AN14953

S32 PFE Technical Reference Manual

Rev. 1.0 — 26 February 2026 **Application note**

Document information

Information	Content
Keywords	PFE, Initialization
Abstract	This document focuses on the initialization of PFE hardware component to help customers build their own PFE driver.

1 Introduction

The S32G Packet Forwarding Engine (PFE) is a key networking component of the NXP S32G automotive network processor family. It is designed to offload the host processors by performing hardware-accelerated packet classification, routing, modification, and forwarding. Positioned between the Ethernet MAC controllers and the host CPU cluster, the PFE delivers deterministic throughput and significantly reduces software processing load across a wide range of networking and gateway applications.

This document provides an overview of the PFE hardware components, describes their initialization sequence, and guides customers in developing their own PFE driver in standalone mode. It also introduces the simplest multi-instance use case.

This document does not cover PFE fast-path mode configuration. To enable advanced PFE features, please refer to the *PFE_FCI_API Reference Manual* and the *PFE RTD example project*.

2 Hardware architecture

2.1 Overview

The PFE is a hardware-based accelerator for classification and forwarding the Ethernet traffic. It has five HIF(Host Interface) and three EMAC(Ethernet Media Access Controller) interfaces. The HIF are used to receive/send frames from/to the host cores and the EMAC interfaces are used to receive send frames from/to the Ethernet network.

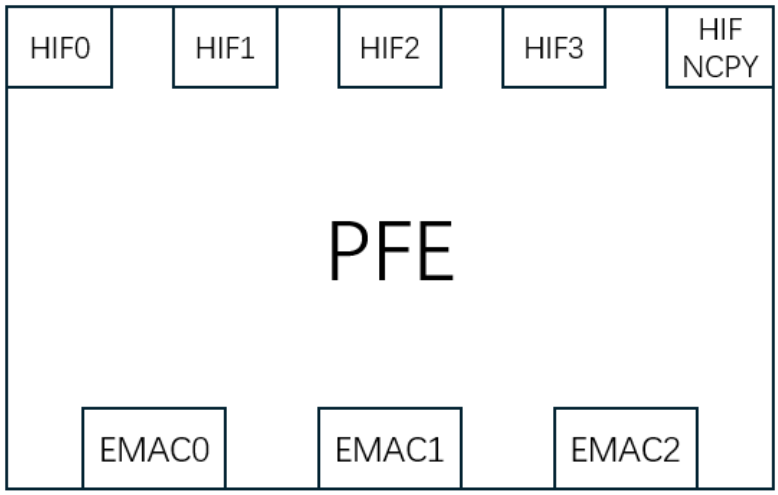

Figure 1. Interfaces

The Classifier (CLASS) module is the heart of the PFE. PFE firmware runs on processing engines in the classifier. Each packet reaching the PFE (ingress/egress) is processed by the PFE firmware that is responsible for packet classification, routing, modifications, and supporting functions. BMU (Buffer Management Unit) allocates the memory pool where Ethernet frames are stored while they are being processed by the PFE. TMU (Traffic Management Unit) is responsible for scheduling the packets out of physical interfaces.

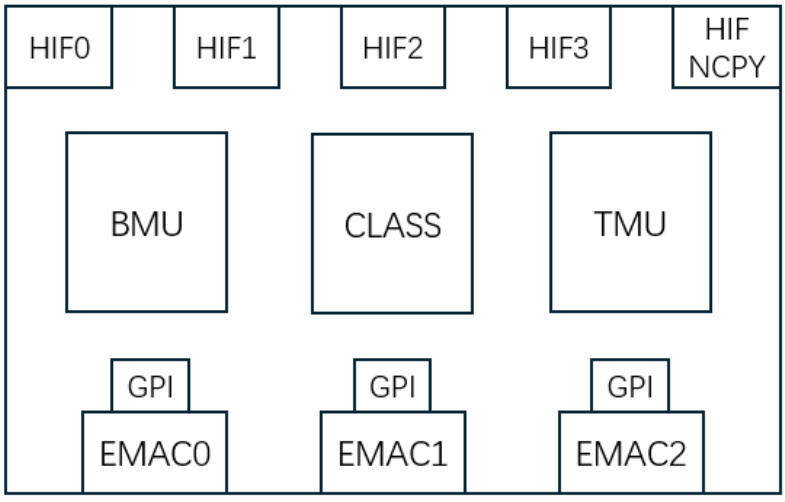

Figure 2. Hardware Architecture

2.2 EMAC

PFE includes 3 Ethernet MAC ports, and it is connected to GPI which has RX Engine that transfer packets to discrete buffers, and packet modifier that fetches packets and performs Layer-2 modifications as per the Classifier.

The EMAC supports following features:

- Compatible with IEEE Standard 802.3
- L2 checksum verification/generation
- Statistics counter registers for RMON/MIB
- RGMII, RMII, GMII, MII, TBI interfaces supported
- Support for 10/100/1000/2500Mbps SGMII
- MDIO interface for physical layer management
- Double VLAN support
- Support for 802.1Q VLAN tagging with recognition of incoming VLAN and priority tagged frames
- Support for jumbo frames
- MAC address filter
- IEEE1588 timestamping (One and two-step supported)

2.3 HIF

The HIF (Host Interface) is a component of the PFE dedicated to provide data interface between host CPUs and the PFE Classifier itself. It means that every time the PFE Classifier decides to propagate an Ethernet packet to host CPU the HIF is used to serve the packet data into host memory and notify the host that a new packet has been delivered. In the opposite direction, once host software needs to send packet to the PFE, it shall use the HIF to do so. The HIF is a data interface. It is capable of transferring packets from PFE HW domain into the host CPU domain by using its internal DMA controller. Once properly configured, the PFE can put (and get) packets to (from) host memory even if it is not the platform memory region explicitly dedicated for the PFE. PCIe memory space is a valid option for the HIF and packet data can be written to (read from) a PCI location. The HIF supports 4 independent channels which can be used by any host CPU with access to HIF register space. This multi-channel approach can be used to implement applications sharing connectivity provided by PFE without need of concurrent accesses to a single HIF instance, or just to increase the HW-SW throughput by implementation of a load-balancing scheme within a multi-core environment. Note that maximum packet size supported by HIF is 2KB and minimum size is 32B.

For more details about HIF, see the *HIF Programmer Guide*.

2.4 BMU

The Buffer Manager Unit (BMU) module allocates and de-allocates buffers for the packets. Its structure is very generic and amenable to any of the operating systems. The peripheral receives and transmits blocks interact at hardware level to allocate and free the memory. The buffer level indication is given to the Classifier for pause frame generation.

The initialization tasks in the software on the host processor configure the buffers with memory range to be used and present them to the BMU. The BMU then manages these buffers as a pool for buffers requested and freed from hardware modules. The access mechanism is a simple register . It read/write register to allocate/free buffers respectively.

There are 2 BMU blocks, BMU1 and BMU2. The BMU1 is used by PFE HW to allocate buffers for the first 64bytes of the frames, as well as parsed header data and pointers. The BMU2 is used by PFE HW to allocate the remaining part of the frames as well as meta-data used by GPI and HIF.

Multicast Handling:

The Buffer Manager block supports multicast of packets. The Buffer Manager block counts the number of usages and frees the buffer only after the packet is transmitted on all the multicast ports.

The reference count is registered by the Classifier code (or the Host software) to indicate the number of instances a packet will be transferred. Subsequently when the packet is transmitted by the ports, the buffers are freed (like the unicast packets – peripherals need not be aware that they are MCAST packets) the count is decremented. The number of multicast flows supported is fixed ahead at RTL compile time as it determines the internal memory size required to store the counts.

Overall, the BMU maintains two sets of buffers for internal memory and external memory. The packet sizes are completely programmable and can be chained for efficient usage. The chaining support for buffers allows support for jumbo frames very easily.

2.5 CLASS

The Classifier (CLASS) module is the heart of the architecture. It is a multi-processor module with hardware assists for packet processing functions of classification and Layer-3 modifications like NAT. The Classifier contains a queue block, a processing block (8 processing engines) and a re-order block. The hardware enables locked accesses to contentious resources like packet, counters etc. fetches the headers from memory and moves the headers in the data memory of the processors. The hardware resolves out of order execution on the way out and interfaces directly to the TMU and host processor.

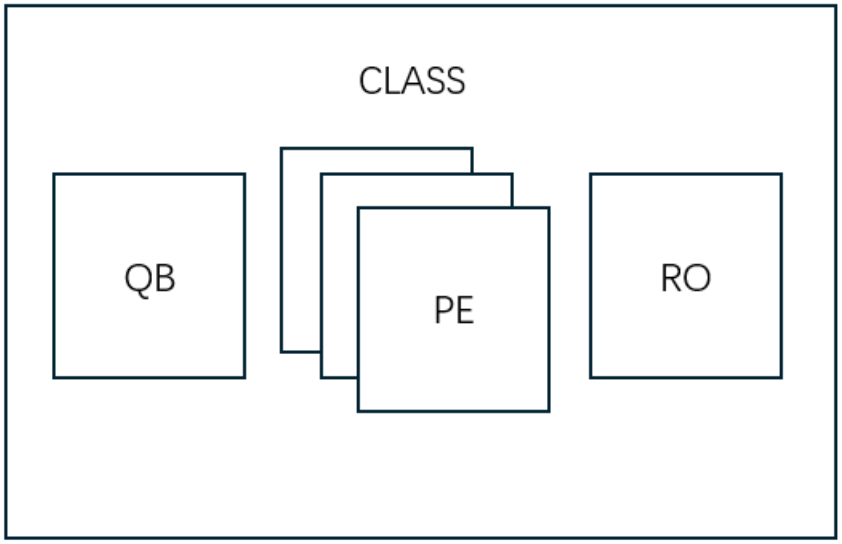

Figure 3. CLASS

The Queuing block takes packets from BMU and schedules the packets to the processing blocks. It moves packet headers from Internal/External memory to the data memory of the Processing Engines (PE). It tags the packets with a sequence number which is used for re-ordering later. This block helps in determining the queuing in the ingress ports, which helps in initiating Pause frames. The Queuing block and DMEM of PE share a ping-pong buffer with status register shared between the two blocks. The Queuing block reads the status indication and fills the DM buffer. The Queuing block has flexibility to assign packets from a particular PHY to (pre-configured) particular PE.

The Processing Engine is a RISC like Micro-Controller with its own data and program memory. The PE has a simple 5 stage pipeline with generic memories (IMEM and DMEM). PE processes the packets for classification, modification and pass them to RO block. Class PE also generates queue number for TMU block based on the TOS (DSCP) filed of IP header.

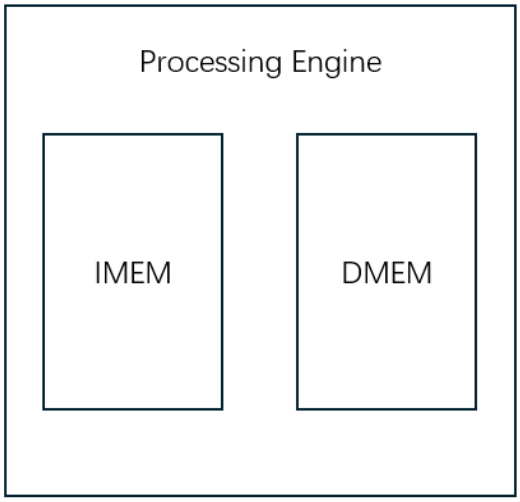

Figure 4. PE memory

The Classifier has multiple processing units that process the packets. Depending on the packet processing, packets can potentially come out of order from the Classifier block. So, the Re-Order block brings the packets back in the order they were received into the Classifier block. The QB block timestamps the packet entering into the Processor Cluster and the RO block at the output, reorders the packets. The Re-order block takes modified header and the Packet Action to perform from the PE and orders them as per the sequence number in the packet. (The max-out of order is only the maximum number of PE's). It transfers the packet headers from DMEM to the allocated RO buffers. It reads pkt header from RO buffer and writes into the location in DDR corresponding to Packet buffer. It triggers TMU module with the Packet pointer.

2.6 TMU

The Traffic Manager Unit (TMU) module implements the QoS (Quality-of-Service) functionality in the packet processing chain. The packets are fed (pushed) to the TMU module from the Classifier after the packet is classified and modified. Classifier determines the queue mapping for the packet. The TMU block works closely with the peripheral blocks with dedicated hardware signals and schedules the packets out. Hardware pulls the packets from the QoS. The hardware based QoS ensures that the packets are scheduled without any latency and the scheduling loop executes every few microseconds. The TMU has programmable queuing discipline (WRR, WFQ or PQ) for implementing QoS. The number of instances of the QoS (within TMU) blocks is parameterizable to suit the application (usually one per PHY). TMU provides 8 queues, 4 shapers and 2 schedulers for each EMAC and provides 2 queues for each HIF channel and all HIFs share one scheduler.

There are 6 PHY in PFE, including 3 EMACs, HIF block, UTIL and HIF No Copy.

TMU provides:

1. 8 queues to each PHY. 8 queues for each EMAC, Util and HIF No Copy, 2 queues for each HIF.
2. Queue depth per PHY per Queue is configurable.
3. Each queue Implements Tail Drop by default.
4. Implements two level schedulers with shapers at each level.
   - Implements PQ algorithm (WRR, WFQ)
   - Token Bucket shapers (Per Port shaper as well)
   - Flexible configuration of Shapers and Schedulers
5. Direct Interface to Egress PHY ports.
6. Support for Host Port.
7. Supports multi-cast packets.

A queueing algorithm sorts the egressing traffic into particular Egress QoS queues in TMU.

Pseudocode which represents the default traffic queueing algorithm:

```
get_queue_for_packet(pkt)
{
    queue = 0;
    if (pkt.isPTP)
    {
        queue = ptp_queue_tbl[ingress_interface];
    }
    else
    {
        if (pkt.hasVlanTag)
        {
            queue = pkt.VlanHdr.PCP;
        }
        else
        {
            if (pkt.isIPv4)
            {
                queue = (pkt.IPv4Hdr.DSCP) / 8;
            }
            if (pkt.isIPv6)
            {
                queue = (pkt.IPv6Hdr.TrafficClass.DS) / 8;
            }
        }
    }

    if (is_hif)
    {
        queue = (queue < 4) ? 0 : 1;
    }

    return queue;
}
```

Figure 5. Queueing algorithm

The purpose of shaper is to space out the frames as much as possible, reduce bursting in case of data peaks, and control maximum egress data rate (in data rate or packet rate units). In general, the shaper operates with configurable clock ticks. With every tick the internal credit value is updated according to the configured parameters. When the input queue has data and the credit value is positive, the queue is serviced, and the credit value is decreased. When credit is negative, the input queue will not be serviced. The sharpers can be positioned on the output of any queue or scheduler.

Scheduler is a component of the Egress QoS which is responsible for traffic aggregation from multiple queues. Its main task is to select the packet from a queue which will be transmitted next. Scheduler 1 is connected to the EMAC port. Each scheduler has up to 8 inputs. Scheduler 1's input can be from any of the 8 queues and the output of scheduler 0. Input 7 has the highest priority by default. The schedulers implement the following scheduling algorithms:

- Simple Round Robin (RR)
- Weighted Round Robin (WRR)
- Deficit Weighted Round Robin (DWRR)

Every queue is configured with a specific percentage of link speed as the weight. Quota and deficit parameters are used to guarantee committed bandwidth and fair chance for each queue.

- Priority Queuing (PQ)

The TMU CONTEXT memory stores queue-related statistics and information relevant to certain scheduler algorithms. Access to these statistics is indirect, requiring a specific sequence of register operations.

To read TMU counters:

1. Initiate access by writing the appropriate context value to the TMU_CNTX_ACCESS_CTRL register.
2. Issue the read command by writing to the TMU_CNTX_ADDR and TMU_CNTX_CMD registers.

3. Retrieve the result from the TMU_CNTX_DATA register after the command execution.

This mechanism allows controlled access to internal TMU statistics for diagnostic or monitoring purposes.

2.7 GPI

The GPI (Generic Packet Interface) block interfaces to blocks like Ethernet, GPON, Host Interface and converts the data into PFE internal structures (and subsequently passed to the Classifier). This block is reused extensively all over the EPP design to transform customer specific peripheral to send data to the Classifier. It has independent transmit and receive channels. The data path (including memory ports) of the GPI is 64 bits wide to suite very high-speed peripherals.

GPI-RX performs following functions:

- Buffer fetches
- Data Transfer
- Chaining of Buffers
- Programmable Offsets
- Status Transfer
- AXI interface

GPI stores the packet data into an internal FIFO realized with a single port arbitrated SRAM. The memory size is configurable at compile time to suit the overall chip architecture. In case of heavy traffic conditions, over-runs occur in this FIFO. The design has configuration registers which determine when to re-start storing data after an over-run condition has occurred. Packets that get into the GPI's FIFO that undergo over-run condition are also sent to the Classifier as the GPI block is a cut through design (data is transferred as it comes in – does not wait for the complete packet to be stored to support Jumbo packets efficiently).

After a programmable threshold bytes are filled in the GPI internal FIFO, The GPI block fetches a buffer to store the data. The initial part of frame is stored in local memory managed by BMU1, and the remaining part is stored in external memory managed by BMU2.

The pre-header sizes within the buffers are configurable. For example, in the most common case of 1 LMEM + N * DMEM buffers, the DMEM data is written after allowing space for the modified LMEM data. For simplicity of the RTL design, all the offsets and buffer start addresses are 64 bits aligned. The design optimizes writes of non-aligned packet sizes taking advantage of the aligned buffer sizes. GPI block chains the buffers and fills in fields according to the pre-defined structures of the Classifiers into a linked list. This feature enables to implement jumbo frames very easily.

The status of the packet is written into the buffer. The status provides the following information:

- Packet Length
- Error Conditions within GPI (over-run)
- Status from Peripheral (ARC Hit, MCAST, BCAST etc.)
- Next Pointer
- PHY Number

The packet is passed to the Classifier after it is completely received. The Classifier's input queue status is available to all the peripherals. Only when the input queue has space, the packet is written to the Classifier. The design ensures that the GPI receiver functions will not block the GPI transmitter.

GPI-TX performs following functions:

- Handshake with TMU
- Layer 2 Action Sequence Fetch
- Layer 2 Actions
- Data Fetch, Programmable Offsets and Chaining

- Buffer Free

The GPI TX block handshakes with Traffic Manager (TMU) and at the end of the sequence, the TMU sends a pointer to the packet to be sent out. The handshake takes into account the current FIFO occupancy status (and compares with a threshold register) to ensure that the decision to transmit by the TMU will be as late as possible ensuring that the full line rate is achievable.

The Layer 2 action sequence block fetches the action header and performs the following tasks as controlled by the action sequence header:

- MAC Source address (SA) Replacement
- MAC Destination Address (DA) Replacement
- VLAN addition (Supports Q in Q scenarios also)
- VLAN deletion
- Priority Field

Action Sequence is 8 bits. Action sequence is concatenation of GPI transmit first read data bits [23:20] and [31:28] bits from the external memory. These bits are updated by Classifier block. Based on the action provided by classifier, corresponding data is available in the transmit structure given below (reserved fields are used – from 0x18).

Table 1. Action sequence bits

Action Sequence Bits	Action
0	SRC MAC Replace
1	VLAN addition
2	TCP and UDP Checksum replacement
3	VLAN Replacement
4	Reserved
5	Reserved
6	IP checksum
7	Reserved

GPI-TX fetches data from the external memory. Data offset needs to be 4 bytes aligned. Buffer offset has to be 8 bytes aligned. The GPI-TX block traverses through the linked list of buffers and transfers the data enabling transfer of Jumbo frames without any additional overheads. The action sequence header and the packet structure are such that only one data fetch is required for sequence header even though the data is accessed by multiple peripherals.

Action sequence is fetched first by GPI. Number of first bytes to be fetched is configurable register GPI_DTX_ASEQ (csr_gpi_dtx_aseq_len). The default value is 0x10.

The offsets to data and subsequent buffers are either programmable or determined as per the action sequence header. Data offset needs to be 4 bytes aligned. Buffer offset has to be 8byte aligned. The data offset for the subsequent buffers is based on CSRs. The GPI-TX block traverses through the linked list of buffers and transfers the data enabling transfer of Jumbo frames without any additional overheads.

The data transfer block of GPI-TX performs high performance DDR, Local Memory and Control Register accesses. The DDR accesses are coalesced and are optimized with safe greedy accesses for reads and writes.

After the packet is filled into the internal buffer, the packet is freed with the BMU. GPI-TX computes the packet start address location and gives it to the BMU for proper de-allocation of the buffers.

2.8 Data Coherence

For Cortex-A53 core specific applications, configure coherency for PFE (see the PFE_COH_EN register's details in the Main General Purpose Registers (Main GPRs) chapter) and platform (see the Ncore chapter in *S32G Reference Manual*). Currently, only supported configurations are (Write 0x1E):

- DDR - Disabled
- HIF0 - HIF4 - Enabled
- Util - Disabled

Table 2. PFE Port Coherency Enable

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
DDR	0	R/W	PFE DDR Coherency Enable 0b - PFE DDR interface not coherent 1b - PFE DDR interface coherent to A53
HIF0	1	R/W	PFE_HIF BD Update Coherency Enable You must set this field to 1 for coherency over HIF channels. Setting this field to 1 changes the awdomain /ardomain of the master to shareable and you must configure Ncore for this case. 0b - PFE HIF0 interface not coherent. 1b - PFE HIF0 interface coherent to A53
HIF1	2	R/W	PFE_HIF Data Write Coherency Enable You must set this field to 1 for coherency over HIF channels. Setting this field to 1 changes the awdomain /ardomain of the master to shareable and you must configure Ncore for this case. 0b - PFE_HIF BD update interface not coherent. 1b - PFE_HIF BD fetch interface coherent to A53
HIF2	3	R/W	PFE_HIF Data Write Coherency Enable. You must set this field to 1 for coherency over HIF channels. Setting this field to 1 changes the awdomain / ardomain of the master to shareable and you must configure Ncore for this case. 0b - PFE_HIF data write interface not coherent 1b - PFE_HIF data write interface coherent to A53
HIF3	4	R/W	PFE_HIF Data Read Coherency Enable You must set this field to 1 for coherency over HIF channels. Setting this field to 1 changes the awdomain /ardomain of the master to shareable and you must configure Ncore for this case. 0b - PFE_HIF data read interface not coherent 1b - PFE_HIF data read interface coherent to A53

Table 2. PFE Port Coherency Enable*...continued*

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
UTIL	5	R/W	PFE UTIL Coherency Enable 0b - PFE UTIL interface not coherent 1b - PFE UTIL interface coherent to A53
Reserved	31:6	R/W	Reserved

2.9 Clock Requirement

Proper clock configuration is essential for the reliable operation of the PFE. This section details the clock sources, frequency requirements, and dependencies for each submodule within the PFE architecture. Ensuring correct clock setup is a prerequisite for successful hardware initialization and driver operation.

1. To enable PFE, PFE_PE_CLK is required first. PFE_PE comes from ACCEL_PLL and enable ACCELL_PLL for PFE_PE_CLK and PFE_SYS_CLK. The only supported frequency for PFE _PE_CLK is 600 MHz and PFE_SYS_CLK is 300 MHz.
2. Configure the MAC interfaces clocking as described in sections in *S32G Refernce Manual*, "PFE_MAC_0 clocking overview", "PFE_MAC_1 clocking overview", and "PFE_MAC_2 clocking overview". For SGMII use case, the SerDes initialization must be done. You can trigger SerDes initialization and PFE initialization independently. However, SerDes initialization completion is a prerequisite for completing PFE initialization (MAC clock configurations). For SerDes initialization and mode specific configuration, see the chapter SerDes Subsystem and the steps described in the *SerDes Subsystem Reference Manual.*
3. For PTP use-cases, enable GMAC_TS_CLK. See the *S32G2 Data Sheet* for the clock range. The recommended frequency is 200 MHz. XBAR_CLK also needs to be set to 400MHz due to hardware constraints.

3 Internal data flow architecture

3.1 EMAC to HIF

1. Packet is received on the GMII/MII port of Ethernet.
2. Ethernet-GPI block of Ethernet Peripheral fetches a LMEM buffer from BMU1.
3. The initial part of the frame is stored in BMU1.
4. Ethernet-GPI block fetches buffer from BMU2.
5. The subsequent part of the packet is stored in the BMU2.
   a. Ethernet-GPI leaves enough space for the LMEM header to be written later.
6. The packet status is filled in the initial 48 bytes of the LMEM buffer.
7. The packet pointer is written to the Classifier.
8. Classifier QB reads the header from BMU1 and writes into the PE of Classifier that is free.
9. Classifier releases the LMEM buffer in BMU1.
10. Classifier Processes the packet. It performs:
    a. Classification
    b. Packet Modifications
    c. Performance Counter Updates
11. Classifier RO block writes the packet to DDR memory (the second buffer of the packet).
12. Classifier passes the packet pointer, packet length, Policy Queue and destination port information to TMU.
13. TMU queues the packet into appropriate bucket.
14. TMU schedules the packet according to the QoS rules.
15. TMU handshakes with the peripheral for optimal time to pick up the next scheduled packet.
16. Peripheral (Ethernet) fetches the data from the packet buffer.
17. HIF-GPI block performs Layer-2 actions on the packet.
18. HIF-GPI block stores the packet data into its internal FIFO.
19. HIF-GPI frees the buffer in BMU2.
20. HIF block sends the packet out.

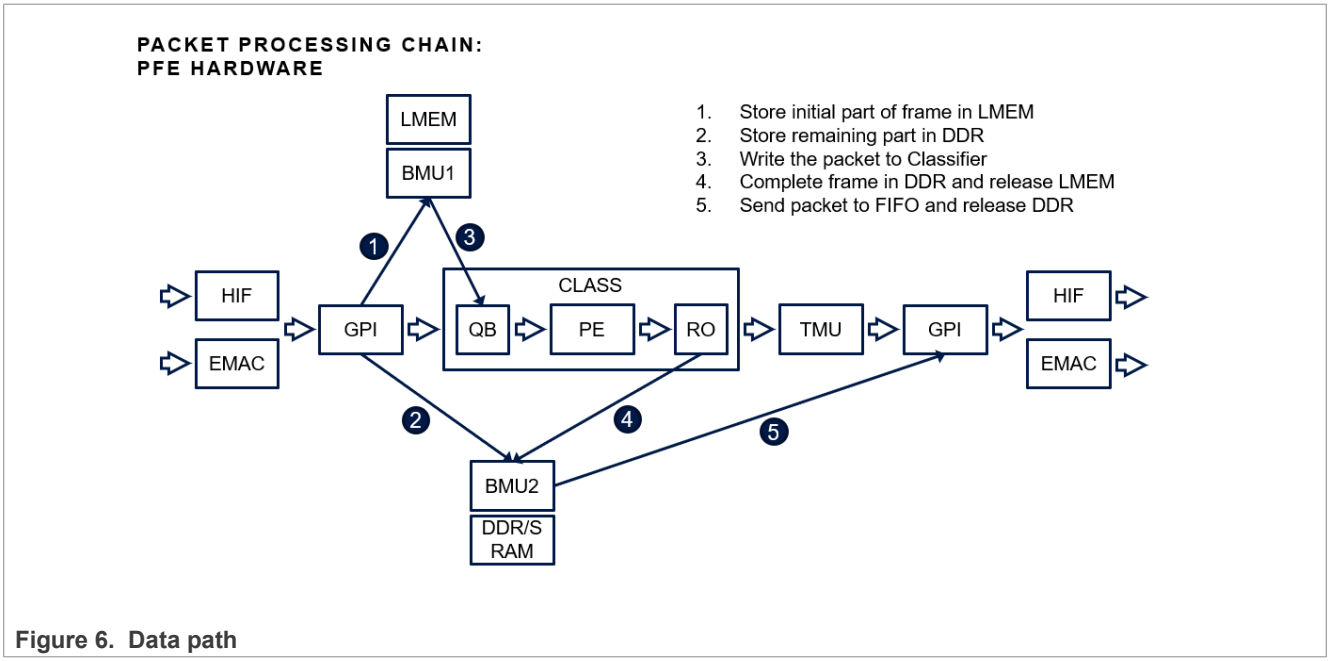

Figure 6. Data path

3.2 HIF to EMAC

1. Packet is received by the HIF block.
2. HIF-GPI block of HIF Peripheral fetches a LMEM buffer from BMU1.
3. The initial part of the frame is stored in LMEM.
4. HIF-GPI block fetches buffer from BMU2.
5. The subsequent part of the packet is stored in the BMU2.
   a. HIF-GPI leaves enough space for the LMEM header to be written later.
6. The packet status is filled in the initial 48 bytes of the LMEM buffer.
7. The packet pointer is written to the Classifier.
8. Classifier QB reads the header from BMU1 and writes into the PE of Classifier that is free.
9. Classifier releases the LMEM buffer in BMU1.
10. Classifier Processes the packet. It performs:
    a. Classification
    b. Packet Modifications
    c. Performance Counter Updates
11. Classifier RO block writes the packet to DDR memory (the second buffer of the packet).
12. Classifier passes the packet pointer, packet length, Policy Queue and destination port information to TMU.
13. TMU queues the packet into appropriate bucket.
14. TMU schedules the packet according to the QoS rules.
15. TMU handshakes with the peripheral for optimal time to pick up the next scheduled packet.
16. Peripheral (Ethernet) fetches the data from the packet buffer.
17. Ethernet-GPI block performs Layer-2 actions on the packet.
18. Ethernet-GPI block stores the packet data into its internal FIFO.
19. Ethernet-GPI frees the buffer in BMU2.
20. Ethernet MAC block sends the packet out.

3.3 Egress Timestamp to HIF

For every PTP frame, the EMAC transmit, it generates an egress timestamp report of 20bytes.

(Prepend Header is 16bytes and 4bytes packet data). Report consists of EMAC timestamp value, receive port number and reference number information, provided by the EMAC. Specific PHY number will be given to this type of packets (ETGPI_PHYNO).

Synchronization of timestamp value, port number and sequence number are done in the egress timestamp report generation module, it works with the GPI to write the generated report into the LMEM buffers. Once the report is written into the LMEM, it triggers the Class HW to transmit the report. Class HW on identifying it to be an egress timestamp report, does no table search and the packet is transmitted as such, without any modifications.

In case, if egress timestamp report generation module FIFO becomes full, it back pressures the EMAC in sending further requests to generate timestamp reports, in this way drops are avoided.

4 Firmware

4.1 Access CLASS PE memory

PE memory includes IMEM and DMEM. The size of IMEM per CLASS PE is 32KB and the size of DMEM per CLASS PE is 16KB. IMEM base address as defined by firmware is 0x9FC00000U and DMEM base address as defined by firmware is 0x20000000. Normally, Accessing CLASS PE memory is not needed except loading the PFE firmware into CLASS and configuring for the VLAN-bridge or Router.

Note: *The address of IMEM and DMEM is not mapped into the bus. It is the address inside the PE. It could not be accessed directly.*

Accessing PE memory has following options to be specified:

1. IMEM or DMEM
2. The address of IMEM or DMEM
3. Write or Read
4. Processors Engine id
5. If write operation is needed, the bytes need to be specified.

Here are two examples of reading and writing the PE memory:

1. Read 4 bytes at the DMEM address 0x20000AD8 of PE [0]:
   a. Calculate a Value:
   b. Writing the Value into address 0x46090100
   c. Read the 4 bytes data at address 0x46090108.

   0x46090100 is the address of the register CLASS_MEM_ACCESS_ADDR and below is the description of the register:
   Bit 31 indicates read operation or write operation. 0 means Internal Memory Read and 1 means Internal memory Write. To access DMEM, bit 18 and bit 17 should be 0x10. To access IMEM, bit 18 and bit 17 should be 0x01. Bit 20-23 indicates PE id. Bit 24-27 is valid only in write operation.
   0x46090108 is the address of the register CLASS_MEM_ACCESS_RDATA (0x46090108). The read data is placed in this register.
2. Write 4 bytes at the DMEM address 0x20000AD8 of PE [0]:
   a. Calculate a Value:
      Value = (0x20000AD8 & 0xfffff) | (1 << 31) | (1 << 17) | (0 << 20) | (0xf << 24)
   b. Write the 4 bytes data at address 0x46090104.
   c. Writing the Value into address 0x46090100.

   In write operation, bit 24 -27 represent the bytes of writing. 1 byte corresponds to 0x1, and 2 bytes corresponds to 0x3, and 3 bytes corresponds to 0x7, and 4 bytes corresponds to 0xf.
   0x46090104 is CLASS_MEM_ACCESS_WDATA (0x46090104). The write data is placed in this register.
   Read operation need to write the CLASS_MEM_ACCESS_ADDR first and read the register CLASS_MEM_ACCESS_RDATA. On the contrary, write operation will write the data into CLASS_MEM_ACCESS_WDATA first, and then write the CLASS_MEM_ACCESS_ADDR.

4.2 Loading and Running FW

The PFE Firmware is a software component running within the PFE. Each packet reaching the PFE (ingress/egress) is processed by the PFE Firmware that is responsible for packet classification, routing, modifications, and supporting functions. The PFE Firmware is under control of NXP and can be modified/extended to target new use cases and add specific features related to Ethernet traffic processing. The PFE Firmware must be loaded into the PFE after each PFE reset. Loading the PFE Firmware is a task of the PFE Driver of the specific operating system transparent to the user. Some PFE Drivers may require storing the PFE Firmware binary into

a predefined location in the file system. All cores of the PFE must be loaded with the appropriate PFE Firmware binaries.

To load the PFE firmware into the PFE, the firmware must first be placed at a predefined address in SRAM. Since the firmware is provided in ELF format, the ELF file must be parsed to extract key components such as the ELF header, program header table, and section header table. After parsing, the instruction memory (IMEM) and data memory (DMEM) of all 8 Processing Elements (PEs) are initialized by clearing them to zero. Before loading the PFE firmware, copy some key sections from ELF file into SRAM, such as pfe_pe_mmap section, message section and feature section. The ELF file contains 17 sections, each of which is processed during the firmware loading procedure.

For each section:

1. Calculate its address in SRAM using the predefined base address plus the sh_offset from the section header.
2. If the section has flags SHF_WRITE, SHF_ALLOC, or SHF_EXECINSTR, skip it.
3. Otherwise, retrieve the section's virtual address and size from the section header.
4. Translate the virtual address by identifying the corresponding program header segment. Once found, compute the offset between the segment's physical and virtual addresses. Apply this offset to the section's virtual address to determine its load address.
5. Based on the section type and load address:
   - If the section type is SHT_PROGBITS (value 1), load it into IMEM or DMEM according to the address for all 8 PEs.
   - If the section type is SHT_NOBITS (value 8) and the load address maps to DMEM, initialize the corresponding memory region to zero for all 8 PEs.

For implementation details, refer to the functions pfe_pe_upload_sections in the pfe_pe.c file within the PFE_MCAL driver.

To execute the PFE firmware, the CLASS module must be enabled by writing the value 0x1 to the register CLASS_TX_CTRL (0x46090004). This action activates all Processing Elements (PEs).

Once the enable bit is set, the driver must poll PE's status to confirm successful initialization. This involves reading the PE state register at address 0x20000E44 in DMEM and verifying that PE is ready for a new frame arrival.

For implementation details, refer to the functions pfe_class_enable and pfe_pe_get_fw_state in the pfe_class.c file within the PFE_MCAL driver.

4.3 Data flow implemented by FW

1. Classifier QB reads the header from LMEM and writes into the PE of Classifier that is free.
2. Classifier releases the LMEM buffer.
3. Classifier Processes the packet. It performs:
   a. Classification
   b. Packet Modifications
   c. Performance Counter Updates
4. Classifier RO block writes the packet to DDR or SRAM memory (the second buffer of the packet).
5. Classifier passes the packet pointer, packet length, Policy Queue and destination port information to TMU.

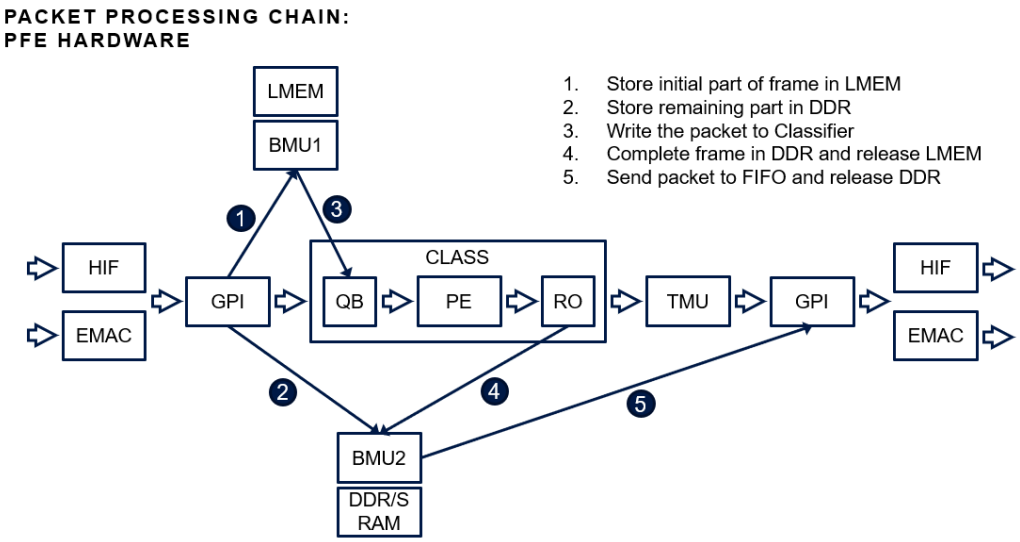

Figure 7. CLASS data path

4.4 Get Class PE Status

A Processing Engine (PE) is considered active when it is executing firmware code and has not been gracefully stopped.

To monitor the firmware state, a state monitor is implemented in the CLASS DMEM. The firmware periodically updates a status variable, which the driver can read to determine the current state.

To retrieve the PE or firmware status:

1. Copy the status variable from CLASS DMEM at address 0x20000E44 (as specified in the PFE memory map section) to SRAM.
2. Parse the copied value to interpret the firmware state.

For implementation reference, see the function pfe_pe_get_state_monitor_nolock in the pfe_pe.c file.

4.5 FW feature management

The PFE Firmware provides the possibility to enable or disable some of the features via a dedicated API. For each configurable feature, it is possible to enable/disable of some features.

There are 17 features in PFE firmware and see the PFE_Firmware_S32G_UserManual for the detailed description of the features.

After loading firmware into PFE CLASS, create and initialize the firmware feature management interface from the feature section copied from ELF file. There are 17 features in PFE firmware, and sufficient storage should be prepared for them. See the pfe_fw_feature.h for the feature structure and refer to the function pfe_class_load_fw_process in pfe_class.c.

5 Initialization

5.1 Overview

This chapter outlines the initialization process of the PFE driver at the hardware level. The sequence begins with configuring the essential hardware blocks, including the BMU, CLASS, TMU, EMAC and GPI modules. Once these components are initialized, the HIF is set up. Following this, the system proceeds to create both physical and logical interfaces and update the status into PFE. The final step involves enabling the HIF and EMAC, completing the hardware initialization phase.

5.2 Prerequisites

Before initializing the PFE clock, check if the PFE partition (partition 2) is enabled. If it is enabled, execute the PFE Software Reset Partition (partition 2) turn-off sequence (see the *S32G Reference Manual*). If it is not enabled (reset status), configure the PHY interface modes (MII modes) . This is done by writing the desired EMAC mode selection to the PFE_EMACX_INTF_SEL register at address 0x4007CA04. Refer to Section 19.2 of the *S32G Reference Manual* for detailed descriptions of this register and the function Eth_43_PFE_PreInit in the Eth_43_PFE.c file.

Once the EMAC interfaces are pre-initialized, proceed to initialize the pins used by the EMACs.

Afterward, initialize the PFE partition (partition 2) and the clocks related to PFE. If the SERDES module is required, initialize it and reconfigure the clock multiplexers to enable SERDES clocks for the EMACs. Refer to the PFE example project and the SERDES RTD example project for guidance on SERDES initialization.

If interrupt mode is used for processing received packets, ensure that the corresponding HIF interrupt handlers are registered and that interrupts are enabled before initializing the PFE driver.

Before initializing any hardware blocks, the PFE error report module should be configured first. Follow the steps below:

1. Disable Fail-Stop Mode (S32G3 Platform Only):
   If running on the S32G3 platform, disable the Fail-Stop mode by writing 0x0 to the following registers:
   - WSP_FAIL_STOP_MODE_INT_EN (0x460940C0)
   - WSP_FAIL_STOP_MODE_EN (0x460940B4)
   - WSP_ECC_ERR_INT_EN (0x46094130)
2. Unmask the PARITY interrupts:
   - Writing 0xFFFFFFFF to the register WSP_PARITY_INT_EN (0x46094050).
3. Configure the WDT (Watch Dog Timer) by writing following register:
   a. Disable the WDT interrupts by writing 0 to bit 0 of the register WDT_INT_EN (0x46094054).
   b. Clear WDT interrupts WDT_INT_SRC (0x46094084).
   c. Set default watchdog timer values:
      - Write 0xFFFFFFFF to the registers WDT_TIMER_VAL_UPE (0x46094088), WDT_TIMER_VAL_BMU (0x4609408C), WDT_TIMER_VAL_HIF (0x46094090)
      - Write 0x00FFFFFF of the register WDT_TIMER_VAL_TLITE (0x46094094).
   d. (S32G3 only) Write 0xFFFFFFFF to the following registers:
      - WDT_TIMER_VAL_HIF_NCPY (0x46094098)
      - WDT_TIMER_VAL_CLASS (0x4609409C)
      - WDT_TIMER_VAL_GPI (0x460940A0)
      - WDT_TIMER_VAL_GPT (0x46094138)
      - WDT_TIMER_VAL_LMEM (0x46094144)

- WDT_TIMER_VAL_ROUTE_LMEM (0x46094148)

e. Enable ALL particular watchdogs by writing the following register
   - 0xFFFFFF to the register CLASS_WDT_INT_EN (0x46094058).
   - 0xF to the register UPE_WDT_INT_EN (0x4609405C).
   - 0xAB to the register HGPI_WDT_INT_EN (0x46094060).
   - 0xC to the register HIF_WDT_INT_EN (0x46094064).
   - 0xFFFFFF to the register TLITE_WDT_INT_EN (0x46094068).
   - 0x3F to the register HNCPY_WDT_INT_EN (0x4609406C).
   - 0xF to the register BMU1_WDT_INT_EN (0x46094070).
   - 0xF to the register BMU2_WDT_INT_EN (0x46094074).
   - 0xFFF to the register EMAC0_WDT_INT_EN (0x46094078).
   - 0xFFF to the register EMAC1_WDT_INT_EN (0x4609407C).
   - 0xFFF to the register EMAC2_WDT_INT_EN (0x46094080).

f. (S32G3 only) Write 0x3 to the following registers:
   - EXT_GPT_WDT_INT_EN (0x46094134)
   - LMEM_WDT_INT_EN (0x46094140)

g. Enable WDT interrupts (excluding the global enable bit) by writing 0xFFFFFFFE to WDT_INT_EN.

4. (S32G3 only) Unmask following interrupts:
   - Write 0x001FFFFF to WSP_BUS_ERR_INT_EN (0x460940D4)
   - Write 0x3 to the following registers:
     - WSP_FW_FAIL_STOP_MODE_INT_EN (0x460940CC)
     - WSP_HOST_FORCE_DEBUG_FAIL_STOP_MODE_INT_EN (0x460940E4)
     - WSP_FAIL_STOP_MODE_INT_EN (0x460940C0)
     - WSP_ECC_ERR_INT_EN (0x46094130)
5. Unmask interrupts by writing 1 to bit 0 of the following registers:
   - WSP_PARITY_INT_EN (0x46094050)
   - WDT_INT_EN (0x46094054).
6. (S32G3 only) Unmask interrupts by writing 1 to bit 0 of the following registers:
   - WSP_BUS_ERR_INT_EN (0x460940D4)
   - WSP_FW_FAIL_STOP_MODE_INT_EN (0x460940CC)
   - WSP_HOST_FORCE_DEBUG_FAIL_STOP_MODE_INT_EN (0x460940E4)
   - WSP_FAIL_STOP_MODE_INT_EN (0x460940C0)
   - WSP_ECC_ERR_INT_EN (0x46094130)

See the function pfe_platform_create_pfe_errors in pfe_platform_master.c file within PFE_MCAL driver.

5.3 Internal RAM initialization

Local Memory (LMEM) is used to store packet headers, frame pointers, and certain table entries. The Classifier accounts for approximately 50% of LMEM accesses, so a dedicated direct access port is provided to optimize performance.

The total LMEM size is 128 KB, and it is memory-mapped to the bus starting at address 0x46000000.

Before initializing any hardware blocks, LMEM must first be initialized by clearing the memory range from 0x46000000 to 0x46002000 to zero.

5.4 BMU initialization

The BMU module is divided into two parts: BMU1 and BMU2, each with its own register base address. The base address for BMU1 is 0x46088000. The base address for BMU2 is 0x4608C000. This distinction is important when accessing or configuring registers specific to each BMU instance. BMU1 is located in local memory in PFE and BMU2 is located in external memory (SRAM/DDR). In PFE_MACL driver, BMU1's address in LMEM in PFE is 0xC0000000 (Seen by PFE) and its maximum buffer count is 512, buff size is 256 bytes. BMU2's address in SRAM is 0x34200000 (configurable) and its maximum buffer count is 256 (configurable), buff size is 2048 bytes.

Before initializing the BMU module, a software reset must be performed by writing the value 0x2 to the BMU_CTRL register (offset 0x04). If the reset is successful, the reset bit will automatically clear. The driver should poll this bit to confirm that the reset has completed. Then, disable the BMU by writing 0x0 to BMU_CTRL.

Note:

1. *BMU2 buffers region must reside within the PFE-accessible range 0x00020000 - 0xBFFFFFFF.*
2. *BMU2 buffers region must be aligned to its size (if the region size is 0x80000, it must be aligned to 0x80000).*

To initialize and configure the each BMU block, follow these steps:

1. **Reset and Disable Interrupts:**
   - Write 0x0 to the BMU_CTRL register.
   - Write 0x0 to the BMU_INT_ENABLE register at offset 0x24.
   - Write 0xFFFFFFFF to the BMU_INT_SRC register at offset 0x20.
2. **Configure Buffer Settings:**
   - Write the base address of the buffer to the BMU_UCAST_BASEADDR register at offset 0x0C.
   - Write the maximum number of buffers to the BMU_UCAST_CONFIG register at offset 0x08.
   - Write the buffer size to the BMU_BUF_SIZE register at offset 0x10.

   Note: *The buffer size must be a power of 2, and the value written should be the exponent (e.g., write 6 for a buffer size of 2^6 = 64 bytes).*
3. **Configure Thresholds:**
   - Write 75% of the maximum number of buffers to the BMU_THRES register at offset 0x18.
   - Write 5% of the free buffers to the BMU_LOW_WATERMARK register at offset 0x50 to define the start of pause frame generation.
   - Write 10% of the free buffers to the BMU_HIGH_WATERMARK register at offset 0x54 to define the stop of pause frame generation.
4. **Clear the BMU internal memory and buffer count memory:**
   There are 64 internal memory blocks and buffer count memory blocks for BMU1 and 1024 internal memory blocks and buffer count memory blocks for BMU2. All blocks need to be clear.
   - Write the block index to the BMU_INT_MEM_ACCESS_ADDR at offset 0x108.
   - Write 0 to the BMU_INT_MEM_ACCESS at offset 0x100.
   - Write 0 to the BMU_INT_MEM_ACCESS2 at offset 0x104.
   - Write the block index to the BMU_BUF_CNT_MEM_ACCESS_ADDR at offset 0x114.
   - Write 0 to the BMU_BUF_CNT_MEM_ACCESS at offset 0x10C.
   - Write 0 to the BMU_BUF_CNT_MEM_ACCESS2 at offset 0x110.
5. **Enable BMU interrupts except the global enable bit:**
   - Write0xFFFFFFFE to the register BMU_INT_ENABLE at offset address 0x24.

For implementation details, refer to the functions pfe_bmu_create in the pfe_bmu.c file within the PFE_MCAL driver.

5.5 Class initialization

Before initializing the Class block, disable the Class block first by writing 0x0 to the register CLASS_TX_CTRL (0x46090004). Then, perform a software rest by writing 0x2 to the register CLASS_TX_CTRL. After soft reset, need to wait for 10us to perform another CSR write/read. Disable the Class block again and set new configuration for the Class block by writing the following register:

1. Write 0xC0088034 to the register CLASS_BMU1_BUF_FREE (0x4609024C).
2. Write 0x02000000 to the register CLASS_PE0_RO_DM_ADDR0 (0x46090060).
3. Write 0x06000400 to the register CLASS_PE0_RO_DM_ADDR1 (0x46090064).
4. Write 0x02000000 to the register CLASS_PE0_QB_DM_ADDR0 (0x46090020).
5. Write 0x06000400 to the register CLASS_PE0_QB_DM_ADDR1 (0x46090024).
6. Write 0xc0080008 to the register CLASS_TM_INQ_ADDR (0x46090114).
7. Write 0x18 to the register CLASS_MAX_BUF_CNT (0x4609020C).
8. Write 0x14 to the register CLASS_AFULL_THRES (0x46090204).
9. Write 0x3C0 to the register CLASS_INQ_AFULL_THRES (0x460902F0).
10. Write 0x1 to the register CLASS_USE_TMU_INQ (0x46090250).
11. Write 0x1 to the register CLASS_PE_SYS_CLK_RATIO (0x46090200).
12. Write 0x0 to the register CLASS_L4_CHKSUM (0x460902A0).
13. Write 0x01000070 to the register CLASS_HDR_SIZE (0x46090014).
14. Write 0x100 to the register CLASS_LMEM_BUF_SIZE (0x46090244).
15. Write 0x88A88100 to the register CLASS_TPID0_TPID1 (0x46090298).
16. Write 0x9100 to the register CLASS_TPID2 (0x4609029C).
17. If the platform is S32G3, write 0x2003 CLASS_AXI_CTRL_ADDR (0x4609050C).
18. Write 0xB010 to the register CLASS_ROUTE_MULTI (0x4609023C).

After initializing the CLASS HW block, the DMEM and IMEM in the CLASS should be clear before loading PFE firmware (See the function pfe_pe_load_firmware).

For implementation details, see the functions pfe_class_create in the pfe_class.c file within the PFE_MCAL driver.

5.6 FW loading

After initializing the CLASS, upload the firmware ELF into CLASS. See Loading and Running FW.

5.7 TMU Initialization

Before initializing the TMU module, a software reset must be performed by writing the value 0x1 to the register TMU_CTRL (0x46080038). If the reset is successful, the reset bit will automatically clear. The driver should poll this bit to confirm that the reset has completed.

After resetting the TMU module, all queues within the TMU must be initialized. Before initializing the queues, the TMU context memory must be switched to direct access mode by writing the value 0x1 to the register TMU_CNTX_ACCESS_CTRL (0x46080134).

Next, initialize 8 queues for each PHY. The system includes 6 PHYs in total:

- 3 EMACs
- All HIFs share 1 PHY
- HIF No Copy
- UTIL

The 4 HIFs share a single PHY, and each HIF is assigned with 2 queues. To initialize a queue in the TMU module, the PHY and queue selection must first be configured. This is done by writing the appropriate values to the register TMU_PHY_QUEUE_SEL (0x46080014). The bit 0-2 is queue selection and bit 8 – 12 is PHY selection. Afterwards, writing 0 to the following register for each queue in every PHY:

- TMU_CURQ_PTR (0x46080018)
- TMU_CURQ_PKT_CNT (0x4608001c)
- TMU_CURQ_DROP_CNT (0x46080020)
- TMU_CURQ_TRANS_CNT (0x46080024)
- TMU_CURQ_QSTAT (0x46080028)
- TMU_HW_PROB_CFG_TBL0 (0x4608002c)
- TMU_HW_PROB_CFG_TBL1 (0x46080030)
- TMU_CURQ_DEBUG (0x46080034)

(S32G2 only) The following operations need to be done for queue 0 of EMAC0:

- Write the value 0x0 to the register TMU_CNTX_ACCESS_CTRL (0x46080134).
- Write the value 0x5 to the register TMU_CNTX_ADDR (0x46080138).
- Write the value 0x0 to the register TMU_CNTX_DATA (0x4608013C).
- Write the value 0x3 to the register TMU_CNTX_CMD (0x46080140).
- Poll to read the register TMU_CNTX_CMD until its value is not 0x4.
- Write the value 0x0 to the register TMU_CNTX_ACCESS_CTRL (0x46080134).
- Write the value 0x6 to the register TMU_CNTX_ADDR (0x46080138).
- Write the value 0x0 to the register TMU_CNTX_DATA (0x4608013C).
- Write the value 0x3 to the register TMU_CNTX_CMD (0x46080140).
- Poll to read the register TMU_CNTX_CMD until its value is not 0x4.
- Write the value 0x0 to the register TMU_CNTX_ACCESS_CTRL (0x46080134).
- Write the value 0x4 to the register TMU_CNTX_ADDR (0x46080138).
- Write the value 0x2 to the register TMU_CNTX_DATA (0x4608013C).
- Write the value 0x3 to the register TMU_CNTX_CMD (0x46080140).
- Poll to read the register TMU_CNTX_CMD until its value is not 0x4.
- Initialize internal TMU FIFO by writing 0 to the register TMU_PHY_INQ_PKTINFO (0x4608000C) for 256 times.
- Write the value 0x0 to the register TMU_CNTX_ACCESS_CTRL (0x46080134).
- Write the value 0x2 to the register TMU_CNTX_ADDR (0x46080138).
- Write the value 0x2 to the register TMU_CNTX_CMD (0x46080140).
- Poll to read the register TMU_CNTX_CMD until its value is not 0x4.
- Read the register TMU_CNTX_DATA (0x4608013C).
- If the value of register TMU_CNTX_DATA is not 256, retry above four steps.
- Clear the bit 0 and 1 of the register TMU_TEQ_CTRL (0x46080047).
- Write the value 0x0 to the register TMU_CNTX_ACCESS_CTRL (0x46080134).
- Write the value 0x4 to the register TMU_CNTX_ADDR (0x46080138).
- Write the value 0x0 to the register TMU_CNTX_DATA (0x4608013C).
- Write the value 0x3 to the register TMU_CNTX_CMD (0x46080140).
- Poll to read the register TMU_CNTX_CMD until its value is not 0x4.

After all preparation jobs are done, execute the following steps to initialize TMU:

1. (S32G3 only) Set the TMU_TEQ_CTRL (0x4608004c) register bit 2 to 1.
2. Write 0 to the following register:
   - TMU_PHY0_TDQ_CTRL (0x460800f0)

- TMU_PHY1_TDQ_CTRL (0x460800f4)
- TMU_PHY2_TDQ_CTRL (0x460800f8)
- TMU_PHY3_TDQ_CTRL (0x460800fc)
- TMU_PHY4_TDQ_CTRL (0x46080100)
- TMU_PHY5_TDQ_CTRL (0x46080104)

3. Perform TMU software reset again.
4. Write 0xC00AC030 to TMU_PHY0_INQ_ADDR (0x46080064).
   - Write 0xC00B0030 to TMU_PHY1_INQ_ADDR (0x46080068).
   - Write 0xC00B4030 to TMU_PHY2_INQ_ADDR (0x4608006c).
   - Write 0xC009C030 to TMU_PHY3_INQ_ADDR (0x46080070).
   - Write 0xC00D0050 to TMU_PHY4_INQ_ADDR (0x46080074).
   - Write 0xC00CC010 to TMU_PHY5_INQ_ADDR (0x46080078).
   - Write 0xC009C030 to TMU_PHY16_INQ_ADDR (0x460800a4).
5. Configure the default egress QoS topology for each PHY. There is a little different between S32G2 and S32G3. TDQ and SHP address range for PHY5(UTIL) does not exist on G2. Skip the configuration for PHY5 about the scheduler and shaper.
   For each PHY (Not required for G2 PHY5(UTIL)):
   - Configure for scheduler 0:
     - Write 0xFFFFFFFF to the register TMU_SCH0_Q_ALLOC0 (0x46081040).
     - Write 0xFFFFFFFF to the register TMU_SCH0_Q_ALLOC1 (0x46081044).
     - Write 0xF to the register TMU_SCH0_POS (0x46081054).
   - Configure for scheduler 1:
     - Write 0xFFFFFFFF to the register TMU_SCH1_Q_ALLOC0 (0x46081140).
     - Write 0xFFFFFFFF to the register TMU_SCH1_Q_ALLOC1 (0x46081144).
   - Configure for shaper 0:
     - Write 0x0 to the register TMU_SHP0_CTRL (0x46081200).
       Write 0x3E to the register TMU_SHP0_CTRL2 (0x4608120C).
       Write 0x0 to the register TMU_SHP0_MAX_CREDIT (0x46081208).
       Write 0x0 to the register TMU_SHP0_MIN_CREDIT (0x46081210).
   - Configure for shaper 1:
     - Write 0x0 to the register TMU_SHP1_CTRL (0x46081300).
     - Write 0x3E to the register TMU_SHP1_CTRL2 (0x4608130C).
     - Write 0x0 to the register TMU_SHP1_MAX_CREDIT (0x46081308).
     - Write 0x0 to the register TMU_SHP1_MIN_CREDIT (0x46081310).
   - Configure for shaper 2:
     - Write 0x0 to the register TMU_SHP2_CTRL (0x46081400).
     - Write 0x3E to the register TMU_SHP2_CTRL2 (0x4608140C).
     - Write 0x0 to the register TMU_SHP2_MAX_CREDIT (0x46081408).
     - Write 0x0 to the register TMU_SHP2_MIN_CREDIT (0x46081410).
   - Configure for shaper 3:
     - Write 0x0 to the register TMU_SHP3_CTRL (0x46081500).
     - Write 0x3E to the register TMU_SHP3_CTRL2 (0x4608150C).
     - Write 0x0 to the register TMU_SHP3_MAX_CREDIT (0x46081508).
     - Write 0x0 to the register TMU_SHP3_MIN_CREDIT (0x46081510).
   - Assign each queue to the scheduler1 input:
     - Write 0x00010203 to the register TMU_SCH1_Q_ALLOC0 (0x46081140).

– Write 0x04050607 to the register TMU_SCH1_Q_ALLOC1 (0x46081144).
- Set data rate mode and RR algorithm for scheduler 1(Not required for G2 PHY5(UTIL)):
  – Write 0x0 to the register TMU_SCH1_BIT_RATE (0x46081148).
  – Write 0x3 to the register MU_SCH1_CTRL (0x46081100).
- The default egress QoS for per PHY in TMU is as below:

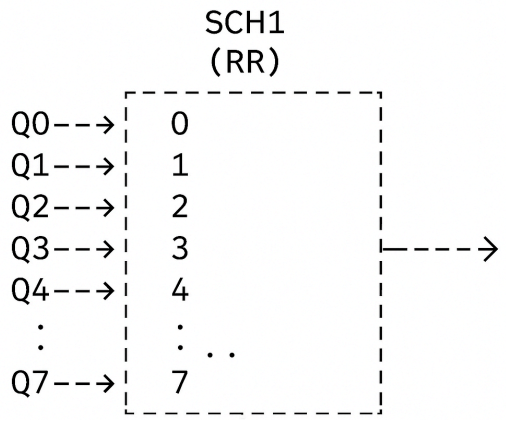

Figure 8. Default egress QoS

- Set tail drop mode for every queue (8 queues in total) in each PHY:
  For PHY3 (HIF), every queue size is 16,
  – Write PHY index to bit 16 – 20 and 8 * queue index + 4 to the register TMU_CNTX_ADDR (0x46080138).
  – Write 0x8001 to the register TMU_CNTX_DATA (0x4608013C).
  – Write the value 0x3 to the register TMU_CNTX_CMD (0x46080140).
  – Poll to read the register TMU_CNTX_CMD until its value is not 0x4.

6. - Write 0xc0088034 to TMU_BMU_INQ_ADDR (0x4608003c).
   - Write 0xc008c034 to TMU_BMU2_INQ_ADDR (0x46080050).
   - Write 0x100 to TMU_AFULL_THRES (0x46080040).
   - Write 0xfc to TMU_INQ_WATERMARK (0x46080040).
   - Write 0xf to TMU_PHY0_TDQ_CTRL (0x460800f0).
   - Write 0xf to TMU_PHY1_TDQ_CTRL (0x460800f4).
   - Write 0xf to TMU_PHY2_TDQ_CTRL (0x460800f8).
   - Write 0xf to TMU_PHY3_TDQ_CTRL (0x460800fc).
   - Write 0xf to TMU_PHY4_TDQ_CTRL (0x46080100).
   - Write 0xf to TMU_PHY5_TDQ_CTRL (0x46080104).
   - Write 0xf to TMU_PHY16_TDQ_CTRL (0x46080130).

   For implementation details, refer to the functions pfe_tmu_create in the pfe_tmu.c file within the PFE_MCAL driver.

5.8 EMAC Initialization

Before initializing three EMAC ports, write 0x1 to the register MTL_DPP_CONTROL (offset 0xCE0), which HW-specific pre-initialization function related to ERR050221. Disable the EMAC controllers by writing 0x0 to the register MAC_CONFIGURATION (offset 0x0).

Note: *EMAC registers info is not described in this document. EMAC has the same registers description as GMAC. See GMAC subsystem RM, which could be downloaded from NXP official website.*

Perform following steps to initialize the HW:

1. Clear the MAC address registers (there are 8 MAC address slot for each MAC). For each MAC address register, high address register offset is 0x300 * n and low address register is 0x304 * n.
   - For the first entry: Write 0x8000FFFF to the MAC high address entry and write 0xFFFFFFFF to the MAC low address entry.
   - For other seven entry: Write 0x0000FFFF to the MAC high address entry and write 0xFFFFFFFF to the MAC low address entry.
   - Clear hash table by writing 0x0 to the register MAC_HASH_TABLE_REG0 (offset 0x10) and MAC_HASH_TABLE_REG1 (offset 0x14).
   - Wait at least 4 clock cycles.
2. Write the following registers:
   - Write 0x0 to the register MAC_CONFIGURATION (offset 0x0).
   - Write 0x0446 to the register MAC_PACKET_FILTER (offset 0x8).
   - Write 0 to the bit 1 of the register MAC_Q0_TX_FLOW_CTRL (offset 0x70).
   - Write 0x0 to the register MAC_INTERRUPT_ENABLE (offset 0xB4).
   - Write 0xFFFFFFFF to the register MMC_RX_INTERRUPT_MASK (offset 0x70C).
   - Write 0xFFFFFFFF to the register MMC_TX_INTERRUPT_MASK (offset 0x710).
   - Write 0xFFFFFFFF to the register MMC_IPC_RX_INTERRUPT_MASK (offset 0x800).
   - Enable ECC, timeout and parity checking by writing 1 to the bit 0, 1, 2, 3 and 4 of the register MTL_ECC_CONTROL (offset 0xCC0).
   - Write 0x22012C to the register MAC_FSM_ACT_TIMER (offset 0x14C).
   - Write 0x5 to the register MTL_DPP_CONTROL (offset 0xCE0).
   - Write 0x3 to the register MAC_FSM_CONTROL (offset 0x148).
   - Write 0x08BA2000 to the register MAC_CONFIGURATION (offset 0x0).
   - Write 1 to the bit 4 of register MTL_RXQ0_OPERATION_MODE (offset 0xD30).
   - Write 0x0 to the register MTL_TXQ0_OPERATION_MODE (offset 0xD00).
   - Write 0x5F2 to the register MAC_EXT_CONFIGURATION (offset 0x4).
   - Write 0x0 to the register MAC_TIMESTAMP_CONTROL (offset 0xB00).
   - Write 0x0 to the register MAC_SUB_SECOND_INCREMENT (offset 0xB04).
3. Disable broadcast:
   - Write 0x0466 to the register MAC_PACKET_FILTER (offset 0x8).
4. Disable loopback mode:
   - Write 0 to bit 12 of the MAC_CONFIGURATION (offset 0x0).
5. Disable promiscuous mode
   - Write 0 to bit 0 of the MAC_CONFIGURATION (offset 0x0).

The register base addresses for each EMAC instance are as follows:

- EMAC0: 0x460A0000
- EMAC1: 0x460A4000
- EMAC2: 0x460A8000

5.9 PFE system soft reset

Steps to Perform PFE System Soft Reset:

1. Write 1 to bit 27 28 29 of WSP_SYS_GENERIC_CONTROL at address 0x46094020 and write 0 again to clear them. (Only S32G3 platform required)
2. Set bit 30 to 1 to perform soft reset.
3. Poll to check the bit 19 of register WSP_DBUG_BUS1_G3 at address 0x460940A4. If the bit 19 is 1, it indicates the soft reset done. Once soft reset is done, write 0 to bit 30 of WSP_SYS_GENERIC_CONTROL and repeat the step 1. (Only S32G3 platform required)

4. After performing step 2, wait 100 us for the reset to occur and write 0 to bit 30 of WSP_SYS_GENERIC_CONTROL. (Only S32G2 platform required)

5.10 GPI Initialization

There are seven GPI instances in the PFE: three GPI, three ETGPI (Egress Timestamp GPI), and one HGPI (High-speed GPI). Each EMAC shares one GPI and one ETGPI. All HIFs share a single HGPI.

To initialize GPI instances, A soft reset is required first, and make sure all HIF/HIF_NOCPY channel DMAs are disabled before the soft reset.

To disable the HIF channel DMA, write 0 to bit 0, 1, 16, and 17 of the HIF_CTRL_CHn register. The register base addresses for each HIF instance are as follows:

- HIF0: 0x46098100
- HIF1: 0x46098200
- HIF2: 0x46098300
- HIF3: 0x46098400

Disable the HIF channel TX and RX IRQ by writing 0 to bit 1, 2, 3 and 4 of the HIF_CHn_INT_EN (offset 0x64) register.

Perform PFE System Soft Reset before initializing every EGPI, HGPI and ETGPI instance. Here are the steps to initialize a GPI instance:

1. Reset GPI first, write 0x2 to the register GPI_CTRL at offset address 0x4. If the reset is successful, the reset bit will automatically clear. The driver should poll this bit to confirm that the reset has completed.
2. Reset EGPI ingress QoS configuration (Only EGPI required).
   a. Clear 64 flow entries. For each entry, write 0x0 to clear all 8 registers CSR_IGQOS_ENTRY_DATA_REGn at offset address 0x188 – 0x1A4.
   b. For each entry, set the entry index to bit 8-14 and set 1 to bit 0, write the value the register CSR_IGQOS_ENTRY_CMDCNTRL at offset address 0x184.
   c. After clear a flow entry, poll to read the bit 0 of register CSR_IGQOS_ENTRY_CMDSTATUS at offset address 0x180. Once the bit 0 is 1, the command is complete.
   d. Clear 64 LRU entries, repeat the step a.
   e. For each entry, set the entry index to bit 8-14 and set 1 to bit 0 and set 1 to bit 16, write the value the register CSR_IGQOS_ENTRY_CMDCNTRL at offset address 0x184.
   f. Repeat step c.
   g. Init the GPI QoS default configuration for each EGPI instance.
   Bypass the ingress QoS by writing 0x0 to the register CSR_IGQOS_CONTROL at the offset address 0x174.
   Initialize the WRED sub block:
   - Write 0x0 to the register CSR_IGQOS_QOS at the offset address 0x17C.
   - Write 0x00018421 to the register CSR_IGQOS_DMEMQ_ZONE_PROB at the offset address 0x150.
   - Write 0x00018421 to the register CSR_IGQOS_LMEMQ_ZONE_PROB at the offset address 0x15C.
   - Write 0x00018421 to the register CSR_IGQOS_RXFQ_ZONE_PROB at the offset address 0x168.
   - Write 0x00002000 to the register CSR_IGQOS_DMEMQ_FULL_THRESH at the offset address 0x154.
   - Write 0x10001FF0 to the register CSR_IGQOS_DMEMQ_DROP_THRESH at the offset address 0x158.
   - Write 0x00000200 to the register CSR_IGQOS_LMEMQ_FULL_THRESH at the offset address 0x160.
   - Write 0x00000200 to the register CSR_IGQOS_RXFQ_FULL_THRESH at the offset address 0x16C.
   - Write 0x010001F0 to the register CSR_IGQOS_LMEMQ_DROP_THRESH at the offset address 0x164.

- Write 0x010001F0 to the register CSR_IGQOS_RXFQ_DROP_THRESH at the offset address 0x170.
- Initialize two shapers:
  - Write the 0x0 to the register CSR_IGQOS_PORT_SHP_WGHT and shaper 0 offset is 0x1E4 and shaper 1 offset is 0x1F0.
  - Write the 0x0 to the register GPI_PORT_SHP_MIN_CREDIT and shaper 0 offset is 0x140 and shaper 1 offset is 0x144.
- Set the shaper type to port level by setting 0 to bit 2 -3 and bit 4 – 6 of the register CSR_IGQOS_PORT_SHP_CONFIG at the offset address 0x1F8. Bit 2 – 3 is for shaper 0 and bit 4 – 6 is for shaper 1.
- Set the shaper mode to bps by setting 0 to bit 0 and bit 1 of the register CSR_IGQOS_PORT_SHP_CONFIG at the offset address 0x1F8. Bit 0 is for shaper 0, and bit 1 is for shaper 1.
- Write 0x0 to the register CSR_IGQOS_PORT_SHP_CTRL shaper 0 offset is 0x1E0 and shaper 1 offset is 0x1EC.

h. Write 0x81008100 to the register CSR_IGQOS_TPID at the offset address 0x218.
i. Write 0x00000030 to the register CSR_IGQOS_CLASS at the offset address 0x178.

3. Disable GPI by writing 0x0 to the register GPI_CTRL.
4. Init GPI by writing following registers:
   a. Write 0x0 to the register GPI_EMAC_1588_TIMESTAMP_EN at offset address 0x13C.
   b. Write 0xE01 to the register GPI_EMAC_1588_TIMESTAMP_EN at offset address 0x13C (HGPI is not required).
   c. Write 0x02000003 to the register GPI_RX_CONFIG at offset address 0x008.
   d. Write 0x02000070 to the register GPI_HDR_SIZE at offset address 0x00C.
   e. Write 0x08000100 to the register GPI_BUF_SIZE at offset address 0x010.
   f. Write 0xC0088030 to the register GPI_LMEM_ALLOC_ADDR at offset address 0x014.
   g. Write 0xC0088034 to the register GPI_LMEM_FREE_ADDR at offset address 0x018.
   h. Write 0xC008C030 to the register GPI_DDR_ALLOC_ADDR at offset address 0x01C.
   i. Write 0xC008C034 to the register GPI_DDR_FREE_ADDR at offset address 0x020.
   j. Write 0xC0090010 to the register GPI_CLASS_ADDR at offset address 0x024.
   k. Write 0x200 to the register GPI_DDR_DATA_OFFSET at offset address 0x034.
   l. Write 0x30 to the register GPI_LMEM_DATA_OFFSET at offset address 0x038.
   m. Write 0x70 to the address GPI_LMEM_SEC_BUF_DATA_OFFSET at offset address 0x060.
   n. Write 0x178 to the register GPI_TMLF_TX at offset address 0x04C.
   o. Write 0x50 to the register GPI_DTX_ASEQ at offset address 0x050.
   p. Write 0x1 to the register GPI_CSR_TOE_CHKSUM_EN at offset address 0x068.
   q. Set 1 to bit 0 and 1 of the register GPI_CSR_AXI_WRITE_DONE_ADDR at offset address 0x14C (Only S32G3 platform required).
   r. Write 0xFFFFFFFF to register CSR_IGQOS_LRU_TIMER_VALUE at offset address 0x208.

Below table is the base address of each GPI instance:

Table 3. Base address of GPI instances

Index	GPI	Base Address
0	EGPI1	0x460AC000
1	EGPI2	0x460B0000
2	EGPI3	0x460B4000
3	HGPI	0x4609C000
4	ETGPI1	0x460B8000

Table 3. Base address of GPI instances...*continued*

Index	GPI	Base Address
5	ETGPI2	0x460BC000
6	ETGPI3	0X460C0000

For implementation details, refer to the functions pfe_gpi_create in the pfe_gpi.c file within the PFE_MCAL driver.

5.11 HIF Initialization

Configure and initialize the global HIF HW block by writing following registers:

1. Disable and clear HIF interrupts:
   - Write 0x0 to the register HIF_ERR_INT_EN (0x4609806C).
   - Write 0x0 to the register HIF_TX_FIFO_ERR_INT_EN (0x46098074).
   - Write 0x0 to the register HIF_RX_FIFO_ERR_INT_EN (0x4609807C).
   - Write 0xFFFFFFFF to the register HIF_ERR_INT_SRC (0x46098068).
   - Write 0xFFFFFFFF to the register HIF_TX_FIFO_ERR_INT_SRC (0x46098070).
   - Write 0xFFFFFFFF to the register HIF_RX_FIFO_ERR_INT_SRC (0x46098078).
2. Perform HIF soft reset (Only S32G2 required):
   - Write 0xF to the register HIF_SOFT_RESET (0x46098014).
     Poll to read HIF_SOFT_RESET until the value is 0.
3. Reset all ring registers of all HIFs (Only S32G3 required):
   - Set RX BD ring address by writing 0x0 to the registers HIF_RX_BDP_RD_LOW_ADDR_CHn (offset 0x0C) and HIF_RX_BDP_RD_HIGH_ADDR_CHn (offset 0x10).
   - Set TX BD ring address by writing 0x0 to the registers HIF_TX_BDP_RD_LOW_ADDR_CHn (offset 0x1C) and HIF_TX_BDP_RD_HIGH_ADDR_CHn (offset 0x20).
   - Set RX WB table by writing 0x0 to the register HIF_RX_BDP_WR_LOW_ADDR_CHn (offset 0x04) and HIF_RX_BDP_WR_HIGH_ADDR_CHn (offset 0x08).
   - Set table length by writing 0x0 to the register HIF_RX_WRBK_BD_CHn_BUFFER_SIZE (offset 0x24).
   - Set TX WB table by writing 0x0 to the register HIF_TX_BDP_WR_LOW_ADDR_CHn (offset 0x14) and HIF_TX_BDP_WR_HIGH_ADDR_CHn (offset 0x18).
   - Set table length by writing 0x0 to the register HIF_TX_WRBK_BD_CHn_BUFFER_SIZE (offset 0x2C).
     The register base addresses for each HIF instance are as follows:
     - HIF0: 0x46098100
     - HIF1: 0x46098200
     - HIF2: 0x46098300
     - HIF3: 0x46098400
4. Write following registers:
   - Write 0x00FF00FF to the register HIF_TX_POLL_CTRL (0x46098004).
   - Write 0x00FF00FF to the register HIF_RX_POLL_CTRL (0x46098008).
   - Write 0x20 to the register HIF_MISC (0x46098008).
   - Write 0x05F5E100 to the register HIF_TIMEOUT_REG (0x46098010).
   - Write 0x33221100 to the register HIF_RX_QUEUE_MAP_CH_NO_ADDR (0x460980CC).
   - Write 0x0 to the register HIF_DMA_BURST_SIZE (0x460980C8).
   - Write 0x0 to the register HIF_DMA_BASE_ADDR (0x460980C4).
   - Write 0x0 to the register HIF_LTC_PKT_CTRL_ADDR (0x460980D0).Write 0xFFFFFFFE to the register HIF_ERR_INT_EN (0x4609806C).
   - Write 0xFFFFFFFE to the register HIF_TX_FIFO_ERR_INT_EN (0x46098074).

- Write 0xFFFFFFFE to the register HIF_RX_FIFO_ERR_INT_EN (0x4609807C).

For step 1 to 4 implementation details, refer to the functions pfe_hif_cfg_init in the pfe_hif_csr.c file within the PFE_MCAL driver

5. Once the HIF HW configuration is set, following steps must be performed for every HIF channel within the given application:
   a. Disable the channel interrupts by writing 0x0 to the register HIF_CHn_INT_EN (offset 0x64) and writing 0xFFFFFFFF to the register HIF_CHn_INT_SRC (0x60).
   b. Disable TX/RX DMA by writing 0x0 to bit 0, bit 1, bit 16 and bit 17 of the register HIF_CTRL_CHn (offset 0x0) and writing 0x0 to bit 1, bit 2, bit 3 and bit 4 of the register HIF_CHn_INT_EN (offset 0x64).
   c. Disable interrupt coalescing by writing 0x0 to the registers HIF_INT_COAL_EN_CHn (offset 0xF0), HIF_ABS_FRAME_COUNT_CHn (offset 0xEC) and HIF_ABS_INT_TIMER_CHn (offset 0xE8).
   d. Enable channel status interrupts in HIF_CHn_INT_EN (offset 0x64) except of the RX/TX and the global enable bit by writing 0xFFFFFFFF and masking bit 0 to 10 (bit 5 to 10 are set 0 for a workaround AAVB-6034).
   e. Bind TX/RX BD rings in the channel registers:
      For TX:
      i. Create the TX BD Ring and TX WB BD Ring. Ensure that descriptors contain valid values (See the *function pfe_hif_ring_create_std* in pfe_hif_ring.c).
      ii. Set HIF_TX_BDP_RD_LOW_ADDR_CHn (offset 0x1C) and HIF_TX_BDP_RD_HIGH_ADDR_CHn (offset 0x20) to start address of the TX BD Ring.
      iii. Set HIF_TX_BDP_WR_LOW_ADDR_CHn (offset 0x14) and HIF_TX_BDP_WR_HIGH_ADDR_CHn (offset 0x18) to start address of the TX WB BD Ring.
      iv. Set HIF_TX_WRBK_BD_CHn_BUFFER_SIZE (offset 0x2C) to number of descriptors provided by the TX WB BD Ring.
      For RX:
      i. Create the RX BD Ring and RX WB BD Ring. Ensure that descriptors contain valid Values (See the function *pfe_hif_ring_create_std in pfe_hif_ring.c*).
      ii. Set HIF_RX_BDP_RD_LOW_ADDR_CHn (offset 0x0C) and HIF_RX_BDP_RD_HIGH_ADDR_CHn (offset 0x10) to start address of the RX BD Ring.
      iii. Set HIF_RX_BDP_WR_LOW_ADDR_CHn (offset 0x04) and HIF_RX_BDP_WR_HIGH_ADDR_CHn (offset 0x08) to start address of the RX WB BD Ring.
      iv. Set HIF_RX_WRBK_BD_CHn_BUFFER_SIZE (offset 0x24) to number of descriptors provided by the RX WB BD Ring.
      v. Assign the RX buffer to the RX BD ring (See the function pfe_hif_chnl_refill_rx_buffers in pfe_hif_ring.c).

For implementation details, refer to the function pfe_hif_chnl_create_mcal_aux in the pfe_hif_chnl.c file within the PFE_MCAL driver.

5.12 Enable each sub-module

After configuring all PFE HW blocks, enable these blocks by following steps:

1. Enable Class module (See the function pfe_class_enable in pfe_class.c):
   - Write 0x1 to the register CLASS_TX_CTRL (0x46090004).

- Once the enable bit is set, the driver must poll PE's status to confirm successful initialization. This involves reading the PE state register at address 0x20000E44 in DMEM and verifying that PE is ready for a new frame arrival.

2. Enable BMU module for both BMU instance ():
   - Write 0x1 to the register BMU_CTRL (offset 0x4).
3. Enable GPI module for 7 GPI instances:
   - Set 1 to bit 1 of register GPI_CTRL (offset 0x4) for each GPI instance.
4. Enable TMU module.
5. Write 0x80000003 to the register WSP_SYS_GENERIC_CONTROL (0x46094020) to indicate the configuration done for PFE.
6. Enable timestamping for three EMACs. (Not necessary. If enabled, see the function *pfe_platform_enable_ts*).
7. Configure PPS output and PPS output is available on EMAC0 only. (Not necessary. If enabled, see the function *pfe_emac_pps0_configure*).
8. Enable the HIF TX/RX DMA and unmask channel TX/RX IRQ:
   - Writing 0x1 to bit 0, bit 1, bit 16 and bit 17 of the register HIF_CTRL_CHn (offset 0x0) and writing 0x1 to bit0, bit 1, bit 2, bit 3 and bit 4 of the register HIF_CHn_INT_EN (offset 0x64).
9. Enable physical interface HIF:
   Set the interface enable flag in the physical interface and write it to the PFE DMEM.
   - Write 1 to bit 0, bit 1, bit 16 and bit 17 of the register HIF_CTRL_CHn.
   - Write 1 to bit 1, bit 2, bit 3 and bit 4 of the register HIF_CHn_INT_EN.
10. Configure each EMAC controller:
    - Set the EMAC link speed and duplex mode to the register MAC_CONFIGURATION (offset 0x0).
    - Configure MAC addresses and direct Rx traffic to our HIF:
      a. Enable the promiscuous mode for physical interface (If required):
         Set the promiscuous enable flag in the physical interface and update it to the CALSS DMEM.
         Enable the PROMISCUOUS_MODE bit of the register MAC_PACKET_FILTER (offset 0x08). (See the function *pfe_phy_if_promisc_enable in pfe_phy_if.c* file).
      b. Add MAC address:
         Check if the address already exists in slot first and check if any slot is available. If no slot is available, calculate the hash value of address and write it into the register MAC_HASH_TABLE_REG (offset 0x10 + n * 4). If there is free address slot, write the address into the register MAC_ADDRESS_HIGH and MAC_ADDRESS_LOW and enable the bit 31 of the MAC_ADDRESS_HIGH.
      c. Direct frames from EMAC to our HIF:
         Configure the logical interface egress port and update to the CLASS DMEM.
    - Configure scheduler and shaper for EMAC ports (If special egress QoS topology needed).
      For implementation details, refer to the functions InterfacePrepare_MasterConfigEMAC in the Eth_PFE_LLD.c file within the PFE_MCAL driver.
11. Enables the given controllers (for EMAC):
    - Set the interface enable flag in the physical interface and write it to the PFE DMEM.
    - Write 1 to bit 0 and 1 of the register MAC_CONFIGURATION (offset 0x0).

5.13 PFE driver shutdown

This recovery procedure refers to a soft reset of the PFE HW. The sequence involves shutting down the PFE, issuing the soft reset via dedicated PFE register, and bringing the PFE back up. Therefore, to issue a PFE soft reset, following steps are required:

1. Preferably gracefully terminate all applications leveraging the PFE.
2. Shut down the PFE driver instance.

3. Start the PFE driver instance.

Shut down the PFE driver:

1. Disable HIF physical interfaces first.
   a. Disable the physical interface flag and update to the CLASS DMEM.
   b. Disable the HIF channel DMA by writing 0 to bit 0, 1, 16, and 17 of the HIF_CTRL_CHn.
   c. Disable the HIF channel TX and RX IRQ by writing 0 to bit 1, 2, 3 and 4 of the HIF_CHn_INT_EN.
2. Clear the register WSP_SYS_GENERIC_CONTROL (0x46094020).
3. Delete associated logical interface list pointer in the physical interface and remove the logical interface in the DMEM heap.
4. Mask HIF interrupts by clearing the bit 0 of HIF_ERR_INT_EN, HIF_TX_FIFO_ERR_INT_EN and HIF_RX_FIFO_ERR_INT_EN.
5. Repeat step 1 for all HIF channels.
6. Disable the HIF interrupts by writing 0x0 to HIF_ERR_INT_EN, HIF_TX_FIFO_ERR_INT_EN and HIF_RX_FIFO_ERR_INT_EN.
7. Disable every GPI instance and reset GPI block.
   a. Write 0 to the bit 0 of GPI_CTRL.
   b. Reset the GPI QoS (Only EGPI is required).
   c. Reset the GPI.
8. Disable 2 BMU instances by writing 0x0 to the register BMU_CTRL and writing 0x0 to the register BMU_INT_ENABLE and writing 0xFFFFFFFF to the register BMU_INT_SRC.
9. Disable CLASS block by writing 0 to the register CLASS_TX_CTRL.
10. Clear the MAC_ADDRESS_HIGH, MAC_ADDRESS_LOW and MAC_HASH_TABLE_REG.
11. Disable the EMAC by writing bit 1 and bit 0 of the register MAC_CONFIGURATION.
12. Disable the EMAC timestamp by writing 0x0 to the register MAC_TIMESTAMP_CONTROL.
13. Mask ECC_ERR interrupts by clearing the bit 0 of WSP_ECC_ERR_INT_EN.
14. Mask FAIL_STOP interrupts by clearing the bit 0 of WSP_FAIL_STOP_MODE_INT_EN.
15. Mask HOST_FAIL_STOP interrupts by clearing the bit 0 of WSP_HOST_FORCE_DEBUG_FAIL_STOP_MODE_INT_EN.
16. Mask FW_FAIL_STOP interrupts by clearing the bit 0 of WSP_FW_FAIL_STOP_MODE_INT_EN.
17. Mask BUS_ERR interrupts by clearing the bit 0 of WSP_BUS_ERR_INT_EN.
18. Mask PARITY interrupts by clearing the bit 0 of WSP_PARITY_INT_EN.
19. Disable the WDT interrupts by writing 0x0 to WDT_INT_EN and Clear WDT interrupts WDT_INT_SRC.

For implementation details, refer to the functions Eth_PFE_LLD_ShutdownDriver in Eth_PFE_LLD.c file within the PFE_MCAL driver.

5.14 Hard reset

The hard reset represents SoC-assisted PFE HW reset. The procedure is utilizing the Software Reset Partition manipulation. To execute the PFE hard reset, following steps must be executed:

1. Preferably gracefully terminate all applications leveraging the PFE.
2. Shut down the PFE driver instance.
3. Perform the PFE HW Reset.
4. Start PFE driver again.

PFE HW Reset:

Following steps reset the PFE hardware into its initial state:

1. Execute the PFE Software Reset Partition (partition 2) turn-off sequence (see the *S32G Reference Manual*).

2. Execute the PFE Software Reset Partition (partition 2) turn-on sequence (see the *S32G Reference Manual*).

5.15 Transmitting and Receiving

See *HIF programmer guide* for packets transmitting and receiving in PFE.

5.16 PFE slave driver initialization

A simple multi-instance PFE driver system design should follow these constraints:

- Each PFE driver instance uses exactly **one HIF** and **one EMAC**.
- EMACs are **not shared** among PFE drivers.

With these system constraints, a PFE slave driver can be initialized based on following points:

1. The slave must not begin hardware initialization until the master is fully functional; otherwise, the PFE may become non-functional. This requirement implies some form of communication between drivers or the execution environment, which must be considered during system design.
2. All PFE internal registers should be configured by the master driver. The PFE slave driver must not write to PFE internal memory.
3. The PFE slave driver is responsible only for configuring the corresponding EMAC registers, HIF interface and managing its Tx/Rx buffers, buffer descriptors, and write-back buffer descriptors. For the PFE slave driver, it must not configure the HIF global registers again and it should only handle the configuration of its corresponding channel.

Note: *Errata 05121 should be enabled If multiple PFE drivers are enabled.*

6 Firmware interface

6.1 PhyIf

Physical interfaces are static objects (defined at startup), which represent hardware interfaces of PFE. They are used by PFE for the ingress/egress of network traffic. Physical interfaces have several configurable properties. Among all these properties, the property .mode is especially important. Mode of a physical interface specifies which classification algorithm is applied on the ingress traffic of the interface.

Every physical interface can have a list of logical interfaces. By default, all physical interfaces are in a default mode (FPP_IF_OP_DEFAULT). In the default mode, ingress traffic of a given physical interface is processed using only the associated default Logical Interface. There are 9 physical interfaces including EMAC0, EMAC1, EMAC2, HIF0, HIF1, HIF2, HIF3, HIF No Copy and UTIL, see Physical Interface Identifiers. The key idea of creating a physical interface in PFE driver is to contains the same data structure as the physical interface inside the CLASS DMEM.

Perform the following steps to create a new physical interface and update to DMEM:

1. Create a physical interface instance with physical interface structure seen by CLASS.
2. Get the memory map of the physical interface in the PFE memory from the memory map data section copied from PFE firmware.
3. Initialize the interface structure with default configuration in classifier.
4. Write the configuration to CLASS.
5. Bind the EMAC or HIF to the physical interface.
6. Set the default operation mode to the physical interface.
7. Update to the CLASS DMEM at the address obtained from memory map.

For implementation details, refer to the functions *pfe_platform_bind_ifaces* and in the *pfe_platform_master.c* file within the *PFE_MCAL driver*.

Structure pfe_ct_phy_if is shared between firmware and the driver. It represents the interface as it is stored in the CLASS DMEM (as specified in the PFE memory map section).

Physical interface structure seen by PFE is as follow:

```
typedef struct __attribute__((packed, aligned(4)))
{
    /*  Pointer to head of list of logical interfaces (DMEM) */
    PFE_PTR(pfe_ct_log_if_t) log_ifs;
    /*  Pointer to default logical interface (DMEM) */
    PFE_PTR(pfe_ct_log_if_t) def_log_if;
    /*  Flags */
    pfe_ct_if_flags_t flags;
    /*  Physical port number */
    pfe_ct_phy_if_id_t id;
    /*  Operational mode */
    pfe_ct_if_op_mode_t mode;
    /*  Block state */
    pfe_ct_block_state_t block_state;
    /*  Mirroring to given port */
    PFE_PTR(pfe_ct_mirror_t) rx_mirrors[PFE_CT_MIRRORS_COUNT];
    PFE_PTR(pfe_ct_mirror_t) tx_mirrors[PFE_CT_MIRRORS_COUNT];
    /*  SPD for IPsec */
    PFE_PTR(pfe_ct_ipsec_spd_t) ipsec_spd;
```

```
    /*  Flexible Filter */
    PFE_PTR(pfe_ct_fp_table_t) filter;
    /* management interface */
    pfe_ct_phy_if_id_t mgmt_interface;
    /*  Gathered statistics */
    pfe_ct_phy_if_stats_t phy_stats __attribute__((aligned(4))); /* Must be aligned to 4 bytes */
} pfe_ct_phy_if_t;
```

List of PFE physical interfaces:

```
typedef enum __attribute__((packed))
{
    /*  HW interfaces */
    PFE_PHY_IF_ID_EMAC0 = 0U,
    PFE_PHY_IF_ID_EMAC1 = 1U,
    PFE_PHY_IF_ID_EMAC2 = 2U,
    PFE_PHY_IF_ID_HIF = 3U,
    PFE_PHY_IF_ID_HIF_NOCPY = 4U,
    /*  UTIL PE - FW internal use */
    PFE_PHY_IF_ID_UTIL = 5U,
    /*  Synthetic interfaces */
    PFE_PHY_IF_ID_HIF0 = 6U,
    PFE_PHY_IF_ID_HIF1 = 7U,
    PFE_PHY_IF_ID_HIF2 = 8U,
    PFE_PHY_IF_ID_HIF3 = 9U,
    /*  Internals */
    PFE_PHY_IF_ID_MAX = PFE_PHY_IF_ID_HIF3,
    PFE_PHY_IF_ID_INVALID
} pfe_ct_phy_if_id_t;
```

6.2 LogIf

Logical Interfaces (LOG_IFs) are attached to Physical Interfaces (PHY_IFs) and specify, based on classification rules, to what PFE port (or ports) should the ingress traffic be directed to. LOG_IFs can be added or removed, configured, enabled or disabled, at runtime. Every physical interface can have a list of associated logical interfaces. The very first logical interface in the list (tail position) is considered the default logical interface of the given physical interface. New logical interfaces are always added to the top of the list (head position), creating a sequence which is ordered from the head (the newest one) back to the tail (the default one). This ordering forms a classification sequence, which is important if the parent physical interface operates in the Flexible Router mode. The names of the default logical interface match the names of associated physical interface (e.g. EMAC0, HIF3, HIF No Copy).

Logical interfaces can be used for the following purposes:

- To forward traffic from PFE to a host processor running a PFE driver.
- To forward traffic or its replicas between physical interfaces (1:N distribution).
- To serve as classification and forwarding rules for Flexible Router.

Similar to physical interfaces, the logical interfaces can be set to a promiscuous mode. For logical interfaces, a promiscuous mode means that the logical interface accepts all the ingress traffic it is tasked to classify, regardless of the actual match rules configured on the interface.

Perform the following steps to create a new logical interface:

1. Before initializing new logical interface instance, allocate the DMEM space for logical interface (See the function *blalloc_alloc_offs in blalloc.c* file)
2. Create and initialize the logical interface structure.
3. Write to class with stats (overriding the statistics with 0).
4. Add logical interface to the physical interface and update to the CLASS DMEM.
5. Enable the promiscuous mode for logical interface and update to the CLASS DMEM.
6. Enable logical interface and update to the CLASS DMEM.

For implementation details, refer to the functions pfe_platform_create_default_log_if in the pfe_platform_master.c file within the PFE_MCAL driver.

Structure pfe_ct_log_if is shared between firmware and the driver. It represents the interface as it is stored in the CLASS DMEM.

Logical interface structure seen by PFE is as follow:

```
typedef struct __attribute__((packed, aligned(4))) pfe_ct_log_if_tag
{
/* Pointer to next logical interface in the list (DMEM) */
PFE_PTR(struct pfe_ct_log_if_tag) next;
/* List of egress physical interfaces. Bit positions correspond
to pfe_ct_phy_if_id_t values (1U << pfe_ct_phy_if_id_t). */
uint32 e_phy_ifs;
/* Flags */
pfe_ct_if_flags_t flags;
/* Match rules. Zero means that matching is disabled and packets
can be accepted on interface in promiscuous mode only. */
pfe_ct_if_m_rules_t m_rules;
/* Interface identifier */
uint8 id;
/* Operational mode */
pfe_ct_if_op_mode_t mode;
/* Reserved */
uint8 res[2];
/* Arguments required by matching rules */
pfe_ct_if_m_args_t __attribute__((aligned(4))) m_args; /* Must be aligned at 4 bytes */
/* Gathered statistics */
pfe_ct_class_algo_stats_t __attribute__((aligned(4))) class_stats; /* Must be aligned at 4 bytes */
} pfe_ct_log_if_t;
```

6.3 Tx header

The PFE Classifier is passing packets via HIF including certain metadata structures to communicate various control information. This metadata is prepended to every ethernet frame and the frame data then follows this header. Format of this metadata is subject of PFE firmware implementation and thus it creates dependency between the HIF driver software and the PFE firmware.

The PFE Classifier expects that every frame transmitted by host CPU via HIF will contain following header prepended to packet data.

Table 4. TX Header Structure

Field	Offset	Description
TX_FLAGS	7:0	Tx Frame Control Flags.

Table 4. TX Header Structure*...continued*

Field	Offset	Description
TX_QUEUE	15:8	Queue number to be used for packet transmission. Default value is zero (the lowest priority) but can be changed according to current QoS requirements. If special value 255 is used, the PFE will determine the queue number using its internal logic. The QoS can be controlled via FCI. See the *FCI Reference Guide document.*
TX_CHID	23:16	Source HIF channel ID. To be used to identify HIF channel being originator of the frame (0, 1, 2, ...).
RESERVED	47:24	Reserved.
TX_REFNUM	63:48	Reference number. In case the Egress Timestamp Report is requested (the HIF_TX_ETS flag is set) the PFE will capture the IEEE1588 timestamp and provide it back to the driver as dedicated Egress Timestamp Report frame. The PFE will include this value in the related report to allow driver to match the request with response. Upper-most 4 bits of the value must always remain 0s.
TX_EGR_PHY_IFS	95:64	Bitmask specifying set of physical interfaces where the frame shall be transmitted. Set bit indicates an interface to be used for transmission. Bit positions are given by the Physical Interface Identifiers. Value (1 « ID) corresponds to physical interface identified by the 'ID' value. Valid and taken into account only if the HIF_TX_INJECT flag is set.
TX_COOKIE	127:96	Arbitrary value which can be used by PFE firmware to route the packet. See the Flexible Router feature within the FCI Reference Guide document.

Table 5. TX Frames Flag

Field	Offset	Description
RESERVED	0:1	Reserved.
HIF_TX_ETS	2	Egress Timestamp. When set the PFE will capture IEEE1588 timestamp and send back special frame called Egress Timestamp Report containing the timestamp value.
HIF_TX_IP_CSUM	3	IP checksum offload flag. If set, then PFE will calculate and insert IP header checksum.
HIF_TX_TCP_CSUM	4	UDP checksum offload flag. If set, then PFE will calculate and insert UDP header checksum.
HIF_TX_UDP_CSUM	5	Transmit Inject flag. 0 - No inject. Let the PFE to steer the packet using PFE configuration (router/bridge/...). 1 - Inject. TX_EGR_PHY_IFS field within the TX header is valid and packet will be transmitted via physical interfaces given by the list.
HIF_TX_INJECT	6	Inter-HIF communication flag. When set the PFE firmware is optimizing the distribution path and tagging the frame with IHC flag being visible via the RX packet header when delivered to the target HIF. 0 - Standard traffic. 1 - HIF-to-HIF traffic. Priority transport. No other routing is possible.

Driver can ask the PFE to capture IEEE1588 timestamp for transmitted PTP frames. For this purpose the TX header contains dedicated flag HIF_TX_ETS. When the flag is set the PFE provides the captured timestamp in form of special RX frame (Egress Timestamp Report) which can be identified by the HIF_RX_ETS flag in the prepend header. The report has following format:

Table 6. Egress Timestamp Report (20 bytes)

Field	Offset	Description
RESERVED	63:0	Reserved.
TS_NSEC	95:64	Timestamp value. Nano-Seconds part.
TS_SEC	127:96	Timestamp value. Seconds part.
RESERVED	135:128	Reserved.
I_PHY_IF	143:136	Physical interface ID. Value corresponds to Physical Interface Identifiers and specifies physical interface which transmitted the frame and captured the timestamp.
REFNUM	159:144	Reference number. This is the TX_REFNUM value from the associated TX header intended to allow driver to match the timestamp request and response. Only lower 12 bits are valid. Other bits must be ignored.

6.4 Rx header

When a frame is received via HIF to host domain, it is prepended by following data structure:

Table 7. RX Header Structure

Field	Offset	Description
RX_PUNT_REASONS	15:0	Punt reason. These flags are being set by the PFE Classifier to indicate why the packet has been forwarded to the host CPU. Assignment is given by the PFE Firmware code. Current meaning of particular flags is described by Punt Reasons. This field is valid only if RX_FLAGS[HIF_RX_PUNT] flag is set.
RX_ING_PHY_IF	23:16	Ingress physical interface identifier set by Classifier. Every received Ethernet frame carries information about on which physical interface it has been received. Value corresponds to Physical Interface Identifiers.
RX_ING_LOG_IF	31:24	Ingress logical interface identifier set by Classifier. See the Logical Interface Identifiers.
RX_FLAGS	47:32	Rx Frame flags.
RX_QUEUE	55:48	Priority/Queue number determined by the Classifier during frame reception.
RESERVED	63:56	RESERVED
RX_TS_NSEC	95:64	Rx timestamp (nanoseconds). Only valid if HIF_RX_TS flag is set.
RX_TS_SEC	127:96	Rx timestamp (seconds). Only valid if HIF_RX_TS flag is set.

Table 8. RX Frame Flags

Field	Offset	Description
HIF_RX_IPV4_CSUM	0	IP checksum valid flag. If set, then IPv4 header checksum is valid.
HIF_RX_TCPV4_ CSUM	1	IPv4 TCP checksum valid flag. If set, then IPv4 TCP header checksum is valid.
HIF_RX_TCPV6_ CSUM	2	IPv6 TCP checksum valid flag. If set, then IPv6 TCP header checksum is valid.
HIF_RX_UDPV4_ CSUM	3	IPv4 UDP checksum valid flag. If set, then IPv4 UDP header checksum is valid.

Table 8. RX Frame Flags*...continued*

Field	Offset	Description
HIF_RX_UDPV6_ CSUM	4	IPv6 UDP checksum valid flag. If set, then IPv6 UDP header checksum is valid.
HIF_RX_PTP	5	PTP flag. 0 - Normal packet. 1 - PTP packet.
HIF_RX_PUNT	6	Punt flag. 0 - Normal Packet. 1 - Punt. The RX_PUNT_REASONS field is valid.
HIF_RX_TS	7	Rx Timestamp flag. 0 - Timestamp not present. 1 - RX_TS_NSEC and RX_TS_SEC fields are valid.
HIF_RX_IHC	8	Inter-HIF communication flag. See the HIF_TX_IHC. 0 - Standard packet. 1 - HIF-to-HIF transport packet.
HIF_RX_ETS	9	Egress timestamp report. 0 - Standard packet. 1 - Frame is Egress Timestamp Report.
RESERVED	15:10	Reserved. Shall be zero.

7 Registers summary

7.1 WSP

Table 9. WSP_FAIL_STOP_MODE_INT_EN :

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
fail_stop_int_en	0	INT_EN	Enable bit for fail stop int
fail_stop_mode_int_en	1	INT_EN	Enable bit for fail stop mode interrupt

Table 10. WSP_FAIL_STOP_MODE_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
fail_stop_mode_en	5:0	R/W	FAIL STOP MODE Enable Register: bit[0] - Parity Error enable, bit[1] - Watchdog Timer enable, bit[2] - Bus Errors enable, bit[3] - ECC multibit error enable, bit[4] - FW fail stop mode enable, bit[5] - Host Force Debug failstop enable

Table 11. WSP_ECC_ERR_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
ecc_err_int_en	0	INT_EN	Enable bit for ecc err int
ecc_multi_err_int_en	1	INT_EN	Enable bit for ecc multi bit err interrupt

Table 12. WSP_PARITY_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
parity_int_en	0	INT_EN	Enable bit for parity_int
master1_int_en	1	INT_EN	Enable bit for master1 parity error interrupt
master2_int_en	2	INT_EN	Enable bit for master2 parity error interrupt
master3_int_en	3	INT_EN	Enable bit for master3 parity error interrupt
master4_int_en	4	INT_EN	Enable bit for master4 parity error interrupt
emac_cbus_int_en	5	INT_EN	Enable bit for EMAC cbus parity error interrupt
emac_dbus_int_en	6	INT_EN	Enable bit for EMAC dbus parity error interrupt
class_cbus_int_en	7	INT_EN	Enable bit for class cbus parity error interrupt
class_dbus_int_en	8	INT_EN	Enable bit for class dbus parity error interrupt
tmu_cbus_int_en	9	INT_EN	Enable bit for TMU cbus parity error interrupt
tmu_dbus_int_en	10	INT_EN	Enable bit for TMU cbus parity error interrupt
hif_cbus_int_en	11	INT_EN	Enable bit for hif cbus parity error interrupt
hif_dbus_int_en	12	INT_EN	Enable bit for hif dbus parity error interrupt

Table 12. WSP_PARITY_INT_EN: *...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_nocpy_cbus_int_en	13	INT_EN	Enable bit for HIF_NOCPY cbus parity error interrupt
hif_nocpy_dbus_int_en	14	INT_EN	Enable bit for HIF_NOCPY dbus parity error interrupt
upe_cbus_int_en	15	INT_EN	Enable bit for util_pe cbus parity error interrupt
upe_dbus_int_en	16	INT_EN	Enable bit for util_pe dbus parity error interrupt
bridge_cbus_int_en	18	INT_EN	Enable bit for bridge cbus parity error interrupt
emac_slv_int_en	19	INT_EN	Enable bit for EMAC slave parity error interrupt
bmu1_slv_int_en	20	INT_EN	Enable bit for bmu1 slave parity error interrupt
bmu2_slv_int_en	21	INT_EN	Enable bit for bmu2 slave parity error interrupt
class_slv_int_en	22	INT_EN	Enable bit for class slave parity error interrupt
hif_slv_int_en	23	INT_EN	Enable bit for hif slave parity error interrupt
hif_nocpy_slv_int_en	24	INT_EN	Enable bit for HIF_NOCPY slave parity error interrupt
lmem_slv_int_en	25	INT_EN	Enable bit for lmem slave parity error interrupt
tmu_slv_int_en	26	INT_EN	Enable bit for TMU slave parity error interrupt
upe_slv_int_en	27	INT_EN	Enable bit for util_pe slave parity error interrupt
wsp_global_slv_int_en	28	INT_EN	Enable bit for wsp_global slave parity error interrupt
gpt1_slv_int_en	29	INT_EN	Enable bit for gpt1 slave parity error interrupt
gpt2_slv_int_en	30	INT_EN	Enable bit for gpt2 slave parity error interrupt
routemem_slv_int_en	31	INT_EN	Enable bit for route lmem slave parity error interrupt

Table 13. WSP_BUS_ERR_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
bus_err_int_en	0	INT_EN	Enable bit for bus error interrupt
m1_bus_rd_err_int_en	1	INT_EN	Enable bit for Master1 bus read error interrupt
m2_bus_wr_err_int_en	2	INT_EN	Enable bit for Master2 bus write error interrupt
m3_bus_wr_err_int_en	3	INT_EN	Enable bit for Master3 bus write error interrupt
m4_bus_rd_err_int_en	4	INT_EN	Enable bit for Master4 bus read error interrupt
hgpi_bus_rd_err_int_en	5	INT_EN	Enable bit for HGPI bus read error interrupt
hgpi_bus_wr_err_int_en	6	INT_EN	Enable bit for HGPI bus write error interrupt
egpi0_bus_rd_err_int_en	7	INT_EN	Enable bit for EMAC 0 bus read error interrupt
egpi0_bus_wr_err_int_en	8	INT_EN	Enable bit for EMAC 0 bus write error interrupt
egpi1_bus_rd_err_int_en	9	INT_EN	Enable bit for EMAC 1 bus read error interrupt
egpi1_bus_wr_err_int_en	10	INT_EN	Enable bit for EMAC 1 bus write error interrupt
egpi2_bus_rd_err_int_en	11	INT_EN	Enable bit for EMAC 2 bus read error interrupt
egpi2_bus_wr_err_int_en	12	INT_EN	Enable bit for EMAC 2 bus write error interrupt

Table 13. WSP_BUS_ERR_INT_EN:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
class_bus_rd_err_int_en	13	INT_EN	Enable bit for Class bus read error interrupt
class_bus_wr_err_int_en	14	INT_EN	Enable bit for Class bus write error interrupt
hif_nocpy_bus_rd_err_int_en	15	INT_EN	Enable bit for HIF_NOCPY bus read error interrupt
hif_nocpy_bus_wr_err_int_en	16	INT_EN	Enable bit for HIF_NOCPY bus write error interrupt
tmu_bus_rd_err_int_en	17	INT_EN	Enable bit for TMU bus read error interrupt
fet_bus_rd_err_int_en	18	INT_EN	Enable bit for FET bus read error interrupt
upe_bus_rd_err_int_en	19	INT_EN	Enable bit for Util PE bus read error interrupt
upe_bus_wr_err_int_en	20	INT_EN	Enable bit for Util PE bus write error interrupt

Table 14. WSP_FW_FAIL_STOP_MODE_INT_EN :

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
fw_fail_stop_int_en	0	INT_EN	Enable bit for fw_fail_stop_int
fw_fail_stop_mode_int_en	1	INT_EN	Enable bit for fw fail stop mode interrupt

Table 15. WSP_HOST_FORCE_DEBUG_FAIL_STOP_MODE_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
host_force_debug_fail_stop_int_en	0	INT_EN	Enable bit for host_force_debug_fail_stop_int
host_force_debug_fail_stop_mode_int_en	1	INT_EN	Enable bit for host force debug fail stop mode interrupt

Table 16. WSP_SYS_GENERIC_CONTROL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
sys_generic_control	31:0	R/W	This programmable control register bits goes as Output ports (sys_generic_control) to the EPP. Bit 31 - is used for software configuration done - not applicable for NXP, Bit 30 - is used for system soft reset, write '1' to soft reset the EPP blocks, Bit 29 - is used for bmu2 soft reset done clear, write '1' to clear the bmu2 soft reset done, Bit 28 - is used for bmu1 soft reset done clear, write '1' to clear the bmu1 soft reset done, Bit 27 - is used for soft reset done clear, write '1' to clear the soft reset done

Table 17. WSP_DBUG_BUS1: (G3 only)

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
wsp_dbg_bus2	15:0	R	Debug bus
hif_busy	16	R	hif_busy indication
utilpe_busy	17	R	utilpe_busy indication

Table 17. WSP_DBUG_BUS1: (G3 only)*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
lmem_init_done	18	R	lmem_init_done indication - lmem memory initialization done
soft_reset_done	19	R	soft_reset_done indication
bmu1_soft_reset_done	20	R	bmu1_soft_reset_done indication
bmu2_soft_reset_done	21	R	bmu2_soft_reset_done indication
class_dbus_busy	22	R	class dbus busy indication
fet_dbus_busy	23	R	fet dbus busy indication
hif_nocpy_dbus_busy	24	R	hif no copy indication
tlite_dbus_busy	25	R	tlite dbus busy indication
hgpi_dbus_busy	26	R	hgpi dbus busy indication
egpi0_dbus_busy	27	R	egpi0 dbus busy indication
egpi1_dbus_busy	28	R	egpi1 dbus busy indication
egpi2_dbus_busy	29	R	egpi2 dbus busy indication
route_lmem_init_done	30	R	route_lmem_init_done indication - route lmem memory initialization done
tmu_reclaim_init_done	31	R	rclm_init_done indication - tmu reclaim memory initialization done

7.2 Watchdog

Table 18. WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
wdt_int_en	0	INT_EN	WDT interrupt enable
wdt_bmu1_wdt_int_en	1	INT_EN	WDT interrupt enable for BMU 1
wdt_bmu2_wdt_int_en	2	INT_EN	WDT interrupt enable for BMU 2
wdt_class_wdt_int_en	3	INT_EN	WDT interrupt enable for CLASS module
wdt_emac0_gpi_wdt_int_en	4	INT_EN	WDT interrupt enable for EMAC0 GPI
wdt_emac1_gpi_wdt_int_en	5	INT_EN	WDT interrupt enable for EMAC1 GPI
wdt_emac2_gpi_wdt_int_en	6	INT_EN	WDT interrupt enable for EMAC2 GPI
wdt_hif_gpi_wdt_int_en	7	INT_EN	WDT interrupt enable for HIF GPI
wdt_hif_nocpy_wdt_int_en	8	INT_EN	WDT interrupt enable for HIF NO CPY
wdt_hif_wdt_int_en	9	INT_EN	WDT interrupt enable for HIF
wdt_tlite_wdt_int_en	10	INT_EN	WDT interrupt enable for TLITE
wdt_util_pe_wdt_int_en	11	INT_EN	WDT interrupt enable for UTIL PE module
wdt_emac0_etgpi_wdt_int_en	12	INT_EN	WDT interrupt enable for EMAC0 ETGPI
wdt_emac1_etgpi_wdt_int_en	13	INT_EN	WDT interrupt enable for EMAC1 ETGPI
wdt_emac2_etgpi_wdt_int_en	14	INT_EN	WDT interrupt enable for EMAC2 ETGPI
wdt_ext_gpt1_wdt_int_en	15	INT_EN	WDT interrupt enable for External GPT1

Table 18. WDT_INT_EN:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
wdt_ext_gpt2_wdt_int_en	16	INT_EN	WDT interrupt enable for External GPT2
wdt_lmem_wdt_int_en	17	INT_EN	WDT interrupt enable for LMEM
wdt_route_lmem_wdt_int_en	18	INT_EN	WDT interrupt enable for ROUTE LMEM

Table 19. WDT_INT_SRC:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
wdt_int	0	INT_EN	WDT interrupt monitoring. Write '1' to clear the interrupt sources.
wdt_bmu1_wdt_int	1	INT_EN	WDT interrupt monitoring for BMU 1
wdt_bmu2_wdt_int	2	INT_EN	WDT interrupt monitoring for BMU 2
wdt_class_wdt_int	3	INT_EN	WDT interrupt monitoring for CLASS module
wdt_emac0_gpi_wdt_int	4	INT_EN	WDT interrupt monitoring for EMAC0 GPI
wdt_emac1_gpi_wdt_int	5	INT_EN	WDT interrupt monitoring for EMAC1 GPI
wdt_emac2_gpi_wdt_int	6	INT_EN	WDT interrupt monitoring for EMAC2 GPI
wdt_hif_gpi_wdt_int	7	INT_EN	WDT interrupt monitoring for HIF GPI
wdt_hif_nocpy_wdt_int	8	INT_EN	WDT interrupt monitoring for HIF NO CPY
wdt_hif_wdt_int	9	INT_EN	WDT interrupt monitoring for HIF
wdt_tlite_wdt_int	10	INT_EN	WDT interrupt monitoring for TLITE
wdt_util_pe_wdt_int	11	INT_EN	WDT interrupt monitoring for UTIL PE module
wdt_emac0_etgpi_wdt_int	12	INT_EN	WDT interrupt monitoring for EMAC0 ETGPI
wdt_emac1_etgpi_wdt_int	13	INT_EN	WDT interrupt monitoring for EMAC1 ETGPI
wdt_emac2_etgpi_wdt_int	14	INT_EN	WDT interrupt monitoring for EMAC2 ETGPI
wdt_ext_gpt1_wdt_int	15	INT_EN	WDT interrupt monitoring for External GPT1
wdt_ext_gpt2_wdt_int	16	INT_EN	WDT interrupt monitoring for External GPT2
wdt_lmem_wdt_int	17	INT_EN	WDT interrupt monitoring for LMEM
wdt_route_lmem_wdt_int	18	INT_EN	WDT interrupt monitoring for ROUTE LMEM

Table 20. WDT_TIMER_VAL_UPE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
util_pe_wdt_val_top	23:0	R/W	Watch dog timer value for UTIL PE.

Table 21. WDT_TIMER_VAL_BMU:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
bmu_wdt_val_top	23:0	R/W	Watch dog timer value for BMU

Table 22. WDT_TIMER_VAL_HIF:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_wdt_val_top	23:0	R/W	Watch dog timer value for HIF

Table 23. WDT_TIMER_VAL_TLITE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
tlite_wdt_val_top	23:0	R/W	Watch dog timer value for TLITE

Table 24. WDT_TIMER_VAL_HIF_NCPY:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_nocpy_wdt_val_top	23:0	R/W	Watch dog timer value for HIF NOCPY

Table 25. WDT_TIMER_VAL_CLASS:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
class_wdt_val_top	23:0	R/W	Watch dog timer value for CLASS

Table 26. WDT_TIMER_VAL_GPI:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
gpi_wdt_val_top	23:0	R/W	Watch dog timer value for GPI

Table 27. WDT_TIMER_VAL_GPT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
ext_gpt_wdt_val_top	23:0	R/W	Watch dog timer value for GPT

Table 28. WDT_TIMER_VAL_LMEM:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
lmem_wdt_val_top	23:0	R/W	Watch dog timer value for LMEM

Table 29. WDT_TIMER_VAL_ROUTE_LMEM:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
route_lmem_wdt_val_top	23:0	R/W	Watch dog timer value for ROUTE LMEM

Table 30. CLASS_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
class_wdt_int_en	23:0	R/W	wdt_int_en for each wdt of class : bit[0] - qinq state en, bit[1] - ro state en, bit[2] - rtfet state en, bit[3] - tsq state en, bit[4] - dmi state en, bit[5] - parse state en, bit[6] - pesm state en, bit[7] - qfet state en, bit[8] - brfet state en, bit[9] - free state en, bit[10] - alloc state en, bit[11] - cbus state en, bit[12] - dbus state en, bit[13] - vlan hash debug state en, bit[14] - mac hash debug state en, bit[15] - ccu state en, bit[16] - pe0 gpt timer state en, bit[17] - pe1 gpt timer state en, bit[18] - pe2 gpt timer state en, bit[19] - pe3 gpt timer state en, bit[20] - pe4 gpt timer state en, bit[21] -pe5 gpt timer state en, bit[22] - pe6 gpt timer state en, bit[23] - pe7 gpt timer state en .

Table 31. UPE_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
util_pe_wdt_int_en	3:0	R/W	wdt_int_en for each wdt of util_pe : bit[0] - gpt timer state en, bit[1] - efet state en, bit[2] - qb state en, bit[3] - ro state en

Table 32. HGPI_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_gpi_wdt_int_en	8:0	R/W	wdt_int_en for each wdt of hif_gpi : bit[0] - tbma state en, bit[1] - rbma state en, bit[2] - dtx dxfr state en, bit[3] - drx dxfr state en, bit[4] - dtx sm state en, bit[5] - drx sm state en, bit[6] - tmlf state en, bit[7] - rmlf state en, bit[8] - dtx aseq state en

Table 33. HIF_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_wdt_int_en	3:0	R/W	wdt_int_en for each wdt of hif : bit[0] - dxr rx state en, bit[1] - bdp rx state en, bit[2] - dxr tx state en, bit[3] - bdp tx state en,

Table 34. TLITE_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
tlite_wdt_int_en	24:0	R/W	wdt_int_en for each wdt of tlite : bit[0] - TMU reclaim state en, bit[1] - phy0 enq state en, bit[2] - phy0 tdq state en, bit[3] - phy0 qos state en, bit[4] - phy0 cntx state en, bit[5] - phy1 enq state en, bit[6] - phy1 tdq state en, bit[7] - phy1 qos state en, bit[8] - phy1 cntx state en, bit[9] - phy2 enq state en, bit[10] - phy2 tdq

Table 34. TLITE_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
			state en, bit[11] - phy2 qos state en, bit[12] - phy2 cntx state en, bit[13] - phy3 enq state en, bit[14] - phy3 tdq state en, bit[15] - phy3 qos state en, bit[16] - phy3 cntx state en, bit[17] - phy4 enq state en, bit[18] - phy4 tdq state en, bit[19] - phy4 qos state en, bit[20] - phy4 cntx state en, bit[21] - phy5 enq state en, bit[22] - phy5 tdq state en, bit[23] - phy5 qos state en, bit[24] - phy5 cntx

Table 35. HNCPY_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_nocpy_wdt_int_en	5:0	R/W	wdt_int_en for each wdt of HIF_NOCPY: bit[0] - dma arb state en, bit[1] - gpi bma state en, bit[2] - mem2mem state en, bit[3] - dma arb tx state en, bit[4] - tx bdp state en, bit[5] - rx bdp state en

Table 36. BMU1_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
bmu1_wdt_int_en	3:0	R/W	wdt_int_en for each wdt of bmu1 : bit[0] - bcnt state en, bit[1] - free state en, bit[2] - alloc state en, bit[3] - rst state en

Table 37. BMU2_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
Bmu2_wdt_int_en	3:0	R/W	wdt_int_en for each wdt of bmu2 : bit[0] - bcnt state en, bit[1] - free state en, bit[2] - alloc state en, bit[3] - rst state en

Table 38. EMAC0_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
emac0_gpi_wdt_int_en	8:0	R/W	wdt_int_en for each wdt of emac0_gpi : bit[0] - tbma state en, bit[1] - rbma state en, bit[2] – dtx dxfr state en, bit[3] - drx dxfr state en, bit[4] - dtx sm state en, bit[5] - drx sm state en, bit[6] - tmlf state en, bit[7] - rmlf state en, bit[8] - dtx aseq state en

Table 38. EMAC0_WDT_INT_EN: *...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
emac0_etgpi_wdt_int_en	11:9	R/W	wdt_int_en for each wdt of emac0_etgpi: bit[9] - etgpi rbma state en, bit[10] - etgpi drx dxfr state en, bit[11] - etgpi drx sm state en

Table 39. EMAC1_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
emac1_gpi_wdt_int_en	8:0	R/W	wdt_int_en for each wdt of emac1_gpi : bit[0] - tbma state en, bit[1] - rbma state en, bit[2] - dtx dxfr state en, bit[3] - drx dxfr state en, bit[4] - dtx sm state en, bit[5] - drx sm state en, bit[6] - tmlf state en, bit[7] - rmlf state en, bit[8] - dtx aseq state en
emac1_etgpi_wdt_int_en	11:9	R/W	wdt_int_en for each wdt of emac1_etgpi : bit[9] - etgpi rbma state en, bit[10] - etgpi drx dxfr state en, bit[11] - etgpi drx sm state en

Table 40. EMAC2_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
emac2_gpi_wdt_int_en	8:0	R/W	wdt_int_en for each wdt of emac2_gpi : bit[0] - tbma state en, bit[1] - rbma state en, bit[2] - dtx dxfr state en, bit[3] - drx dxfr state en, bit[4] - dtx sm state en, bit[5] - drx sm state en, bit[6] - tmlf state en, bit[7] - rmlf state en, bit[8] - dtx aseq state en
emac2_etgpi_wdt_int_en	11:9	R/W	wdt_int_en for each wdt of emac2_etgpi : bit[9] - etgpi rbma state en, bit[10] - etgpi drx dxfr state en, bit[11] -etgpi drx sm state en

Table 41. EXT_GPT_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
ext_gpt_wdt_int_en	1:0	R/W	wdt_int_en for each wdt of ext gpt bit[0] - gpt1 en, bit[1] - gpt2 en

Table 42. LMEM_WDT_INT_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
lmem_wdt_int_en	1:0	R/W	wdt_int_en for lmem and route_lmem, bit[0] - lmem, bit[1] - route lmem

7.3 BMU

Table 43. BMU_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_en	0	R/W	BMU Enable. This signal needs to be set to enable the BMU module.

Table 43. BMU_CTRL:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_sw_rst	1	R/W	BMU Reset. This signal is used to initiate soft reset of the BMU. This is a self-clear signal. Software should only set this register and should not attempt to clear it. Soft reset is mandatory to have the internal bitmap memory cleared at reset.

Table 44. BMU_INT_ENABLE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
bmu_int_en	0	INT_EN	Master Enable for BMU interrupt
bmu_empty_int_en	1	INT_EN	Enable for the single cast buffer Empty Interrupt
bmu_full_int_en	2	INT_EN	Enable for the single cast buffer full Interrupt
bmu_thres_int_en	3	INT_EN	Enable for the single cast buffer Threshold Interrupt
bmu_free_err_int_en	4	INT_EN	Enable for the when single cast Buffer is freed again
bmu_mcast_empty_int_en	5	INT_EN	Enable for the Multi cast buffer Empty Interrupt. Reserved - Has no effect on Interrupts
bmu_mcast_full_int_en	6	INT_EN	Enable for the Multi cast buffer full Interrupt. Reserved - Has no effect on Interrupts
bmu_mcast_thres_int_en	7	INT_EN	Enable for the Multi cast buffer Threshold Interrupt. Reserved - Has no effect on Interrupts
bmu_mcast_free_int_en	8	INT_EN	Enable for the when Multi cast Buffer is freed again. Reserved - Has no effect on Interrupts

Table 45. BMU_INT_SRC:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
bmu_int	0	INT	BMU Global Interrupt Status
bmu_empty_int	1	INT	BMU Buffer Empty Interrupt. Initially BMU Empty Interrupt will be generated, incase if interrupt enables are set, then this register will hold a value of 9'h1, after reset
bmu_full_int	2	INT	BMU Buffer Full Interrupt
bmu_thres_int	3	INT	BMU Buffer Threshold Interrupt
bmu_free_err_int	4	INT	This interrupt is generated when a free is generated for an already free buffer
bmu_mcast_empty_int	5	INT	BMU MCAST Buffer Empty Interrupt. Reserved - Always reads '0'
bmu_mcast_full_int	6	INT	BMU MCAST Buffer Full Interrupt. Reserved - Always reads '0'

Table 45. BMU_INT_SRC:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
bmu_mcast_thres_int	7	INT	BMU MCAST Buffer Threshold Interrupt. Reserved - Always reads '0'
bmu_mcast_free_int	8	INT	BMU MCAST free error interrupt is generated when a free is generated for an already free buffer. Reserved - Always reads '0'

Table 46. BMU_UCAST_BASEADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_base_addr	31:0	R/W	Base address of the Memory buffer for addresses. The least significant bits till the buffer size are invalid. The base address needs to be aligned to max_buf_cnt * buf_size.

Table 47. BMU_UCAST_CONFIG:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_max_buf_cn	15:0	R/W	Maximum number of buffers that can be used. This number must be smaller than the hardware can support. After programming -BMU1 Max_buf_cnt is 16'h200 & -BMU2 Max_buf_cnt is 16'h2000.

Table 48. BMU_BUF_SIZE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_buf_size	15:0	R/W	This indicates the buffer size of each of the buffers allocated and freed. After programming, - BMU1_ BUF_SIZE is 16'h8 (2 power 8 => 256 bytes). -BMU2_BUF_SIZE is 16'hb (2 power 11 => 2K bytes). Note: BMU1 buffer size is fixed as 256bytes and BMU2 buffer size is 2Kbytes. This register value should not be changed under any circumstance.

Table 49. BMU_THRES:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_ucast_thres	15:0	R/W	Threshold of number of Uni-Cast buffers to generate an Interrupt. After programming, BMU_ THRES value for - BMU1 is 32'h0200_0200 & - BMU2 is 32'h2000_2000.
csr_bmu_mcast_thres	31:16	R/W	Threshold of number of multi-cast buffers to generate an Interrupt. This is reserved and writing to this register has no effect.

Table 50. BMU_LOW_WATERMARK:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_low_watermark	15:0	R/W	CSR BMU Low Watermark

Table 51. BMU_HIGH_WATERMARK:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_high_watermark	15:0	R/W	CSR BMU High Watermark

Table 52. BMU_INT_MEM_ACCESS_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_mem_addr	31:0	R/W	This provides access to the internal memory.

Table 53. BMU_INT_MEM_ACCESS:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
ucast_int_mem	31:0	R/W	This provides access to the internal memory. It is used only for debug purpose. Memory data [31:0].

Table 54. BMU_INT_MEM_ACCESS2:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
ucast_int_mem2	7:0	R/W	This provides access to the internal memory. It is used only for debug purpose. Memory data [39:32].

Table 55. BMU_BUF_CNT_MEM_ACCESS_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_buf_cnt_mem_addr	31:0	R/W	This provides access to the buf_cnt memory.

Table 56. BMU_BUF_CNT_MEM_ACCESS:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
bmu_buf_cnt_mem	31:0	R/W	This provides access to the buf_cnt memory. It is used only for debug purpose. Memory data [31:0].

Table 57. BMU_BUF_CNT_MEM_ACCESS2:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
bmu_buf_cnt_mem2	7:0	R/W	This provides access to the buf_cnt memory. It is used only for debug purpose. Memory data [39:32].

7.4 CLASS

Table 58. CLASS_TX_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_class_en	0	R/W	Classifier Enable bit
csr_class_swrst	1	R/W	Classifier Software reset

Table 59. CLASS_BMU1_BUF_FREE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu1_buf_free	31:0	R/W	Imem buffer free address

Table 60. CLASS_PE0_RO_DM_ADDR0:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_pe0_ro_dm_start_addr0	15:0	R/W	DMEM Address of first buffer on the Re-order side
csr_pe0_ro_dm_start_addr1	31:16	R/W	DMEM Address of second buffer on the re-order side

Table 61. CLASS_PE0_RO_DM_ADDR1:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_pe0_ro_dm_start_addr2	15:0	R/W	DMEM Address of third buffer on the Re-order side
csr_pe0_ro_dm_start_addr3	31:16	R/W	DMEM Address of fourth buffer on the re-order side

Table 62. CLASS_PE0_QB_DM_ADDR0:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_pe0_qb_dm_start_addr0	15:0	R/W	DMEM Address of first buffer on the queuing side
csr_pe0_qb_dm_start_addr1	31:16	R/W	DMEM Address of second buffer on the queuing side

Table 63. CLASS_PE0_QB_DM_ADDR1:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_pe0_qb_dm_start_addr2	15:0	R/W	DMEM Address of third buffer on the queuing side
csr_pe0_qb_dm_start_addr3	31:16	R/W	DMEM Address of fourth buffer on the queuing side

Table 64. CLASS_TM_INQ_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tm_inq_addr	31:0	R/W	Address of the Traffic Manager's input queue

Table 65. CLASS_MAX_BUF_CNT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_max_buf_cnt	5:0	R/W	Maximum buffer count for llm fifo

Table 66. CLASS_AFULL_THRES：

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_afull_thres	4:0	R/W	Threshold for llm fifo

Table 67. CLASS_INQ_AFULL_THRES：

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_inq_afull_thres	10:0	R/W	Class inq fifo is almost full threshold

Table 68. CLASS_USE_TMU_INQ：

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_use_tmu_inq_afull	0	R/W	tmu inq full

Table 69. CLASS_PE_SYS_CLK_RATIO：

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_clmode	1:0	R	Clock mode ratio for sys_clk and pe_clk. 00 means same freq and 11 means divided 4 freq. This register is made as Read only. freq, 01 means divided by 2 freq, 10 means divided by 3

Table 70. CLASS_L4_CHKSUM:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tcp_chksum_drop	0	R/W	Tcp checksum drop
csr_udp_chksum_drop	1	R/W	Udp checksum drop
csr_short_pkt_thres	8:2	R/W	Threshold for short packets
csr_ipv4_chksum_drop	9	R/W	Ipv4 checksum drop

Table 71. CLASS_HDR_SIZE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_lmem_hdr_size	5:0	R/W	LMEM Header Size. Data in the LMEM is written from this offset. The first location of LMEM is however written with the subsequent buffer address if it exists. This location is a 32 bit aligned address. If two bytes offset is required, the register in the EMAC is programmed to add two bytes.

Table 71. CLASS_HDR_SIZE:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_ddr_hdr_size	25:16	R/W	DDR Header Size. Data in the DDR is written from this offset. The subsequent buffer address if it exists is written before the offset

Table 72. CLASS_LMEM_BUF_SIZE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_lmem_buf_size	15:0	R/W	lmem buffer size

Table 73. CLASS_TPID0_TPID1:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tpid0	15:0	R/W	Vlan protocol for port0
csr_tpid1	31:16	R/W	Vlan protocol for port1

Table 74. CLASS_TPID2:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tpid2	15:0	R/W	Vlan protocol for port2

Table 75. CLASS_AXI_CTRL_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_axi_ctrl	15:0	R/W	Bit 0 csr_axi_write_done, if this bit is zero app interface does not wait for bvalid for issuing next write command, if this bit is 1 app interface waits for bvalid for issuing next comman. Bit 1 class msif axi_write_done_mas, if this bit is zero msif interface does not wait for bvalid for issuing wlast, if this bit is 1 msif interface waits for bvalid for issuing wlast Bits 3:2 - Reserved Bits 13:4 - axi dbus burst size

Table 76. CLASS_ROUTE_MULTI:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_rt_two_level_ref	0	R/W	1: Route two level reference, 0: single level reference
csr_phyno_in_hash	1	R/W	Phy no in hash calculation in route for asymmetric hash
csr_parse_route_en	3	R/W	H/W route fetch enable
csr_vlan_aware_bridge	4	R/W	Vlan aware bridge
csr_parse_bridge_en	5	R/W	H/W bridge lookup enable

Table 76. CLASS_ROUTE_MULTI:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_ipaligned_pkt	6	R/W	Sets Ip aligned packet
csr_arc_hit_check_en	7	R/W	Sets Arc hit
csr_vlan_aware_bridge_phy1	8	R/W	Port1 is aware of Vlan pkt. For future use
csr_vlan_aware_bridge_phy2	9	R/W	Port2 is aware of Vlan pkt. For future use
csr_vlan_aware_bridge_phy3	10	R/W	Port3 is aware of Vlan pkt. For future use
csr_class_toe	11	R/W	Sets TCP offload process
csr_asym_hash	13:12	R/W	Asymmetric hash calculation. 00 means normal hash, 01 means CRC on srcport, 10 means CRC on src_ip address, 11 means CRC on exclusive or of srcip, srcport
csr_sym_rtentry	14	R/W	Symmentric route entry
csr_qb2bus_endianess	15	R/W	Specifies route entry endianness access from DDR for QFET. Default it is one.
csr_len_check	16	R/W	Length check enable
csr_config_gen1	17	R/W	General config reg. For future use
csr_config_gen2	18	R/W	General config reg. For future use
csr_config_gen3	19	R/W	General config reg. For future use
csr_config_gen4	20	R/W	General config reg. For future use

Table 77. CLASS_MEM_ACCESS_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_pe_mem_addr	23:0	R/W	Internal memory address.Its an indirect R/W access from Host. Bits[23:20] indicate the processor to write the data to
csr_pe_mem_wren	27:24	R/W	Byte Enables of the Internal memory access
csr_pe_mem_cmd	31	R/W	Mem Access Command. 0 = Internal Memory Read,1 = Internal memory Write. To Access IRAM bit [18:17] must be 2'h01 and to access DRAM bit[18:17] must be 2'h10

Table 78. CLASS_MEM_ACCESS_RDATA:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_mem_access_rdata	31:0	R	Internal Memory Access Read Data. The commands are blocked at the mem_access only

Table 79. CLASS_MEM_ACCESS_WDATA:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_pe_mem_wdata	31:0	R/W	Internal Memory Access Read Data. The commands are blocked at the mem_access only

7.5 TMU

Table 80. TMU_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tmu_ctrl	31:0	R/W	Controls the reset for multiple blocks; 0 --> S/W reset of tlite module

Table 81. TMU_CNTX_ACCESS_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_cntx_access_ctrl	0	R/W	Controls the direct/indirect access to context memory, 0 --> indirect, 1 --> direct. SW should only use indirect access

Table 82. TMU_CNTX_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_cntx_ind_addr	15:0	R/W	Write the context memory address to this register. Per PHY context memory of 64 is divided to have 8 registers per Queue with Q0 registers occupying 0-7 locations and Q7 occupying 56:63. Each Queue has following registers <0:{curQ_head_ptr, curQ_tail_ptr} < 1:curQ_pkt_cnt < 2:curQ_drop_cnt 3:curQ_trans_cnt 4:{curQ_Qmax, curQ_Qmin, curQ_cfg} 5:curQ_hw_prob_cfg_tbl0 6:curQ_hw_prob_cfg_tbl1 7:curQ_dbg
csr_cntx_ind_phy_no	20:16	R/W	Phy no field

Table 83. TMU_CNTX_DATA:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_cntx_ind_data	31:0	R/W	It will have the read/write data based on the command

Table 84. TMU_CNTX_CMD:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_cntx_ind_cmd	0	R/W	0--> for read,1--> for write
csr_cntx_ind_start	1	R/W	Write 1 to start the context access
csr_cntx_ind_done	2	R/W	Context access operation done signal; poll for this bit for access completion

Table 85. TMU_PHY_QUEUE_SEL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_queue_sel	2:0	R/W	Direct Access: One has to write this register with PHY_NUM and QNUM to configure respective queue of that phy. For example, to configure hardware drop probability table of a particular queue of a particular phy, first step should be to write phy_num and queue_num to this register and then whatever is written to CFG0-CFG3 will be written in probability table of that queue for configured phy.
csr_phy_sel	12:8	R/W	Direct Access: One has to write this register with PHY_NUM and QNUM to configure respective queue of that phy. For example, to configure hardware drop probability table of a particular queue of a particular phy, first step should be to write phy_num and queue_num to this register and then whatever is written to CFG0-CFG3 will be written in probability table of that queue for configured phy.

Table 86. TMU_CURQ_PTR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_curq_ptr	31:0	R/W	Direct Access: used to configure queue for either tail drop or wred drop. For further information refer TMU spec doc.

Table 87. TMU_CURQ_PKT_CNT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_pkt_cnt	31:0	R/W	Direct Access: used to configure queue for either tail drop or wred drop. For further information refer TMU spec doc.

Table 88. TMU_CURQ_DROP_CNT :

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_drop_cnt	31:0	R/W	Direct Access: used to configure queue for either tail drop or wred drop. For further information refer TMU spec doc.

Table 89. TMU_CURQ_TRANS_CNT :

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_trans_cnt	31:0	R/W	Direct Access: used to configure queue for either tail drop or wred drop. For further information refer TMU spec doc.

Table 90. TMU_CURQ_QSTAT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_qstat	31:0	R/W	Direct Access: used to configure queue for either tail drop or wred drop. For further information refer TMU spec doc.

Table 91. TMU_HW_PROB_CFG_TBL0:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_prob_cfg_tbl0	31:0	R/W	Direct Access: used to configure queue for either tail drop or wred drop. For further information refer TMU spec doc.

Table 92. TMU_HW_PROB_CFG_TBL1:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_prob_cfg_tbl1	31:0	R/W	Direct Access: used to configure queue for either tail drop or wred drop. For further information refer TMU spec doc.

Table 93. TMU_CURQ_DEBUG:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_curq_debug	31:0	R/W	Direct Access: used to configure queue for either tail drop or wred drop. For further information refer TMU spec doc.

Table 94. TMU_PHY_INQ_PKTINFO:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_phy_inq_pktinfo	31:0	R/W	Classifier (or Host) writes packet length to this address. This will be a blocking request, if the queue is full. However, queuing discipline will clear the buffers and free them. It is essential that the master does an INCR (2) to write to this register and the previous one. It needs to be a locked access write. All masters accessing these registers in WSP perform INCR (2) Accesses 15:0 Pkt length 18 :16 Queue number 19 Reserved 22 :20 Phy number 31 :23 Reserved

Table 95. TMU_TEQ_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_llm_en	0	R/W	Enables the llm in teq module
csr_drop_efifo_full	1	R/W	Setting this bit would result in dropping of packets in TEQ, if the llm becomes full

Table 95. TMU_TEQ_CTRL:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_class_bvalid_cnt_en	2	R/W	Setting this bit would result in waiting for at least one valid from dbus2class before triggering gpi

Table 96. TMU_PHYn_TDQ_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_phyn_tdq_ctrl	31:0	R/W	0-->shp_clk_cntrl_en 1-->hw_en for tdq_sch_shaper, 3:2:--> for enabling the schedulers; 4-->allw_tdq_prog(Read only)

Table 97. TMU_PHYn_INQ_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_phyn_inq_addr	31:0	R/W	INQ address of Egress port n

Table 98. TMU_SCHn_Q_ALLOC0:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_sch0_q_alloc0	31:0	R/W	Program a value above 7 to make the input invalid. 3:0 --> que no on inp0 11:8 --> que no on inp1 19:16 --> que no on inp2 27:24 --> que no on inp3

Table 99. TMU_SCHn_Q_ALLOC1:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_sch0_q_alloc1	31:0	R/W	Program a value above 7 to make the input invalid. 3:0 --> que no on inp4 11:8 --> que no on inp5 19:16 --> que no on inp6 27:24 --> que no on inp7

Table 100. TMU_SCHn_POS:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_sch0_pos	3	R/W	Controls the o/p of sch0 position connection to sch1; valid values 0 to 7;

TMU_SHPn_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_shpn_en	0	R/W	Enable signal for shaper

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_shpn_clk_div	31:1	R/W	Clock division value at which credits will be added. 4:1--valid; clock division would be 2^(clkdiv value +1)

Table 101. TMU_SHPn_MIN_CREDIT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_shpn_min_credit	21:0	R/W	Min credit. If credit counter value is more than min credit value, then credit counter will freeze to be of min credit value

Table 102. TMU_SHPn_MAX_CREDIT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_shpn_max_credit	31:0	R/W	Max credit. If credit counter value is less than max credit value, then int_wght or frac_wght credits will be added to it.

Table 103. TMU_SHPn_CTRL2:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_shpn_bit_rate	0	R/W	Controls the data rate or pkt rate for the pure pkt length, 0--> datarate, 1 -->pktrate
csr_shpn_pos	5:1	R/W	Controls the shp n position for Qos; valid values are 0-16
csr_shpn_mode	6	R/W	0 -->Bursty mode 1 -->Zero-burst

Table 104. TMU_SCHn_BIT_RATE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_schn_bit_rate	0	R/W	Controls the data rate or pkt rate for the pure pkt length, 0--> datarate, 1 -->pktrate

Table 105. TMU_BMU_INQ_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_bmu_inq_addr	31:0	R/W	Address of BMU, where buffer should be freed in case of drop.

Table 106. TMU_AFULL_THRES:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_afull_thres	15:0	R/W	Controls the LLM aful threshold. As of now not used.

Table 107. TMU_INQ_WATERMARK:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_inq_watermark	9:0	R/W	Threshold point above which, RTL shows INQ fifo is full.

Table 108. TMU_PHYn_TDQ_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_phy0_tdq_ctrl	31:0	R/W	0-->shp_clk_cntrl_en 1-->hw_en for tdq_sch_shaper, 3:2-->for enabling the schedulers 4-->allw_tdq_prog(Read only)

7.6 GPI

Table 109. GPI_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_gpi_en	0	R/W	GPI Enable. This signal needs to be set to enable the gpi module
csr_gpi_swrs	1	R/W	GPI Reset. This signal is used to initiate soft reset of the gpi. This is a self-clear signal. Software should only set this register and should not attempt to clear it.

Table 110. GPI_PORT_SHPn_MIN_CREDIT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_port_shp0_min_credit	21:0	R/W	Shaper and min weight

Table 111. GPI_EMAC_1588_TIMESTAMP_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_emac_1588_timestamp_en	0	R/W	This is for 1588 timestamp enable
csr_emac_one_step_timestamp_en	1	R/W	This is enable bit for One-Step Timestamp
csr_emac_txqueueflush	2	R/W	Tx queue flush enable
csr_emac_rxpktflush	3	R/W	Rx pkt flush enable
csr_emac_crc_pad_ctrl	5:4	R/W	CRC pad control 2'b00: CRC and Pad Insertion 2'b01: CRC Insertion

Table 111. GPI_EMAC_1588_TIMESTAMP_EN:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
			(Disable Pad Insertion) 2'b10: Disable CRC Insertion 2'b11: CRC Replacement
csr_emac_tx_pbl	15:9	R/W	Transmit pbl
csr_emac_rx_pbl	22:16	R/W	Receive pbl

Table 112. GPI_RX_CONFIG:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_lmem_buf_en	0	R/W	Enable for LMEM buffer. If this bit is set, the hardware fetches the LMEM for the first buffer. If the Enable for the DDR buffer is not set, the data will be fetched only from LMEM for first and subsequent buffers too. For NPU all the buffers are LMEM enabled.
csr_ddr_buf_en	1	R/W	Enable for DDR Buffer. If the enable for LMEM is set, this will be the second buffer fetched. Otherwise, this will be the only buffer fetched. If the packet size is larger than the DDR BUF size, multiple buffers are fetched and chained together. For NPU DDR buffers are not enabled.
csr_dis_rmlf_mask_wd	15	R/W	Disable rmlf mask wdt when the rmlf state is in HUNTEOF state.
csr_alloc_retry_cycles	25:16	R/W	This register has the number of system clock cycles, the state machine has to wait before retrying incase the buffers are full at the buffer manager.

Table 113. GPI_HDR_SIZE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_lmem_hdr_size	7:0	R/W	LMEM Header Size. Data in the LMEM is written from this offset. The first location of LMEM is written with the subsequent buffer address if it exists. This location is a 32 bit aligned address.
csr_ddr_hdr_size	25:16	R/W	DDR Header Size. Data in the DDR is written from this offset. The subsequent buffer address if it exists is written before the offset

Table 114. GPI_BUF_SIZE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_lmem_bufsize	15:0	R/W	Buffer size of an LMEM buffer
csr_ddr_bufsize	31:16	R/W	Buffer size of an DDR buffer

Table 115. GPI_LMEM_ALLOC_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_lmem_bm_alloc_addr	31:0	R/W	Address Location within the BMU to fetch an LMEM buffer

Table 116. GPI_LMEM_FREE_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_lmem_bm_free_addr	31:0	R/W	Address Location within the BMU to free an LMEM buffer

Table 117. GPI_DDR_ALLOC_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_ddr_bm_alloc_addr	31:0	R/W	Address Location within the BMU to fetch a DDR buffer

Table 118. GPI_DDR_FREE_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_ddr_bm_alloc_addr	31:0	R/W	Address Location within the BMU to free a DDR buffer

Table 119. GPI_CLASS_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_class_addr	31:0	R/W	Address of the location in the Classifier where packet from peripherals is sent to

Table 120. GPI_DDR_DATA_OFFSET:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_ddr_data_offset	9:0	R/W	CSR DDR data offset

Table 121. GPI_LMEM_DATA_OFFSET:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_lmem_data_offset	7:0	R/W	CSR LMEM data offset

Table 122. GPI_TMLF_TX:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_gpi_tmlf_txthres	15:0	R/W	Threshold number of TMLF words (64bit size) to be in the TMLF FIFO before transmission starts

Table 123. GPI_DTX_ASEQ:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_gpi_dtx_aseq_len	7:0	R/W	Length of action sequence. This register is used for prefetching the action sequence data bytes from the DDR. This is mainly to specify the first burst size of first read transaction of the packet. To support short packets less than 64 bytes, check sum calculation. This needs to be configured as 0x40 bytes. If GPI based checksums are not used. This field has to be kept as 0x50 only

Table 124. GPI_CSR_TOE_CHKSUM_EN:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_gpi_toe_chksum_en	0	R/W	csr_gpi_toe_chksum_en is enabled, the hardware module in gpi_dtx_aseq calculates ip/tcp/udp checks and updated the packet

Table 125. GPI_CSR_AXI_WRITE_DONE_ADDR:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_axi_write_done	15:0	R/W	If bit0 axi_write_done is zero, app interface does not wait for bvalid for issuing next write command, if this bit is 1 app interface waits for bvalid for issuing next command for bit1 axi_write_done_mas, if this bit is zero msif interface does not wait for bvalid for issuing wlast, if this bit is 1 msif interface waits for valid for issuing last Bits 15:3 – Reserved

7.7 IGQOS

Table 126. CSR_IGQOS_ENTRY_DATA_REGn:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_entry_data_reg	31:0	R/W	Entry memory register n

Table 127. CSR_IGQOS_ENTRY_CMDCNTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_entry_cmdcntrl	31:0	R/W	Entry command register, used to write into Entry memory and LRU memory. Bit 7:0 - Command, 8'h1 is write command 8'h2 - read command Bit 14:8 - Table address Bit 16th - Table selects. If set, LRU table is selected, else entry table selected.

Table 128. CSR_IGQOS_ENTRY_CMDSTATUS:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_entry_cmdstatus	31:0	R	Entry status register. Bit 0 is set, once command is completed. It is read on clear register.

Table 129. CSR_IGQOS_CONTROL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_control	15:0	R/W	Bit 0 - Ingress QOS total module enable. If this bit is not set, ingress qos is bypassed.

Table 130. SR_IQGOS_DMEMQ_ZONE_PROB:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_dmemq_zone1_prob	3:0	R/W	DMEM queue zone1 probabilities
csr_igqos_dmemq_zone2_prob	7:4	R/W	DMEM queue zone2 probabilities
csr_igqos_dmemq_zone3_prob	11:8	R/W	DMEM queue zone3 probabilities
csr_igqos_dmemq_zone4_prob	15:12	R/W	DMEM queue zone4 probabilities
csr_igqos_dmemq_weight	31:16	R/W	DMEM queue weight

Table 131. CSR_IGQOS_LMEMQ_ZONE_PROB:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_lmemq_zone1_prob	3:0	R/W	LMEM queue zone1 probabilities
csr_igqos_lmemq_zone2_prob	7:4	R/W	LMEM queue zone2 probabilities
csr_igqos_lmemq_zone3_prob	11:8	R/W	DMEM queue zone3 probabilities
csr_igqos_lmemq_zone4_prob	15:12	R/W	LMEM queue zone4 probabilities
csr_igqos_lmemq_weight	31:16	R/W	LMEM queue weight

Table 132. CSR_IGQOS_RXFQ_ZONE_PROB:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_rxfq_zone1_prob	3:0	R/W	RXF FIFO queue zone1 probabilities
csr_igqos_rxfq_zone2_prob	7:4	R/W	RXF FIFO queue zone2 probabilities
csr_igqos_rxfq_zone3_prob	11:8	R/W	RXF FIFO queue zone3 probabilities
csr_igqos_rxfq_zone4_prob	15:12	R/W	RXF FIFO queue zone4 probabilities
csr_igqos_rxfq_weight	31:16	R/W	RXF queue weight

Table 133. CSR_IGQOS_DMEMQ_FULL_THRESH:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_dmemqfull_thresh	15:0	R/W	DMEM queue full threshold, when dmem queue is greater than the full threshold, dmem queue full condition is generated and all the packets will be dropped.

Table 134. CSR_IGQOS_DMEMQ_DROP_THRESH:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_dmemqdrop_max_thresh	15:0	R/W	Maximum Drop threshold for DMEM queue. When dmem queue crosses the Max threshold, only reserve packets are accepted other dropped.
csr_igqos_dmemqdrop_min_thresh	31:16	R/W	Minimum Drop threshold for DMEM queue. When DMEM queue is greater than, minimum threshold only unmagement frames based on probability is dropped other packets are accepted.

Table 135. CSR_IGQOS_LMEMQ_FULL_THRESH:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_lmemqfull_thresh	15:0	R/W	LMEM queue full threshold. When lmem queue is greater than the full threshold, lmem queue full condition is generated and all the packets will be dropped. This value should be programmed with equal to value csr_bmu_max_buf_cnt.

Table 136. CSR_IGQOS_LMEMQ_DROP_THRESH:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_lmemqdrop_max_thresh	15:0	R/W	Maximum Drop threshold for LMEM queue. When lmem queue crosses the Max threshold, only reserve packets are accepted other dropped.
csr_igqos_lmemqdrop_min_thresh	31:16	R/W	Minimum Drop threshold for LMEM queue. When LMEM queue is greater than, minimum threshold only unmagement frames based on probability is dropped other packets are accepted.

Table 137. CSR_IGQOS_RXFQ_FULL_THRESH:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_rxfqfull_thresh	15:0	R/W	RXF FIFO queue full threshold. when RXF FIFO queue is greater than the full threshold, RXF FIFO queue full condition is generated, and all the packets will be dropped. This value should be programmed with equal to value csr_bmu_max_buf_cnt.

Table 138. CSR_IGQOS_RXFQ_DROP_THRESH:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_rxfqdrop_max_thresh	15:0	R/W	Maximum Drop threshold for RXF FIFO queue. When RXF FIFO queue crosses the Max threshold, only reserve packets are accepted other dropped.
csr_igqos_rxfqdrop_min_thresh	31:16	R/W	Minimum Drop threshold for LMEM queue. When LMEM queue is greater than, minimum threshold

Table 138. CSR_IGQOS_RXFQ_DROP_THRESH:*...continued*

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
			only unmagement frames based on probability is dropped other packets are accepted.

Table 139. CSR_IGQOS_PORT_SHP_WGHT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_port_shp_wght	31:0	R/W	Ingress qos, shaper weight register. Bits 7:0 frag wght Bits 10:8 Integer wght Bits 31:11 - Reserved

Table 140. GPI_PORT_SHP_MIN_CREDIT:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_port_shp0_min_credit	21:0	R/W	Shaper min weight

Table 141. CSR_IGQOS_PORT_SHP_CONFIG:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_port_shp_config	7:0	R/W	Ingress qos, shape config register. Bit 0 value of 0 shaper0 is - Shaper is bps, value of 1 Shaper0 is pps. Bit 1 value of 0 Shaper1 is bps, value of 1 Shaper1 is pps, Bits [3:2] value 00 Shaper0 is for port level data rate, value 01 Shaper0 is for BCAST Packets, value 10 Shaper0 is for MCAST Packets. Bits [6:4] value 00 Shaper1 is for port level data rate value 01 Shaper1 is for BCAST Packets value 10 Shaper1 is for MCAST Packets
csr_igqos_port_shp_ifg	15:8	R/W	IFG between packets. Used In ingress shaper

Table 142. CSR_IGQOS_PORT_SHP_CTRL:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_port_shp_ctrl	31:0	R/W	Ingress qos, shaper control register. Bit 0 Shaper Enable Bits 16:1 CLKDIV factor. Bits 31:17 - Reserved

Table 143. CSR_IGQOS_TPID:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_tpid	31:0	R/W	TPID for VLAN packets 15: 0 - TPID0 31:16 - TPID1

Table 144. CSR_IGQOS_CLASS:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_class	7:0	R/W	Bit0 - reserved bit1 - tcp checksum drop enable bit2 - udp checksum drop enable bit3 - ipv4 checksum drop enable bit4-tpid0 enable bit5-tpid1 enable bit6 - lru enable bit7-entry size 128

Table 145. CSR_IGQOS_LRU_TIMER_VALUE:

FIELD	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_igqos_lru_timer_value	31:0	R/W	lru timer value that is compared with the lru timer to generate timeout event.

7.8 HIF

Table 146. HIF_MISC:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_seq_num_check_en	0	R/W	Sequence number check enable/disable. When enabled, the HIF if checking sequence number provided with BD against internal reference and in case it does not follow the sequential order the BD is considered to be invalid. 1 - Enable sequence number checking of fetched BDs. 0 - Disable sequence number checking of fetched BDs.
csr_bdprd_axi_write_done	1	R/W	If write_done is set to 1, new request will not be acknowledged until old request is written out. If write done is set to 0, request will be accepted once data is written to internal FIFO without really going out for buffer descriptor read.
csr_bdpwr_axi_write_done	2	R/W	If write_done is set to 1, new request will not be acknowledged until old request is written out. If write done is set to 0, request will be accepted once data is written to internal FIFO without really going out for buffer descriptor write.
csr_rxdxr_axi_write_done	3	R/W	If write_done is set to 1, new request will not be acknowledged until old request is written out. If write done is set to 0, request will be accepted once data is written to internal FIFO without really going out for data write.

Table 147. HIF_SOFT_RESET:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
sys_sw_reset_rx_path	0	R/W	Soft reset for RX path.
sys_sw_reset_tx_path	1	R/W	Soft reset for TX path.
sys_sw_reset_tx_rx_path	2	R/W	Soft reset.
csr_sw_reset	3	R/W	Soft reset for csr.

Table 148. HIF_ERR_INT_SRC:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_err_int	0	INT	'1' indicates that an interrupt from HIF block is pending. This value is an OR of all error interrupts in HIF block.
dxr_csr_tx_pkt_len_err_int	1	INT	HIF DXR TX Packet length error. It happens when the packet length is greater than packet length mentioned in BD.
dxr_csr_tx_sof_err_in	2	INT	HIF DXR TX SOF error. It happens when SOF is followed by SOF with no EOF (app side).
dxr_csr_tx_data_err_int	3	INT	HIF DXR TX Data error. It happens When we EOF is followed by data with no SOF in between (app side).
dxr_csr_tx_eof_err_int	4	INT	HIF DXR TX EOF error. It happens when EOF is followed by EOF with no SOF (app side).
dxr_csr_rx_pkt_len_err_int	5	INT	HIF DXR RX Packet length error. It happens when the packet length is greater than packet length mentioned in BD.
dxr_csr_rx_sof_err_int	6	INT	HIF DXR RX SOF error. It happens when SOF is followed by SOF with no EOF (app side).
dxr_csr_rx_data_err_int	7	INT	HIF DXR RX Data error. It happens when EOF is followed by data with no SOF in between (app side).
dxr_csr_rx_eof_err_int	8	INT	HIF DXR RX EOF error. It happens when EOF is followed by EOF with no SOF (app side).
bdp_csr_tx_rdaxi_err_int	9	INT	HIF DXR RX EOF error. It happens when EOF is followed by EOF with no SOF (app side).
bdp_csr_tx_wr_axi_err_int	10	INT	HIF BDP TX RD AXI error. It happens when TX BDP read block sends request to AXI with zero as data count.
bdp_csr_rx_rd_axi_err_int	11	INT	HIF BDP RX RD AXI error. It happens when RX BDP read block sends request to AXI with zero as data count.
bdp_csr_rx_wr_axi_err_int	12	INT	HIF BDP RX WR AXI error. It happens when RX BDP write block sends request to AXI with zero as data count.
dxr_csr_tx_axi_err_int	13	INT	HIF DXR TX AXI error. It happens when TX DXR block sends request to AXI with zero as data count.
dxr_csr_rx_axi_err_int	14	INT	HIF DXR RX AXI error. It happens when RX DXR block sends request to AXI with zero as data count.

Table 149. HIF_ERR_INT_EN:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_err_int_en	0	R/W	Master Enable bit for HIF ERR interrupt.
dxr_csr_tx_pkt_len_err_int_en	1	R/W	Enable bit for DXR TX packet length error interrupt.
dxr_csr_tx_sof_err_int_en	2	R/W	Enable bit for DXR TX SOF error interrupt.

Table 149. HIF_ERR_INT_EN:*...continued*

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
dxr_csr_tx_data_err_int_en	3	R/W	Enable bit for DXR TX Data error interrupt.
dxr_csr_tx_eof_err_int_en	4	R/W	Enable bit for DXR TX EOF error interrupt.
dxr_csr_rx_pkt_len_err_int_en	5	R/W	Enable bit for DXR RX packet length error interrupt.
dxr_csr_rx_sof_err_int_en	6	R/W	Enable bit for DXR RX SOF error interrupt
dxr_csr_rx_data_err_int_en	7	R/W	Enable bit for DXR RX Data error interrupt.
dxr_csr_rx_eof_err_int_en	8	R/W	Enable bit for DXR RX EOF error interrupt.
bdp_csr_tx_rd_axi_err_int_en	9	R/W	Enable bit for TX BDP read AXI error interrupt.
bdp_csr_tx_wr_axi_err_int_en	10	R/W	Enable bit for TX BDP write AXI error interrupt.
bdp_csr_rx_rd_axi_err_int_en	11	R/W	Enable bit for RX BDP read AXI error interrupt.
bdp_csr_rx_wr_axi_err_int_en	12	R/W	Enable bit for RX BDP write AXI error interrupt.
dxr_csr_tx_axi_err_int_en	13	R/W	Enable bit for TX DXR AXI error interrupt.
dxr_csr_rx_axi_err_int_en	14	R/W	Enable bit for RX DXR AXI error interrupt.

Table 150. HIF_TX_FIFO_ERR_INT_SRC:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_tx_fifo_err_int	0	INT	’1’ indicates that an interrupt from HIF block is pending. This value is an OR of all FIFO error interrupts in HIF block for TX.
bdp_csr_tx_rd_fifo_overrun_int	1	INT	TX BD read FIFO overrun
bdp_csr_tx_wr_fifo_overrun_int	2	INT	TX BD write FIFO overrun
dxr_csr_tx_fifo_overrun_int	3	INT	TX data FIFO overrun
dxr_csr_tx_lbuf_overrun_int	4	INT	TX data lbuf FIFO overrun
dxr_csr_tx_sof_ctrl_word_fifo_ overrun_int	5	INT	TX SOF control word FIFO overrun
bdp_dxr_csr_tx_bd_ctrl_fifo_ overrun_int	6	INT	TX BD control FIFO overrun
dxr_csr_tx_sad_fifo_overrun_int	7	INT	TX SAD FIFO overrun
bdp_csr_tx_bvalid_fifo_overrun_int	8	INT	TX bvalid FIFO overrun
hif_axi_bdp_csr_tx_bvalid_fifo_ overrun_int	9	INT	TX HIF AXI bvalid FIFO overrun
bdp_csr_tx_rd_fifo_underrun_int	10	INT	TX BD read FIFO underrun
bdp_csr_tx_wr_fifo_underrun_int	11	INT	TX BD write FIFO underrun
dxr_csr_tx_fifo_underrun_int	12	INT	TX data FIFO underrun
dxr_csr_tx_lbuf_underrun_int	13	INT	TX data lbuf FIFO underrun
dxr_csr_tx_sof_ctrl_word_fifo_ underrun_int	14	INT	TX SOF control word FIFO underrun
bdp_dxr_csr_tx_bd_ctrl_fifo_ underrun_int	15	INT	TX BD control FIFO underrun

Table 150. HIF_TX_FIFO_ERR_INT_SRC:*...continued*

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
dxr_csr_tx_sad_fifo_underrun_int	16	INT	TX SAD FIFO underrun
bdp_csr_tx_bvalid_fifo_underrun_int	17	INT	TX bvalid FIFO underrun
hif_axi_bdp_csr_tx_bvalid_fifo_underrun_int	18	INT	TX HIF AXI bvalid FIFO underrun

Table 151. HIF_TX_FIFO_ERR_INT_EN:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_tx_fifo_err_int_en	0	R/W	Master Enable bit for HIF TX FIFO ERR interrupt
bdp_csr_tx_rd_fifo_overrun_int_en	1	R/W	Enable bit for TX BD read FIFO overrun
bdp_csr_tx_wr_fifo_overrun_int_en	2	R/W	Enable bit for TX BD write FIFO overrun
dxr_csr_tx_fifo_overrun_int_en	3	R/W	Enable bit for TX data FIFO overrun
dxr_csr_tx_lbuf_overrun_int_en	4	R/W	Enable bit for TX data lbuf FIFO overrun
dxr_csr_tx_sof_ctrl_word_fifo_overrun_int_en	5	R/W	Enable bit for TX SOF control word FIFO overrun
bdp_dxr_csr_tx_bd_ctrl_fifo_overrun_int_en	6	R/W	Enable bit for TX BD control FIFO overrun
dxr_csr_tx_sad_fifo_overrun_int_en	7	R/W	Enable bit for TX SAD FIFO overrun
bdp_csr_tx_bvalid_fifo_overrun_int_en	8	R/W	Enable bit for TX bvalid FIFO overrun
hif_axi_bdp_csr_tx_bvalid_fifo_overrun_int_en	9	R/W	Enable bit for TX HIF AXI bvalid FIFO overrun
bdp_csr_tx_rd_fifo_underrun_int_en	10	R/W	Enable bit for TX BD read FIFO underrun
bdp_csr_tx_wr_fifo_underrun_int_en	11	R/W	Enable bit for TX BD write FIFO underrun
dxr_csr_tx_fifo_underrun_int_en	12	R/W	Enable bit for TX data FIFO underrun
dxr_csr_tx_lbuf_underrun_int_en	13	R/W	Enable bit for TX data lbuf FIFO underrun
dxr_csr_tx_sof_ctrl_word_fifo_underrun_int_en	14	R/W	Enable bit for TX SOF control word FIFO underrun
bdp_dxr_csr_tx_bd_ctrl_fifo_underrun_int_en	15	R/W	Enable bit for TX BD control FIFO underrun
dxr_csr_tx_sad_fifo_underrun_int_en	16	R/W	Enable bit for TX SAD FIFO underrun.
bdp_csr_tx_bvalid_fifo_underrun_int_en	17	R/W	Enable bit for TX bvalid FIFO underrun
hif_axi_bdp_csr_tx_bvalid_fifo_underrun_int_en	18	R/W	Enable bit for TX HIF AXI bvalid FIFO underrun

Table 152. HIF_RX_FIFO_ERR_INT_SRC:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_rx_fifo_err_int	0	INT	’1’ indicates that an interrupt from HIF block is pending. This value is an OR of all FIFO error interrupts in HIF block for RX.
bdp_csr_rx_rd_fifo_overrun_int	1	INT	RX BD read FIFO overrun.
bdp_csr_rx_wr_fifo_overrun_int	2	INT	RX BD write FIFO overrun.
dxr_csr_rx_fifo_overrun_int	3	INT	RX data FIFO overrun.
dxr_csr_rx_lbuf_overrun_int	4	INT	RX data lbuf FIFO overrun.
dxr_csr_rx_sof_ctrl_word_fifo_ overrun_int	5	INT	RX SOF control word FIFO overrun.
bdp_dxr_csr_rx_bd_ctrl_fifo_ overrun_int	6	INT	RX BD control FIFO overrun.
dxr_csr_rx_sad_fifo_overrun_int	7	INT	RX SAD FIFO overrun.
bdp_csr_rx_bvalid_fifo_overrun_int	8	INT	RX bvalid FIFO overrun.
hif_axi_bdp_csr_rx_bvalid_fifo_ overrun_int	9	INT	RX HIF AXI bvalid FIFO overrun.
bdp_csr_rx_rd_fifo_underrun_int	10	INT	RX BD read FIFO underrun
bdp_csr_rx_wr_fifo_underrun_int	11	INT	RX BD write FIFO underrun
dxr_csr_rx_fifo_underrun_int	12	INT	RX data FIFO underrun.
dxr_csr_rx_lbuf_underrun_int	13	INT	RX data lbuf FIFO underrun.
dxr_csr_rx_sof_ctrl_word_fifo_ underrun_int	14	INT	RX SOF control word FIFO underrun.
bdp_dxr_csr_rx_bd_ctrl_fifo_ underrun_int	15	INT	RX BD control FIFO underrun.
dxr_csr_rx_sad_fifo_underrun_int	16	INT	RX SAD FIFO underrun.
bdp_csr_rx_bvalid_fifo_underrun_int	17	INT	RX bvalid FIFO underrun.
hif_axi_bdp_csr_rx_bvalid_fifo_ underrun_int	18	INT	RX HIF AXI bvalid FIFO underrun.

Table 153. HIF_RX_FIFO_ERR_INT_EN:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_rx_fifo_err_int _en	0	R/W	Master Enable bit for HIF RX FIFO ERR interrupt
bdp_csr_rx_rd_fifo_overrun_int _en	1	R/W	Enable bit for RX BD read FIFO overrun.
bdp_csr_rx_wr_fifo_overrun_int _en	2	R/W	Enable bit for RX BD write FIFO overrun.
dxr_csr_rx_fifo_overrun_int _en	3	R/W	Enable bit for RX data FIFO overrun.
dxr_csr_rx_lbuf_overrun_int _en	4	R/W	Enable bit for RX data lbuf FIFO overrun.
dxr_csr_rx_sof_ctrl_word_fifo_ overrun_int _en	5	R/W	Enable bit for RX SOF control word FIFO overrun.
bdp_dxr_csr_rx_bd_ctrl_fifo_ overrun_int _en	6	R/W	Enable bit for RX BD control FIFO overrun.

Table 153. HIF_RX_FIFO_ERR_INT_EN:*...continued*

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
dxr_csr_rx_sad_fifo_overrun_int _en	7	R/W	Enable bit for RX SAD FIFO overrun.
bdp_csr_rx_bvalid_fifo_overrun_int _en	8	R/W	Enable bit for RX bvalid FIFO overrun.
hif_axi_bdp_csr_rx_bvalid_fifo_ overrun_int _en	9	R/W	Enable bit for RX HIF AXI bvalid FIFO overrun.
bdp_csr_rx_rd_fifo_underrun_int _en	10	R/W	Enable bit for RX BD read FIFO underrun
bdp_csr_rx_wr_fifo_underrun_int _en	11	R/W	Enable bit for RX BD write FIFO underrun
dxr_csr_rx_fifo_underrun_int _en	12	R/W	Enable bit for RX data FIFO underrun.
dxr_csr_rx_lbuf_underrun_int _en	13	R/W	Enable bit for RX data lbuf FIFO underrun.
dxr_csr_rx_sof_ctrl_word_fifo_ underrun_int _en	14	R/W	Enable bit for RX SOF control word FIFO underrun.
bdp_dxr_csr_rx_bd_ctrl_fifo_ underrun_int _en	15	R/W	Enable bit for RX BD control FIFO underrun.
dxr_csr_rx_sad_fifo_underrun_int _en	16	R/W	Enable bit for RX SAD FIFO underrun.
bdp_csr_rx_bvalid_fifo_underrun_int _en	17	R/W	Enable bit for RX bvalid FIFO underrun.
hif_axi_bdp_csr_rx_bvalid_fifo_ underrun_int _en	18	R/W	Enable bit for RX HIF AXI bvalid FIFO underrun.

Table 154. HIF_RX_BDP_RD_LOW_ADDR_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_rx_bdp_rd_bd_low_addr_chn_ out	31:0	R/W	Lower part of base address of the RX BD Ring. This register must be updated only when the RX DMA is IDLE.

Table 155. HIF_RX_BDP_RD_HIGH_ADDR_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_rx_bdp_rd_bd_high_addr_chn_ out	31:0	R/W	Upper part of base address of the RX BD Ring. This register must be updated only when the RX DMA is IDLE.

Table 156. HIF_TX_BDP_RD_LOW_ADDR_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tx_bdp_rd_bd_low_addr_chn_ out	31:0	R/W	Lower part of base address of the TX BD Ring. This register must be updated only when the RX DMA is IDLE.

Table 157. HIF_TX_BDP_RD_HIGH_ADDR_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tx_bdp_rd_ bd_high_addr_chn_ out	31:0	R/W	Upper part of base address of the TX BD Ring. This register must be updated only when the RX DMA is IDLE.

Table 158. HIF_RX_BDP_WR_LOW_ADDR_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_rx_bdp_wr_bd_low_addr_chn_ out	31:0	R/W	Lower part of base address of the RX WB BD Ring. This register must be updated only when the RX DMA is IDLE

Table 159. HIF_RX_BDP_WR_HIGH_ADDR_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_rx_bdp_wr_bd_high_addr_chn_ out	31:0	R/W	Upper part of base address of the RX WB BD Ring. This register must be update only when the RX DMA is IDLE.

Table 160. HIF_TX_BDP_WR_LOW_ADDR_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tx_bdp_wr_bd_low_addr_chn_ out	31:0	R/W	Lower part of base address of the TX WB BD Ring. This register must be updated only when the TX DMA is IDLE.

Table 161. HIF_TX_BDP_WR_HIGH_ADDR_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tx_bdp_wr_bd_high_addr_chn_ out	31:0	R/W	Upper part of base address of the TX WB BD Ring. This register must be updated only when the TX DMA is IDLE

Table 162. HIF_RX_WRBK_BD_CHn_BUFFER_SIZE:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_rx_wr_bck_bd_chn_buffer_size	15:0	R/W	RX write-back BD buffer size to wrap around. It is the number of write-back buffer descriptors in RX WB BD Ring.

Table 163. HIF_TX_WRBK_BD_CHn_BUFFER_SIZE:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tx_wr_bck_bd_chn_buffer_size	15:0	R/W	TX write-back BD buffer size to wrap around. It is the number of write-back buffer descriptors in TX WB BD Ring.

Table 164. HIF_TX_POLL_CTRL:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tx_bdp_poll_cntr1	15:0	R/W	Number of cycles that HIF TX BDP block would wait before re-fetching the BD control word if the previously fetched BD was disabled.
csr_tx_bdp_poll_cntr2	31:16	R/W	Number of cycles that HIF TX BDP block would wait before writing updated BD back to system memory when number of BDs are less than threshold value.

Table 165. HIF_RX_POLL_CTRL:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_rx_bdp_poll_cntr1	15:0	R/W	Number of cycles that HIF RX BDP block would wait before re-fetching the BD control word if the previously fetched BD was disabled.
csr_rx_bdp_poll_cntr2	31:16	R/W	Number of cycles that HIF RX BDP block would wait before writing updated BD back to system memory when number of BDs are less than threshold value.

Table 166. HIF_TIMEOUT_REG:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_hif_timeout_val	31:0	R/W	Timeout value for HIF when timeout is enabled in HIF_MISC in number of system clock ticks (pfe_sys_clk).

Table 167. HIF_RX_QUEUE_MAP_CH_NO_ADDR:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_hif_rx_queue_map_ch_no	3:0	R/W	TMU queue mapping to HIF RX channels1. When PFE sends frame to TMU Queue 0 it is forwarded to HIF channel given by this bitfield. HIF RX channel number mapped to TMU Queue 0.
	7:4	R/W	HIF RX channel number mapped to TMU Queue 1.
	11:8	R/W	HIF RX channel number mapped to TMU Queue 2.
	15:12	R/W	HIF RX channel number mapped to TMU Queue 3.
	19:16	R/W	HIF RX channel number mapped to TMU Queue 4.
	23:20	R/W	HIF RX channel number mapped to TMU Queue 5.
	27:24	R/W	HIF RX channel number mapped to TMU Queue 6.
	31:28	R/W	HIF RX channel number mapped to TMU Queue 7.

Table 168. HIF_DMA_BURST_SIZE:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_dma_burst_size	2:0	R/W	DMA burst size in number of bytes. 00 - 128 01 - 256 10 - 512 11 - 1024 Default: 128 bytes

Table 169. HIF_DMA_BASE_ADDR:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_dma_base_addr	31:0	R/W	The upper 32 bits of BD or data buffer address. This address will be concatenated with BD or data buffer address resulting in 64-bit addressing capability.

Table 170. HIF_LTC_PKT_CTRL_ADDR:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_hif_tx_ltc_pkt_flow_en	0	R/W	HIF TX packet flow through Launch time control (LTC) module flow enable. When this bit is enabled, HIF TX will accept maximum "csr_hif_tx_ltc_max_pkt_cnt_per_ch" Packets after that wait for LTC read enable signal to fetch next packet else packets will be accepted by HIF TX till it has space in internal memory.

Table 171. HIF_CHn_INT_EN:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_chn_int_en	0	R/W	Master Enable bit for HIF channel interrupts.
bdp_csr_rx_cbd_chn_int_en	1	R/W	Enable bit for HIF RX BDP interrupt.
bdp_csr_rx_pkt_chn_int_en	2	R/W	Enable bit for HIF RX Packet completion interrupt.
bdp_csr_tx_cbd_chn_int_en	3	R/W	Enable bit for HIF TX BDP interrupt.
bdp_csr_tx_pkt_chn_int_en	4	R/W	Enable bit for HIF TX Packet completion interrupt.
bdp_rd_csr_rx_timeout_chn_int_en	5	R/W	Enable bit for HIF RX BDP Read timeout interrupt.
bdp_wr_csr_rx_timeout_chn_int_en	6	R/W	Enable bit for HIF RX BDP Write timeout interrupt.
bdp_rd_csr_tx_timeout_chn_int_en	7	R/W	Enable bit for HIF RX BDP Read timeout interrupt.
bdp_wr_csr_tx_timeout_chn_int_en	8	R/W	Enable bit for HIF RX BDP Write timeout interrupt.
dxr_csr_rx_timeout_chn_int_en	9	R/W	Enable bit for DXR RX timeout interrupt.
dxr_csr_tx_timeout_chn_int_en	10	R/W	Enable bit for DXR TX timeout interrupt

Table 172. HIF_CHn_INT_SRC:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_chn_int	0	R/W	'1' indicates that an interrupt from HIF channel block is pending. This value is an OR of all interrupts in HIF channel block.
bdp_csr_rx_cbd_chn_int	1	R/W	This bit is set if any of CBD_INT_EN in BD is set within burst of BDs written out. See RX BD Ring.
bdp_csr_rx_pkt_chn_int	2	R/W	This bit is set if any of PKT_INT_EN in BD is set within burst of BDs written out. See RX BD Ring.
bdp_csr_tx_cbd_chn_int	3	R/W	This bit is set if any of CBD_INT_EN in BD is set within burst of BDs written out. See TX BD Ring.
bdp_csr_tx_pkt_chn_int	4	R/W	This bit is set if any of PKT_INT_EN in BD is set within burst of BDs written out. See TX BD Ring.
bdp_rd_csr_rx_timeout_chn_int	5	R/W	HIF RX BD Read Timeout.
dp_wr_csr_rx_timeout_chn_int	6	R/W	HIF RX BD Write Timeout.
bdp_rd_csr_tx_timeout_chn_int	7	R/W	HIF TX BD Read Timeout.
bdp_wr_csr_tx_timeout_chn_int	8	R/W	HIF TX BD Write Timeout.
dxr_csr_rx_timeout_chn_int	9	R/W	HIF DXR RX Timeout.
dxr_csr_tx_timeout_chn_int	10	R/W	HIF DXR TX Timeout (common for both data as well as SAD).

Table 173. HIF_CTRL_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
csr_tx_dma_en_chn_out	0	R/W	TX DMA enable bit. 1 - Enabled 0 - Disabled
csr_tx_bdp_poll_cntr_en_chn_out	1	R/W	Setting this bit to '1' triggers the Poll Counter Method for TX BD fetch (see the BD Fetch). If not set, then to trigger a single TX BD fetch one need to write the HIF_TX_CHn_START.
csr_rx_dma_en_chn_out	16	R/W	RX DMA enable bit. 1 - Enabled 0 - Disabled
csr_rx_bdp_poll_cntr_en_chn_out	17	R/W	Setting this bit to '1' trigger the Poll Counter Metchod for RX BD fetch (see the BD Fetch). If not set, then to trigger a single TX BD fetch one need to write the HIF_RX_CHn_START.

Table 174. HIF_INT_COAL_EN_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_int_coal_en_chn	1:0	R/W	Interrupt Coalescing Enable. Bit 0 for coalescing enable based on timeout. Bit 1 for coalescing enable based on frame count.

Table 175. HIF_ABS_FRAME_COUNT_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_abs_frame_count_chn	31:0	R/W	Frame counter for coalescing. If number of interrupts generated is equal to this counter, coalesce interrupt will be generated.

Table 176. HIF_ABS_INT_TIMER_CHn:

Field	OFFSET	ACCESS_TYPE	DESCRIPTION
hif_abs_int_ timer_chn	31:0	R/W	Coalescing timer value

7.9 EMAC

See the *GMAC subsystem reference manual*. EMAC share the same register as GMAC.

Reference document :

PFE Host Interface programmer's guide

GMAC subsystem reference manual

8 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2026 NXP 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 must be provided with the distribution.
3. Neither the name of the copyright holder 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 HOLDER 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.

9 Acronyms and definitions

Table 177. Acronyms and definitions

ACRONYM	DEFINITION
BMU	Buffer Manager Unit
CLASS	Classifier
DMEM	Data Memory
EMAC	Ethernet Media Access Controller
GPI	Generic Packet Interface
HIF	Host Interface
IGQOS	Ingress Quality of Service
IMEM	Instruction Memory
LMEM	Local Memory
Logif	Logical Interface
PE	Processing Engine
PFE	Packet Forward Engine
Phyif	Physical Interface
TMU	Traffic Manager Unit

10 Revision history

Table 178. Revision history

Document ID	Release date	Description
AN14953 v.1.0	26 February 2026	Initial release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

Suitability for use in automotive applications (functional safety) — This NXP product has been qualified for use in automotive applications. It has been developed in accordance with ISO 26262, and has been ASIL classified accordingly. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Tab. 1. Action sequence bits 9
Tab. 2. PFE Port Coherency Enable 10
Tab. 3. Base address of GPI instances 26
Tab. 4. TX Header Structure 34
Tab. 5. TX Frames Flag 35
Tab. 6. Egress Timestamp Report (20 bytes) 36
Tab. 7. RX Header Structure 36
Tab. 8. RX Frame Flags 36
Tab. 9. WSP_FAIL_STOP_MODE_INT_EN : 38
Tab. 10. WSP_FAIL_STOP_MODE_EN: 38
Tab. 11. WSP_ECC_ERR_INT_EN: 38
Tab. 12. WSP_PARITY_INT_EN: 38
Tab. 13. WSP_BUS_ERR_INT_EN: 39
Tab. 14. WSP_FW_FAIL_STOP_MODE_INT_EN : 40
Tab. 15. WSP_HOST_FORCE_DEBUG_FAIL_ STOP_MODE_INT_EN: 40
Tab. 16. WSP_SYS_GENERIC_CONTROL: 40
Tab. 17. WSP_DBUG_BUS1: (G3 only) 40
Tab. 18. WDT_INT_EN: 41
Tab. 19. WDT_INT_SRC: 42
Tab. 20. WDT_TIMER_VAL_UPE: 42
Tab. 21. WDT_TIMER_VAL_BMU: 42
Tab. 22. WDT_TIMER_VAL_HIF: 43
Tab. 23. WDT_TIMER_VAL_TLITE: 43
Tab. 24. WDT_TIMER_VAL_HIF_NCPY: 43
Tab. 25. WDT_TIMER_VAL_CLASS: 43
Tab. 26. WDT_TIMER_VAL_GPI: 43
Tab. 27. WDT_TIMER_VAL_GPT: 43
Tab. 28. WDT_TIMER_VAL_LMEM: 43
Tab. 29. WDT_TIMER_VAL_ROUTE_LMEM: 43
Tab. 30. CLASS_WDT_INT_EN: 44
Tab. 31. UPE_WDT_INT_EN: 44
Tab. 32. HGPI_WDT_INT_EN: 44
Tab. 33. HIF_WDT_INT_EN: 44
Tab. 34. TLITE_WDT_INT_EN: 44
Tab. 35. HNCPY_WDT_INT_EN: 45
Tab. 36. BMU1_WDT_INT_EN: 45
Tab. 37. BMU2_WDT_INT_EN: 45
Tab. 38. EMAC0_WDT_INT_EN: 45
Tab. 39. EMAC1_WDT_INT_EN: 46
Tab. 40. EMAC2_WDT_INT_EN: 46
Tab. 41. EXT_GPT_WDT_INT_EN: 46
Tab. 42. LMEM_WDT_INT_EN: 46
Tab. 43. BMU_CTRL: 46
Tab. 44. BMU_INT_ENABLE: 47
Tab. 45. BMU_INT_SRC: 47
Tab. 46. BMU_UCAST_BASEADDR: 48
Tab. 47. BMU_UCAST_CONFIG： 48
Tab. 48. BMU_BUF_SIZE: 48
Tab. 49. BMU_THRES: 48
Tab. 50. BMU_LOW_WATERMARK: 49
Tab. 51. BMU_HIGH_WATERMARK: 49
Tab. 52. BMU_INT_MEM_ACCESS_ADDR: 49
Tab. 53. BMU_INT_MEM_ACCESS: 49
Tab. 54. BMU_INT_MEM_ACCESS2: 49
Tab. 55. BMU_BUF_CNT_MEM_ACCESS_ADDR: 49
Tab. 56. BMU_BUF_CNT_MEM_ACCESS: 49
Tab. 57. BMU_BUF_CNT_MEM_ACCESS2: 49
Tab. 58. CLASS_TX_CTRL： 50
Tab. 59. CLASS_BMU1_BUF_FREE： 50
Tab. 60. CLASS_PE0_RO_DM_ADDR0: 50
Tab. 61. CLASS_PE0_RO_DM_ADDR1: 50
Tab. 62. CLASS_PE0_QB_DM_ADDR0: 50
Tab. 63. CLASS_PE0_QB_DM_ADDR1: 50
Tab. 64. CLASS_TM_INQ_ADDR: 50
Tab. 65. CLASS_MAX_BUF_CNT: 51
Tab. 66. CLASS_AFULL_THRES： 51
Tab. 67. CLASS_INQ_AFULL_THRES： 51
Tab. 68. CLASS_USE_TMU_INQ： 51
Tab. 69. CLASS_PE_SYS_CLK_RATIO： 51
Tab. 70. CLASS_L4_CHKSUM: 51
Tab. 71. CLASS_HDR_SIZE: 51
Tab. 72. CLASS_LMEM_BUF_SIZE: 52
Tab. 73. CLASS_TPID0_TPID1: 52
Tab. 74. CLASS_TPID2: 52
Tab. 75. CLASS_AXI_CTRL_ADDR: 52
Tab. 76. CLASS_ROUTE_MULTI: 52
Tab. 77. CLASS_MEM_ACCESS_ADDR: 53
Tab. 78. CLASS_MEM_ACCESS_RDATA: 53
Tab. 79. CLASS_MEM_ACCESS_WDATA: 53
Tab. 80. TMU_CTRL: 54
Tab. 81. TMU_CNTX_ACCESS_CTRL: 54
Tab. 82. TMU_CNTX_ADDR: 54
Tab. 83. TMU_CNTX_DATA: 54
Tab. 84. TMU_CNTX_CMD: 54
Tab. 85. TMU_PHY_QUEUE_SEL: 55
Tab. 86. TMU_CURQ_PTR: 55
Tab. 87. TMU_CURQ_PKT_CNT: 55
Tab. 88. TMU_CURQ_DROP_CNT : 55
Tab. 89. TMU_CURQ_TRANS_CNT : 55
Tab. 90. TMU_CURQ_QSTAT: 56
Tab. 91. TMU_HW_PROB_CFG_TBL0: 56
Tab. 92. TMU_HW_PROB_CFG_TBL1: 56
Tab. 93. TMU_CURQ_DEBUG: 56
Tab. 94. TMU_PHY_INQ_PKTINFO: 56
Tab. 95. TMU_TEQ_CTRL: 56
Tab. 96. TMU_PHYn_TDQ_CTRL: 57
Tab. 97. TMU_PHYn_INQ_ADDR: 57
Tab. 98. TMU_SCHn_Q_ALLOC0: 57
Tab. 99. TMU_SCHn_Q_ALLOC1: 57
Tab. 100. TMU_SCHn_POS: 57
Tab. 101. TMU_SHPn_MIN_CREDIT: 58
Tab. 102. TMU_SHPn_MAX_CREDIT: 58
Tab. 103. TMU_SHPn_CTRL2: 58
Tab. 104. TMU_SCHn_BIT_RATE: 58
Tab. 105. TMU_BMU_INQ_ADDR: 58
Tab. 106. TMU_AFULL_THRES: 59
Tab. 107. TMU_INQ_WATERMARK: 59
Tab. 108. TMU_PHYn_TDQ_CTRL: 59
Tab. 109. GPI_CTRL: 59
Tab. 110. GPI_PORT_SHPn_MIN_CREDIT: 59

Tab. 111. GPI_EMAC_1588_TIMESTAMP_EN: 59
Tab. 112. GPI_RX_CONFIG: 60
Tab. 113. GPI_HDR_SIZE: 60
Tab. 114. GPI_BUF_SIZE: 60
Tab. 115. GPI_LMEM_ALLOC_ADDR: 61
Tab. 116. GPI_LMEM_FREE_ADDR: 61
Tab. 117. GPI_DDR_ALLOC_ADDR: 61
Tab. 118. GPI_DDR_FREE_ADDR: 61
Tab. 119. GPI_CLASS_ADDR: 61
Tab. 120. GPI_DDR_DATA_OFFSET: 61
Tab. 121. GPI_LMEM_DATA_OFFSET: 61
Tab. 122. GPI_TMLF_TX: 61
Tab. 123. GPI_DTX_ASEQ: 62
Tab. 124. GPI_CSR_TOE_CHKSUM_EN: 62
Tab. 125. GPI_CSR_AXI_WRITE_DONE_ADDR: 62
Tab. 126. CSR_IGQOS_ENTRY_DATA_REGn: 62
Tab. 127. CSR_IGQOS_ENTRY_CMDCNTRL: 62
Tab. 128. CSR_IGQOS_ENTRY_CMDSTATUS: 62
Tab. 129. CSR_IGQOS_CONTROL: 63
Tab. 130. SR_IQGOS_DMEMQ_ZONE_PROB: 63
Tab. 131. CSR_IGQOS_LMEMQ_ZONE_PROB: 63
Tab. 132. CSR_IGQOS_RXFQ_ZONE_PROB: 63
Tab. 133. CSR_IGQOS_DMEMQ_FULL_THRESH: 63
Tab. 134. CSR_IGQOS_DMEMQ_DROP_THRESH: 64
Tab. 135. CSR_IGQOS_LMEMQ_FULL_THRESH: 64
Tab. 136. CSR_IGQOS_LMEMQ_DROP_THRESH: 64
Tab. 137. CSR_IGQOS_RXFQ_FULL_THRESH: 64
Tab. 138. CSR_IGQOS_RXFQ_DROP_THRESH: 64
Tab. 139. CSR_IGQOS_PORT_SHP_WGHT: 65
Tab. 140. GPI_PORT_SHP_MIN_CREDIT: 65
Tab. 141. CSR_IGQOS_PORT_SHP_CONFIG: 65
Tab. 142. CSR_IGQOS_PORT_SHP_CTRL: 65
Tab. 143. CSR_IGQOS_TPID: 65
Tab. 144. CSR_IGQOS_CLASS: 66
Tab. 145. CSR_IGQOS_LRU_TIMER_VALUE: 66
Tab. 146. HIF_MISC: 66
Tab. 147. HIF_SOFT_RESET: 66
Tab. 148. HIF_ERR_INT_SRC: 67
Tab. 149. HIF_ERR_INT_EN: 67
Tab. 150. HIF_TX_FIFO_ERR_INT_SRC: 68
Tab. 151. HIF_TX_FIFO_ERR_INT_EN: 69
Tab. 152. HIF_RX_FIFO_ERR_INT_SRC: 70
Tab. 153. HIF_RX_FIFO_ERR_INT_EN: 70
Tab. 154. HIF_RX_BDP_RD_LOW_ADDR_CHn: 71
Tab. 155. HIF_RX_BDP_RD_HIGH_ADDR_CHn: 71
Tab. 156. HIF_TX_BDP_RD_LOW_ADDR_CHn: 71
Tab. 157. HIF_TX_BDP_RD_HIGH_ADDR_CHn: 72
Tab. 158. HIF_RX_BDP_WR_LOW_ADDR_CHn: 72
Tab. 159. HIF_RX_BDP_WR_HIGH_ADDR_CHn: 72
Tab. 160. HIF_TX_BDP_WR_LOW_ADDR_CHn: 72
Tab. 161. HIF_TX_BDP_WR_HIGH_ADDR_CHn: 72
Tab. 162. HIF_RX_WRBK_BD_CHn_BUFFER_SIZE: 72
Tab. 163. HIF_TX_WRBK_BD_CHn_BUFFER_SIZE: 72
Tab. 164. HIF_TX_POLL_CTRL: 73
Tab. 165. HIF_RX_POLL_CTRL: 73
Tab. 166. HIF_TIMEOUT_REG: 73
Tab. 167. HIF_RX_QUEUE_MAP_CH_NO_ADDR: 73
Tab. 168. HIF_DMA_BURST_SIZE: 74
Tab. 169. HIF_DMA_BASE_ADDR: 74
Tab. 170. HIF_LTC_PKT_CTRL_ADDR: 74
Tab. 171. HIF_CHn_INT_EN: 74
Tab. 172. HIF_CHn_INT_SRC: 75
Tab. 173. HIF_CTRL_CHn: 75
Tab. 174. HIF_INT_COAL_EN_CHn: 75
Tab. 175. HIF_ABS_FRAME_COUNT_CHn: 76
Tab. 176. HIF_ABS_INT_TIMER_CHn: 76
Tab. 177. Acronyms and definitions 78
Tab. 178. Revision history 79

Figures

Fig. 1. Interfaces 3
Fig. 2. Hardware Architecture 3
Fig. 3. CLASS 5
Fig. 4. PE memory 6
Fig. 5. Queueing algorithm 7
Fig. 6. Data path 12
Fig. 7. CLASS data path 16
Fig. 8. Default egress QoS 23

Contents

1 Introduction 2
2 Hardware architecture 3
2.1 Overview 3
2.2 EMAC 4
2.3 HIF 4
2.4 BMU 4
2.5 CLASS 5
2.6 TMU 6
2.7 GPI 8
2.8 Data Coherence 10
2.9 Clock Requirement 11
3 Internal data flow architecture 12
3.1 EMAC to HIF 12
3.2 HIF to EMAC 13
3.3 Egress Timestamp to HIF 13
4 Firmware 14
4.1 Access CLASS PE memory 14
4.2 Loading and Running FW 14
4.3 Data flow implemented by FW 15
4.4 Get Class PE Status 16
4.5 FW feature management 16
5 Initialization 17
5.1 Overview 17
5.2 Prerequisites 17
5.3 Internal RAM initialization 18
5.4 BMU initialization 19
5.5 Class initialization 20
5.6 FW loading 20
5.7 TMU Initialization 20
5.8 EMAC Initialization 23
5.9 PFE system soft reset 24
5.10 GPI Initialization 25
5.11 HIF Initialization 27
5.12 Enable each sub-module 28
5.13 PFE driver shutdown 29
5.14 Hard reset 30
5.15 Transmitting and Receiving 31
5.16 PFE slave driver initialization 31
6 Firmware interface 32
6.1 PhyIf 32
6.2 LogIf 33
6.3 Tx header 34
6.4 Rx header 36
7 Registers summary 38
7.1 WSP 38
7.2 Watchdog 41
7.3 BMU 46
7.4 CLASS 50
7.5 TMU 54
7.6 GPI 59
7.7 IGQOS 62
7.8 HIF 66
7.9 EMAC 76
8 Note about the source code in the document 77
9 Acronyms and definitions 78
10 Revision history 79
Legal information 80