```
**********************************************************
     477 Fri Oct  5 15:14:40 2018
new/exception_lists/wscheck
9724 qede needs updates for newer GCC
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #
  11 # Copyright 2018 Joyent, Inc.
  12 #
  13 syntax: glob

  15 usr/src/uts/common/io/qede/*
```

```
**********************************************************
   44712 Fri Oct  5 15:14:40 2018
new/usr/src/uts/common/io/qede/579xx/drivers/ecore/ecore_dcbx.c
9724 qede needs updates for newer GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, v.1,  (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://opensource.org/licenses/CDDL-1.0.
 * 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 2014-2017 Cavium, Inc.
 * The contents of this file are subject to the terms of the Common Development
 * and Distribution License, v.1,  (the "License").

 * You may not use this file except in compliance with the License.

 * You can obtain a copy of the License at available
 * at http://opensource.org/licenses/CDDL-1.0

 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/*
 * Copyright 2018 Joyent, Inc.
 */

#include "bcm_osal.h"
#include "ecore.h"
#include "ecore_sp_commands.h"
#include "ecore_dcbx.h"
#include "ecore_cxt.h"
#include "ecore_gtt_reg_addr.h"
#include "ecore_iro.h"
#ifdef CONFIG_ECORE_ROCE
#include "ecore_roce.h"
#endif
#include "ecore_iov_api.h"

#define ECORE_DCBX_MAX_MIB_READ_TRY     (100)
#define ECORE_ETH_TYPE_DEFAULT          (0)
#define ECORE_ETH_TYPE_ROCE             (0x8915)
#define ECORE_UDP_PORT_TYPE_ROCE_V2     (0x12B7)
#define ECORE_ETH_TYPE_FCOE             (0x8906)
#define ECORE_TCP_PORT_ISCSI            (0xCBC)

#define ECORE_DCBX_INVALID_PRIORITY     0xFF

/* Get Traffic Class from priority traffic class table, 4 bits represent
```



```
 * the traffic class corresponding to the priority.
 */
#define ECORE_DCBX_PRIO2TC(prio_tc_tbl, prio) \
                ((u32)(prio_tc_tbl >> ((7 - prio) * 4)) & 0x7)

static bool ecore_dcbx_app_ethtype(u32 app_info_bitmap)
{
        return !!(ECORE_MFW_GET_FIELD(app_info_bitmap, DCBX_APP_SF) ==
                  DCBX_APP_SF_ETHTYPE);
}
```

*unchanged_portion_omitted_*

```
/* Update app protocol data and hw_info fields with the TLV info */
static void
ecore_dcbx_update_app_info(struct ecore_dcbx_results *p_data,
                           struct ecore_hwfn *p_hwfn,
                           bool enable, u8 prio, u8 tc,
                           enum dcbx_protocol_type type)
{
        enum ecore_pci_personality personality;
        enum dcbx_protocol_type id;
        char *name;
        int i;

        for (i = 0; i < OSAL_ARRAY_SIZE(ecore_dcbx_app_update); i++) {
                id = ecore_dcbx_app_update[i].id;

                if (type != id)
                        continue;

                personality = ecore_dcbx_app_update[i].personality;
                name = ecore_dcbx_app_update[i].name;

                ecore_dcbx_set_params(p_data, p_hwfn, enable,
                                      prio, tc, type, personality);
        }
}
```

*unchanged_portion_omitted_*

```
/*
 * Read updated MIB.
 * Reconfigure QM and invoke PF update ramrod command if operational MIB
 * change is detected.
 */
enum _ecore_status_t
ecore_dcbx_mib_update_event(struct ecore_hwfn *p_hwfn, struct ecore_ptt *p_ptt,
                            enum ecore_mib_read_type type)
{
        enum _ecore_status_t rc = ECORE_SUCCESS;

        rc = ecore_dcbx_read_mib(p_hwfn, p_ptt, type);
        if (rc)
                return rc;

        if (type == ECORE_DCBX_OPERATIONAL_MIB) {
                ecore_dcbx_get_dscp_params(p_hwfn, p_ptt,
                                           &p_hwfn->p_dcbx_info->get);

                rc = ecore_dcbx_process_mib_info(p_hwfn);
                if (!rc) {
                        bool enabled __unused;
                        bool enabled;

                        /* reconfigure tcs of QM queues according
                         * to negotiation results
                         */
```

```
1009                        ecore_qm_reconf(p_hwfn, p_ptt);

1011                        /* update storm FW with negotiation results */
1012                        ecore_sp_pf_update_dcbx(p_hwfn);

1014                        /* set eagle enigne 1 flow control workaround
1015                         * according to negotiation results
1016                         */
1017                        enabled = p_hwfn->p_dcbx_info->results.dcbx_enabled;

1019 #ifdef CONFIG_ECORE_ROCE
1020                        /* for roce PFs, we may want to enable/disable DPM
1021                         * when DCBx change occurs
1022                         */
1023                        if (ECORE_IS_ROCE_PERSONALITY(p_hwfn))
1024                                ecore_roce_dpm_dcbx(p_hwfn, p_ptt);
1025 #endif
1026                }
1027        }

1029        ecore_dcbx_get_params(p_hwfn, p_ptt, &p_hwfn->p_dcbx_info->get, type);

1031        if (type == ECORE_DCBX_OPERATIONAL_MIB) {
1032                struct ecore_dcbx_results *p_data;
1033                u16 val;

1035                /* Update the DSCP to TC mapping bit if required */
1036                if (p_hwfn->p_dcbx_info->dscp_nig_update) {
1037                        ecore_wr(p_hwfn, p_ptt, NIG_REG_DSCP_TO_TC_MAP_ENABLE,
1038                                 0x1);
1039                        p_hwfn->p_dcbx_info->dscp_nig_update = false;
1040                }

1042                /* Configure in NIG which protocols support EDPM and should
1043                 * honor PFC.
1044                 */
1045                p_data = &p_hwfn->p_dcbx_info->results;
1046                val = (0x1 << p_data->arr[DCBX_PROTOCOL_ROCE].tc) |
1047                        (0x1 << p_data->arr[DCBX_PROTOCOL_ROCE_V2].tc);
1048                val <<= NIG_REG_TX_EDPM_CTRL_TX_EDPM_TC_EN_SHIFT;
1049                val |= NIG_REG_TX_EDPM_CTRL_TX_EDPM_EN;
1050                ecore_wr(p_hwfn, p_ptt, NIG_REG_TX_EDPM_CTRL, val);
1051        }

1053        OSAL_DCBX_AEN(p_hwfn, type);

1055        return rc;
1056 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  120523 Fri Oct  5 15:14:41 2018
new/usr/src/uts/common/io/qede/579xx/drivers/ecore/ecore_mcp.c
9724 qede needs updates for newer GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, v.1,  (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://opensource.org/licenses/CDDL-1.0.
 * 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 2014-2017 Cavium, Inc.
 * The contents of this file are subject to the terms of the Common Development
 * and Distribution License, v.1,  (the "License").

 * You may not use this file except in compliance with the License.

 * You can obtain a copy of the License at available
 * at http://opensource.org/licenses/CDDL-1.0

 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/*
 * Copyright 2018 Joyent, Inc.
 */

#include "bcm_osal.h"
#include "ecore.h"
#include "ecore_status.h"
#include "nvm_map.h"
#include "nvm_cfg.h"
#include "ecore_mcp.h"
#include "mcp_public.h"
#include "reg_addr.h"
#include "ecore_hw.h"
#include "ecore_init_fw_funcs.h"
#include "ecore_sriov.h"
#include "ecore_vf.h"
#include "ecore_iov_api.h"
#include "ecore_gtt_reg_addr.h"
#include "ecore_iro.h"
#include "ecore_dcbx.h"
#include "ecore_sp_commands.h"

#define CHIP_MCP_RESP_ITER_US 10
#define EMUL_MCP_RESP_ITER_US 1000 * 1000

#define ECORE_DRV_MB_MAX_RETRIES        (500 * 1000) /* Account for 5 sec */
```



```
#define ECORE_MCP_RESET_RETRIES         (50 * 1000) /* Account for 500 msec */

#define DRV_INNER_WR(_p_hwfn, _p_ptt, _ptr, _offset, _val) \
        ecore_wr(_p_hwfn, _p_ptt, (_p_hwfn->mcp_info->_ptr + _offset), \
                 _val)

#define DRV_INNER_RD(_p_hwfn, _p_ptt, _ptr, _offset) \
        ecore_rd(_p_hwfn, _p_ptt, (_p_hwfn->mcp_info->_ptr + _offset))

#define DRV_MB_WR(_p_hwfn, _p_ptt, _field, _val) \
        DRV_INNER_WR(p_hwfn, _p_ptt, drv_mb_addr, \
                     OFFSETOF(struct public_drv_mb, _field), _val)

#define DRV_MB_RD(_p_hwfn, _p_ptt, _field) \
        DRV_INNER_RD(_p_hwfn, _p_ptt, drv_mb_addr, \
                     OFFSETOF(struct public_drv_mb, _field))

#define PDA_COMP (((FW_MAJOR_VERSION) + (FW_MINOR_VERSION << 8)) << \
        DRV_ID_PDA_COMP_VER_SHIFT)

#define MCP_BYTES_PER_MBIT_SHIFT 17

#ifndef ASIC_ONLY
static int loaded;
static int loaded_port[MAX_NUM_PORTS] = { 0 };
#endif

bool ecore_mcp_is_init(struct ecore_hwfn *p_hwfn)
{
        if (!p_hwfn->mcp_info || !p_hwfn->mcp_info->public_base)
                return false;
        return true;
}
```

*unchanged_portion_omitted_*

```
static enum _ecore_status_t ecore_do_mcp_cmd(struct ecore_hwfn *p_hwfn,
                                             struct ecore_ptt *p_ptt,
                                             u32 cmd, u32 param,
                                             u32 *o_mcp_resp, u32 *o_mcp_param)
{
        u32 delay = CHIP_MCP_RESP_ITER_US;
        u32 max_retries = ECORE_DRV_MB_MAX_RETRIES;
        u32 seq, cnt = 1, actual_mb_seq __unused;
        u32 seq, cnt = 1, actual_mb_seq;
        enum _ecore_status_t rc = ECORE_SUCCESS;

#ifndef ASIC_ONLY
        if (CHIP_REV_IS_EMUL(p_hwfn->p_dev))
                delay = EMUL_MCP_RESP_ITER_US;
        /* There is a built-in delay of 100usec in each MFW response read */
        if (CHIP_REV_IS_FPGA(p_hwfn->p_dev))
                max_retries /= 10;
#endif

        /* Get actual driver mailbox sequence */
        actual_mb_seq = DRV_MB_RD(p_hwfn, p_ptt, drv_mb_header) &
                        DRV_MSG_SEQ_NUMBER_MASK;

        /* Use MCP history register to check if MCP reset occurred between
         * init time and now.
         */
        if (p_hwfn->mcp_info->mcp_hist !=
            ecore_rd(p_hwfn, p_ptt, MISCS_REG_GENERIC_POR_0)) {
                DP_VERBOSE(p_hwfn, ECORE_MSG_SP, "Rereading MCP offsets\n");
                ecore_load_mcp_offsets(p_hwfn, p_ptt);
                ecore_mcp_cmd_port_init(p_hwfn, p_ptt);
```

```
 392         }
 393         seq = ++p_hwfn->mcp_info->drv_mb_seq;

 395         /* Set drv param */
 396         DRV_MB_WR(p_hwfn, p_ptt, drv_mb_param, param);

 398         /* Set drv command along with the updated sequence */
 399         DRV_MB_WR(p_hwfn, p_ptt, drv_mb_header, (cmd | seq));

 401         DP_VERBOSE(p_hwfn, ECORE_MSG_SP,
 402                    "wrote command (%x) to MFW MB param 0x%08x\n",
 403                    (cmd | seq), param);

 405         do {
 406                 /* Wait for MFW response */
 407                 OSAL_UDELAY(delay);
 408                 *o_mcp_resp = DRV_MB_RD(p_hwfn, p_ptt, fw_mb_header);

 410                 /* Give the FW up to 5 second (500*10ms) */
 411         } while ((seq != (*o_mcp_resp & FW_MSG_SEQ_NUMBER_MASK)) &&
 412                  (cnt++ < max_retries));

 414         DP_VERBOSE(p_hwfn, ECORE_MSG_SP,
 415                    "[after %d ms] read (%x) seq is (%x) from FW MB\n",
 416                    cnt * delay, *o_mcp_resp, seq);

 418         /* Is this a reply to our command? */
 419         if (seq == (*o_mcp_resp & FW_MSG_SEQ_NUMBER_MASK)) {
 420                 *o_mcp_resp &= FW_MSG_CODE_MASK;
 421                 /* Get the MCP param */
 422                 *o_mcp_param = DRV_MB_RD(p_hwfn, p_ptt, fw_mb_param);
 423         } else {
 424                 /* FW BUG! */
 425                 DP_ERR(p_hwfn, "MFW failed to respond [cmd 0x%x param 0x%x]\n",
 426                        cmd, param);
 427                 ecore_mcp_print_cpu_info(p_hwfn, p_ptt);
 428                 *o_mcp_resp = 0;
 429                 rc = ECORE_AGAIN;
 430                 ecore_hw_err_notify(p_hwfn, ECORE_HW_ERR_MFW_RESP_FAIL);
 431         }
 432         return rc;
 433 }
______unchanged_portion_omitted_

1436 static void ecore_mcp_send_protocol_stats(struct ecore_hwfn *p_hwfn,
1437                                           struct ecore_ptt *p_ptt,
1438                                           enum MFW_DRV_MSG_TYPE type)
1439 {
1440         enum ecore_mcp_protocol_type stats_type __unused;
1436         enum ecore_mcp_protocol_type stats_type;
1441         union ecore_mcp_protocol_stats stats;
1442         struct ecore_mcp_mb_params mb_params;
1443         u32 hsi_param;
1444         enum _ecore_status_t rc;

1446         switch (type) {
1447         case MFW_DRV_MSG_GET_LAN_STATS:
1448                 stats_type = ECORE_MCP_LAN_STATS;
1449                 hsi_param = DRV_MSG_CODE_STATS_TYPE_LAN;
1450                 break;
1451         case MFW_DRV_MSG_GET_FCOE_STATS:
1452                 stats_type = ECORE_MCP_FCOE_STATS;
1453                 hsi_param = DRV_MSG_CODE_STATS_TYPE_FCOE;
1454                 break;
1455         case MFW_DRV_MSG_GET_ISCSI_STATS:
1456                 stats_type = ECORE_MCP_ISCSI_STATS;
```



```
1457                 hsi_param = DRV_MSG_CODE_STATS_TYPE_ISCSI;
1458                 break;
1459         case MFW_DRV_MSG_GET_RDMA_STATS:
1460                 stats_type = ECORE_MCP_RDMA_STATS;
1461                 hsi_param = DRV_MSG_CODE_STATS_TYPE_RDMA;
1462                 break;
1463         default:
1464                 DP_NOTICE(p_hwfn, false, "Invalid protocol type %d\n", type);
1465                 return;
1466         }

1468         OSAL_GET_PROTOCOL_STATS(p_hwfn->p_dev, stats_type, &stats);

1470         OSAL_MEM_ZERO(&mb_params, sizeof(mb_params));
1471         mb_params.cmd = DRV_MSG_CODE_GET_STATS;
1472         mb_params.param = hsi_param;
1473         mb_params.p_data_src = &stats;
1474         mb_params.data_src_size = sizeof(stats);
1475         rc = ecore_mcp_cmd_and_union(p_hwfn, p_ptt, &mb_params);
1476         if (rc != ECORE_SUCCESS)
1477                 DP_ERR(p_hwfn, "Failed to send protocol stats, rc = %d\n", rc);
1478 }
______unchanged_portion_omitted_

3129 enum _ecore_status_t ecore_mcp_phy_sfp_read(struct ecore_hwfn *p_hwfn,
3130                                             struct ecore_ptt *p_ptt,
3131                                             u32 port, u32 addr, u32 offset,
3132                                             u32 len, u8 *p_buf)
3133 {
3134         struct ecore_mcp_nvm_params params;
3131         enum _ecore_status_t rc;
3135         u32 bytes_left, bytes_to_copy, buf_size;

3137         OSAL_MEMSET(&params, 0, sizeof(struct ecore_mcp_nvm_params));
3138         params.nvm_common.offset =
3139                 (port << DRV_MB_PARAM_TRANSCEIVER_PORT_SHIFT) |
3140                 (addr << DRV_MB_PARAM_TRANSCEIVER_I2C_ADDRESS_SHIFT);
3141         addr = offset;
3142         offset = 0;
3143         bytes_left = len;
3144         params.type = ECORE_MCP_NVM_RD;
3145         params.nvm_rd.buf_size = &buf_size;
3146         params.nvm_common.cmd = DRV_MSG_CODE_TRANSCEIVER_READ;
3147         while (bytes_left > 0) {
3148                 bytes_to_copy = OSAL_MIN_T(u32, bytes_left,
3149                                            MAX_I2C_TRANSACTION_SIZE);
3150                 params.nvm_rd.buf = (u32 *)(p_buf + offset);
3151                 params.nvm_common.offset &=
3152                         (DRV_MB_PARAM_TRANSCEIVER_I2C_ADDRESS_MASK |
3153                          DRV_MB_PARAM_TRANSCEIVER_PORT_MASK);
3154                 params.nvm_common.offset |=
3155                         ((addr + offset) <<
3156                          DRV_MB_PARAM_TRANSCEIVER_OFFSET_SHIFT);
3157                 params.nvm_common.offset |=
3158                         (bytes_to_copy << DRV_MB_PARAM_TRANSCEIVER_SIZE_SHIFT);
3159                 (void) ecore_mcp_nvm_command(p_hwfn, p_ptt, &params);
3156                 rc = ecore_mcp_nvm_command(p_hwfn, p_ptt, &params);
3160                 if ((params.nvm_common.resp & FW_MSG_CODE_MASK) ==
3161                     FW_MSG_CODE_TRANSCEIVER_NOT_PRESENT) {
3162                         return ECORE_NODEV;
3163                 } else if ((params.nvm_common.resp & FW_MSG_CODE_MASK) !=
3164                            FW_MSG_CODE_TRANSCEIVER_DIAG_OK)
3165                         return ECORE_UNKNOWN_ERROR;

3167                 offset += *params.nvm_rd.buf_size;
3168                 bytes_left -= *params.nvm_rd.buf_size;
```

```
3169         }

3171         return ECORE_SUCCESS;
3172 }

3174 enum _ecore_status_t ecore_mcp_phy_sfp_write(struct ecore_hwfn *p_hwfn,
3175                                              struct ecore_ptt *p_ptt,
3176                                              u32 port, u32 addr, u32 offset,
3177                                              u32 len, u8 *p_buf)
3178 {
3179         struct ecore_mcp_nvm_params params;
3177         enum _ecore_status_t rc;
3180         u32 buf_idx, buf_size;

3182         OSAL_MEMSET(&params, 0, sizeof(struct ecore_mcp_nvm_params));
3183         params.nvm_common.offset =
3184                 (port << DRV_MB_PARAM_TRANSCEIVER_PORT_SHIFT) |
3185                 (addr << DRV_MB_PARAM_TRANSCEIVER_I2C_ADDRESS_SHIFT);
3186         params.type = ECORE_MCP_NVM_WR;
3187         params.nvm_common.cmd = DRV_MSG_CODE_TRANSCEIVER_WRITE;
3188         buf_idx = 0;
3189         while (buf_idx < len) {
3190                 buf_size = OSAL_MIN_T(u32, (len - buf_idx),
3191                                       MAX_I2C_TRANSACTION_SIZE);
3192                 params.nvm_common.offset &=
3193                         (DRV_MB_PARAM_TRANSCEIVER_I2C_ADDRESS_MASK |
3194                          DRV_MB_PARAM_TRANSCEIVER_PORT_MASK);
3195                 params.nvm_common.offset |=
3196                         ((offset + buf_idx) <<
3197                          DRV_MB_PARAM_TRANSCEIVER_OFFSET_SHIFT);
3198                 params.nvm_common.offset |=
3199                         (buf_size << DRV_MB_PARAM_TRANSCEIVER_SIZE_SHIFT);
3200                 params.nvm_wr.buf_size = buf_size;
3201                 params.nvm_wr.buf = (u32 *)&p_buf[buf_idx];
3202                 (void) ecore_mcp_nvm_command(p_hwfn, p_ptt, &params);
3200                 rc = ecore_mcp_nvm_command(p_hwfn, p_ptt, &params);
3203                 if ((params.nvm_common.resp & FW_MSG_CODE_MASK) ==
3204                     FW_MSG_CODE_TRANSCEIVER_NOT_PRESENT) {
3205                         return ECORE_NODEV;
3206                 } else if ((params.nvm_common.resp & FW_MSG_CODE_MASK) !=
3207                            FW_MSG_CODE_TRANSCEIVER_DIAG_OK)
3208                         return ECORE_UNKNOWN_ERROR;

3210                 buf_idx += buf_size;
3211         }

3213         return ECORE_SUCCESS;
3214 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   65388 Fri Oct  5 15:14:41 2018
new/usr/src/uts/common/io/qede/579xx/drivers/ecore/ecore_phy.c
9724 qede needs updates for newer GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, v.1,  (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://opensource.org/licenses/CDDL-1.0.
 * 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 2014-2017 Cavium, Inc.
 * The contents of this file are subject to the terms of the Common Development
 * and Distribution License, v.1,  (the "License").

 * You may not use this file except in compliance with the License.

 * You can obtain a copy of the License at available
 * at http://opensource.org/licenses/CDDL-1.0

 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/*
 * Copyright 2018 Joyent, Inc.
 */

#include "bcm_osal.h"
#include "ecore.h"
#include "reg_addr.h"
#include "ecore_hw.h"
#include "ecore_hsi_common.h"
#include "ecore_mcp.h"
#include "nvm_cfg.h"
#include "ecore_phy_api.h"

#define SERDESID 0x900e


enum _ecore_status_t ecore_phy_read(struct ecore_hwfn *p_hwfn,
                                    struct ecore_ptt *p_ptt, u32 port, u32 lane,
                                    u32 addr, u32 cmd, u8 *buf)
{
        return ecore_mcp_phy_read(p_hwfn->p_dev, cmd,
                          addr | (lane << 16) | (1<<29) | (port << 30), buf, 8);
}
```

*_____unchanged_portion_omitted_*

```
/* get mac status */
```



```
static int ecore_ah_e5_phy_mac_stat(struct ecore_hwfn *p_hwfn,
                                    struct ecore_ptt *p_ptt, u32 port,
                                    char *p_phy_result_buf)
{
        u32 length, reg_id, addr, data_hi __unused, data_lo;
        u32 length, reg_id, addr, data_hi, data_lo;

        length = OSAL_SPRINTF(p_phy_result_buf,
                              "MAC stats for port %d (only non-zero)\n", port);

        for (reg_id = 0; reg_id < OSAL_ARRAY_SIZE(ah_stat_regs); reg_id++) {
                addr = ah_stat_regs[reg_id].reg;
                data_lo = ecore_rd(p_hwfn, p_ptt,
                                   NWM_REG_MAC0_K2_E5 +
                                   NWM_REG_MAC0_SIZE * 4 * port +
                                   addr);
                data_hi = ecore_rd(p_hwfn, p_ptt,
                                   NWM_REG_MAC0_K2_E5 +
                                   NWM_REG_MAC0_SIZE * 4 * port +
                                   addr + 4);

                if (data_lo) {  /* Only non-zero */
                        length += OSAL_SPRINTF(&p_phy_result_buf[length],
                                               "%-10s: 0x%08x (%s)\n",
                                               ah_stat_regs[reg_id].name,
                                               data_lo,
                                               ah_stat_regs[reg_id].desc);
                }
        }

        return ECORE_SUCCESS;
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   63360 Fri Oct  5 15:14:42 2018
new/usr/src/uts/common/io/qede/qede_gld.c
9724 qede needs updates for newer GCC
**********************************************************
```

```
/*
* CDDL HEADER START
*
* The contents of this file are subject to the terms of the
* Common Development and Distribution License, v.1,  (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://opensource.org/licenses/CDDL-1.0.
* 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 2014-2017 Cavium, Inc.
* The contents of this file are subject to the terms of the Common Development
* and Distribution License, v.1,  (the "License").

* You may not use this file except in compliance with the License.

* You can obtain a copy of the License at available
* at http://opensource.org/licenses/CDDL-1.0

* See the License for the specific language governing permissions and
* limitations under the License.
*/

/*
 * Copyright 2018 Joyent, Inc.
 */

#include "qede.h"

#define FP_LOCK(ptr)    \
mutex_enter(&ptr->fp_lock);
#define FP_UNLOCK(ptr)  \
mutex_exit(&ptr->fp_lock);

int
qede_ucst_find(qede_t *qede, const uint8_t *mac_addr)
{
        int slot;

        for(slot = 0; slot < qede->ucst_total; slot++) {
                if (bcmp(qede->ucst_mac[slot].mac_addr.ether_addr_octet,
                    mac_addr, ETHERADDRL) == 0) {
                        return (slot);
                }
        }
        return (-1);

}
```

*unchanged_portion_omitted_*


```
/*
 * Set Loopback mode
 */

static enum ioc_reply
qede_set_loopback_mode(qede_t *qede, uint32_t mode)
{
        int i = 0;
        int ret, i = 0;
        struct ecore_dev *edev = &qede->edev;
        struct ecore_hwfn *hwfn;
        struct ecore_ptt *ptt = NULL;
        struct ecore_mcp_link_params *link_params;

        hwfn = &edev->hwfns[0];
        link_params = ecore_mcp_get_link_params(hwfn);
        ptt = ecore_ptt_acquire(hwfn);

        switch(mode) {
        default:
                qede_info(qede, "unknown loopback mode !!");
                ecore_ptt_release(hwfn, ptt);
                return IOC_INVAL;

        case QEDE_LOOP_NONE:
                ecore_mcp_set_link(hwfn, ptt, 0);

                while (qede->params.link_state && i < 5000) {
                        OSAL_MSLEEP(1);
                        i++;
                }
                i = 0;

                link_params->loopback_mode = ETH_LOOPBACK_NONE;
                qede->loop_back_mode = QEDE_LOOP_NONE;
                (void) ecore_mcp_set_link(hwfn, ptt, 1);
                ret = ecore_mcp_set_link(hwfn, ptt, 1);
                ecore_ptt_release(hwfn, ptt);

                while (!qede->params.link_state && i < 5000) {
                        OSAL_MSLEEP(1);
                        i++;
                }
                return IOC_REPLY;

        case QEDE_LOOP_INTERNAL:
                qede_print("!%s(%d) : loopback mode (INTERNAL) is set!",
                    __func__, qede->instance);
                ecore_mcp_set_link(hwfn, ptt, 0);

                while(qede->params.link_state && i < 5000) {
                        OSAL_MSLEEP(1);
                        i++;
                }
                i = 0;
                link_params->loopback_mode = ETH_LOOPBACK_INT_PHY;
                qede->loop_back_mode = QEDE_LOOP_INTERNAL;
                (void) ecore_mcp_set_link(hwfn, ptt, 1);
                ret = ecore_mcp_set_link(hwfn, ptt, 1);
                ecore_ptt_release(hwfn, ptt);

                while(!qede->params.link_state && i < 5000) {
                        OSAL_MSLEEP(1);
                        i++;
                }
```

```
1324                 return IOC_REPLY;

1326         case QEDE_LOOP_EXTERNAL:
1327                 qede_print("!%s(%d) : External loopback mode is not supported",
1328                     __func__, qede->instance);
1329                 ecore_ptt_release(hwfn, ptt);
1330                 return IOC_INVAL;
1331         }
1332 }
______unchanged_portion_omitted_

1464 static int
1465 qede_ioctl_rd_wr_nvram(qede_t *qede, mblk_t *mp)
1466 {
1467         qede_nvram_data_t *data1 = (qede_nvram_data_t *)(mp->b_cont->b_rptr);
1468         qede_nvram_data_t *data2, *next_data;
1469         struct ecore_dev *edev = &qede->edev;
1470         uint32_t hdr_size = 24, bytes_to_copy, copy_len = 0;
1467         uint32_t ret = 0, hdr_size = 24, bytes_to_copy, copy_len = 0;
1471         uint32_t copy_len1 = 0;
1472         uint32_t addr = data1->off;
1473         uint32_t size = data1->size, i, buf_size;
1474         uint8_t cmd, cmd2;
1475         uint8_t *buf, *tmp_buf;
1476         mblk_t *mp1;

1478         cmd = (uint8_t)data1->unused1;

1480         switch(cmd) {
1481         case QEDE_NVRAM_CMD_READ:
1482                 buf = kmem_zalloc(size, GFP_KERNEL);
1483                 if(buf == NULL) {
1484                         cmn_err(CE_WARN, "memory allocation failed"
1485                         " in nvram read ioctl\n");
1486                         return (DDI_FAILURE);
1487                 }
1488                 (void) ecore_mcp_nvm_read(edev, addr, buf, data1->size);
1485                 ret = ecore_mcp_nvm_read(edev, addr, buf, data1->size);

1490                 copy_len = (MBLKL(mp->b_cont)) - hdr_size;
1491                 if(copy_len > size) {
1492                         (void) memcpy(data1->uabc, buf, size);
1493                         kmem_free(buf, size);
1494                         //OSAL_FREE(edev, buf);
1492                         ret = 0;
1495                         break;
1496                 }
1497                 (void) memcpy(data1->uabc, buf, copy_len);
1498                 bytes_to_copy = size - copy_len;
1499                 tmp_buf = ((uint8_t *)buf) + copy_len;
1500                 copy_len1 = copy_len;
1501                 mp1 = mp->b_cont;
1502                 mp1 = mp1->b_cont;

1504                 while (mp1) {
1505                         copy_len = MBLKL(mp1);
1506                         if(mp1->b_cont == NULL) {
1507                                 copy_len = MBLKL(mp1) - 4;
1508                         }
1509                         data2 = (qede_nvram_data_t *)mp1->b_rptr;
1510                         if (copy_len > bytes_to_copy) {
1511                                 (void) memcpy(data2->uabc, tmp_buf,
1512                                     bytes_to_copy);
1513                                 kmem_free(buf, size);
1514                                 //OSAL_FREE(edev, buf);
1515                                 break;
```



```
1516                         }
1517                         (void) memcpy(data2->uabc, tmp_buf, copy_len);
1518                         tmp_buf = tmp_buf + copy_len;
1519                         copy_len += copy_len;
1520                         mp1 = mp1->b_cont;
1521                         bytes_to_copy = bytes_to_copy - copy_len;
1522                 }
1523
1524                 kmem_free(buf, size);
1525                 //OSAL_FREE(edev, buf);
1526                 break;
1527
1528         case QEDE_NVRAM_CMD_WRITE:
1529                 cmd2 = (uint8_t )data1->cmd2;
1530                 size = data1->size;
1531                 addr = data1->off;
1532                 buf_size =  size; //data1->buf_size;
1533                 //buf_size =  data1->buf_size;
1532                 ret = 0;

1535                 switch(cmd2){
1536                 case START_NVM_WRITE:
1537                         buf = kmem_zalloc(size, GFP_KERNEL);
1538                         //buf = qede->reserved_buf;
1539                         qede->nvm_buf_size = data1->size;
1540                         if(buf == NULL) {
1541                                 cmn_err(CE_WARN,
1542                                 "memory allocation failed in START_NVM_WRITE\n")
1543                                 return DDI_FAILURE;
1544                         }
1545                         qede->nvm_buf_start = buf;
1546                         cmn_err(CE_NOTE,
1547                             "buf = %p, size = %x\n", qede->nvm_buf_start, size);
1548                         qede->nvm_buf = buf;
1549                         qede->copy_len = 0;
1550                         //tmp_buf = buf + addr;
1550                         ret = 0;
1551                         break;
1552
1553                 case ACCUMULATE_NVM_BUF:
1554                         tmp_buf = qede->nvm_buf;
1555                         copy_len = MBLKL(mp->b_cont) - hdr_size;
1556                         if(copy_len > buf_size) {
1557                                 if (buf_size < qede->nvm_buf_size) {
1558                                 (void) memcpy(tmp_buf, data1->uabc, buf_size);
1559                                         qede->copy_len = qede->copy_len +
1560                                             buf_size;
1561                                 } else {
1562                                         (void) memcpy(tmp_buf,
1563                                             data1->uabc, qede->nvm_buf_size);
1564                                         qede->copy_len =
1565                                             qede->copy_len + qede->nvm_buf_size;
1566                                 }
1567                                 tmp_buf = tmp_buf + buf_size;
1568                                 qede->nvm_buf = tmp_buf;
1569                                 //qede->copy_len = qede->copy_len + buf_size;
1570                                 cmn_err(CE_NOTE,
1571                                     "buf_size from app = %x\n", copy_len);
1572                                 ret = 0;
1572                                 break;
1573                         }
1574                         (void) memcpy(tmp_buf, data1->uabc, copy_len);
1575                         tmp_buf = tmp_buf + copy_len;
1576                         bytes_to_copy = buf_size - copy_len;
1577                         mp1 = mp->b_cont;
1578                         mp1 = mp1->b_cont;
```

```
1579                        copy_len1 = copy_len;
1580
1581                        while (mp1) {
1582                                copy_len = MBLKL(mp1);
1583                                if (mp1->b_cont == NULL) {
1584                                        copy_len = MBLKL(mp1) - 4;
1585                                }
1586                                next_data = (qede_nvram_data_t *) mp1->b_rptr;
1587                                if (copy_len > bytes_to_copy){
1588                                        (void) memcpy(tmp_buf, next_data->uabc,
1589                                            bytes_to_copy);
1590                                        qede->copy_len = qede->copy_len +
1591                                            bytes_to_copy;
1593                                        ret = 0;
1592                                        break;
1593                                }
1594                                (void) memcpy(tmp_buf, next_data->uabc,
1595                                    copy_len);
1596                                qede->copy_len = qede->copy_len + copy_len;
1597                                tmp_buf = tmp_buf + copy_len;
1598                                copy_len = copy_len1 + copy_len;
1599                                bytes_to_copy = bytes_to_copy - copy_len;
1600                                mp1 = mp1->b_cont;
1601                        }
1602                        qede->nvm_buf = tmp_buf;
1605                        ret = 0;
1603                        break;

1605                case STOP_NVM_WRITE:
1606                        //qede->nvm_buf = tmp_buf;
1610                        ret = 0;
1607                        break;
1608                case READ_BUF:
1609                        tmp_buf = (uint8_t *)qede->nvm_buf_start;
1610                        for(i = 0; i < size ; i++){
1611                                cmn_err(CE_NOTE,
1612                                    "buff (%d) : %d\n", i, *tmp_buf);
1613                                tmp_buf ++;
1614                        }
1619                        ret = 0;
1615                        break;
1616                }
1617                break;
1618        case QEDE_NVRAM_CMD_PUT_FILE_DATA:
1619                tmp_buf = qede->nvm_buf_start;
1620                (void) ecore_mcp_nvm_write(edev, ECORE_PUT_FILE_DATA,
1625                ret = ecore_mcp_nvm_write(edev, ECORE_PUT_FILE_DATA,
1621                        addr, tmp_buf, size);
1622                kmem_free(qede->nvm_buf_start, size);
1623                //OSAL_FREE(edev, tmp_buf);
1624                cmn_err(CE_NOTE, "total size = %x, copied size = %x\n",
1625                    qede->nvm_buf_size, qede->copy_len);
1626                tmp_buf = NULL;
1627                qede->nvm_buf = NULL;
1628                qede->nvm_buf_start = NULL;
1634                ret = 0;
1629                break;

1631        case QEDE_NVRAM_CMD_SET_SECURE_MODE:
1632                (void) ecore_mcp_nvm_set_secure_mode(edev, addr);
1638                ret = ecore_mcp_nvm_set_secure_mode(edev, addr);
1633                break;

1635        case QEDE_NVRAM_CMD_DEL_FILE:
1636                (void) ecore_mcp_nvm_del_file(edev, addr);
1642                ret = ecore_mcp_nvm_del_file(edev, addr);
```



```
1637                break;

1639        case QEDE_NVRAM_CMD_PUT_FILE_BEGIN:
1640                (void) ecore_mcp_nvm_put_file_begin(edev, addr);
1646                ret = ecore_mcp_nvm_put_file_begin(edev, addr);
1641                break;

1643        case QEDE_NVRAM_CMD_GET_NVRAM_RESP:
1644                buf = kmem_zalloc(size, KM_SLEEP);
1645                (void) ecore_mcp_nvm_resp(edev, buf);
1651                ret = ecore_mcp_nvm_resp(edev, buf);
1646                (void)memcpy(data1->uabc, buf, size);
1647                kmem_free(buf, size);
1648                break;

1650        default:
1651                cmn_err(CE_WARN,
1652                    "wrong command in NVRAM read/write from application\n");
1653                break;
1654        }
1655        return (DDI_SUCCESS);
1656 }
```

*unchanged_portion_omitted_*