),
1401            ((pba << 10) - 2 * igb->max_frame_size));

1403        if (hw->mac.type < e1000_82576) {
1404                hw->fc.high_water = hwm & 0xFFF8;  /* 8-byte granularity */
1405                hw->fc.low_water = hw->fc.high_water - 8;
1406        } else {
1407                hw->fc.high_water = hwm & 0xFFF0;  /* 16-byte granularity */
1408                hw->fc.low_water = hw->fc.high_water - 16;
1409        }

1411        hw->fc.pause_time = E1000_FC_PAUSE_TIME;
1412        hw->fc.send_xon = B_TRUE;

1414        (void) e1000_validate_mdi_setting(hw);

1416        /*
1417         * Reset the chipset hardware the second time to put PBA settings
1418         * into effect.
1419         */
1420        if (e1000_reset_hw(hw) != E1000_SUCCESS) {
1421                igb_error(igb, "Second reset failed");
1422                goto init_adapter_fail;
1423        }

1425        /*
1426         * Don't wait for auto-negotiation to complete
1427         */
1428        hw->phy.autoneg_wait_to_complete = B_FALSE;

1430        /*
1431         * Copper options
1432         */
1433        if (hw->phy.media_type == e1000_media_type_copper) {
1434                hw->phy.mdix = 0;       /* AUTO_ALL_MODES */
1435                hw->phy.disable_polarity_correction = B_FALSE;
1436                hw->phy.ms_type = e1000_ms_hw_default; /* E1000_MASTER_SLAVE */
1437        }

1439        /*
1440         * Initialize link settings
1441         */
1442        (void) igb_setup_link(igb, B_FALSE);

1444        /*
1445         * Configure/Initialize hardware
1446         */
1447        if (e1000_init_hw(hw) != E1000_SUCCESS) {
1448                igb_error(igb, "Failed to initialize hardware");
1449                goto init_adapter_fail;
1450        }
```



```
1452        /*
1453         *  Start the link setup timer
1454         */
1455        igb_start_link_timer(igb);

1457        /*
1458         * Disable wakeup control by default
1459         */
1460        E1000_WRITE_REG(hw, E1000_WUC, 0);

1462        /*
1463         * Record phy info in hw struct
1464         */
1465        (void) e1000_get_phy_info(hw);

1467        /*
1468         * Make sure driver has control
1469         */
1470        igb_get_driver_control(hw);

1472        /*
1473         * Restore LED settings to the default from EEPROM
1474         * to meet the standard for Sun platforms.
1475         */
1476        (void) e1000_cleanup_led(hw);

1478        /*
1479         * Setup MSI-X interrupts
1480         */
1481        if (igb->intr_type == DDI_INTR_TYPE_MSIX)
1482                igb->capab->setup_msix(igb);

1484        /*
1485         * Initialize unicast addresses.
1486         */
1487        igb_init_unicst(igb);

1489        /*
1490         * Setup and initialize the mctable structures.
1491         */
1492        igb_setup_multicst(igb);

1494        /*
1495         * Set interrupt throttling rate
1496         */
1497        for (i = 0; i < igb->intr_cnt; i++)
1498                E1000_WRITE_REG(hw, E1000_EITR(i), igb->intr_throttling[i]);

1500        /*
1501         * Read identifying information and place in devinfo.
1502         */
1503        nvmword = 0xffff;
1504        (void) e1000_read_nvm(&igb->hw, NVM_OEM_OFFSET_0, 1, &nvmword);
1505        oemid[0] = (int)nvmword;
1506        (void) e1000_read_nvm(&igb->hw, NVM_OEM_OFFSET_1, 1, &nvmword);
1507        oemid[1] = (int)nvmword;
1508        (void) ddi_prop_update_int_array(DDI_DEV_T_NONE, igb->dip,
1509            "oem-identifier", oemid, 2);

1511        pbanum[0] = '\0';
1512        (void) e1000_read_pba_string(&igb->hw, pbanum, sizeof (pbanum));
1513        if (*pbanum != '\0') {
1514                (void) ddi_prop_update_string(DDI_DEV_T_NONE, igb->dip,
1515                    "printed-board-assembly", (char *)pbanum);
1516        }
```

```
1518         nvmword = 0xffff;
1519         (void) e1000_read_nvm(&igb->hw, NVM_VERSION, 1, &nvmword);
1520         if ((nvmword & 0xf00) == 0) {
1521                 (void) snprintf(eepromver, sizeof (eepromver), "%x.%x",
1522                     (nvmword & 0xf000) >> 12, (nvmword & 0xff));
1523                 (void) ddi_prop_update_string(DDI_DEV_T_NONE, igb->dip,
1524                     "nvm-version", eepromver);
1525         }

1527         /*
1528          * Save the state of the phy
1529          */
1530         igb_get_phy_state(igb);

1532         igb_param_sync(igb);

1534         return (IGB_SUCCESS);

1536 init_adapter_fail:
1537         /*
1538          * Reset PHY if possible
1539          */
1540         if (e1000_check_reset_block(hw) == E1000_SUCCESS)
1541                 (void) e1000_phy_hw_reset(hw);

1543         return (IGB_FAILURE);
1544 }
```

unchanged_portion_omitted_

```
1805 /*
1806  * igb_start - Start the driver/chipset
1807  */
1808 int
1809 igb_start(igb_t *igb, boolean_t alloc_buffer)
1810 {
1811         int i;

1813         ASSERT(mutex_owned(&igb->gen_lock));

1815         if (alloc_buffer) {
1816                 if (igb_alloc_rx_data(igb) != IGB_SUCCESS) {
1817                         igb_error(igb,
1818                             "Failed to allocate software receive rings");
1819                         return (IGB_FAILURE);
1820                 }

1822                 /* Allocate buffers for all the rx/tx rings */
1823                 if (igb_alloc_dma(igb) != IGB_SUCCESS) {
1824                         igb_error(igb, "Failed to allocate DMA resource");
1825                         return (IGB_FAILURE);
1826                 }

1828                 igb->tx_ring_init = B_TRUE;
1829         } else {
1830                 igb->tx_ring_init = B_FALSE;
1831         }

1833         for (i = 0; i < igb->num_rx_rings; i++)
1834                 mutex_enter(&igb->rx_rings[i].rx_lock);
1835         for (i = 0; i < igb->num_tx_rings; i++)
1836                 mutex_enter(&igb->tx_rings[i].tx_lock);

1838         /*
1839          * Start the adapter
1840          */
```



```
1841         if ((igb->attach_progress & ATTACH_PROGRESS_INIT_ADAPTER) == 0) {
1842                 if (igb_init_adapter(igb) != IGB_SUCCESS) {
1843                         igb_fm_ereport(igb, DDI_FM_DEVICE_INVAL_STATE);
1844                         goto start_failure;
1845                 }
1846                 igb->attach_progress |= ATTACH_PROGRESS_INIT_ADAPTER;
1847         }

1849         /*
1850          * Setup the rx/tx rings
1851          */
1852         igb_setup_rings(igb);

1854         /*
1855          * Enable adapter interrupts
1856          * The interrupts must be enabled after the driver state is START
1857          */
1858         igb->capab->enable_intr(igb);

1860         if (igb_check_acc_handle(igb->osdep.cfg_handle) != DDI_FM_OK)
1861                 goto start_failure;

1863         if (igb_check_acc_handle(igb->osdep.reg_handle) != DDI_FM_OK)
1864                 goto start_failure;

1866         if (igb->hw.mac.type == e1000_i350)
1867                 (void) e1000_set_eee_i350(&igb->hw);
1868         else if (igb->hw.mac.type == e1000_i354)
1869                 (void) e1000_set_eee_i354(&igb->hw);

1871         for (i = igb->num_tx_rings - 1; i >= 0; i--)
1872                 mutex_exit(&igb->tx_rings[i].tx_lock);
1873         for (i = igb->num_rx_rings - 1; i >= 0; i--)
1874                 mutex_exit(&igb->rx_rings[i].rx_lock);

1876         return (IGB_SUCCESS);

1878 start_failure:
1879         for (i = igb->num_tx_rings - 1; i >= 0; i--)
1880                 mutex_exit(&igb->tx_rings[i].tx_lock);
1881         for (i = igb->num_rx_rings - 1; i >= 0; i--)
1882                 mutex_exit(&igb->rx_rings[i].rx_lock);

1884         ddi_fm_service_impact(igb->dip, DDI_SERVICE_LOST);

1886         return (IGB_FAILURE);
1887 }
```

unchanged_portion_omitted_

```
**********************************************************
    9697 Wed Feb 26 09:49:38 2014
new/usr/src/uts/common/io/igb/igb_stat.c
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
/*
 * CDDL HEADER START
 *
 * Copyright(c) 2007-2009 Intel Corporation. All rights reserved.
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2014 Pluribus Networks Inc.
 */

#include "igb_sw.h"

/*
 * Update driver private statistics.
 */
static int
igb_update_stats(kstat_t *ks, int rw)
{
        igb_t *igb;
        struct e1000_hw *hw;
        igb_stat_t *igb_ks;
        uint32_t val_low, val_high;
#ifdef IGB_DEBUG
        int i;
#endif

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

        igb = (igb_t *)ks->ks_private;
        igb_ks = (igb_stat_t *)ks->ks_data;
        hw = &igb->hw;

        mutex_enter(&igb->gen_lock);

        /*
         * Basic information.
         */
        igb_ks->link_speed.value.ui64 = igb->link_speed;
        igb_ks->reset_count.value.ui64 = igb->reset_count;
```



```
        igb_ks->dout_sync.value.ui64 = igb->dout_sync;

#ifdef IGB_DEBUG
        igb_ks->rx_frame_error.value.ui64 = 0;
        igb_ks->rx_cksum_error.value.ui64 = 0;
        igb_ks->rx_exceed_pkt.value.ui64 = 0;
        for (i = 0; i < igb->num_rx_rings; i++) {
                igb_ks->rx_frame_error.value.ui64 +=
                    igb->rx_rings[i].stat_frame_error;
                igb_ks->rx_cksum_error.value.ui64 +=
                    igb->rx_rings[i].stat_cksum_error;
                igb_ks->rx_exceed_pkt.value.ui64 +=
                    igb->rx_rings[i].stat_exceed_pkt;
        }

        igb_ks->tx_overload.value.ui64 = 0;
        igb_ks->tx_fail_no_tbd.value.ui64 = 0;
        igb_ks->tx_fail_no_tcb.value.ui64 = 0;
        igb_ks->tx_fail_dma_bind.value.ui64 = 0;
        igb_ks->tx_reschedule.value.ui64 = 0;
        for (i = 0; i < igb->num_tx_rings; i++) {
                igb_ks->tx_overload.value.ui64 +=
                    igb->tx_rings[i].stat_overload;
                igb_ks->tx_fail_no_tbd.value.ui64 +=
                    igb->tx_rings[i].stat_fail_no_tbd;
                igb_ks->tx_fail_no_tcb.value.ui64 +=
                    igb->tx_rings[i].stat_fail_no_tcb;
                igb_ks->tx_fail_dma_bind.value.ui64 +=
                    igb->tx_rings[i].stat_fail_dma_bind;
                igb_ks->tx_reschedule.value.ui64 +=
                    igb->tx_rings[i].stat_reschedule;
        }

        /*
         * Hardware calculated statistics.
         */
        igb_ks->gprc.value.ul += E1000_READ_REG(hw, E1000_GPRC);
        igb_ks->gptc.value.ul += E1000_READ_REG(hw, E1000_GPTC);
        igb_ks->prc64.value.ul += E1000_READ_REG(hw, E1000_PRC64);
        igb_ks->prc127.value.ul += E1000_READ_REG(hw, E1000_PRC127);
        igb_ks->prc255.value.ul += E1000_READ_REG(hw, E1000_PRC255);
        igb_ks->prc511.value.ul += E1000_READ_REG(hw, E1000_PRC511);
        igb_ks->prc1023.value.ul += E1000_READ_REG(hw, E1000_PRC1023);
        igb_ks->prc1522.value.ul += E1000_READ_REG(hw, E1000_PRC1522);
        igb_ks->ptc64.value.ul += E1000_READ_REG(hw, E1000_PTC64);
        igb_ks->ptc127.value.ul += E1000_READ_REG(hw, E1000_PTC127);
        igb_ks->ptc255.value.ul += E1000_READ_REG(hw, E1000_PTC255);
        igb_ks->ptc511.value.ul += E1000_READ_REG(hw, E1000_PTC511);
        igb_ks->ptc1023.value.ul += E1000_READ_REG(hw, E1000_PTC1023);
        igb_ks->ptc1522.value.ul += E1000_READ_REG(hw, E1000_PTC1522);

        /*
         * The 64-bit register will reset whenever the upper
         * 32 bits are read. So we need to read the lower
         * 32 bits first, then read the upper 32 bits.
         */
        val_low = E1000_READ_REG(hw, E1000_GORCL);
        val_high = E1000_READ_REG(hw, E1000_GORCH);
        igb_ks->gor.value.ui64 += (uint64_t)val_high << 32 | (uint64_t)val_low;

        val_low = E1000_READ_REG(hw, E1000_GOTCL);
        val_high = E1000_READ_REG(hw, E1000_GOTCH);
        igb_ks->got.value.ui64 += (uint64_t)val_high << 32 | (uint64_t)val_low;
#endif

        igb_ks->symerrs.value.ui64 += E1000_READ_REG(hw, E1000_SYMERRS);
```

```
125         igb_ks->mpc.value.ui64 += E1000_READ_REG(hw, E1000_MPC);
126         igb_ks->rlec.value.ui64 += E1000_READ_REG(hw, E1000_RLEC);
127         igb_ks->fcruc.value.ui64 += E1000_READ_REG(hw, E1000_FCRUC);
128         igb_ks->rfc.value.ul += E1000_READ_REG(hw, E1000_RFC);
129         igb_ks->tncrs.value.ul += E1000_READ_REG(hw, E1000_TNCRS);
130         igb_ks->tsctc.value.ul += E1000_READ_REG(hw, E1000_TSCTC);
131         igb_ks->tsctfc.value.ul += E1000_READ_REG(hw, E1000_TSCTFC);
132         igb_ks->xonrxc.value.ui64 += E1000_READ_REG(hw, E1000_XONRXC);
133         igb_ks->xontxc.value.ui64 += E1000_READ_REG(hw, E1000_XONTXC);
134         igb_ks->xoffrxc.value.ui64 += E1000_READ_REG(hw, E1000_XOFFRXC);
135         igb_ks->xofftxc.value.ui64 += E1000_READ_REG(hw, E1000_XOFFTXC);

137         mutex_exit(&igb->gen_lock);

139         if (igb_check_acc_handle(igb->osdep.reg_handle) != DDI_FM_OK) {
140                 ddi_fm_service_impact(igb->dip, DDI_SERVICE_DEGRADED);
141                 return (EIO);
142         }

144         return (0);
145 }

147 /*
148  * Create and initialize the driver private statistics.
149  */
150 int
151 igb_init_stats(igb_t *igb)
152 {
153         kstat_t *ks;
154         igb_stat_t *igb_ks;

156         /*
157          * Create and init kstat
158          */
159         ks = kstat_create(MODULE_NAME, ddi_get_instance(igb->dip),
160             "statistics", "net", KSTAT_TYPE_NAMED,
161             sizeof (igb_stat_t) / sizeof (kstat_named_t), 0);

163         if (ks == NULL) {
164                 igb_error(igb,
165                     "Could not create kernel statistics");
166                 return (IGB_FAILURE);
167         }

169         igb->igb_ks = ks;

171         igb_ks = (igb_stat_t *)ks->ks_data;

173         /*
174          * Initialize all the statistics.
175          */
175         kstat_named_init(&igb_ks->link_speed, "link_speed",
176             KSTAT_DATA_UINT64);
176         kstat_named_init(&igb_ks->reset_count, "reset_count",
177             KSTAT_DATA_UINT64);
178         kstat_named_init(&igb_ks->dout_sync, "DMA_out_sync",
179             KSTAT_DATA_UINT64);

181 #ifdef IGB_DEBUG
182         kstat_named_init(&igb_ks->rx_frame_error, "rx_frame_error",
183             KSTAT_DATA_UINT64);
184         kstat_named_init(&igb_ks->rx_cksum_error, "rx_cksum_error",
185             KSTAT_DATA_UINT64);
186         kstat_named_init(&igb_ks->rx_exceed_pkt, "rx_exceed_pkt",
187             KSTAT_DATA_UINT64);
188         kstat_named_init(&igb_ks->tx_overload, "tx_overload",
```



```
189             KSTAT_DATA_UINT64);
190         kstat_named_init(&igb_ks->tx_fail_no_tbd, "tx_fail_no_tbd",
191             KSTAT_DATA_UINT64);
192         kstat_named_init(&igb_ks->tx_fail_no_tcb, "tx_fail_no_tcb",
193             KSTAT_DATA_UINT64);
194         kstat_named_init(&igb_ks->tx_fail_dma_bind, "tx_fail_dma_bind",
195             KSTAT_DATA_UINT64);
196         kstat_named_init(&igb_ks->tx_reschedule, "tx_reschedule",
197             KSTAT_DATA_UINT64);

199         kstat_named_init(&igb_ks->gprc, "good_pkts_recvd",
200             KSTAT_DATA_UINT64);
201         kstat_named_init(&igb_ks->gptc, "good_pkts_xmitd",
202             KSTAT_DATA_UINT64);
203         kstat_named_init(&igb_ks->gor, "good_octets_recvd",
204             KSTAT_DATA_UINT64);
205         kstat_named_init(&igb_ks->got, "good_octets_xmitd",
206             KSTAT_DATA_UINT64);
207         kstat_named_init(&igb_ks->prc64, "pkts_recvd_(  64b)",
208             KSTAT_DATA_UINT64);
209         kstat_named_init(&igb_ks->prc127, "pkts_recvd_(  65- 127b)",
210             KSTAT_DATA_UINT64);
211         kstat_named_init(&igb_ks->prc255, "pkts_recvd_( 127- 255b)",
212             KSTAT_DATA_UINT64);
213         kstat_named_init(&igb_ks->prc511, "pkts_recvd_( 256- 511b)",
214             KSTAT_DATA_UINT64);
215         kstat_named_init(&igb_ks->prc1023, "pkts_recvd_( 511-1023b)",
216             KSTAT_DATA_UINT64);
217         kstat_named_init(&igb_ks->prc1522, "pkts_recvd_(1024-1522b)",
218             KSTAT_DATA_UINT64);
219         kstat_named_init(&igb_ks->ptc64, "pkts_xmitd_(  64b)",
220             KSTAT_DATA_UINT64);
221         kstat_named_init(&igb_ks->ptc127, "pkts_xmitd_(  65- 127b)",
222             KSTAT_DATA_UINT64);
223         kstat_named_init(&igb_ks->ptc255, "pkts_xmitd_( 128- 255b)",
224             KSTAT_DATA_UINT64);
225         kstat_named_init(&igb_ks->ptc511, "pkts_xmitd_( 255- 511b)",
226             KSTAT_DATA_UINT64);
227         kstat_named_init(&igb_ks->ptc1023, "pkts_xmitd_( 512-1023b)",
228             KSTAT_DATA_UINT64);
229         kstat_named_init(&igb_ks->ptc1522, "pkts_xmitd_(1024-1522b)",
230             KSTAT_DATA_UINT64);
231 #endif

233         kstat_named_init(&igb_ks->symerrs, "recv_symbol_errors",
234             KSTAT_DATA_UINT64);
235         kstat_named_init(&igb_ks->mpc, "recv_missed_packets",
236             KSTAT_DATA_UINT64);
237         kstat_named_init(&igb_ks->rlec, "recv_length_errors",
238             KSTAT_DATA_UINT64);
239         kstat_named_init(&igb_ks->fcruc, "recv_unsupport_FC_pkts",
240             KSTAT_DATA_UINT64);
241         kstat_named_init(&igb_ks->rfc, "recv_frag",
242             KSTAT_DATA_UINT64);
243         kstat_named_init(&igb_ks->tncrs, "xmit_with_no_CRS",
244             KSTAT_DATA_UINT64);
245         kstat_named_init(&igb_ks->tsctc, "xmit_TCP_seg_contexts",
246             KSTAT_DATA_UINT64);
247         kstat_named_init(&igb_ks->tsctfc, "xmit_TCP_seg_contexts_fail",
248             KSTAT_DATA_UINT64);
249         kstat_named_init(&igb_ks->xonrxc, "XONs_recvd",
250             KSTAT_DATA_UINT64);
251         kstat_named_init(&igb_ks->xontxc, "XONs_xmitd",
252             KSTAT_DATA_UINT64);
253         kstat_named_init(&igb_ks->xoffrxc, "XOFFs_recvd",
254             KSTAT_DATA_UINT64);
```

```
255         kstat_named_init(&igb_ks->xofftxc, "XOFFs_xmitd",
256             KSTAT_DATA_UINT64);

258         /*
259          * Function to provide kernel stat update on demand
260          */
261         ks->ks_update = igb_update_stats;

263         ks->ks_private = (void *)igb;

265         /*
266          * Add kstat to systems kstat chain
267          */
268         kstat_install(ks);

270         return (IGB_SUCCESS);
271 }
```

unchanged_portion_omitted_

```
**********************************************************
   25598 Wed Feb 26 09:49:38 2014
new/usr/src/uts/common/io/igb/igb_sw.h
4431 igb support for I354
4616 igb has uninitialized kstats
**********************************************************
```

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

/*
 * Copyright(c) 2007-2010 Intel Corporation. All rights reserved.
 */

/*
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2014 Pluribus Networks Inc.
 */

#ifndef _IGB_SW_H
#define _IGB_SW_H

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <sys/conf.h>
#include <sys/debug.h>
#include <sys/stropts.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/strlog.h>
#include <sys/kmem.h>
#include <sys/stat.h>
#include <sys/kstat.h>
#include <sys/modctl.h>
#include <sys/errno.h>
#include <sys/dlpi.h>
#include <sys/mac_provider.h>
#include <sys/mac_ether.h>
#include <sys/vlan.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/pci.h>
#include <sys/pcie.h>
#include <sys/sdt.h>
#include <sys/ethernet.h>
#include <sys/pattr.h>
```



```
#include <sys/strsubr.h>
#include <sys/netlb.h>
#include <sys/random.h>
#include <inet/common.h>
#include <inet/tcp.h>
#include <inet/ip.h>
#include <inet/mi.h>
#include <inet/nd.h>
#include <sys/ddifm.h>
#include <sys/fm/protocol.h>
#include <sys/fm/util.h>
#include <sys/fm/io/ddi.h>
#include "e1000_api.h"
#include "e1000_82575.h"


#define MODULE_NAME                     "igb"   /* module name */

#define IGB_SUCCESS                     DDI_SUCCESS
#define IGB_FAILURE                     DDI_FAILURE

#define IGB_UNKNOWN                     0x00
#define IGB_INITIALIZED                 0x01
#define IGB_STARTED                     0x02
#define IGB_SUSPENDED                   0x04
#define IGB_STALL                       0x08
#define IGB_ERROR                       0x80

#define IGB_RX_STOPPED                  0x1

#define IGB_INTR_NONE                   0
#define IGB_INTR_MSIX                   1
#define IGB_INTR_MSI                    2
#define IGB_INTR_LEGACY                 3

#define IGB_ADAPTER_REGSET              1       /* mapping adapter registers */
#define IGB_ADAPTER_MSIXTAB             4       /* mapping msi-x table */

#define IGB_NO_POLL                     -1
#define IGB_NO_FREE_SLOT                -1

#define MAX_NUM_UNICAST_ADDRESSES       E1000_RAR_ENTRIES
#define MCAST_ALLOC_COUNT               256
#define MAX_COOKIE                      18
#define MIN_NUM_TX_DESC                 2

/*
 * Number of settings for interrupt throttle rate (ITR).  There is one of
 * these per msi-x vector and it needs to be the maximum of all silicon
 * types supported by this driver.
 */
#define MAX_NUM_EITR                    25

/*
 * Maximum values for user configurable parameters
 */
#define MAX_TX_RING_SIZE                4096
#define MAX_RX_RING_SIZE                4096
#define MAX_RX_GROUP_NUM                4

#define MAX_MTU                         9000
#define MAX_RX_LIMIT_PER_INTR           4096

#define MAX_RX_COPY_THRESHOLD           9216
#define MAX_TX_COPY_THRESHOLD           9216
#define MAX_TX_RECYCLE_THRESHOLD        DEFAULT_TX_RING_SIZE
```

```
127 #define MAX_TX_OVERLOAD_THRESHOLD       DEFAULT_TX_RING_SIZE
128 #define MAX_TX_RESCHED_THRESHOLD        DEFAULT_TX_RING_SIZE
129 #define MAX_MCAST_NUM                   8192

131 /*
132  * Minimum values for user configurable parameters
133  */
134 #define MIN_TX_RING_SIZE                64
135 #define MIN_RX_RING_SIZE                64
136 #define MIN_RX_GROUP_NUM                1

138 #define MIN_MTU                         ETHERMIN
139 #define MIN_RX_LIMIT_PER_INTR           16

141 #define MIN_RX_COPY_THRESHOLD           0
142 #define MIN_TX_COPY_THRESHOLD           0
143 #define MIN_TX_RECYCLE_THRESHOLD        MIN_NUM_TX_DESC
144 #define MIN_TX_OVERLOAD_THRESHOLD       MIN_NUM_TX_DESC
145 #define MIN_TX_RESCHED_THRESHOLD        MIN_NUM_TX_DESC
146 #define MIN_MCAST_NUM                   8

148 /*
149  * Default values for user configurable parameters
150  */
151 #define DEFAULT_TX_RING_SIZE            512
152 #define DEFAULT_RX_RING_SIZE            512
153 #define DEFAULT_RX_GROUP_NUM            1

155 #define DEFAULT_MTU                     ETHERMTU
156 #define DEFAULT_RX_LIMIT_PER_INTR       256

158 #define DEFAULT_RX_COPY_THRESHOLD       128
159 #define DEFAULT_TX_COPY_THRESHOLD       512
160 #define DEFAULT_TX_RECYCLE_THRESHOLD    (MAX_COOKIE + 1)
161 #define DEFAULT_TX_OVERLOAD_THRESHOLD   MIN_NUM_TX_DESC
162 #define DEFAULT_TX_RESCHED_THRESHOLD    128
163 #define DEFAULT_TX_RESCHED_THRESHOLD_LOW        32
164 #define DEFAULT_MCAST_NUM               4096

166 #define IGB_LSO_MAXLEN                  65535

168 #define TX_DRAIN_TIME                   200
169 #define RX_DRAIN_TIME                   200

171 #define STALL_WATCHDOG_TIMEOUT          8       /* 8 seconds */

173 /*
174  * Defined for IP header alignment.
175  */
176 #define IPHDR_ALIGN_ROOM                2

178 /*
179  * Bit flags for attach_progress
180  */
181 #define ATTACH_PROGRESS_PCI_CONFIG      0x0001  /* PCI config setup */
182 #define ATTACH_PROGRESS_REGS_MAP        0x0002  /* Registers mapped */
183 #define ATTACH_PROGRESS_PROPS           0x0004  /* Properties initialized */
184 #define ATTACH_PROGRESS_ALLOC_INTR      0x0008  /* Interrupts allocated */
185 #define ATTACH_PROGRESS_ALLOC_RINGS     0x0010  /* Rings allocated */
186 #define ATTACH_PROGRESS_ADD_INTR        0x0020  /* Intr handlers added */
187 #define ATTACH_PROGRESS_LOCKS           0x0040  /* Locks initialized */
188 #define ATTACH_PROGRESS_INIT_ADAPTER    0x0080  /* Adapter initialized */
189 #define ATTACH_PROGRESS_STATS           0x0200  /* Kstats created */
190 #define ATTACH_PROGRESS_MAC             0x0800  /* MAC registered */
191 #define ATTACH_PROGRESS_ENABLE_INTR     0x1000  /* DDI interrupts enabled */
192 #define ATTACH_PROGRESS_FMINIT          0x2000  /* FMA initialized */
```



```
194 #define PROP_ADV_AUTONEG_CAP            "adv_autoneg_cap"
195 #define PROP_ADV_1000FDX_CAP            "adv_1000fdx_cap"
196 #define PROP_ADV_1000HDX_CAP            "adv_1000hdx_cap"
197 #define PROP_ADV_100FDX_CAP             "adv_100fdx_cap"
198 #define PROP_ADV_100HDX_CAP             "adv_100hdx_cap"
199 #define PROP_ADV_10FDX_CAP              "adv_10fdx_cap"
200 #define PROP_ADV_10HDX_CAP              "adv_10hdx_cap"
201 #define PROP_DEFAULT_MTU                "default_mtu"
202 #define PROP_FLOW_CONTROL               "flow_control"
203 #define PROP_TX_RING_SIZE               "tx_ring_size"
204 #define PROP_RX_RING_SIZE               "rx_ring_size"
205 #define PROP_MR_ENABLE                  "mr_enable"
206 #define PROP_RX_GROUP_NUM               "rx_group_number"

208 #define PROP_INTR_FORCE                 "intr_force"
209 #define PROP_TX_HCKSUM_ENABLE           "tx_hcksum_enable"
210 #define PROP_RX_HCKSUM_ENABLE           "rx_hcksum_enable"
211 #define PROP_LSO_ENABLE                 "lso_enable"
212 #define PROP_TX_HEAD_WB_ENABLE          "tx_head_wb_enable"
213 #define PROP_TX_COPY_THRESHOLD          "tx_copy_threshold"
214 #define PROP_TX_RECYCLE_THRESHOLD       "tx_recycle_threshold"
215 #define PROP_TX_OVERLOAD_THRESHOLD      "tx_overload_threshold"
216 #define PROP_TX_RESCHED_THRESHOLD       "tx_resched_threshold"
217 #define PROP_RX_COPY_THRESHOLD          "rx_copy_threshold"
218 #define PROP_RX_LIMIT_PER_INTR          "rx_limit_per_intr"
219 #define PROP_INTR_THROTTLING            "intr_throttling"
220 #define PROP_MCAST_MAX_NUM              "mcast_max_num"

222 #define IGB_LB_NONE                     0
223 #define IGB_LB_EXTERNAL                 1
224 #define IGB_LB_INTERNAL_PHY             3
225 #define IGB_LB_INTERNAL_SERDES          4

227 enum ioc_reply {
228         IOC_INVAL = -1, /* bad, NAK with EINVAL */
229         IOC_DONE,       /* OK, reply sent */
230         IOC_ACK,        /* OK, just send ACK */
231         IOC_REPLY       /* OK, just send reply */
232 };
```

*unchanged_portion_omitted_*

```
559 typedef struct igb {
560         int                     instance;
561         mac_handle_t            mac_hdl;
562         dev_info_t              *dip;
563         struct e1000_hw         hw;
564         struct igb_osdep        osdep;

566         adapter_info_t          *capab;         /* adapter capabilities */

568         uint32_t                igb_state;
569         link_state_t            link_state;
570         uint32_t                link_speed;
571         uint32_t                link_duplex;
572         boolean_t               link_complete;
573         timeout_id_t            link_tid;

575         uint32_t                reset_count;
576         uint32_t                attach_progress;
577         uint32_t                loopback_mode;
578         uint32_t                default_mtu;
579         uint32_t                max_frame_size;
580         uint32_t                dout_sync;

582         uint32_t                rcb_pending;
```

```
584         uint32_t                mr_enable;      /* Enable multiple rings */
585         uint32_t                vmdq_mode;      /* Mode of VMDq */

587         /*
588          * Receive Rings and Groups
589          */
590         igb_rx_ring_t           *rx_rings;      /* Array of rx rings */
591         uint32_t                num_rx_rings;   /* Number of rx rings in use */
592         uint32_t                rx_ring_size;   /* Rx descriptor ring size */
593         uint32_t                rx_buf_size;    /* Rx buffer size */
594         igb_rx_group_t          *rx_groups;     /* Array of rx groups */
595         uint32_t                num_rx_groups;  /* Number of rx groups in use */

597         /*
598          * Transmit Rings
599          */
600         igb_tx_ring_t           *tx_rings;      /* Array of tx rings */
601         uint32_t                num_tx_rings;   /* Number of tx rings in use */
602         uint32_t                tx_ring_size;   /* Tx descriptor ring size */
603         uint32_t                tx_buf_size;    /* Tx buffer size */

605         boolean_t               tx_ring_init;
606         boolean_t               tx_head_wb_enable; /* Tx head wrtie-back */
607         boolean_t               tx_hcksum_enable; /* Tx h/w cksum offload */
608         boolean_t               lso_enable;     /* Large Segment Offload */
609         uint32_t                tx_copy_thresh; /* Tx copy threshold */
610         uint32_t                tx_recycle_thresh; /* Tx recycle threshold */
611         uint32_t                tx_overload_thresh; /* Tx overload threshold */
612         uint32_t                tx_resched_thresh; /* Tx reschedule threshold */
613         boolean_t               rx_hcksum_enable; /* Rx h/w cksum offload */
614         uint32_t                rx_copy_thresh; /* Rx copy threshold */
615         uint32_t                rx_limit_per_intr; /* Rx pkts per interrupt */

617         uint32_t                intr_throttling[MAX_NUM_EITR];
618         uint32_t                intr_force;

620         int                     intr_type;
621         int                     intr_cnt;
622         int                     intr_cap;
623         size_t                  intr_size;
624         uint_t                  intr_pri;
625         ddi_intr_handle_t       *htable;
626         uint32_t                eims_mask;
627         uint32_t                ims_mask;

629         kmutex_t                gen_lock; /* General lock for device access */
630         kmutex_t                watchdog_lock;
631         kmutex_t                link_lock;
632         kmutex_t                rx_pending_lock;

634         boolean_t               watchdog_enable;
635         boolean_t               watchdog_start;
636         timeout_id_t            watchdog_tid;

638         boolean_t               unicst_init;
639         uint32_t                unicst_avail;
640         uint32_t                unicst_total;
641         igb_ether_addr_t        unicst_addr[MAX_NUM_UNICAST_ADDRESSES];
642         uint32_t                mcast_count;
643         uint32_t                mcast_alloc_count;
644         uint32_t                mcast_max_num;
645         struct ether_addr       *mcast_table;

647         /*
648          * Kstat definitions
```



```
649          */
650         kstat_t                 *igb_ks;

652         /*
653          * Backing store for MAC stats.  These are reported via GLDv3, instead o
654          * via our private kstat structure.
655          */
656         uint64_t                stat_tor;       /* rbytes */
657         uint64_t                stat_tpr;       /* rpackets */
658         uint64_t                stat_tot;       /* obytes */
659         uint64_t                stat_tpt;       /* opackets */
660         uint64_t                stat_colc;      /* collisions */
661         uint64_t                stat_mcc;       /* multi colls */
662         uint64_t                stat_scc;       /* single colls */
663         uint64_t                stat_ecol;      /* excessive colls */
664         uint64_t                stat_latecol;   /* late colls */
665         uint64_t                stat_bptc;      /* xmit bcast */
666         uint64_t                stat_mptc;      /* xmit bcast */
667         uint64_t                stat_bprc;      /* recv bcast */
668         uint64_t                stat_mprc;      /* recv mcast */
669         uint64_t                stat_rnbc;      /* recv nobuf */
670         uint64_t                stat_roc;       /* recv toolong */
671         uint64_t                stat_sec;       /* sqe errors */
672         uint64_t                stat_dc;        /* defer */
673         uint64_t                stat_algnerrc;  /* align errors */
674         uint64_t                stat_crcerrs;   /* crc errors */
675         uint64_t                stat_cexterr;   /* carrier extension errors */
676         uint64_t                stat_ruc;       /* recv tooshort */
677         uint64_t                stat_rjc;       /* recv jabber */
678         uint64_t                stat_rxerrc;    /* recv errors */

680         uint32_t                param_en_1000fdx_cap:1,
681                                 param_en_1000hdx_cap:1,
682                                 param_en_100t4_cap:1,
683                                 param_en_100fdx_cap:1,
684                                 param_en_100hdx_cap:1,
685                                 param_en_10fdx_cap:1,
686                                 param_en_10hdx_cap:1,
687                                 param_1000fdx_cap:1,
688                                 param_1000hdx_cap:1,
689                                 param_100t4_cap:1,
690                                 param_100fdx_cap:1,
691                                 param_100hdx_cap:1,
692                                 param_10fdx_cap:1,
693                                 param_10hdx_cap:1,
694                                 param_autoneg_cap:1,
695                                 param_pause_cap:1,
696                                 param_asym_pause_cap:1,
697                                 param_rem_fault:1,
698                                 param_adv_1000fdx_cap:1,
699                                 param_adv_1000hdx_cap:1,
700                                 param_adv_100t4_cap:1,
701                                 param_adv_100fdx_cap:1,
702                                 param_adv_100hdx_cap:1,
703                                 param_adv_10fdx_cap:1,
704                                 param_adv_10hdx_cap:1,
705                                 param_adv_autoneg_cap:1,
706                                 param_adv_pause_cap:1,
707                                 param_adv_asym_pause_cap:1,
708                                 param_adv_rem_fault:1,
709                                 param_lp_1000fdx_cap:1,
710                                 param_lp_1000hdx_cap:1,
711                                 param_lp_100t4_cap:1;

713         uint32_t                param_lp_100fdx_cap:1,
714                                 param_lp_100hdx_cap:1,
```

```
                                param_lp_10fdx_cap:1,
                                param_lp_10hdx_cap:1,
                                param_lp_autoneg_cap:1,
                                param_lp_pause_cap:1,
                                param_lp_asym_pause_cap:1,
                                param_lp_rem_fault:1,
                                param_pad_to_32:24;

        /*
         * FMA capabilities
         */
        int                     fm_capabilities;

        ulong_t                 page_size;
} igb_t;

typedef struct igb_stat {

        kstat_named_t link_speed;       /* Link Speed */
        kstat_named_t reset_count;      /* Reset Count */
        kstat_named_t dout_sync;        /* DMA out of sync */
#ifdef IGB_DEBUG
        kstat_named_t rx_frame_error;   /* Rx Error in Packet */
        kstat_named_t rx_cksum_error;   /* Rx Checksum Error */
        kstat_named_t rx_exceed_pkt;    /* Rx Exceed Max Pkt Count */

        kstat_named_t tx_overload;      /* Tx Desc Ring Overload */
        kstat_named_t tx_fail_no_tcb;   /* Tx Fail Freelist Empty */
        kstat_named_t tx_fail_no_tbd;   /* Tx Fail Desc Ring Empty */
        kstat_named_t tx_fail_dma_bind; /* Tx Fail DMA bind */
        kstat_named_t tx_reschedule;    /* Tx Reschedule */

        kstat_named_t gprc;     /* Good Packets Received Count */
        kstat_named_t gptc;     /* Good Packets Xmitted Count */
        kstat_named_t gor;      /* Good Octets Received Count */
        kstat_named_t got;      /* Good Octets Xmitd Count */
        kstat_named_t prc64;    /* Packets Received - 64b */
        kstat_named_t prc127;   /* Packets Received - 65-127b */
        kstat_named_t prc255;   /* Packets Received - 127-255b */
        kstat_named_t prc511;   /* Packets Received - 256-511b */
        kstat_named_t prc1023;  /* Packets Received - 511-1023b */
        kstat_named_t prc1522;  /* Packets Received - 1024-1522b */
        kstat_named_t ptc64;    /* Packets Xmitted (64b) */
        kstat_named_t ptc127;   /* Packets Xmitted (64-127b) */
        kstat_named_t ptc255;   /* Packets Xmitted (128-255b) */
        kstat_named_t ptc511;   /* Packets Xmitted (255-511b) */
        kstat_named_t ptc1023;  /* Packets Xmitted (512-1023b) */
        kstat_named_t ptc1522;  /* Packets Xmitted (1024-1522b */
#endif
        kstat_named_t crcerrs;  /* CRC Error Count */
        kstat_named_t symerrs;  /* Symbol Error Count */
        kstat_named_t mpc;      /* Missed Packet Count */
        kstat_named_t scc;      /* Single Collision Count */
        kstat_named_t ecol;     /* Excessive Collision Count */
        kstat_named_t mcc;      /* Multiple Collision Count */
        kstat_named_t latecol;  /* Late Collision Count */
        kstat_named_t colc;     /* Collision Count */
        kstat_named_t dc;       /* Defer Count */
        kstat_named_t sec;      /* Sequence Error Count */
        kstat_named_t rlec;     /* Receive Length Error Count */
        kstat_named_t xonrxc;   /* XON Received Count */
        kstat_named_t xontxc;   /* XON Xmitted Count */
        kstat_named_t xoffrxc;  /* XOFF Received Count */
        kstat_named_t xofftxc;  /* Xoff Xmitted Count */
        kstat_named_t fcruc;    /* Unknown Flow Conrol Packet Rcvd Count */
        kstat_named_t bprc;     /* Broadcasts Pkts Received Count */
```



```
        kstat_named_t mprc;     /* Multicast Pkts Received Count */
        kstat_named_t rnbc;     /* Receive No Buffers Count */
        kstat_named_t ruc;      /* Receive Undersize Count */
        kstat_named_t rfc;      /* Receive Frag Count */
        kstat_named_t roc;      /* Receive Oversize Count */
        kstat_named_t rjc;      /* Receive Jabber Count */
        kstat_named_t tor;      /* Total Octets Recvd Count */
        kstat_named_t tot;      /* Total Octets Xmted Count */
        kstat_named_t tpr;      /* Total Packets Received */
        kstat_named_t tpt;      /* Total Packets Xmitted */
        kstat_named_t mptc;     /* Multicast Packets Xmited Count */
        kstat_named_t bptc;     /* Broadcast Packets Xmited Count */
        kstat_named_t algnerrc; /* Alignment Error count */
        kstat_named_t rxerrc;   /* Rx Error Count */
        kstat_named_t tncrs;    /* Transmit with no CRS */
        kstat_named_t cexterr;  /* Carrier Extension Error count */
        kstat_named_t tsctc;    /* TCP seg contexts xmit count */
        kstat_named_t tsctfc;   /* TCP seg contexts xmit fail count */
} igb_stat_t;
______unchanged_portion_omitted_
```