Profinet and Ethernet/IP Initialization Guide

ASM Software Apps

Exported on 08/14/2025

Table of Contents

1 ICSS EMAC Initialization

1.1 Creating and initializing ICSS_EMAC_Params

ICSS_EMAC Parameters are used with the ICSS_EMAC_open() call for initialization of ICSS EMAC. Default values for these parameters are set using ICSS_EMAC_Params_init().

```
1  ICSS_EMAC_Params        icssEmacParams;
2  ICSS_EMAC_Params_init(&icssEmacParams);
```

1.2 PRUICSS Interrupt controller and Memory Map initialization data

Load the PRUICSS Interrupt controller initialization data from the tiswitch_pruss_intc_mapping.h file.

For the Memory Map initialization data, the icss_emac_mmap.h file can be referred to.

```
1  PRUICSS_IntcInitData pruss_intc_initdata = PRUSS_INTC_INITDATA;
2  icssEmacParams.pruicssIntcInitData       = &pruss_intc_initdata;
3  icssEmacParams.fwStaticMMap              = &(icss_emacFwStaticCfg[1]);
4  icssEmacParams.fwDynamicMMap             = &icss_emacFwDynamicCfg;
5  icssEmacParams.pruicssHandle             = prusshandle;
```

1.3 Configure the callback function

1.3.1 Link callback configuration

Application specific functions can be programmed as Link callback functions individually for the 2 ports in case of a link change.

```
1  icssEmacParams.callBackObject.port0LinkCallBack.callBack =
   (ICSS_EMAC_CallBack)app_Specific_Port1linkResetCallBack;
2  icssEmacParams.callBackObject.port0LinkCallBack.userArg = (void*)
   (app_Specific_Port1linkHandle);
3  icssEmacParams.callBackObject.port1LinkCallBack.callBack =
   (ICSS_EMAC_CallBack)app_Specific_Port2linkResetCallBack;
4  icssEmacParams.callBackObject.port1LinkCallBack.userArg = (void*)
   (app_Specific_Port2linkHandle);
```

1.3.2 Rx RT and NRT callback configuration

When packets are received with queueNumber >= ethPrioQueue, Rx RT callback is invoked. Otherwise, the Rx NRT callback is invoked.

Any application specific function can be programmed as the Rx RT callback.

When using the lwIP TCP/IP stack, the NRT callback should be Lwip2Emac_serviceRx with Lwipif_handle as the userArg.

```
1  icssEmacParams.callBackObject.rxRTCallBack.callBack =
   (ICSS_EMAC_CallBack)app_Specific_RxRTCallBack;
2  icssEmacParams.callBackObject.rxRTCallBack.userArg = (void*)
   (app_Specific_RxRTHandle);
3  icssEmacParams.callBackObject.rxNRTCallBack.callBack =
   (ICSS_EMAC_CallBack)Lwip2Emac_serviceRx;
4  icssEmacParams.callBackObject.rxNRTCallBack.userArg = (void*)
   (Lwipif_handle);
```

NOTE: Similarly, different application specific callback functions can be programmed as callbacks for Tx completion or as custom Tx or Rx callbacks. Refer to the "ICSS_EMAC_CallBackObject" structure for more details.

1.4 Configure the Ethernet PHY Handles

Assign the Ethernet PHY handles. ETHPHY instances are added from the SysConfig.

```
1  icssEmacParams.ethphyHandle[0] = gEthPhyHandle[CONFIG_ETHPHY0];
2  icssEmacParams.ethphyHandle[1] = gEthPhyHandle[CONFIG_ETHPHY1];
```

1.5 Configure the MAC Address of the Interface

The MAC address is programmed in the EEPROM of the board. One can use a helper function to retrieve it.

An example helper function is as follows.

```
1  /* EEPROM data offset in I2C EEPROM Flash */
2  #define I2C_EEPROM_DATA_OFFSET          (0x8000)
3  #define I2C_EEPROM_MAC_DATA_OFFSET      (0x42)
4
5  /* Global variable to store interface MAC address */
6  uint8_t lclMac[6];
7
8  void PN_socgetMACAddress(uint8_t *lclMac)
9  {
```

```
10      EEPROM_read(gEepromHandle[CONFIG_EEPROM0],
I2C_EEPROM_MAC_DATA_OFFSET, lclMac, 6U);
11  }
```

Once the MAC address is retrieved, assign it to the icssEmacParams. In the above code block, the MAC address will be copied to the lclMac array.

```
1  memcpy(&(icssEmacParams.macId[0]), &(lclMac[0]), 6);
```

1.6 Call the ICSS_EMAC_open API

Once the initialization parameters are configured in icssEmacParams, call the ICSS_EMAC_open API with these parameters.

```
1  emachandle = ICSS_EMAC_open(CONFIG_ICSS_EMAC0, &icssEmacParams);
```

2 Profinet Handle Initialization

2.1 Profinet Handle creation

The Profinet related include files have to be included in the file and global instance of Profinet Handle should be created.

```
/* Profinet related includes */
#include <industrial_comms/profinet_device/icss_fwhal/iPNDrv.h>
#include <industrial_comms/profinet_device/icss_fwhal/PN_HandleDef.h>
#include <industrial_comms/profinet_device/icss_fwhal/PN_Handle.h>
#include <industrial_comms/profinet_device/icss_fwhal/iPNLegacy.h>

/* Global Variables */
PN_Handle appPnHandle;
```

2.2 Memory allocation for Profinet Handle

Make sure memory allocation is done for the Profinet Handle in the application code.

```
appPnHandle = (PN_Handle)malloc(sizeof(PN_Config));
```

2.3 Profinet Handle Initialization

The following code block can be taken as a reference to do required Profinet Handle Initialization. This function can be invoked in the application and accepts the Profinet Handle, ICSS EMAC Handle and PRUICSS Handle as required arguments.

```
void PN_initDefaultValues(PN_Handle appPnHandle, ICSS_EMAC_Handle
emachandle, PRUICSS_Handle prusshandle)
{
    /* Assign EMAC and PRU handles */
    appPnHandle->emacHandle = emachandle;
    appPnHandle->pruicssHandle  = prusshandle;

    appPnHandle->pLegPkt = NULL;

    /* Configure the pulse width for sync0: 5000+1 cycles i.e. 25us */
    appPnHandle->pnPtcpConfig.ptcpSync0PinPulseWidth = 5000;
```

```
11      /* Program cmp1 reg with period, used for sync0 signal generation: 10
    us */
12      appPnHandle->pnPtcpConfig.ptcpSync0PinStart = 10000;
13      appPnHandle->pnPtcpConfig.ptcpSyncFilterfactor = 8;
14
15      /* Initialize ISOM Config */
16      appPnHandle->pnIsoMConfig.isoMIntCreateFlag = 0;
17      appPnHandle->pnIsoMConfig.isoMNumEvents = 0;
18      appPnHandle->pnIsoMConfig.isoMIntConfig.isrFnPtr = NULL;
19      appPnHandle->pnIsoMConfig.isoMIntConfig.args = NULL;
20
21      /* Initialize default values */
22      appPnHandle->initRtcDrvFlag = TRUE;
23      appPnHandle->initRtcMemFlag = 0;
24      appPnHandle->mrpState = MRPREADY;
25      appPnHandle->legState = NOINIT;
26
27      /* Initialize the structure storing the PTCP info */
28      appPnHandle->pnPtcpConfig.cycleCtrInitPending = 0;
29      appPnHandle->pnPtcpConfig.calculatedCycleCtr = 0;
30      appPnHandle->pnPtcpConfig.masterChange = 0;
31      appPnHandle->pnPtcpConfig.phaseCtrChange = 0;
32      appPnHandle->pnPtcpConfig.maxSeqId = 0;
33      appPnHandle->pnPtcpConfig.minSeqId = 0;
34
35      /* Initialize the structure storing the different interrupt
    configurations of Profinet */
36      appPnHandle->pnIntConfig.ppmIntConfig.isrFnPtr = &PN_ppmIsrHandler;
37      appPnHandle->pnIntConfig.ppmIntConfig.args = appPnHandle;
38      appPnHandle->pnIntConfig.cpmIntConfig.isrFnPtr = &PN_cpmIsrHandler;
39      appPnHandle->pnIntConfig.cpmIntConfig.args = appPnHandle;
40      appPnHandle->pnIntConfig.dhtIntConfig.isrFnPtr = &PN_dhtIsrHandler;
41      appPnHandle->pnIntConfig.dhtIntConfig.args = appPnHandle;
42  #ifdef PTCP_SUPPORT
43      appPnHandle->pnIntConfig.ptcpIntConfig.isrFnPtr = &PN_PTCP_isrHandler;
44      appPnHandle->pnIntConfig.ptcpIntConfig.args = appPnHandle;
45  #endif
46
47      /* Function pointer to get MAC address */
48      appPnHandle->getMACAddress = &PN_getMACAddr;
49
50      /* Do the required Interrupt configuration */
51      PN_socinitIRTHandle(appPnHandle);
52  }
```

In the above the code block, in line 47, a reference to the "PN_getMACAddr" is passed to the function pointer to get MAC address in the Profinet handle. This is a sample application specific function to get the MAC address explained below.

Please note that this function makes use of a global variable "IntfMacAddress_s" which contains the same MAC address as "icssEmacParams.macId" used for ICSS_EMAC_open (Refer to line 15 of the next code block).

```
/* Global Variables */

/* MAC address length in bytes */
#define PRU_PN_MAC_ADDR_LEN             6
/* Global array to store the interface MAC address */
static uint8_t IntfMacAddress_s[PRU_PN_MAC_ADDR_LEN];
/* Interface MAC Addr index */
#define INTERFACE_MAC                   0
/* Port1 MAC Addr index */
#define PORT1_MAC                       1
/* Port2 MAC Addr index */
#define PORT2_MAC                       2

/* Copy the Interface MAC address from icssEmacParams*/
memmove (icssEmacParams.macId, IntfMacAddress_s, PRU_PN_MAC_ADDR_LEN);

/* PN_getMACAddr function definition */
void PN_getMACAddr(uint8_t index, uint8_t *lclMac)
{
    uint32_t *MAC_0_3 = NULL;
    uint16_t *MAC_4_5 = NULL;
    uint8_t i;

    /* Below code shall never be in production environment - to handle boards with uninitialized EEPROM MAC ID */
    MAC_0_3 = (uint32_t *)(IntfMacAddress_s);
    MAC_4_5 = (uint16_t *)(&IntfMacAddress_s[4]);

    if((*MAC_0_3 == 0xFFFFFFFFU) && (*MAC_4_5 == 0xFFFFU))
    {
        DebugP_log("\r\n Warning : Board EEPROM is not initialized correctly with MAC address - please fix this ");
        DebugP_log("\r\n Generating random MAC address for now \n");
        *MAC_0_3 = rand();
        *MAC_4_5 = rand() & 0xFFFF;
        IntfMacAddress_s[0] &= 0xFE; /* Force unicast MAC address */
    }
    switch(index)
    {
        case INTERFACE_MAC:
            break;
        case PORT1_MAC:
            IntfMacAddress_s[5] += 1;
            break;
        case PORT2_MAC:
            IntfMacAddress_s[5] += 2;
            break;
        default:
            break;
    }

    for(i=0;i<6;i++)
```

```
    {
        lclMac[i]=IntfMacAddress_s[i];
    }
}
```

2.4 Interrupt Configuration

The next step is to do the required interrupt configuration in the Profinet Handle. For this, a sample function called "PN_socinitIRTHandle" is created and invoked towards the end of the "PN_initDefaultValues" function referred in the previous section. It accepts the Profinet Handle as required argument.

Please note that some of the interrupt configurations differ depending on ICSS instance being used by the application. This can be taken care of in the following code block.

```
#define PN_RX_INT_OFFSET                            (0)
#define PN_PPM_INT_OFFSET                           (1)
#define PN_CPM_INT_OFFSET                           (2)
#define PN_DHT_INT_OFFSET                           (3)
#define PN_PTCP_INT_OFFSET                          (4)
#define PN_LINK_INT_OFFSET                          (6)
#define PN_ISOM_INT_OFFSET                          (7)

#define PRU_RX_INT_NUM                               20
#define PRU_LINK_INT_NUM                             26
#define TIEIP_MDIO_CLKDIV                            79 //For 2.5MHz MDIO
clock: 200/(TIESC_MDIO_CLKDIV+1)
#define PRU_PN_PTCP_TASK_PRIO                        11
#define PRU_PN_PTCP_SYNC_MONITOR_TASK_PRIO           11
#define PRU_PN_LINK_TASK_PRIO                        12
#define PRU_PN_RX_TASK_PRIO                          12

#define PN_PTCP_TIMERID                              0 // TimerP_ANY
#define PN_PTCP_TIMER_FREQ                           25000000

void PN_socinitIRTHandle (PN_Handle appPnHandle)
{
    PRUICSS_HwAttrs const *pruicssHwAttrs =
PRUICSS_getAttrs(CONFIG_PRU_ICSS1);

    /* Interrupt configuration of PN PPM */
    appPnHandle->pnIntConfig.ppmIntConfig.pruIntNum = 21;
    appPnHandle->pnIntConfig.ppmIntConfig.socEvId = 21;
    appPnHandle->pnIntConfig.ppmIntConfig.intPrio = 11;

    /* Interrupt configuration of PN CPM */
    appPnHandle->pnIntConfig.cpmIntConfig.pruIntNum = 22;
    appPnHandle->pnIntConfig.cpmIntConfig.socEvId = 22;
    appPnHandle->pnIntConfig.cpmIntConfig.intPrio = 12;

    /* Interrupt configuration of PN DHT */
```

```
35      appPnHandle->pnIntConfig.dhtIntConfig.pruIntNum = 23;
36      appPnHandle->pnIntConfig.dhtIntConfig.socEvId = 23;
37      appPnHandle->pnIntConfig.dhtIntConfig.intPrio = 10;
38
39      /* Interrupt configuration of PN PTCP */
40  #ifdef PTCP_SUPPORT
41      appPnHandle->pnIntConfig.ptcpIntConfig.pruIntNum = 24;
42      appPnHandle->pnIntConfig.ptcpIntConfig.socEvId = 24;
43      appPnHandle->pnIntConfig.ptcpIntConfig.intPrio = 13;
44      appPnHandle->pnPtcpConfig.ptcpEnableSlowCompensation = 1;
45      appPnHandle->pnPtcpConfig.ptcpTimer.ptcpTimerID = PN_PTCP_TIMERID;
46  #endif
47
48      /* Interrupt configuration of PN ISOM */
49      appPnHandle->pnIsoMConfig.isoMIntConfig.pruIntNum = 27;
50      appPnHandle->pnIsoMConfig.isoMIntConfig.socEvId = 27;
51      appPnHandle->pnIsoMConfig.isoMIntConfig.intPrio = 14;
52
53      if(pruicssHwAttrs->instance == 0)
54      {
55          /* Interrupt configuration for ICSSG0 */
56          appPnHandle->pnIntConfig.ppmIntConfig.coreIntNum =
    CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
    PN_PPM_INT_OFFSET;
57          appPnHandle->pnIntConfig.cpmIntConfig.coreIntNum =
    CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
    PN_CPM_INT_OFFSET;
58          appPnHandle->pnIntConfig.dhtIntConfig.coreIntNum =
    CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
    PN_DHT_INT_OFFSET;
59  #ifdef PTCP_SUPPORT
60          appPnHandle->pnIntConfig.ptcpIntConfig.coreIntNum =
    CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
    PN_PTCP_INT_OFFSET;
61  #endif
62          appPnHandle->pnIsoMConfig.isoMIntConfig.coreIntNum =
    CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
    PN_ISOM_INT_OFFSET;
63      }
64      else
65      {
66          /* Interrupt configuration for ICSSG1 */
67          appPnHandle->pnIntConfig.ppmIntConfig.coreIntNum =
    CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
    PN_PPM_INT_OFFSET;
68          appPnHandle->pnIntConfig.cpmIntConfig.coreIntNum =
    CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
    PN_CPM_INT_OFFSET;
69          appPnHandle->pnIntConfig.dhtIntConfig.coreIntNum =
    CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
    PN_DHT_INT_OFFSET;
70  #ifdef PTCP_SUPPORT
```

```
71          appPnHandle->pnIntConfig.ptcpIntConfig.coreIntNum =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
PN_PTCP_INT_OFFSET;
72  #endif
73          appPnHandle->pnIsoMConfig.isoMIntConfig.coreIntNum =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
PN_ISOM_INT_OFFSET;
74      }
75  }
```

2.5 Invoke the PN_initDrv API

Once the Profinet Handle is initialized, the next step is to call the PN_initDrv API from the Profinet Driver. This will load the PRU firmware and do the required initialization.

```
1   int32_t  PN_drvStatus;
2
3   PN_drvStatus = PN_initDrv(appPnHandle);
4
5   if(PN_drvStatus == 0)
6   {
7       DebugP_log("\r\nProfinet Driver initialization successful!!\r\n");
8   }
9   else
10  {
11      DebugP_log("\r\nProfinet Driver initialization unsuccessful!!\r\n");
12  }
```

Once this is done, the Profinet firmware should be running on the PRU cores.

3 Ethernet/IP (EIP) Initialization Guide

3.1 Ethernet/IP Handle creation

The Ethernet/IP related include files have to be included in the file and global instances of the following handles should be created:

- EIP Handle
- DLR Handle
- TimeSync Handle

```
1  /* Ethernet/IP related includes */
2  #include <industrial_comms/ethernetip_adapter/icss_fwhal/
   icss_eip_driver.h>
3  #include <industrial_comms/ethernetip_adapter/icss_fwhal/firmware/
   icss_emac_mmap.h>
4  #include <networking/icss_emac/source/icss_emac_local.h>
5
6  EIP_Handle icssEipHandle;
7  EIP_DLRHandle dlrHandle;
8  TimeSync_ParamsHandle_t timeSyncHandle;
```

3.2 Memory allocation for EIP Handle

Make sure memory allocation is done for the EIP Handle in the application code.

```
1  icssEipHandle = (EIP_Handle)malloc(sizeof(struct eip_Config_s));
```

3.3 EIP Handle Initialization

The following code block can be taken as a reference to do required EIP Handle Initialization. This function can be invoked in the application and accepts the EIP Handle, ICSS EMAC Handle and PRUICSS Handle as required arguments.

```
1  int8_t EIP_initDefaultValues(EIP_Handle icssEipHandle,ICSS_EMAC_Handle
   emachandle,PRUICSS_Handle prusshandle)
2  {
3      int8_t status = SystemP_SUCCESS;
4      /* Initialize EIP Handle */
```

```
5       memset(icssEipHandle,0,sizeof(struct eip_Config_s));
6       /* Assign ICSS EMAC handle */
7       icssEipHandle->emacHandle = emachandle;
8       /* Assign PRU-ICSS handle */
9       icssEipHandle->pruicssHandle  = prusshandle;
10      /* Allocate Memory for DLR Handle*/
11      dlrHandle = (EIP_DLRHandle)malloc(sizeof(dlr_Config));
12      /* Allocate Memory for TimeSync Handle*/
13      timeSyncHandle = (TimeSync_ParamsHandle_t)malloc(sizeof(TimeSync_Param
    sHandle));
14      /* Initialize DLR Handle*/
15      EIP_initICSSDlrHandle(dlrHandle, emachandle);
16      /* Initialize TimeSync Handle*/
17      status = EIP_initICSSPtpHandle(timeSyncHandle, emachandle);
18      return status;
19  }
```

NOTE: In the above code block in lines 15 and 17, there are calls made to the EIP_initICSSDlrHandle and EIP_initICSSPtpHandle for DLR and TimeSync initialization respectively. More information about these functions are provided in the later sections.

3.4 DLR Handle Initialization

The following code block can be taken as a reference to initialize the DLR handle. The function EIP_initICSSDlrHandle takes in the DLR and ICSS EMAC handles as function parameters.

```
1   void EIP_initICSSDlrHandle(EIP_DLRHandle dlrHandle, ICSS_EMAC_Handle
    emachandle)
2   {
3       /* Assign ICSS EMAC handle */
4       dlrHandle->emacHandle = emachandle;
5       /* Perform memory allocation for the DLR parent structure */
6       dlrHandle->dlrObj = (dlrStruct *)malloc(sizeof(dlrStruct));
7       /* Perform memory allocation for the DLR exclusion list */
8       dlrHandle->exclusionList = (exceptionList *)malloc(sizeof(exceptionLis
    t));
9   }
```

3.5 TimeSync Handle Initialization

For TimeSync initialization, a lot of parameters are application specific, but the following function : EIP_initICSSPtpHandle can be referenced. It takes in TimeSync and ICSS EMAC handles as function parameters.

```
1   int8_t EIP_initICSSPtpHandle(TimeSync_ParamsHandle_t timeSyncHandle,
    ICSS_EMAC_Handle emachandle)
2   {
```

```
    int8_t returnVal = SystemP_FAILURE;
    timeSyncHandle->emacHandle = emachandle;

    /*Configure PTP. These variables must be configured before doing anything else*/
    timeSyncHandle->timeSyncConfig.config = BOTH;
    timeSyncHandle->timeSyncConfig.type = E2E;
    timeSyncHandle->timeSyncConfig.protocol = UDP_IPV4;
    timeSyncHandle->timeSyncConfig.tickPeriod = 500;
    timeSyncHandle->txprotocol = 0;

    timeSyncHandle->timeSyncConfig.delayReqSendTaskPriority = 10;
    timeSyncHandle->timeSyncConfig.txTsTaskPriority = 10;
    timeSyncHandle->timeSyncConfig.nrtTaskPriority = 8;
    timeSyncHandle->timeSyncConfig.backgroundTaskPriority = 7;

    /* Memory allocation for different sub-structures */
    timeSyncHandle->tsSyntInfo = (timeSync_SyntInfo_t *)malloc(sizeof(
                                    timeSync_SyntInfo_t));

    if(timeSyncHandle->tsSyntInfo == NULL)
    {
        DebugP_log("\n\rIssue in allocating memory for timeSyncHandle->tsSyntInfo");
    }

    timeSyncHandle->tsNrrInfo[0] = (timeSync_NrrInfo_t *)malloc(sizeof(
                                       timeSync_NrrInfo_t));

    if(timeSyncHandle->tsNrrInfo[0] == NULL)
    {
        DebugP_log("\n\rIssue in allocating memory for timeSyncHandle->tsNrrInfo[0]");
    }

    timeSyncHandle->tsNrrInfo[1] = (timeSync_NrrInfo_t *)malloc(sizeof(
                                       timeSync_NrrInfo_t));

    if(timeSyncHandle->tsNrrInfo[1] == NULL)
    {
        DebugP_log("\n\rIssue in allocating memory for timeSyncHandle->tsNrrInfo[1]");
    }

    timeSyncHandle->syncParam[0] = (syncParam_t *)malloc(sizeof(syncParam_t));

    if(timeSyncHandle->syncParam[0] == NULL)
    {
        DebugP_log("\n\rIssue in allocating memory for timeSyncHandle->syncParam[0]");
    }

```

```
50      timeSyncHandle->syncParam[1] = (syncParam_t *)malloc(sizeof(syncParam_
    t));
51
52      if(timeSyncHandle->syncParam[1] == NULL)
53      {
54          DebugP_log("\n\rIssue in allocating memory for timeSyncHandle-
    >syncParam[1]");
55      }
56
57      timeSyncHandle->tsRunTimeVar = (timeSync_RuntimeVar_t *)malloc(sizeof(
58                                          timeSync_RuntimeVar_t));
59
60      if(timeSyncHandle->tsRunTimeVar == NULL)
61      {
62          DebugP_log("\n\rIssue in allocating memory for timeSyncHandle-
    >tsRunTimeVar");
63      }
64
65      timeSyncHandle->delayParams = (delayReqRespParams_t *)malloc(sizeof(
66                                         delayReqRespParams_t));
67
68      if(timeSyncHandle->delayParams == NULL)
69      {
70          DebugP_log("\n\rIssue in allocating memory for timeSyncHandle-
    >delayParams");
71      }
72
73      timeSyncHandle->offsetAlgo  = (timeSync_Offset_Stable_Algo_t *)malloc(s
    izeof(
74                                         timeSync_Offset_Stable_Algo_t));
75
76      if(timeSyncHandle->offsetAlgo == NULL)
77      {
78          DebugP_log("\n\rIssue in allocating memory for timeSyncHandle-
    >offsetAlgo");
79      }
80
81      /*Allocate Rx and Tx packet buffers*/
82      returnVal = TimeSync_alloc_PktBuffer(timeSyncHandle);
83
84      /*Set only if a custom Tx LLD API is used*/
85      timeSyncHandle->timeSyncConfig.custom_tx_api = 0;
86      /*Set to 1 if Rx timestamps are coming from shared RAM*/
87      timeSyncHandle->timeSyncConfig.timestamp_from_shared_ram = 1;
88
89      /*If there is no forwarding between ports then set this*/
90      timeSyncHandle->timeSyncConfig.emac_mode = 0;
91      timeSyncHandle->timeSyncConfig.hsrEnabled = 0;
92
93      timeSyncHandle->rxTimestamp_gPTP = (timeStamp *)malloc(sizeof(timeStam
    p));
94
95      timeSyncHandle->timeSyncConfig.domainNumber[0] = 0;
```

```
96
97      timeSyncHandle->timeSyncConfig.logAnnounceRcptTimeoutInterval =
    DEFAULT_ANNOUNCE_TIMEOUT_LOG_INTERVAL;
98      timeSyncHandle->timeSyncConfig.logAnnounceSendInterval =
    DEFAULT_ANNOUNCE_SEND_LOG_INTERVAL;
99      timeSyncHandle->timeSyncConfig.logPDelReqPktInterval =
    DEFAULT_PDELAY_REQ_LOG_INTERVAL;
100     timeSyncHandle->timeSyncConfig.logSyncInterval =
    DEFAULT_SYNC_SEND_LOG_INTERVAL;
101
102     /*No asymmetry*/
103     timeSyncHandle->timeSyncConfig.asymmetryCorrection[0] = 0;
104     timeSyncHandle->timeSyncConfig.asymmetryCorrection[1] = 0;
105     timeSyncHandle->timeSyncConfig.pdelayBurstNumPkts = 3;      /*3 frames
    sent in a burst*/
106     timeSyncHandle->timeSyncConfig.pdelayBurstInterval = 200;   /*gap
    between each frame is 100ms*/
107     timeSyncHandle->timeSyncConfig.sync0_interval = 1000000;      /*1
    milisec value*/
108
109     /*Register callback*/
110     timeSyncHandle->timeSyncConfig.timeSyncSyncLossCallBackfn =
    (TimeSync_SyncLossCallBack_t)EIP_syncLossCallback;
111
112     timeSyncHandle->timeSyncConfig.masterParams.priority1 =
    TIMESYNC_DEFAULT_PRIO_1;
113     timeSyncHandle->timeSyncConfig.masterParams.priority2 =
    TIMESYNC_DEFAULT_PRIO_2;
114     timeSyncHandle->timeSyncConfig.masterParams.clockAccuracy =
    TIMESYNC_DEFAULT_CLOCK_ACCURACY; /*greater than 10s */
115     timeSyncHandle->timeSyncConfig.masterParams.clockClass =
    TIMESYNC_DEFAULT_CLOCK_CLASS;
116     timeSyncHandle->timeSyncConfig.masterParams.clockVariance =
    TIMESYNC_DEFAULT_CLOCK_VARIANCE;
117     timeSyncHandle->timeSyncConfig.masterParams.stepRemoved =
    TIMESYNC_DEFAULT_STEPS_REMOVED;
118     timeSyncHandle->timeSyncConfig.masterParams.UTCOffset =
    TIMESYNC_UTC_OFFSET;
119     timeSyncHandle->timeSyncConfig.masterParams.timeSource =
    TIMESYNC_DEFAULT_TIME_SOURCE; /*Internal oscillator*/
120
121     timeSyncHandle-
    >timeSyncConfig.masterParams.ptp_flags[TS_PTP_TIMESCALE_INDEX] = 1;
122     timeSyncHandle-
    >timeSyncConfig.masterParams.ptp_flags[TS_PTP_TWO_STEP_INDEX] = 1;
123
124     timeSyncHandle->timeSyncConfig.rxPhyLatency = 220;
125     timeSyncHandle->timeSyncConfig.txPhyLatency = 64;
126
127     returnVal = SystemP_SUCCESS;
128     return returnVal;
129 }
130
```

```
void EIP_syncLossCallback()
{
    /*This indicates a loss of time sync
      Call your function here which handles sync loss*/

    return;
}
```

3.6 MAC Address Configuration

The Interface MAC address programmed during the ICSS EMAC Initialization should also be copied to the DLR and TimeSync handles. The following code-block can be taken as a reference.

```
/* Copy the MAC ID passed to ICSS-EMAC during ICSS-EMAC initialization*/
memcpy(dlrHandle->macId, &(icssEmacParams.macId[0]), 6);
memcpy(timeSyncHandle->macId, &(icssEmacParams.macId[0]), 6);
```

3.7 Configure the Interrupts

The next step is to configure the DLR and TimeSync interrupts. For this, a helper function like EIP_configureInterrupts in the following code-block can be used. This function takes in the DLR and TimeSync handles as function parameters.

Please note that some of the interrupt configurations differ depending on ICSS instance being used by the application. This can be taken care of in the following code block.

```
/* Interrupt related defines EthernetIP */
#define EIP_PTP_INT_NUM                     (3)
#define EIP_DLR_P0_INT_OFFSET               (1)
#define EIP_DLR_P1_INT_OFFSET               (2)
#define EIP_BEACON_TIMEOUT_INT_OFFSET_P0    (4)
#define EIP_BEACON_TIMEOUT_INT_OFFSET_P1    (7)

/*Configure the DLR and PTP Interrupts */
void EIP_configureInterrupts(EIP_DLRHandle dlrHandle,
TimeSync_ParamsHandle_t timeSyncHandle)
{
    PRUICSS_HwAttrs const *pruicssHwAttrs =
PRUICSS_getAttrs(CONFIG_PRU_ICSS1);

    if(pruicssHwAttrs->instance == 0)
    {
        /*Configure Time Sync interrupts*/
        timeSyncHandle->timeSyncConfig.txIntNum =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 + EIP_PTP_INT_NUM;

        /*Configure DLR*/
```

```
        dlrHandle->dlrObj->port0IntNum =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
EIP_DLR_P0_INT_OFFSET;
        dlrHandle->dlrObj->port1IntNum =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
EIP_DLR_P1_INT_OFFSET;
        dlrHandle->dlrObj->beaconTimeoutIntNum_P0 =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
EIP_BEACON_TIMEOUT_INT_OFFSET_P0;
        dlrHandle->dlrObj->beaconTimeoutIntNum_P1 =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG0_PR1_HOST_INTR_PEND_0 +
EIP_BEACON_TIMEOUT_INT_OFFSET_P1;
    }
    else
    {
        /*Configure Time Sync interrupts*/
        timeSyncHandle->timeSyncConfig.txIntNum =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 + EIP_PTP_INT_NUM;

        /*Configure DLR*/
        dlrHandle->dlrObj->port0IntNum =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
EIP_DLR_P0_INT_OFFSET;
        dlrHandle->dlrObj->port1IntNum =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
EIP_DLR_P1_INT_OFFSET;
        dlrHandle->dlrObj->beaconTimeoutIntNum_P0 =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
EIP_BEACON_TIMEOUT_INT_OFFSET_P0;
        dlrHandle->dlrObj->beaconTimeoutIntNum_P1 =
CSLR_R5FSS0_CORE0_INTR_PRU_ICSSG1_PR1_HOST_INTR_PEND_0 +
EIP_BEACON_TIMEOUT_INT_OFFSET_P1;
    }
}
```

3.8 Perform the EIP Driver Init

After the interrupts are configured properly, the next step is to assign the DLR and TimeSync handles back to the EIP handle and invoke the EIP_drvInit API to load the EIP firmware, and call the DLR and TimeSync Driver init APIs. Finally, invoke the EIP_drvStart API to start DLR and enable PTP.

The following code block can be taken as a reference.

```

/* Assign the DLR handle */
icssEipHandle->dlrHandle = dlrHandle;
/* Assign the TimeSync handle */
icssEipHandle->timeSyncHandle = timeSyncHandle;

/* Init EIP Driver */
EIP_drvInit(icssEipHandle);
```

```
8   EIP_drvStart(icssEipHandle);
```

3.9 Register Application Specific callback APIs

The next step is to register different callback APIs related to the Ethernet/IP application.

3.9.1 Link Callbacks

The first step is to set up link callbacks. These are used for some special handling from DLR and PTP perspective. The ICSS EMAC Link callbacks can be assigned as the code block below.

```
1   /* Register Callback APIs for Link Break*/
2   ((((ICSS_EMAC_Object *)emachandle->object)->callBackObject).port0LinkCallBack).callBack = (ICSS_EMAC_CallBack)EIP_DLR_TIMESYNC_port0ProcessLinkBrk;
3   ((((ICSS_EMAC_Object *)emachandle->object)->callBackObject).port0LinkCallBack).userArg  = (void*)icssEipHandle;
4   ((((ICSS_EMAC_Object *)emachandle->object)->callBackObject).port1LinkCallBack).callBack = (ICSS_EMAC_CallBack)EIP_DLR_TIMESYNC_port1ProcessLinkBrk;
5   ((((ICSS_EMAC_Object *)emachandle->object)->callBackObject).port1LinkCallBack).userArg  = (void*)icssEipHandle;
```

The callback APIs EIP_DLR_TIMESYNC_port0ProcessLinkBrk and EIP_DLR_TIMESYNC_port1ProcessLinkBrk are defined as follows.

```
1   /*Registering Port 1 Link break callback */
2   void EIP_DLR_TIMESYNC_port0ProcessLinkBrk(void* icssEmacVoidPtr, uint8_t linkStatus, void *eipVoidPtr)
3   {
4       EIP_Handle eipHandle = (EIP_Handle)eipVoidPtr;
5       /* Process DLR Link Break */
6       EIP_DLR_port0ProcessLinkBrk(linkStatus, (void *)(eipHandle->dlrHandle));
7       /* Process TimeSync Link Break */
8       TimeSync_Port1linkResetCallBack(linkStatus, (void *)(eipHandle->timeSyncHandle));
9   }
10
11  /*Registering Port 2 Link break callback */
12  void EIP_DLR_TIMESYNC_port1ProcessLinkBrk(void* icssEmacVoidPtr, uint8_t linkStatus, void *eipVoidPtr)
13  {
14       EIP_Handle eipHandle = (EIP_Handle)eipVoidPtr;
15      /* Process DLR Link Break */
16      EIP_DLR_port1ProcessLinkBrk(linkStatus, (void *)(eipHandle->dlrHandle));
17      /* Process TimeSync Link Break */
```

```
    TimeSync_Port2linkResetCallBack(linkStatus, (void *)(eipHandle->timeSyncHandle));
}
```

3.9.2 Rx RT callback

The ICSS EMAC Rx RT callback can be set like the code-block below.

```
/* Register Rx RT callback API */
((((ICSS_EMAC_Object *)emachandle->object)->callBackObject).rxRTCallBack).callBack = (ICSS_EMAC_CallBack)EIP_RtRxCallback;
((((ICSS_EMAC_Object *)emachandle->object)->callBackObject).rxRTCallBack).userArg  = (void*)icssEipHandle;
```

A sample Rx RT callback API for EIP can be referenced below.

```
/* Global counter to counter Rx RT errors */
static uint32_t rx_rt_errors_s  = 0;
/**PTP MAC ID for comparison*/
uint8_t ptpMAC_EIP[6] = {0x1, 0x0, 0x5e, 0x0, 0x1, 0x81};
/**DLR MAC ID for comparison*/
uint8_t dlrMAC_EIP[6] = {0x1, 0x21, 0x6c, 0x0, 0x0, 0x2};

int8_t EIP_RtRxCallback (void *emacHandleVoidPtr, uint8_t portNumber, void * eipHandleVoidPtr)
{
    uint32_t packetLen;
    int32_t  port;
    int8_t   returnVal = SystemP_SUCCESS;
    int32_t  queue;
    int32_t  len;
    ICSS_EMAC_RxArgument rxArgs;

    EIP_Handle            eipHandle = (EIP_Handle)eipHandleVoidPtr;
    ICSS_EMAC_Handle      emacHandle = (ICSS_EMAC_Handle)emacHandleVoidPtr;
    ICSS_EMAC_PortParams  rxPort   = ((ICSS_EMAC_Object*)(emacHandle->object))->switchPort[0];
    uint32_t              rxErrors = rxPort.queue[0].qStat.errCount +
                                     rxPort.queue[1].qStat.errCount +
                                     rxPort.queue[2].qStat.errCount;
    uint8_t               *dstMacId = eipHandle->tempFrame;

    packetLen = ICSS_EMAC_rxPktInfo (emacHandle, &port, &queue);

    if (packetLen == SystemP_FAILURE)
    {
        returnVal = SystemP_FAILURE;
```

```
30          return returnVal;
31      }
32
33      if (rxErrors != rx_rt_errors_s)
34      {
35          rx_rt_errors_s = rxErrors;
36
37          DebugP_log("RX real-time queue errors: %d", rxErrors);
38      }
39
40      switch (queue)
41      {
42          case ICSS_EMAC_QUEUE1:
43          case ICSS_EMAC_QUEUE2:
44          case ICSS_EMAC_QUEUE3:
45          case ICSS_EMAC_QUEUE4:
46          {
47              rxArgs.icssEmacHandle = emacHandleVoidPtr;
48              rxArgs.destAddress    = (uint32_t)(icssEipHandle->tempFrame);
49              rxArgs.queueNumber    = queue;
50              rxArgs.more           = 0;
51              rxArgs.port           = 0;
52
53              len = ICSS_EMAC_rxPktGet (&rxArgs, NULL);
54
55              /*Compare Destination MAC ID and determine if this is a DLR
    packet*/
56              if((memcmp((void *)dstMacId, (void *)dlrMAC_EIP, 6U) == 0))
57              {
58                  EIP_DLR_processDLRFrame(eipHandle->dlrHandle, eipHandle-
    >tempFrame,
59                                          rxArgs.port - 1, len);
60              }
61              /*Compare Destination MAC ID and determine if this is a PTP
    packet*/
62              else if((memcmp((void *)dstMacId, (void *)ptpMAC_EIP, 6U) ==
    0) &&
63                      (rxArgs.port >= ICSS_EMAC_PORT_1) &&
64                      (rxArgs.port <= ICSS_EMAC_PORT_2))
65              {
66                  /*Link local field doesn't matter in case of EIP*/
67                  TimeSync_processPTPFrame(eipHandle->timeSyncHandle,
68                                          eipHandle->tempFrame, rxArgs.port,
    len, FALSE);
69              }
70
71          }   break;
72          default:
73          {
74              returnVal = SystemP_FAILURE;
75          }   break;
76      }
77
```

```
78      return returnVal;
79  }
```

3.10 IP Address Configuration

After the host IP address configured in the EEPROM is read, it should also be configured in the DLR and TimeSync handles. The following code block can be taken as a reference.

```
1  /* Global variable to store the IP address */
2  uint32_t ipAddress;
3  /* Offset where IP address is stored on flash */
4  #define SPI_EEPROM_DEVICEIP_OFFSET  0x8000
5
6  /* Assign the IP read from EEPROM */
7  EEPROM_read(gEepromHandle[CONFIG_EEPROM0], SPI_EEPROM_DEVICEIP_OFFSET,
   (uint8_t *)&ipAddress, 4);
8  /* Call the helper function to configure the assign IP address to the DLR
   and TimeSync handles */
9  EIP_DLR_TIMESYNC_assignIP();
```

The helper function EIP_DLR_TIMESYNC_assignIP is defined below.

```
1  void EIP_DLR_TIMESYNC_assignIP()
2  {
3      EIP_DLR_addModuleIPAddress(icssEipHandle->dlrHandle, ipAddress);
4      TimeSync_addIP(icssEipHandle->timeSyncHandle, ipAddress);
5  }
```

After following all these steps, your Ethernet/IP application should be up and running along with DLR and PTP.