```
**********************************************************
   48638 Tue Jan 15 10:18:57 2019
new/usr/src/uts/common/io/mii/mii.c
10089 phy_check() is bitwise, should be streetwise
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  */

  26 /*
  27  * Copyright (c) 2018, Joyent, Inc.
  28  */

  30 /*
  31  * mii - MII/PHY support for MAC drivers
  32  *
  33  * Utility module to provide a consistent interface to a MAC driver accross
  34  * different implementations of PHY devices
  35  */

  37 #include <sys/types.h>
  38 #include <sys/debug.h>
  39 #include <sys/errno.h>
  40 #include <sys/param.h>
  41 #include <sys/kmem.h>
  42 #include <sys/conf.h>
  43 #include <sys/ddi.h>
  44 #include <sys/sunddi.h>
  45 #include <sys/modctl.h>
  46 #include <sys/cmn_err.h>
  47 #include <sys/policy.h>
  48 #include <sys/note.h>
  49 #include <sys/strsun.h>
  50 #include <sys/miiregs.h>
  51 #include <sys/mac_provider.h>
  52 #include <sys/mac_ether.h>
  53 #include <sys/mii.h>
  54 #include "miipriv.h"

  56 #define MII_SECOND      1000000

  58 /* indices into error array */
  59 enum {
  60         MII_EOK = 0,
  61         MII_ERESET,
```



```
  62         MII_ESTART,
  63         MII_ENOPHY,
  64         MII_ECHECK,
  65         MII_ELOOP,
  66 };
______unchanged_portion_omitted_


1378 int
1379 phy_check(phy_handle_t *ph)
1380 {
1381         uint16_t control, status, lpar, msstat, anexp;
1382         int debounces = 100;

1384         ASSERT(mutex_owned(&ph->phy_mii->m_lock));

1386 debounce:
1387         status = phy_read(ph, MII_STATUS);
1388         control = phy_read(ph, MII_CONTROL);

1390         if (status & MII_STATUS_EXTENDED) {
1391                 lpar = phy_read(ph, MII_AN_LPABLE);
1392                 anexp = phy_read(ph, MII_AN_EXPANSION);
1393         } else {
1394                 lpar = 0;
1395                 anexp = 0;
1396         }

1398         /*
1399          * We reread to clear any latched bits.  This also debounces
1400          * any state that might be in transition.
1401          */
1402         drv_usecwait(10);
1403         if ((status != phy_read(ph, MII_STATUS)) && debounces) {
1404                 debounces--;
1405                 goto debounce;
1406         }

1408         /*
1409          * Detect the situation where the PHY is removed or has died.
1410          * According to spec, at least one bit of status must be set,
1411          * and at least one bit must be clear.
1412          */
1413         if ((status == 0xffff) || (status == 0)) {
1414                 ph->phy_speed = 0;
1415                 ph->phy_duplex = LINK_DUPLEX_UNKNOWN;
1416                 ph->phy_link = LINK_STATE_UNKNOWN;
1417                 ph->phy_present = B_FALSE;
1418                 return (DDI_FAILURE);
1419         }

1421         /* We only respect the link flag if we are not in loopback. */
1422         if ((ph->phy_loopback != PHY_LB_INT_PHY) &&
1423             ((status & MII_STATUS_LINKUP) == 0)) {
1424                 ph->phy_speed = 0;
1425                 ph->phy_duplex = LINK_DUPLEX_UNKNOWN;
1426                 ph->phy_link = LINK_STATE_DOWN;
1427                 return (DDI_SUCCESS);
1428         }

1430         ph->phy_link = LINK_STATE_UP;

1432         if ((control & MII_CONTROL_ANE) == 0) {

1434                 ph->phy_lp_aneg = B_FALSE;
1435                 ph->phy_lp_10_hdx = B_FALSE;
```

```
1436                 ph->phy_lp_10_fdx = B_FALSE;
1437                 ph->phy_lp_100_t4 = B_FALSE;
1438                 ph->phy_lp_100_hdx = B_FALSE;
1439                 ph->phy_lp_100_fdx = B_FALSE;
1440                 ph->phy_lp_1000_hdx = B_FALSE;
1441                 ph->phy_lp_1000_fdx = B_FALSE;

1443                 /*
1444                  * We have no idea what our link partner might or might
1445                  * not be able to support, except that it appears to
1446                  * support the same mode that we have forced.
1447                  */
1448                 if (control & MII_CONTROL_1GB) {
1449                         ph->phy_speed = 1000;
1450                 } else if (control & MII_CONTROL_100MB) {
1451                         ph->phy_speed = 100;
1452                 } else {
1453                         ph->phy_speed = 10;
1454                 }
1455                 ph->phy_duplex = control & MII_CONTROL_FDUPLEX ?
1456                     LINK_DUPLEX_FULL : LINK_DUPLEX_HALF;

1458                 return (DDI_SUCCESS);
1459         }

1461         if (ph->phy_type == XCVR_1000X) {

1463                 ph->phy_lp_10_hdx = B_FALSE;
1464                 ph->phy_lp_10_fdx = B_FALSE;
1465                 ph->phy_lp_100_t4 = B_FALSE;
1466                 ph->phy_lp_100_hdx = B_FALSE;
1467                 ph->phy_lp_100_fdx = B_FALSE;

1469                 /* 1000BASE-X requires autonegotiation */
1470                 ph->phy_lp_aneg = B_TRUE;
1471                 ph->phy_lp_1000_fdx = !!(lpar & MII_ABILITY_X_FD);
1472                 ph->phy_lp_1000_hdx = !!(lpar & MII_ABILITY_X_HD);
1473                 ph->phy_lp_pause = !!(lpar & MII_ABILITY_X_PAUSE);
1474                 ph->phy_lp_asmpause = !!(lpar & MII_ABILITY_X_ASMPAUSE);

1476         } else if (ph->phy_type == XCVR_100T2) {
1477                 ph->phy_lp_10_hdx = B_FALSE;
1478                 ph->phy_lp_10_fdx = B_FALSE;
1479                 ph->phy_lp_100_t4 = B_FALSE;
1480                 ph->phy_lp_1000_hdx = B_FALSE;
1481                 ph->phy_lp_1000_fdx = B_FALSE;
1482                 ph->phy_lp_pause = B_FALSE;
1483                 ph->phy_lp_asmpause = B_FALSE;

1485                 /* 100BASE-T2 requires autonegotiation */
1486                 ph->phy_lp_aneg = B_TRUE;
1487                 ph->phy_lp_100_fdx = !!(lpar & MII_ABILITY_T2_FD);
1488                 ph->phy_lp_100_hdx = !!(lpar & MII_ABILITY_T2_HD);

1490         } else if (anexp & MII_AN_EXP_PARFAULT) {
1491                 /*
1492                  * Parallel detection fault!  This happens when the
1493                  * peer does not use autonegotiation, and the
1494                  * detection logic reports more than one type of legal
1495                  * link is available.  Note that parallel detection
1496                  * can only happen with half duplex 10, 100, and
1497                  * 100TX4.  We also should not have got here, because
1498                  * the link state bit should have failed.
1499                  */
1500 #ifdef  DEBUG
1501                 phy_warn(ph, "Parallel detection fault!");
```



```
1502 #endif
1503                 ph->phy_lp_10_hdx = B_FALSE;
1504                 ph->phy_lp_10_fdx = B_FALSE;
1505                 ph->phy_lp_100_t4 = B_FALSE;
1506                 ph->phy_lp_100_hdx = B_FALSE;
1507                 ph->phy_lp_100_fdx = B_FALSE;
1508                 ph->phy_lp_1000_hdx = B_FALSE;
1509                 ph->phy_lp_1000_fdx = B_FALSE;
1510                 ph->phy_lp_pause = B_FALSE;
1511                 ph->phy_lp_asmpause = B_FALSE;
1512                 ph->phy_speed = 0;
1513                 ph->phy_duplex = LINK_DUPLEX_UNKNOWN;
1514                 return (DDI_SUCCESS);

1516         } else {
1517                 ph->phy_lp_aneg = !!(anexp & MII_AN_EXP_LPCANAN);

1519                 /*
1520                  * Note: If the peer doesn't support autonegotiation, then
1521                  * according to clause 28.5.4.5, the link partner ability
1522                  * register will still have the right bits set.  However,
1523                  * gigabit modes cannot use legacy parallel detection.
1524                  */

1526                 if ((ph->phy_type == XCVR_1000T) &&
1522                 if ((ph->phy_type == XCVR_1000T) &
1527                     (anexp & MII_AN_EXP_LPCANAN)) {

1529                         /* check for gige */
1530                         msstat = phy_read(ph, MII_MSSTATUS);

1532                         ph->phy_lp_1000_hdx =
1533                             !!(msstat & MII_MSSTATUS_LP1000T);

1535                         ph->phy_lp_1000_fdx =
1536                             !!(msstat & MII_MSSTATUS_LP1000T_FD);
1537                 }

1539                 ph->phy_lp_100_fdx = !!(lpar & MII_ABILITY_100BASE_TX_FD);
1540                 ph->phy_lp_100_hdx = !!(lpar & MII_ABILITY_100BASE_TX);
1541                 ph->phy_lp_100_t4 = !!(lpar & MII_ABILITY_100BASE_T4);
1542                 ph->phy_lp_10_fdx = !!(lpar & MII_ABILITY_10BASE_T_FD);
1543                 ph->phy_lp_10_hdx = !!(lpar & MII_ABILITY_10BASE_T);
1544                 ph->phy_lp_pause = !!(lpar & MII_ABILITY_PAUSE);
1545                 ph->phy_lp_asmpause = !!(lpar & MII_ABILITY_ASMPAUSE);
1546         }

1548         /* resolve link pause */
1549         if ((ph->phy_en_flowctrl == LINK_FLOWCTRL_BI) &&
1550             (ph->phy_lp_pause)) {
1551                 ph->phy_flowctrl = LINK_FLOWCTRL_BI;
1552         } else if ((ph->phy_en_flowctrl == LINK_FLOWCTRL_RX) &&
1553             (ph->phy_lp_pause || ph->phy_lp_asmpause)) {
1554                 ph->phy_flowctrl = LINK_FLOWCTRL_RX;
1555         } else if ((ph->phy_en_flowctrl == LINK_FLOWCTRL_TX) &&
1556             (ph->phy_lp_pause)) {
1557                 ph->phy_flowctrl = LINK_FLOWCTRL_TX;
1558         } else {
1559                 ph->phy_flowctrl = LINK_FLOWCTRL_NONE;
1560         }

1562         if (ph->phy_adv_1000_fdx && ph->phy_lp_1000_fdx) {
1563                 ph->phy_speed = 1000;
1564                 ph->phy_duplex = LINK_DUPLEX_FULL;

1566         } else if (ph->phy_adv_1000_hdx && ph->phy_lp_1000_hdx) {
```

```
1567                 ph->phy_speed = 1000;
1568                 ph->phy_duplex = LINK_DUPLEX_HALF;

1570         } else if (ph->phy_adv_100_fdx && ph->phy_lp_100_fdx) {
1571                 ph->phy_speed = 100;
1572                 ph->phy_duplex = LINK_DUPLEX_FULL;

1574         } else if (ph->phy_adv_100_t4 && ph->phy_lp_100_t4) {
1575                 ph->phy_speed = 100;
1576                 ph->phy_duplex = LINK_DUPLEX_HALF;

1578         } else if (ph->phy_adv_100_hdx && ph->phy_lp_100_hdx) {
1579                 ph->phy_speed = 100;
1580                 ph->phy_duplex = LINK_DUPLEX_HALF;

1582         } else if (ph->phy_adv_10_fdx && ph->phy_lp_10_fdx) {
1583                 ph->phy_speed = 10;
1584                 ph->phy_duplex = LINK_DUPLEX_FULL;

1586         } else if (ph->phy_adv_10_hdx && ph->phy_lp_10_hdx) {
1587                 ph->phy_speed = 10;
1588                 ph->phy_duplex = LINK_DUPLEX_HALF;

1590         } else {
1591 #ifdef  DEBUG
1592                 phy_warn(ph, "No common abilities.");
1593 #endif
1594                 ph->phy_speed = 0;
1595                 ph->phy_duplex = LINK_DUPLEX_UNKNOWN;
1596         }

1598         return (DDI_SUCCESS);
1599 }
```

_____***unchanged_portion_omitted_***