20450 Century Boulevard
Germantown, MD 20874

EMAC Low Level Driver
Software Design Document

Revision 1.7

May 28, 2020

Document License

Contributors to this document

	Revision Record	
	Document Title: **Software Design Specification**	
Date	Revision	Description of Change
04-04-2017	0.1	Initial Release
04-10-2017	0.2	Added feature list
07-07-2017	0.3	Addressed all the peer review comments
06-13-2018	0.3	Added feature list for AM65XX
07-13-2018	0.4	Added driver details for AM65xx
10-04-2018	0.5	Added document license for Creative Commons
10-05-2018	0.6	WIP draft Incorporated subset of review comments
11-27-2018	0.7	Added sections for IOCTL commands, driver initialization , TX Time Stamp, TX Software Descriptor Return Queue Processing, RX Time Stamp
02-21-2019	0.8	Incorporate follow up comments from customer
3-28-2019	0.9	Incorporate additional follow up comments from customer
4-2-2019	.95	Additional IOCTL updates
5-08-2019	1.0	Updates to error types, Clarification of each IOCTL type to be either synchronous versus asynchronous, adding details for EMAC_IOCTL_INTERFACE_MAC_CONFIG
5-15-2019	1.1	IOCTL table updates. Enhanced emac poll link status section. Enhanced Packet RX section. Miscellaneous updates.
5-31-2019	1.2	Added sections 7.4.2.1, 7.4.2.1.1 and 7.4.2.2 Added clarification to emac_poll_ctrl API and ring bitmap. Updated FDB ADD IOCTL details
9-6-2019	1.3	Section 7.4.3: Updated clarification of return codes to application during emac_send Section 7.4.5: Further clarification to TX timestamp implementation Section 7.4.6: Added details of how ring number is returned in rx callback

9-24-19	1.4	Section 14.4.9.1: Added details about touch and valid bits of FID_C2 bitfield for FDB ADD IOCTL Section 7.4: Clarified max number of RX subchannels to be 16. Section 7.4.8 Clarification of TX channel priority for dual mac and switch use cases.
2-17-20	1.5	Removed reference to PSI based Management messaging
4-14-20	1.6	Removed all references to Interposer card configuration Added new virtual port enums for ICSSG switch and ICSSG dual mac use cases
5-28-20	1.7	Section 7.4.4: Added clarification to priority to PktChannel for emac_send API Section 7.4.10.1: Updated Switch IOCTL table with EMAC_IOCTL_FDB_AGEING_TIMEOUT_CTRL details.

TABLE OF CONTENTS

1 Purpose

This document describes the functionality, architecture, and operation of the Ethernet Media Access Controller（EMAC) Low Level Driver. Also the data types, data structures and application programming interfaces (API) provided by the EMAC driver are explained in this document.

2 Functional Overview

EMAC driver provides a well-defined API layer which allows applications to use the EMAC peripheral to control the flow of packet data from the processor to the PHY and the MDIO module to control PHY configuration and status monitoring.

3 Assumptions

NA

4 Definitions, Abbreviations, Acronyms

Term	Description
API	Application Programming Interface
CSL	Chip Support Library
EMAC	Ethernet Media Access Controller
LLD	Low Level Driver Design
ISR	Interrupt Service Routine
MDIO	Managed Data Input Output
MMR	Memory Mapped Registers
NDK	Network Development Kit
NIMU	Network Interface Management Unit
OSAL	Operating System Adaptation Layer
PHY	Physical layer
MGMT	Management
FW	Firmware

Table 1 : Abbreviations and acronyms

5 References

Following references are related to the features described in this document and shall be consulted as necessary.

- TRM for SoCs being supported by EMAC LLD

- Migrating_Applications_from_EDMA_to_UDMA_using_TI-RTOS.pdf (ti/drv/udma/docs)

6 Design Constraints

6.1 External Constraints / Features

- EMAC LLD should access OS components only through OSAL.

6.2 External Constraints / System Performance

EMAC LLD should allow applications to transfer and receive through Ethernet port and communicate with the network devices at maximum possible speed as supported by HW.

6.3 Internal Constraints / Requirements

EMAC LLD should use CSL layer for register access to abstract the HW dependencies and maintain portability across the platforms.

7 System Overview

7.1 System Context

EMAC LLD is designed to be functional as part of TI processor SDK driver package. There will be several components in the processor SDK, apart from applications, which uses EMAC LLD. Driver design ensures that it fits into system properly and provides suitable APIs for utilizing EMAC HW functionality.
The following figure shows the architecture of processor SDK sub-system around the LLD modules.

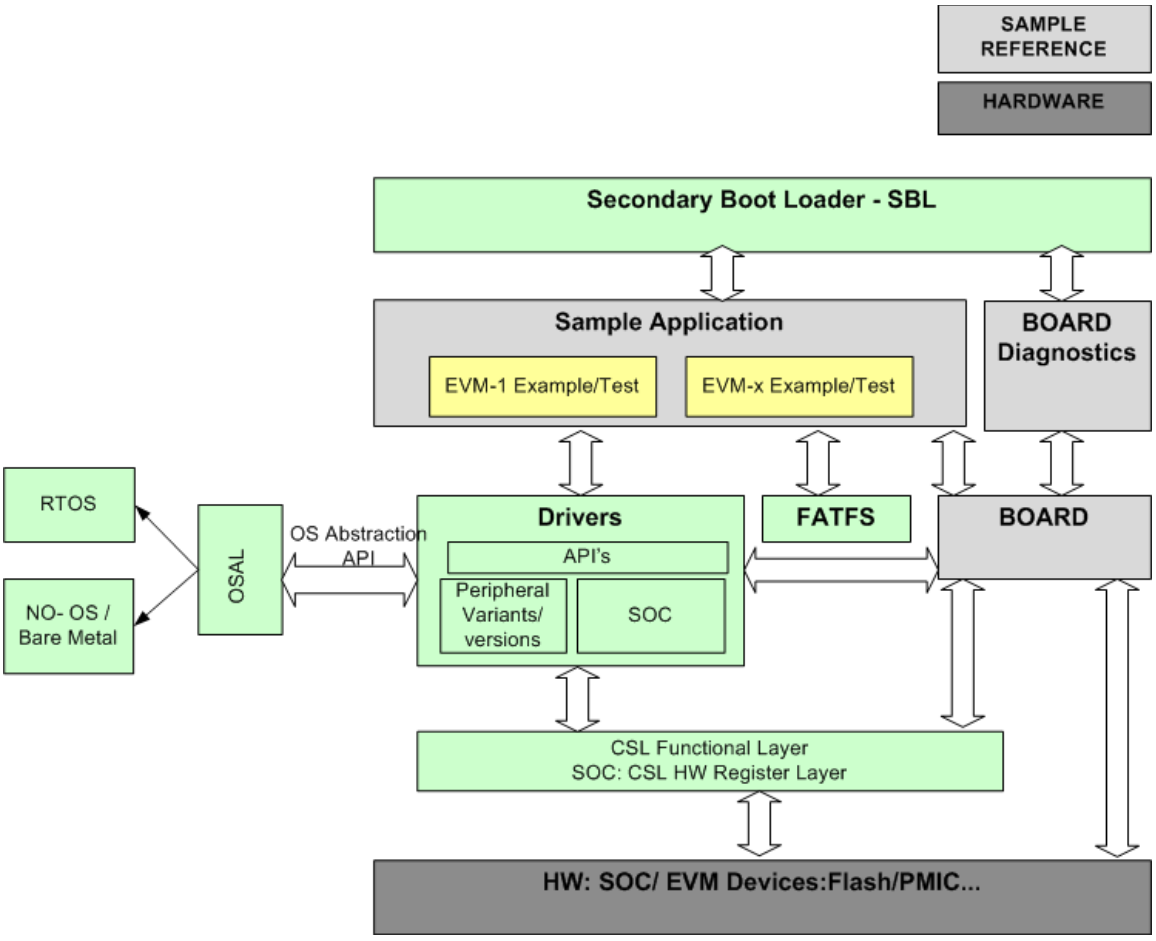


Figure 1 : Process SDK driver subsystem architecture

7.2 Functional Description

The EMAC driver is responsible for the following:-

- EMAC/MDIO configuration & Queue Management
- Providing a well-defined API to interface with the applications
- Well defined operating system adaptation layer API which supports single core and multiple core critical section protection

The next couple of sections document each of the above mentioned responsibilities in greater detail:

7.3 CPPI Based IP Driver: IP Version 0/1/4

7.3.1 EMAC Peripheral configuration

The EMAC driver test application provides a sample implementation sequence which initializes and configures the EMAC IP block. This implementation is sample only and application developers are recommended to modify it as deemed fit.

The initialization sequence is not a part of the EMAC driver library. This was done because the EMAC initialization sequence has to be modified and customized by application developers.
The following figure shows the EMAC API the application can call to initialize the EMAC peripheral:-

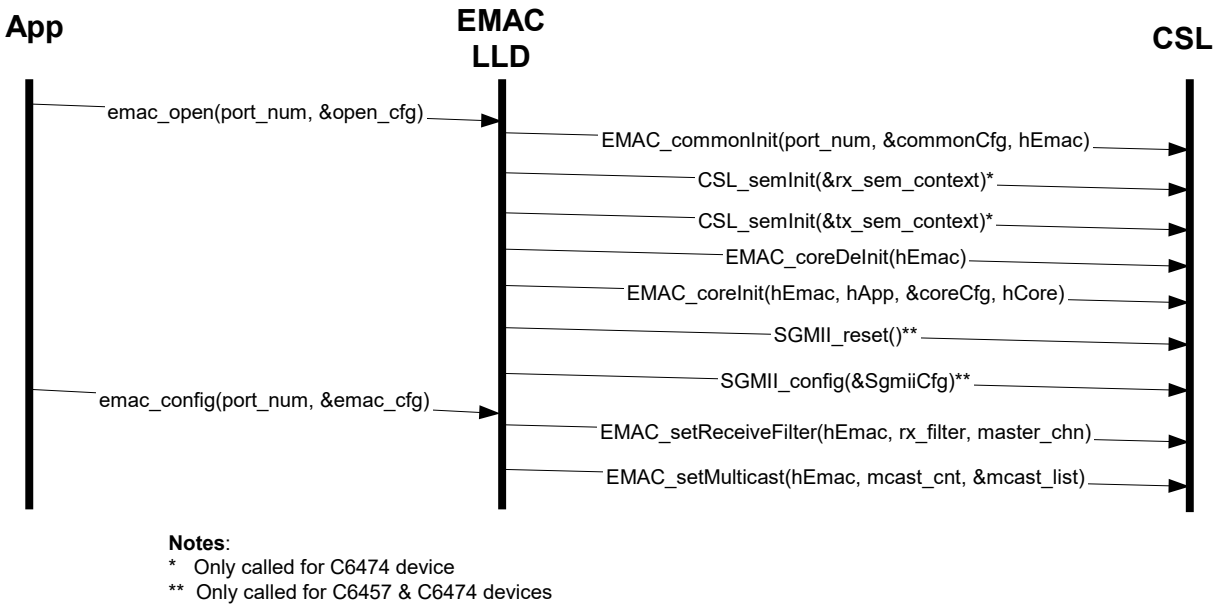


Figure 2 : EMAC configuration

Please note that the call flow dedicated above is basic illustration of how *emac_open* is handled internally and may differ from amongst different IP versions of the driver. At the API level from application point of view, it's the same.

Refer to the EMAC_OPEN_CONFIG_INFO_T as defined in emac_drv.h for details of configuration parameters passed into the driver at the time of *emac_open* API call.

When this API is called, the EMAC driver will first initialize common EMAC configurations (e.g. loopback mode, MDIO enable, PHY address, packet size, etc.) which applies to all the cores, and then initialize the core specific configurations (e.g. channel/MAC address configuration, TX/RX packet descriptor queue size, call back functions, etc.). The driver may

also need to do some device specific configurations (e.g. C6457 & C6474 have a SGMII interface in the EMAC peripheral which need to be configured, and C6474 has a hardware semaphore which also needs to be configured).

The *emac_config*() API passes the following configuration parameters to the EMAC driver:

- EMAC port number
- EMAC packet receive filter level
- Multicast configurations

NOTE: This API is currently only implemented for v0 version of the driver.

7.3.2 Queue Management

The EMAC driver manages one TX packet descriptor queue and one RX packet descriptor queue per each EMAC port, the TX/RX queue size is initialized by the application. The driver pre-allocates the packet buffer for each packet descriptor pushed to the RX queue when an EMAC port is opened. The driver frees both TX/RX queues when an EMAC port is closed.

7.3.3 Packet Descriptor

By default, the EMAC driver uses CPPI RAM(8K-byte) for EMAC IP managed Packet Descriptor memory. This internal 8K-byte memory is used to manage the buffer descriptors that are 4-word(16-bytes) deep. The maximum number of descriptors that can be used for managing the packets being transferred is 512. Application shall allocate the packet descriptors for TX, RX and should pass the information to driver using EMAC_OPEN_CONFIG_INFO_T structure during driver open.

7.3.4 Packet TX

The application can send a packet by calling *emac_send* () API, the application needs to allocate an application managed packet descriptor from the application queue, copy the packet data and convert it to the EMAC driver managed packet descriptor format.

The following figure shows the EMAC/CSL API for a packet sent:-

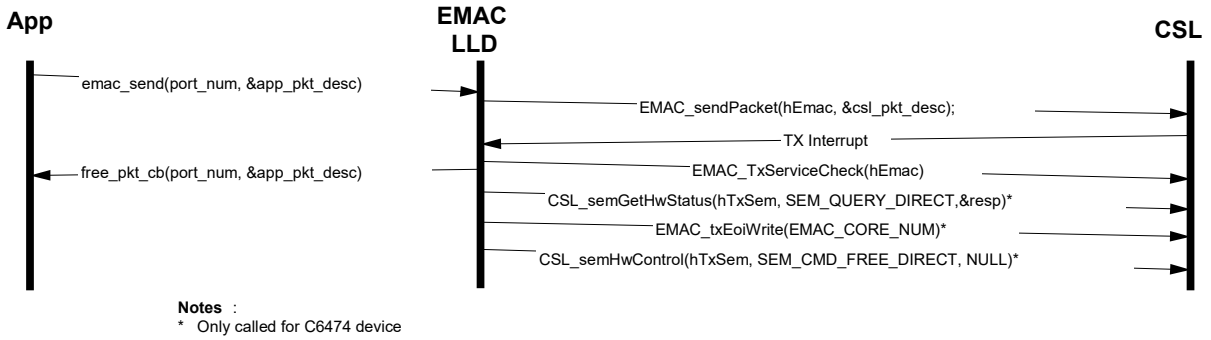


Figure 3 : EMAC TX function

7.3.5 Packet RX

When a packet is received, the EMAC driver will convert the packet descriptor received to the application managed packet descriptor format and pass it to the application by calling the rx_pkt_cb() callback function.

The following figure shows the EMAC/CSL API for a packet received:-

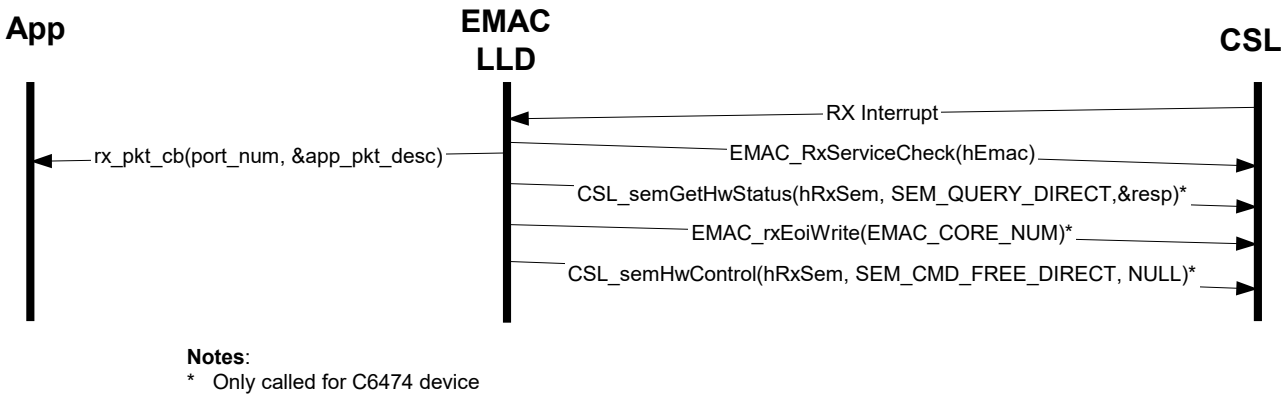


Figure 4 : EMAC RX function

7.3.6 Single Critical Section

The EMAC driver maintains certain per core specific data structures. These data structures need to be protected from access by multiple users running on the same core. Users are defined as entities in the system which uses the EMAC Driver API's. The critical section defined here should also take into account the context of these users (Thread or Interrupt) and define the critical sections appropriately.

For example: In the EMAC RX interrupt service routine, if RX interrupt is not disabled, a new RX interrupt may pre-empt the existing RX ISR and cause data corruption in CSL CPPI packet descriptors.

The EMAC driver uses the Emac_osalEnterSingleCoreCriticalSection() API to enter the single core critical section and Emac_osalExitSingleCoreCriticalSection to exit the single core critical section.

7.3.7 Multi-core Critical section

The EMAC driver supports multiple cores sharing the same EMAC port. The driver defines the following common data structures that are shared by all the cores:

- EMAC_Device emac_comm_dev
- EMAC_COMMON_PCB_T emac_comm_pcb

emac_comm_dev contains common EMAC device instance information, it is defined in the EMAC driver, but is managed by the EMAC CSL.
emac_comm_pcb contains common port control block information that is managed by the EMAC driver.

The EMAC driver defines a pragma data section “emacComm” for these two data structures, the application needs to put “emacComm” data section in the shared memory (either shared L2 data if available or external memory)

The EMAC driver calls Emac_osalEnterMultipleCoreCriticalSection() and Emac_osalExitMultipleCoreCriticalSection() API to enter and exit critical section to access shared resource by multiple cores. The EMAC multicore test application shows an example how to implement semaphore protection for shared resource access among multiple cores. C6472 uses IPC GateMP module to implement a software semaphore, and C6474 uses CSL hardware semaphore.

For shared memory access, the EMAC driver calls Emac_osalBeginMemAccess() and Emac_osalEndMemAccess() to protect cache coherence when cache is enabled. The driver always performs an invalidate cache operation before reading data and write back cache operation after writing data. The start address of emac_comm_dev and emac_comm_pcb need to be set aligned to the cache line size of the device by the application.

The following figure shows an example how the EMAC driver can access the shared resource:-

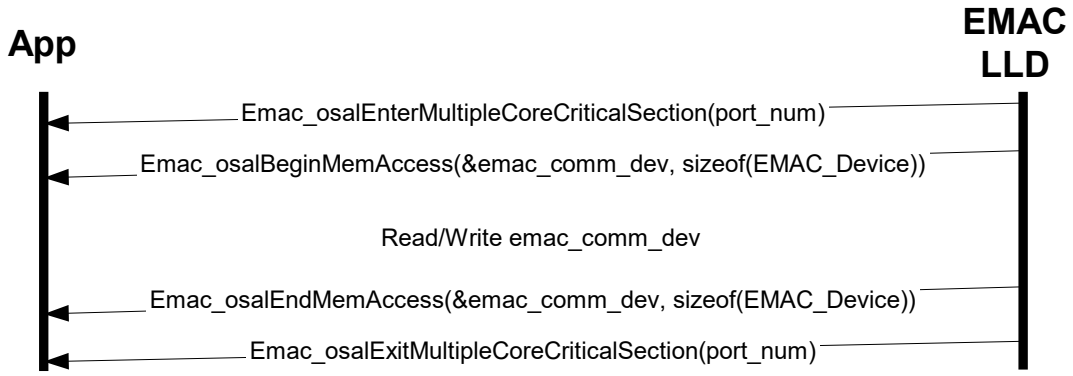


Figure 5 : EMAC Critical section access

7.3.8 Interrupts

Interrupt configuration is specified in SOC’s init configuration and is provided to the EMAC LLD at time of *emac_open* API. Interrupt registration is done within the LLD at time of *emac_open*. Once interrupt is received, application provided callbacks are invoked.

7.4 **UDMA/NavSS based IP Driver: IP Version 5**

IP version 5 supports SOCs based on NavSS/UDMA based DMA interface eg:AM65XX. The LLD provides a common set of APIs to service both CPSW and ICSS-G hardware IP ports. This is possible due to the NavSS IP which groups together various different hardware IP blocks (in this case CPSW/ICSS-G) and whose purpose is to support the efficient transfer of data between various software, firmware and hardware entities via the use of channels.

A channel is a DMA instance/resource of which there are the following 2 types:

1. Receive (RX) Channel
    a. RX Packet Channel: EMAC LLD receives packets from the network via ICSSG/CPSW subsystem

ICSS Switch supports 2 RX channels distributed as follows:

- Physical port 0 data packets from network
- Physical Port 1 data packets from network

Each RX channel can be "divided" into to N (9) sub channels where each sub-channel can be considered a distinct flow having each having its own free and completion ring pair. This allow for "bining" packets of different types to be delivered to the EMAC LLD. ICSS-G f/w provides a flow id as packet meta data that the DMA uses to determine the ultimate free/completion ring pair to use. Note that when a channel is created using the UDMA driver, a flow is created by "default" and is considered by the driver is the 1st sub-channel or flow. The remaining N-1 flows are sequential.
Default SOC configuration as specified by the emac_soc.c file (see sub-sequent section for overview) provides configuration for the LLD to create 1 RX Packet Channel with 9 sub-channels per slice.

2. Transmit (TX) Channel: EMAC LLD transmits packets to the network via ICSSG/CPSW subsystem

Default SOC configuration as specified in the emac_soc.c file provides configuration for the LLD to create 4 TX channels per physical port (slice). LLD provides the application the option to choose which TX channel to transmit in the packet send function. More details about transmitting packets in subsequent sections below.

7.4.1 Memory

There is no constraint on where Packet Descriptor, packet buffer, and Ring memory resides except that the memory block/region must cache size aligned on cores which are not hardware cache coherent such as the R5F (there may be performance impacts however). Application shall allocate the packet descriptors for TX, RX and Ring memory and should pass the information to LLD using EMAC_OPEN_CONFIG_INFO_T structure during driver open. See section 7.4.2.1 for details.

7.4.2 EMAC Driver Initialization configuration

The *emac_open()* API is used for driver initialization. Refer to the EMAC_OPEN_CONFIG_INFO_T as defined in emac_drv.h for details of configuration parameters passed into the driver at the time of *emac_open* API call.
The EMAC driver unit test application provides a sample implementation sequence which initializes and configures the EMAC driver. This implementation is sample only and application developers are recommended to modify it as deemed fit.

The following flow diagram illustrates a sample calling sequence of API calls that can be used as a guideline for application development in addition to what is provided with the EMAC driver unit test application.

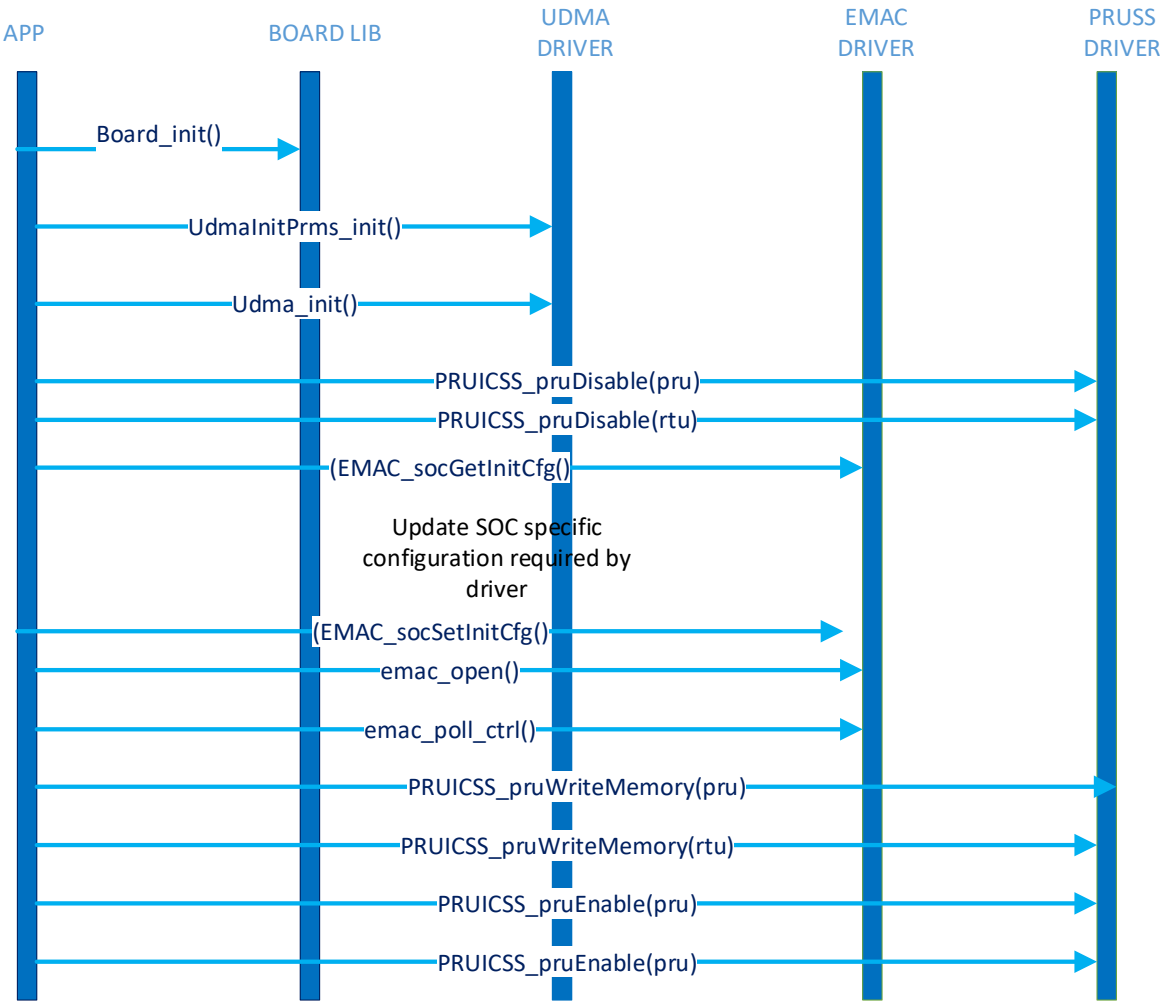


Figure 6 API Call Flow Sequence

7.4.2.1 EMAC Open Configuration Details.

EMAC_OPEN_CONFIG_INFO_T is passed into the driver at time of *emac_open()* and is per port. The following table describe each of the fields of this structure.

Field	Description
master_core_flag	Indicates core as master, used for multiple core use case and legacy SOCs. For Maxwell, only single core use case is currently supported, set this to TRUE.
mdio_flag	Note that this does not apply for Maxwell, always set this to true required to poll for link status
phy_addr	Note that this does not apply for Maxwell, phy address comes for SOC configuration mentioned in below table.
num_of_tx_pkt_desc	Total # of pkt desc initialized for all the TX chans in TX queues/rings
num_of_rx_pkt_desc	Total # of pkt desc initialized for all the RX chans in RX queues/rings
max_pkt_size	Maximum size of the packet in bytes
num_of_chans	Total number of TX/RX channels configured for this core, only applicable for SOC_C6657,for Maxwell, channel configuration comes from SOC configuration mentioned in table below.
p_chan_mac_addr	Note that this does not apply for Maxwell switch use case. Use EMAC_IOCTL_INTERFACE_MAC_CONFIG
rx_pkt_cb	EMAC RX call back function to receive packets

alloc_pkt_cb	EMAC allocate packet call back function, used to setup RX channels/rings at start up and replenish rx free channel/ring at time of packet receive.
free_pkt_cb	EMAC free packet call back function, used to free packet to application during tx completion event, and time of driver close(at close for both rx and tx packets)
rx_mgmt_response_cb	EMAC receive call back function for management response from ICSSG FW
tx_ts_cb	EMAC transmit timestamp response callback function
hwAttrs	Pointer to a driver specific hardware attributes structure, see below table for details
appPrivate	DEPRECATED: Application specific handle, not used by driver
mode_of_operation	DEPRECATED: should always be set to EMAC_MODE_INTERRUPT, to poll the driver for packets, use emac_poll_ctrl API.
udmaHandle	Handle to UDMA driver
drv_trace_cb	EMAC driver trace callback function

7.4.2.1.1 SOC Specific Configuration

At time of *emac_open*, the driver requires SOC specific configuration. This is passed to the driver via *hwAttrs* field of *EMAC_OPEN_CONFIG_INFO_T*. Defaults for soc configuration are in ti/drv/emac/soc/am65xx/emac_soc.c. Since the emac drivers supports many SOC types, the hwAttrs field is passed in as a void * and then internally casted by the driver to that specific SOC hw attributes structure. The hwAttrs struct defined for SOC_AM65XX is EMAC_HwAttrs_V5

The following is EMAC_HwAttrs_V5 declaration:

```
typedef struct EMAC_HwAttrs_V5_s {
   /*! Per Port configuration */
```

```
    EMAC_PER_PORT_CFG portCfg[EMAC_MAX_PORTS];
} EMAC_HwAttrs_V5;
```

This configuration consists of the following fields and is per port.
NOTE that unless stated, those fields should not be modified, fields which can be modified are in bold:

Field	Description
phyAddr	PHY address (board specific)
nTxChans	number of transmit channels
mdioRegsBaseAddr	base address of MDIO sub-system
icssSharedRamBaseAddr	base address of ICSSG shared memory, N/A for CPSW2G
icssgCfgRegBaseAddr	base address of ICSSG configuration registers, N/A for CPSW2G
icssDram0BaseAddr	base address of ICSSG data ram 0 memory, N/A for CPSW2G
txChannel[4] of type EMAC_PER_CHANNEL_CFG_TX	configuration required to setup up to 4 transmit channels which is to be provided by the application. This includes **chHandle** :memory for Udma_ChObj **freeRingMem**: pointer to memory for free UDMA ring, sized to **elementCount** * size of uint64_t . **compRingMem**: pointer to memory for UDMA completion ring, sized to number of ring entries * size of uint64_t . NOTE: size should be the same as used for free ring mem. **hPdMem**: pointer to memory for packet descriptors that will be pushed to the TX free descriptor software ring maintained by the driver. This needs to be sized by **num_of_tx_pkt_desc** (field of *EMAC_OPEN_CONFIG_INFO_T)* * 128. **eventHandle**: placeholder for adding interrupt support at ring level, currently not used.

	elementCount: Set to queue depth of the ring/number of ring elements (this can be modified, up to 128) threadId: thread ID to connect cpdma to/from emac psi
rxChannel of type EMAC_PER_CHANNEL_CFG_RX	configuration required to setup upto a receive channel which is to be provided by the application. This includes **chHandle** : pointer to memory for Udma_ChObj **flowHandle**: pointer to memory for Udma_FlowObj to setup additional flows/rings in additional to default flow/ring. nsubChan: number of sub-channels or flows. subChan: sub-channel specific configuration as specified by EMAC_RX_SUBCHAN which has the following fields: nfreeRings: number of free rings associated with sub-channel, for now set to 1 but can be upto4 to support advanced features of flow. Note: EMAC_MAX_FREE_RINGS_PER_SUBCHAN is set to 1. **freeRingHandle**[EMAC_MAX_FREE_RINGS_PER_SUBCHAN]: pointer to memory for free UDMA ring handle **freeRingMem**[EMAC_MAX_FREE_RINGS_PER_SUBCHAN]: pointer to memory for free UDMA ring, sized to **elementCount** * size of uint64_t . **compRingHandle**: pointer to memory for free UDMA completion handle **compRingMem**: pointer to memory for UDMA completion ring, sized to number of ring entries * size of uint64_t .. **hPdMem**[EMAC_MAX_FREE_RINGS_PER_SUBCHAN]: memory for packet descriptors that will be pushed to the rx free ring. This needs to be sized by **num_of_rx_pkt_desc** (field of

	EMAC_OPEN_CONFIG_INFO_T) * 128. **eventHandle**: placeholder for adding interrupt support at ring level, currently not used. **elementCountFree**[EMAC_MAX_FREE_RINGS_PER_SUBCHAN]: set to queue depth of the rx free ring **elementCountCompletion**: set to queue depth of the rx completion ring threadId: thread ID to connect cpdma to/from emac psi
rxChannelCfgOverPSI	Not used
rxChannel2CfgOverPSI	Not used
getFwCfg	Function pointer provided by the application and used by the driver to get firmware specific configuration, required to setup the environment for FW to operate. Must be set to *emacGetSwitchFwConfig* or switch use case and *emacGetDualMacFwConfig* for dual mac use case.

7.4.2.2 ICSSG Port Queue Configuration

The firmware configuration is directly tied to the ICSSG firmware and must not be modified accept for addresses of application provided memory for ICSSG port queues. This memory must be allocated from MSMC SRAM and128 byte aligned. The port buffer size mentioned below is specific to the firmware dependent Please refer to the firmware documentation for details.

The following sequence is an example illustration of how to accomplish this (NOTE the ports used in this illustration are software logical port 0-1, not EMAC_ICSSGX_SWITCH_PORT1/ EMAC_ICSSGX_SWITCH_PORT2 and also shown only for ICSSG_0.

```
/* memory of port queue, from msmc memory, 128 byte aligned */
uint8_t icss_tx_port_queue_icssg0[1][PORT_BUFFER_SIZE] __attribute__ ((aligned
(UDMA_CACHELINE_ALIGNMENT))) __attribute__ ((section (".bss:emac_msmc_mem")));
EMAC_FW_APP_CONFIG *pFwAppCfg;
/* Get application part of init config */
emacGetSwitchFwAppInitCfg(port_num, &pFwAppCfg);
/* update the address lo/hi with address of port queue */
pFwAppCfg->txPortQueueLowAddr = 0xFFFFFFFF & ((uint32_t) &icss_tx_port_queue_icssg0[0][0]);
pFwAppCfg->txPortQueueHighAddr = 0;

/* set/store the addresses of the port queue*/
emacSetSwitchFwAppInitCfg(port_num, pFwAppCfg);
```

7.4.3 EMAC Port Enumeration Details

7.4.3.1 ICSSG DUAL MAC Mode

The following macro definitions should be used as the port number field for all of the EMAC LLD API's when running in DUAL MAC mode.

```
#define EMAC_ICSSG0_DUALMAC_PORT0  ((uint32_t)0U)  /* ICSSG instance 0 port 0 */
#define EMAC_ICSSG0_DUALMAC_PORT1  ((uint32_t)1U)  /* ICSSG instance 0 port 1 */
#define EMAC_ICSSG1_DUALMAC_PORT0  ((uint32_t)2U)  /* ICSSG instance 1 port 0 */
#define EMAC_ICSSG1_DUALMAC_PORT1  ((uint32_t)3U)  /* ICSSG instance 1 port 1 */
#define EMAC_ICSSG2_DUALMAC_PORT0  ((uint32_t)4U)  /* ICSSG instance 2 port 0 */
#define EMAC_ICSSG2_DUALMAC_PORT1  ((uint32_t)5U)  /* ICSSG instance 2 port 1 */
```

These macros can be used directly when invoking any API call depending on the dual mac port being referenced.

7.4.3.2 ICSG Switch Mode

The following macro definitions should be used as the port number field for all the EMAC LLD API's when running in Switch Mode.

```
#define EMAC_ICSSG0_SWITCH_PORT  ((uint32_t)8U)     /* for emac open/close/certain IOCTL's, flooded packet */
#define EMAC_ICSSG0_SWITCH_PORT0 ((uint32_t)9U)     /* host port for certain IOCTL's */
#define EMAC_ICSSG0_SWITCH_PORT1 ((uint32_t)10U)    /* for directed send, stats, polling, for certain IOCTL's */
#define EMAC_ICSSG0_SWITCH_PORT2 ((uint32_t)11U)    /* for directed send, stats, for certain IOCTL's */

#define EMAC_ICSSG1_SWITCH_PORT ((uint32_t)12U)     /* for emac open/close/certain IOCTL's, flooded packet */
#define EMAC_ICSSG1_SWITCH_PORT0 ((uint32_t)13U)    /* host port for certain IOCTL's */
#define EMAC_ICSSG1_SWITCH_PORT1 ((uint32_t)14U)    /* for directed send, stats, polling for certain IOCTL's */
#define EMAC_ICSSG1_SWITCH_PORT2 ((uint32_t)15U)    /* for directed send, stats, for certain IOCTL's */

#define EMAC_ICSSG2_SWITCH_PORT ((uint32_t)16U)     /* for emac open/close/certain IOCTL's, flooded packet */*/
#define EMAC_ICSSG2_SWITCH_PORT0 ((uint32_t)17U)    /* host port for certain IOCTL's */
#define EMAC_ICSSG2_SWITCH_PORT1 ((uint32_t)18U)    /* for directed send, stats, polling,  for certain IOCTL's */
#define EMAC_ICSSG2_SWITCH_PORT2 ((uint32_t)19U)    /* for directed send, stats, for certain IOCTL's */
```

Independent switches can be configured, 1 per each ICSSG instance. Details of usage will be provided in subsequent sections and the following convention will be used where “X “ denotes the ICSSG instance

EMAC_ICSSGX_SWITCH_PORT
EMAC_ICSSGX_SWITCH_PORT0
EMAC_ICSSGX_SWITCH_PORT1
EMAC_ICSSGX_SWITCH_PORT2

Overview of how virtual ports should be used when invoking API calls, details are provided in subsequent sections where applicable.

API	Virtual Port Number
emac_open	EMAC_ICSSGX_SWITCH_PORT
emac_close	EMAC_ICSSGX_SWITCH_PORT
emac_poll_ctrl	physical port 0/ETH0: EMAC_ICSSGX_SWITCH_PORT1 physical port 1/ETH1: EMAC_ICSSGX_SWITCH_PORT2
emac_send	physical port 0/ETH0: EMAC_ICSSGX_SWITCH_PORT1 physical port 1/ETH1: EMAC_ICSSGX_SWITCH_PORT2 un-directed packet: EMAC_ICSSGX_SWITCH_PORT
emac_ioctl	Refer to table in subsequent sections
emac_poll	physical port 0/ETH0: EMAC_ICSSGX_SWITCH_PORT1 physical port 1/ETH1: EMAC_ICSSGX_SWITCH_PORT2

emac_get_statistics_icssg	physical port 0/ETH0: EMAC_ICSSGX_SWITCH_PORT1 physical port 1/ETH1: EMAC_ICSSGX_SWITCH_PORT2

7.4.4 Packet TX

EMAC LLD for AM65XX will support UDMAP operations to transfer data between the host processor and network peripherals. A valid port number and EMAC_PKT_DESC_T are required arguments to the *emac_send* API.
For DUAL MAC use case, the port number is the physical port used to transmit the packet to the network.
For switch use case, we will use virtual port concept for the port number when calling *emac_send*. For directed packet to a specific physical port, use [virtual] port
EMAC_ICSSGX_SWITCH_PORT1
to send on physical port 0/ETH0 of the switch and use [virtual] port
EMAC_ICSSGX_SWITCH_PORT2
to send on physical port 1/ETH1 of the switch.

For un-directed packets use virtual port
EMAC_ICSSGX_SWITCH_PORT
In this case, the driver and switch firmware will take care of transmitting the packet on the correct port(s).

The transmit submit ring to be used for the transmission can be specified in the PktChannel field of the EMAC_PKT_DESC_T passed in. In other words, the application may decide on which of the Transmit Channels(rings) to use. The driver will support configuration of up to 8 TX channels per switch or 4 TX channels per dual mac port (at time of *emac_open*) each associated with a transmit submit ring/completion ring pair.

For switch use case, the first 4 channels (0-3) are associated with slice 0 and second 4 channels(4-7) associated with slice 1 for a total of 8 channels per switch.

Here is en example of a mapping of priority to PtkChannel field that can be used by an application. Using this mapping, each priority goes on its own channel and the priorities are dived over the 2 slices.

Priority	PktChannel	Slice #	Physical Channel Number
0 (lowest)	0	0	0
1	4	1	0

2	1	0	1
3	5	1	1
4	2	0	2
5	6	1	2
6	3	0	3
7 (highest)	7	1	3

The TX port queue (0-7) inside ICSSG that is used to transmit the packet from the ICSSG firmware to the PHY can be specified in the *TxPktTc* field of the EMAC_PKT_DESC_T passed in in the *emac_send* API call. In other words, the application can select which TX port queue to use.

Future Development: The application can set the TxPktTc field to 0xFF and in this case, the firmware will determine which port queue to use based on the priority REGEN(remap) and PORT PRIORITY mapping that is configured for the host port.

The following sequence occurs during *emac_send* API call:

1. LLD/driver maintains hardware TX descriptor free linked list (in software) setup at time of *emac_open*
2. At time of *emac_send* API call, LLD will pop a free TX descriptor from free linked list and populate free TX descriptor with packet length, pointer to packet buffer and any META data that is passed in the application descriptor. This provides ZERO copy transfer of data. ZERO copy is achieved by transferring ownership of the passed in descriptor and linked packet buffer to the LLD which is directly "linked" to the TX descriptor which is queued on the Transmit Submit ring. Note that no "memcpy" is performed by the driver during *emac_send* API call.
3. LLD will push the TX descriptor using UDMA ring queue API to the specified transmit submit ring associated with the port number passed into the API call.
4. The *emac_send* will return failure codes to the calling application based on the failure encountered as follows:
    a. Port is closed : EMAC_DRV_RESULT_ERR_PORT_CLOSED
    b. Unable to submit the packet for transmission on the transmit ring due to the ring being full (in this case the buffer ownership is returned to the application): EMAC_DRV_RESULT_ERR_UDMA_RING_ENQUEUE
    c. No free TX descriptor available: EMAC_DRV_RESULT_ERR_NO_FREE_DESC

d. Invalid packet channel specified in EMAC_PKT_DESC: EMAC_DRV_RESULT_ERR_INVALID_CHANNEL

NOTE: The DMA is used to move the packet from the host owned buffer to ICSSG firmware, which then will copy the packet to the TX port queue. Thus the DMA for TX packets in TX ring is controlled by firmware and firmware will not allow the DMA of the packet if there is no room in TX port queue in ICSS. So packets will remain in the TX ring in this event and firmware will service another TX channel.

7.4.5 TX Software Descriptor Return Queue Processing

Note that at the time of *emac_send()*, the software descriptor passed in's ownership is given to the driver and needs to returned back to the calling application as specified in the section above. The following mean is provided by the driver to provide the software descriptor back to the calling application.

- emac_poll_ctrl(port_num, rxPktRings, rxMgmtRings, txRings) where txRings field is a bitmap of TX channel completion rings to poll. API will directly query the TX channel completion ring and invoke the TX callback if packet is present in the completion ring. The following defines should be used when invoking the *emac_poll_ctrl* API for *txRings* argument:

```
EMAC_POLL_TX_COMPLETION_RING1      /* poll TX completion events for channel 0
EMAC_POLL_TX_COMPLETION_RING2      /* poll TX completion events for channel 1
EMAC_POLL_TX_COMPLETION_RING3      /* poll TX completion events for channel 2
EMAC_POLL_TX_COMPLETION_RING4      /* poll TX completion events for channel 3
EMAC_POLL_TX_COMPLETION_ALL        /* poll TX completion events for all channels
```

7.4.6 TX Time Stamp

The *emac_send()* API will allow a packet to be marked as TX timestamp required and will allow the application to provide a piece of opaque data (i.e. timestamp id) that can be used to associate a TX timestamp with the packet when the TX timestamp is delivered later. Application will need to register a callback at time of *emac_open* which driver will call to provide the timestamp. The following is the prototype of the callback:

```
typedef void EMAC_TX_TS_CALLBACK_FN_T
(
  uint32_t          port_num,
   /**< EMAC port number                                    */
  uint32_t ts_id,
   /**< timestamp id to correlate TS response with TX request      */
  uint64_t ts
   /**< 64 bit timestamp provided by ICSSG FW                    */
   bool isValid;
   /**< flag to indicate if packet was transmitted and timestamp is valid */
);
```

The callback to the application to free the TX descriptor is completely independent of the callback to the application for TX timestamp response. There is no synchronization between when the TX packet descriptor is returned to the application via the packet free callback and when the TX Timestamp response callback is issued.

The application should not re-use a TX packet descriptor which has a pending TX timestamp request until it receives the TX timestamp response even though the TX packet descriptor has been returned to the application via the packet free callback.

In order to request the TX timestamp, the application will need to update the following fields of the EMAC_PKT_DESC_T when calling the *emac_send* API:

1. Update Flags field in EMAC_PKT_DESC_T with EMAC_PKT_FLAG_TX_TS_REQ
2. Update ts_id field in EMAC_PKT_FLAG_TX_TS_REQ with 32 bit id which will be returned with timestamp and can be used by application to correlate TX timestamp request with response

In order to retrieve the timestamp, the application will need to use the *emac_poll_ctrl* API as follows:

- emac_poll_ctrl(port_num, rxPktRings, rxMgmtRings, txRings) where rxMgmtRings is a bitmap of RX Management rings to poll. If timestamp management packet is available, registered TX timestamp callback will be invoked The following define should be used when invoking the *emac_poll_ctrl* API for rxMgmtRings argument : EMAC_POLL_RX_MGMT_TX_TS or EMAC_POLL_RX_MGMT_RING3

7.4.7 Packet RX

The driver supports multiple receive rings per port. In the case of ICSSG Switch, 9 receive rings are supported. Currently, only 8 of the receive rings are used and packets are directed to the rings based on the PCP to port queue mapping as specified by the following IOCTL: EMAC_IOCTL_PORT_PRIO_MAPPING_CTRL. For ICSSG Dual MAC, 1 receive ring is currently supported.

When a packet is received, the EMAC driver will convert the packet descriptor received to the application managed packet descriptor format (EMAC_PKT_DESC_T) and pass it to the application by calling the rx_pkt_cb() callback function. The receive ring the packet arrives on will be updated in the PktChannel field of the application managed packet descriptor being returned via the callback.
Registration of rx_pkt_cb() is done at time of *emac_open()* API call. For both mode of operation specified below, rx_pkt_cb() will get called to provide the packet to the application.

For receive packets, the following 2 modes of operation will be supported and can be configured at time of *emac_open()* for specified port (note the default mode is POLL).

1. EMAC_MODE_INTERRUPT: *emac_poll_pkt*() API will PEND on a SEMAPHORE which is posted by RX ISR(ISR registration/SEMAPHORE creation done at time of *emac_open*), invoke RX callback and again PEND on SEMAHPORE. This is a blocking API call and will only return to user application if PORT status is closed. Task context is required in INTERRUPT mode.

NOTE: Interrupt mode of operation is not currently supported, will be made available in future release.

2. EMAC_MODE_POLL: *emac_poll_ctrl*() API will directly query the RX completion queue and invoke the RX callback if packet is received. Will return to user application after each *emac_poll_ctrl*() call.

7.4.8 RX Time Stamp

The 64 bit RX timestamp can be extracted from the psinfo[] field of the EMAC_CPPI_DESC_T hardware descriptor which is dequeued from the UDMA ring. psinfo[1] contains the upper 32 bits of timestamp and psinfo[0] contains lower 32 bits of timestamp.
The RxTimeStamp field of the EMAC_PKT_DESC_T will be updated with the 64 bit timestamp from the hardware descriptor and provided for each packet received and provided to the calling application via RX callback.

7.4.9 New APIs

API to retrieve ICSS-G statistics. :

1. EMAC_DRV_ERR_E *emac_get_statistics_icssg*(uint32_t port_num, EMAC_STATISTICS_ICSS-G_T *p_stats, bool clear)

Statistics will include ICSSG hardware and PA_STATS counters supported by firmware. Please refer to EMAC_STATISTICS_ICSSG_T in emac_drv.h for details.

NOTE
1. Some of the hardware stats saturates much faster due to 16-bit counter in ICSS-G hardware
2. PA STATS counters are currently not supported will be available starting MS3 release per current plans.

API to poll receive packet rings, receive management response rings and tx completion rings. Note that a task context is required to make this call and this function does not return until all of the rings per the configured bitmaps have been examined.
Also note that not all rings need to be polled when calling *emac_poll_ctrl* API and should be based on application use case to achieve optimum performance.

2. EMAC_DRV_ERR_E emac_poll_ctrl(uint32_t port_num, uint32_t rxPktRings, uint32_t rxMgmtRings, uint32_t txRings)

   rxPktRings is a bitmap of which packet completion rings to poll. Please refer to EMAC_POLL_RX_PKT_RINGS enum for configuration values. To poll multiple rings, these enum values can be “orred” together. First ring (EMAC_POLL_RX_PKT_RING1) is not used by firmware, The assignment of packets to rings EMAC_POLL_RX_PKT_RING2 - EMAC_POLL_RX_PKT_RING9 is based on EMAC_IOCTL_PORT_PRIO_MAPPING_CTRL configuration.

   rxgmtRings is a bitmap of which management rings to poll. Please refer to EMAC_POLL_RX_MGMT_RINGS enum for configuration values. To poll multiple

rings, these enum values can be “orred” together. First ring (EMAC_POLL_RX_MGMT_RING1) is not used by firmware, to receive mgmt response from FW, use EMAC_POLL_RX_MGMT_RING2 and to receive transmit timestamp response from FW, use EMAC_POLL_RX_MGMT_TX_TS or EMAC_POLL_RX_MGMT_RING3.

txRings is a bitmap of which packet transmit completion rings to poll. Please refer to EMAC_POLL_TX_COMPL_RINGS enum for configuration values. To poll multiple rings, these enum values can be “orred” together.

EMAC_POLL_TX_COMPLETION_RING1 is the lowest priority TX channel and EMAC_POLL_TX_COMPLETION_RING4 is the highest priority TX channel..

API for issuing IOCTL commands for ICSSG ports

3. EMAC_DRV_ERR_E *emac_ioctl*(uint32_t port_num, EMAC_IOCTL_CMD emacIoctlCmd, EMAC_IOCTL_PARAMS *emacIoctlParams)

Callback API for receiving IOCTL command response from FW

4. EMAC_RX_MGMT_CALLBACK_FN_T

```
typedef void EMAC_RX_MGMT_CALLBACK_FN_T
(
  uint32_t                          port_num,
  /**< EMAC port number  */
  EMAC_IOCTL_CMD_RESP_T*       pCmdResp
  /**< Pointer to the IOCTL command reponse                */
);
```

Application will need to register this callback function with driver at time of *emac_open()* by populating rx_mgmt_response_cb of EMAC_OPEN_CONFIG_INFO_T. The

7.4.10 IOCTL API Details

IOCTL can be classified into the following 2 types:

1. Application invokes IOCTL which results in driver issuing MMR update, non-blocking where IOCTL is synchronous in nature and executes immediately with return code status.

2. Application invokes an IOCTL which results in the following which are both asynchronous calls :
    a. driver issuing management (MGMT) message using Hardware Queue Manager. this does not complete until MGMT response is returned to calling application by the firmware..
    b. driver updates specified location in PRU-ICSS DMEM with IOCTL command. This does not complete until driver read the PRUI-ICSS DMEM location to verify the command has completed.

As part of the IOCTL, a sequence number is used for each IOCTL which is returned to the calling application to correlate an IOCTL request with a response. NOTE: At any given time, only 1 IOCTL request can be outstanding. If 1 IOCTL request is in progress and application issues a 2nd one, it will get rejected with error code EMAC_DRV_RESULT_IOCTL_ERR. If the driver is able to issue the IOCTL request to the FW, it will return EMAC_DRV_RESULT_IOCTL_IN_PROGRESS.

The result of the IOCTL call is made available to the calling application with EMAC_RX_MGMT_CALLBACK_FN_T. This is a callback function the application is required to register with the driver at time of *emac_open()*. Refer to the *status* field of EMAC_IOCTL_CMD_RESP_T

```
typedef void EMAC_RX_MGMT_CALLBACK_FN_T
(
  uint32_t              port_num,
  /**< EMAC port number                                    */
  EMAC_IOCTL_CMD_RESP_T*     pCmdResp
  /**< Pointer to the IOCTL command reponse                */
);
```

NOTE: IOCTL tables below will indicate IOCTL type to be either synchronous or asynchronous, also provide details of the *status* field of EMAC_IOCTL_CMD_RESP_T.

Refer to the following ladder diagram for details of asynchronous IOCTL call flow when using HWQ's.

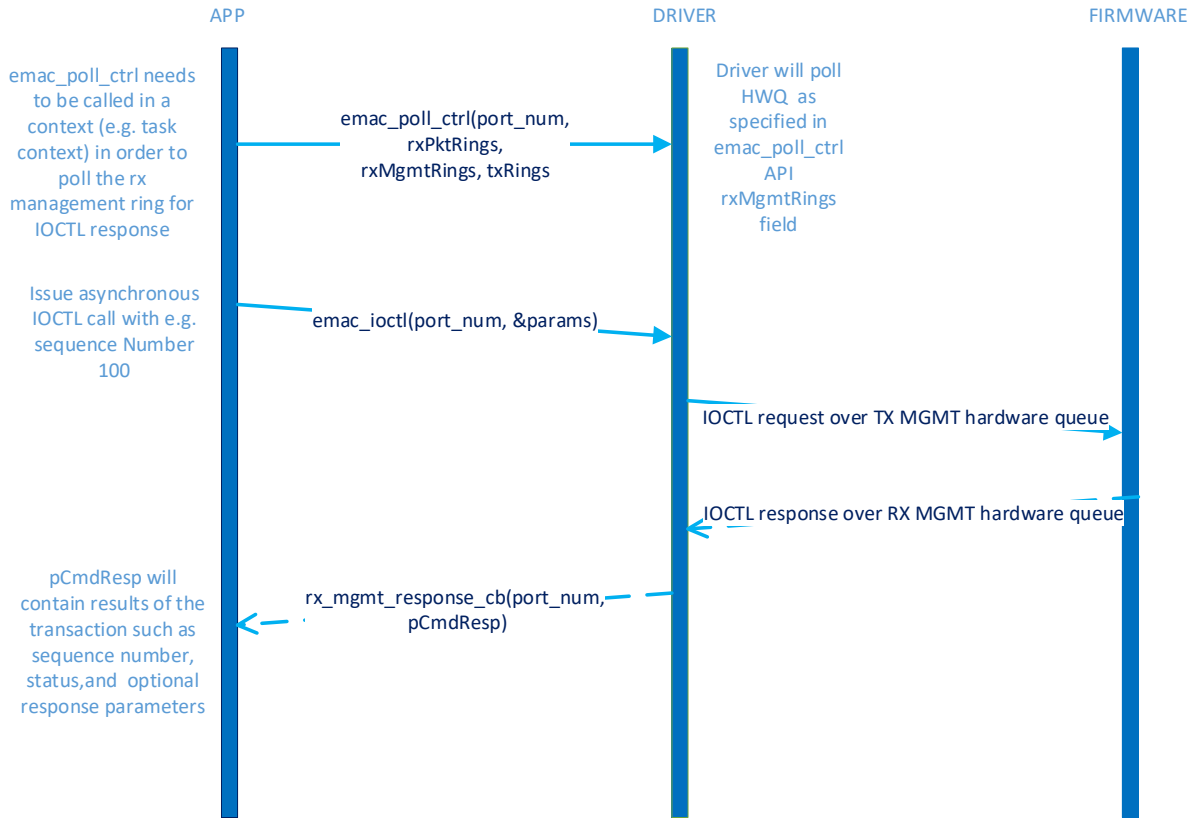


Figure 7 Asynchronous IOCTL Call Flow

The tables below provide a list of IOCTLS currently supported for switch and dual mac use case.

7.4.10.1 Switch Use Case

The following virtual ports should be used when making IOCTL calls as indicated in the table below where "X" denotes the ICSSG instance:

EMAC_ICSSGX_SWITCH_PORT0
➔ Host Port

EMAC_ICSSGX_SWITCH_PORT1
➔ ETH0/SW0 configuration

EMAC_ICSSGX_SWITCH_PORT2
➔ ETH1/SW1 configuration

EMAC_ICSSGX_SWITCH_PORT
➔ Switch centric configuration

IOCTL Command/Sub Command	Description	IOCTL Parameters	RETURN TYPE
EMAC_IOCTL_FDB_ENTRY_CTRL/ EMAC_IOCTL_FDB_ENTRY_ADD Type: asynchronous IOCTL MGMT message using Hardware Queue Manager to FW, MGMT response from FW is received via Hardware Queue Manager NOTE: To program FDB entry as a special management multicast frame set the block and secure bits in FID_C2. The touched bit in FID_C2 must not be set. This bit is used by PRU internally. The valid bit in FID_C2 must always be set. Please refer FID_C2 Bitfield in AM654x_PROFINET_Switch_Non_Real_time_Interface_Design pdf. fdbEntry field (also the FID_C2) is now an array of 2 and it's possible to assign two values to the two physical ports. Index 0 is for EMAC_ICSSGX_SWITCH_PORT1 and Index 1 is for EMAC_ICSSGX_SWITCH_PORT2.	Add forward data base entry to internal ICSSG memory.	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT **EMAC_IOCTL_FDB_ENTRY:** uint8_t mac[6] int16_t vlanId vlanId Range: 0 to 4095 uint8_t fdbEntry[2]	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success EMAC_DRV_RESULT_IOCTL_ERR_INVALID_VLAN_ID on vlanId out of range. EMAC_DRV_RESULT_IOCTL_ERR_SEND_MGMT_MSG if unable to send MGMT message over Hardware Queue Manager Status field values returned via EMAC_RX_MGMT_CALLBACK_FN_T : 0x1: returned on success. 0x3: returned if an ageable FDB entry is removed in order to ADD the new entry. Aged out entry will be returned to the in the 1st 2 bytes of the *respParams* field of the

			EMAC_IOCTL_CMD_RESP_T. 0x10: returned on error (no free entry available)
EMAC_IOCTL_FDB_ENTRY_CTRL/ EMAC_IOCTL_FDB_ENTRY_DEL Type: asynchronous IOCTL MGMT message using Hardware Queue Manager to FW, MGMT response from FW is received via Hardware Queue Manager	Delete forward data base entry from internal ICSSG memory	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT **EMAC_IOCTL_FDB_ENTRY:** uint8_t mac[6] int16_t vlanId vlanId Range: 0 to 4095 uint8_t fdbEntry Note: No need to populate fdb_entry field	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success EMAC_DRV_RESULT_IOCTL_ERR_INVALID_VLAN_ID on vlanId out of range. EMAC_DRV_RESULT_IOCTL_ERR_SEND_MGMT_MSG if unable to send MGMT msg over Hardware Queue Manager Status field values returned via EMAC_RX_MGMT_CALLBACK_FN_T : 0x1: returned on success. 0x10: returned on error (entry to delete not found)
EMAC_IOCTL_FDB_ENTRY_CTRL/ EMAC_IOCTL_FDB_ENTRY_DELETE_ALL or EMAC_IOCTL_FDB_ENTRY_AGEABLE Type: asynchronous IOCTL MGMT message using Hardware Queue Manager to FW, MGMT response from FW is received via Hardware Queue Manager **For future deliverable, subject to change.**	Delete all forward data base entries from internal ICSSG memory	**PORT_NUM:** EMAC_SWITCH_PORT	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success Status field values returned via EMAC_RX_MGMT_CALLBACK_FN_T : 0x1: returned on success. 0x10: returned on error
EMAC_IOCTL_FDB_AGEING_TIMEOUT_CTRL Type: synchronous IOCTL MMR update, non-blocking	Configure FDB ageing interval (in nanoseconds)	**PORT_NUM:** EMAC_SWITCH_PORT **EMAC_IOCTL_FDB_AGEING_INTERVAL** uint64_t fdbAgeingInterval	EMAC_DRV_RESULT_OK on success
EMAC_IOCTL_PORT_STATE_CTRL/ EMAC_IOCTL_PORT_STATE_DISABLE Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion.	Place PORT is disabled state	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success Status field values returned via EMAC_RX_MGMT_CALLBACK_FN_T : 0x1: returned on success.
EMAC_IOCTL_PORT_STATE_CTRL/ EMAC_IOCTL_PORT_STATE_BLOCKING	Place PORT is blocking state	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success

Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion.		EMAC_ICSSGX_SWITCH_PORT 2	Status field values returned via EMAC_RX_MGMT_ CALLBACK_FN_T : 0x1: returned on success.
EMAC_IOCTL_PORT_STATE_CTRL/ EMAC_IOCTL_PORT_STATE_FORWARD Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion	Place PORT is forwarding state	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESUL T_IOCTL_IN_PROG RESS on success Status field values returned via EMAC_RX_MGMT_ CALLBACK_FN_T : 0x1: returned on success.
EMAC_IOCTL_PORT_STATE_CTRL/ EMAC_IOCTL_PORT_STATE_FORWARD_WO_L EARNING Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion	Place PORT is forwarding state without learning	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESUL T_IOCTL_IN_PROG RESS on success Status field values returned via EMAC_RX_MGMT_ CALLBACK_FN_T : 0x1: returned on success.
EMAC_IOCTL_VLAN_CTRL/ EMAC_IOCTL_VLAN_SET_DEFAULT_TBL Type: synchronous IOCTL MMR update, non-blocking	Update ICSSG shared memory with default vlan fid table entries (4096 entries set to default settings)	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT	EMAC_DRV_RESUL T_OK on success
EMAC_IOCTL_VLAN_CTRL/ EMAC_IOCTL_VLAN_SET_ENTRY Type: synchronous IOCTL MMR update, non-blocking	Set entry in vlan table for specified vlan id value where vlan id is from 0 to 4095).	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT **EMAC_IOCTL_VLAN_FID_EN TRY** int16_vlanId vlanId Range: 0 to 4095 EMAC_IOCTL_VLAN_FID_PAR AMS vlanFidPrams –used to populate vlan_fid and vlan_info	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_IOCTL_ERR_INV ALID_VLAN_ID on failure
EMAC_IOCTL_VLAN_CTRL/ EMAC_IOCTL_VLAN_SET_DEFAULT_VLAN_ID Type: synchronous IOCTL MMR update, non-blocking	Set default VLAN ID and PCP bits for specified switch port.	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 0(host port) or EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_VLAN_DEFAU LT_ENTRY** int16_vlanId vlanId Range: 0 to 4095 int8_t pcp range 0-7	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_IOCTL_ERR_INV ALID_VLAN_ID on failure
EMAC_IOCTL_VLAN_CTRL/	Get entry in vlan table for	**PORT_NUM:**	EMAC_DRV_RESUL

EMAC_IOCTL_VLAN_GET_ENTRY Type: synchronous IOCTL MMR update, non-blocking	specified vlan id value where vlan id is from 0 to 4095).	EMAC_ICSSGX_SWITCH_PORT **EMAC_IOCTL_VLAN_FID_ENTRY** to be populated by the driver.	T_OK on success EMAC_DRV_RESULT_IOCTL_ERR_INVALID_VLAN_ID on failure
EMAC_IOCTL_VLAN_CTRL/ EMAC_IOCTL_VLAN_AWARE_MODE Type: asynchronous IOCTL MMR update and Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion. **For future deliverable, subject to change.**	Enable/disable VLAN aware mode.	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT , enable(1), disable(0)	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success
EMAC_IOCTL_PRIO_REGEN_CTRL Type: synchronous IOCTL MMR update, non-blocking	Configure the priority regeneration table for a port including the host port.	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 OR EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_PRIO_REGEN_MAP:** 8-byte priority regen array indexed by PCP value. Index 0 of the array corresponds to PCP 0 so if you want to change PCP 0 to 7 then you would write a value of 7 at index 0	EMAC_DRV_RESULT_OK on success
EMAC_IOCTL_PORT_PRIO_MAPPING_CTRL Type: synchronous IOCTL MMR update, non-blocking	Configure the mapping of PCP to port queue. Also configures the mapping of PCP to CPPI flow/host receive ring.	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_PORT_PRIO_MAP:** 8-byte port priority to port queue mapping indexed by PCP value. Also the 8-byte priority to CPPI flow/receive ring. One-to-one mapping from PCP to output Queue is managed using FT3[0:7] and Classifier[0:7] MMRs.	EMAC_DRV_RESULT_OK on success
EMAC_IOCTL_ACCEPTABLE_FRAME_CHECK_CTRL/EMAC_IOCTL_ACCEPTABLE_FRAME_CHECK_ONLY_VLAN_TAGGED Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion. .	Admit only VLAN-tagged frames	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success Status field values returned via EMAC_RX_MGMT_CALLBACK_FN_T : 0x1: returned on success.
EMAC_IOCTL_ACCEPTABLE_FRAME_CHECK_CTRL/ EMAC_IOCTL_ACCEPTABLE_FRAME_CHECK_ONLY_UN_TAGGED_PRIO_TAGGED	Admit Only Untagged and Priority-tagged frames	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success

Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion. .			Status field values returned via EMAC_RX_MGMT_CALLBACK_FN_T : 0x1: returned on success.
EMAC_IOCTL_ACCEPTABLE_FRAME_CHECK_CTRL/ EMAC_IOCTL_ACCEPTABLE_FRAME_CHECK_ALL_FRAMES Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion.	Admit all frames (default setting)	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success Status field values returned via EMAC_RX_MGMT_CALLBACK_FN_T : 0x1: returned on success.
EMAC_IOCTL_INTERFACE_MAC_CONFIG Type: synchronous IOCTL MMR update, non-blocking	Interface MAC address configuration in hardware MMR's.	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 0 (host port) or EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_MAC_ADDR_T:** uint8_t addr[EMAC_MAC_ADDR_LENTH]; where EMAC_MAC_ADDR_LENTH is 6	EMAC_DRV_RESULT_OK on success EMAC_DRV_RESULT_INVALID_PORT on failure
EMAC_IOCTL_SAV_CHECK_CTRL Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion. **For future deliverable, subject to change.**	Source address violation check enable/disable control.	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 enable(1), disable(0)	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success EMAC_DRV_RESULT_IOCTL_ERR on failure
EMAC_IOCTL_CUT_THROUGH_CTRL Could use a classi to determine if a packet type is eligible for cut-through. Could also be 1 byte in SMEM per port which FW reads to determine if a packet type is eligible for cut-through **For future deliverable, subject to change.**	TBD	TBD	TBD
EMAC_IOCTL_HOST_TX_RATE_LIMITER_CTRL **For future deliverable, subject to change.**	TBD	TBD	TBD
EMAC_IOCTL_HOST_RX_RATE_LIMTER_CTLR **For future deliverable, subject to change.**	TBD	TBD	TBD
EMAC_IOCTL_PKT_TO_FLOW_CLASSI_CTRL **For future deliverable, subject to change.**	TBD	TBD	TBD
EMAC_IOCTL_UC_FLOODING_CTRL/ EMAC_IOCTL_PORT_UC_FLOODING_ENABLE Type: asynchronous IOCTL Driver will update update the DMEM location to	Enable flooding of unknown unicast packets to host port	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success EMAC_DRV_RESULT_INVALID_PORT

convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion. .			on failure
EMAC_IOCTL_UC_FLOODING_CTRL/ EMAC_IOCTL_PORT_UC_FLOODING_DISABLE Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion. .	Disable flooding of unknown unicast packets to host port	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESUL T_IOCTL_IN_PROG RESS on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_TX_ENABLE Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion.	Enable pre-emption on TX	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESUL T_IOCTL_IN_PROG RESS on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_TX_DISABLE Type: asynchronous IOCTL Driver will update update the DMEM location to convey the command, and poll the DMEM location at time of emac_poll_ctrl API for completion. .	Disable pre-emption on TX	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESUL T_IOCTL_IN_PROG RESS on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_GET_TX_ENABLE_ST ATUS Type: synchronous IOCTL, non-blocking	Get status of pre-emption on TX	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_PREEMPTION _ENTRY:** **premt_tx_enabled_status** field will be populated by the driver as follows **:** 1 if active, 0 if not active	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_GET_TX_ACTIVE_STA TUS Type: synchronous IOCTL, non-blocking	Get status of weather pre-emption is active	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_PREEMPTION _ENTRY:** **premt_tx_active_status** field will be populated by the driver as follows **:** 1 if active, 0 if not active	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_VERIFY_ENABLE Type: synchronous IOCTL MMR update, non-blocking	Enable verify state machine	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_VERIFY_DISABLE	Disable verify state machine	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL

Type: synchronous IOCTL MMR update, non-blocking		EMAC_ICSSGX_SWITCH_PORT 2	T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_GET_VERIFY_STATE Type: synchronous IOCTL, non-blocking	Get the verify state machine current state	EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_PREEMPTION _ENTRY: preempt_verify_state** enum field will be populated by the driver as follows: STATE_UNKNOWN STATE_INITIAL STATE_VERIFYING STATE_SUCCEEDED STATE FAILED STATE DISABLED	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_GET_MIN_FRAG_SIZE _LOCAL Type: synchronous IOCTL, non-blocking	Get minimum fragment size supported by firmware	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_PREEMPTION _ENTRY: premt_min_fragment_size** field will be populated with min fagment size supported	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_FRAME_PREEMPTION_CTRL/ EMAC_IOCTL_PREEMPT_SET_MIN_FRAG_SIZE _REMOTE Type: synchronous IOCTL MMR update, non-blocking	Configure the minimum non final fragment size supported by remote link partner in units of 64	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_PREEMPTION _ENTRY: premt_min_fragment_size** field will be populated with min fagment size supported	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_INVALID_PORT on failure
EMAC_IOCTL_CUT_THROUGH_PREEMPT_SELE CT Type: synchronous IOCTL MMR update, non-blocking	Configures queues are pre-emptive/express and/or as cut-through/Store&Forward	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_PREMPT_OR_ CUT_THROUGH_MAP –** The struct takes two arrays as inputs pcpPreemptMap is a byte map for each queue where 1 indicates that the queue is a pre-emptive queue. (This is not operational right now)	EMAC_DRV_RESUL T_OK on success EMAC_DRV_RESUL T_INVALID_PORT on failure

		pcpCutThroughMap is similar but for determining if a queue should be cut-through or not	
EMAC_IOCTL_SPECIAL_FRAME_PRIO_CONFIG Type: synchronous IOCTL MMR update, non-blocking	Specifies the queue number to be used for special packets	**PORT_NUM:** EMAC_ICSSGX_SWITCH_PORT 1 or EMAC_ICSSGX_SWITCH_PORT 2 **EMAC_IOCTL_SPECIAL_FRAME_DEFAULT_PRIO –** specifies the queue number to be used for special packets	EMAC_DRV_RESULT_OK on success EMAC_DRV_RESULT_INVALID_PORT on failure

Table 2 Switch IOCTL Commands

7.4.10.2 DUAL MAC Use Case (Single Instance ICSS FW)

For DUAL MAC use case, use the software port number in the IOCTL call as follows:

```
EMAC_ICSSG0_DUALMAC_PORT0  ((uint32_t)0U)  /* ICSSG instance 0 port 0 */
EMAC_ICSSG0_DUALMAC_PORT1  ((uint32_t)1U)  /* ICSSG instance 0 port 1 */
EMAC_ICSSG1_DUALMAC_PORT0  ((uint32_t)2U)  /* ICSSG instance 1 port 0 */
EMAC_ICSSG1_DUALMAC_PORT1  ((uint32_t)3U)  /* ICSSG instance 1 port 1 */
EMAC_ICSSG2_DUALMAC_PORT0  ((uint32_t)4U)  /* ICSSG instance 2 port 0 */
EMAC_ICSSG2_DUALMAC_PORT1  ((uint32_t)5U)  /* ICSSG instance 2 port 1 */
```

IOCTL Command/Sub Command	Description	IOCTL Parameters	RETURN TYPE
EMAC_IOCTL_PROMISCOUS_MODE_CTRL (MMR update, non-blocking) Type: synchronous IOCTL MMR update, non-blocking	Enable/disable promiscuous mode of operation	Port number enable(1), disable(0)	EMAC_DRV_RESULT_OK on success EMAC_DRV_RESULT_IOCTL_ERR on failure
EMAC_IOCTL_PORT_CTRL/ EMAC_IOCTL_PORT_STATE_DISABLE	Place PORT is disabled state	port number	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success EMAC_DRV_RESULT_IOCTL_ERR on failure
EMAC_IOCTL_PORT_CTRL/ EMAC_IOCTL_PORT_STATE_ENABLE	Re-enable the PORT	port number	EMAC_DRV_RESULT_IOCTL_IN_PROGRESS on success EMAC_DRV_RESULT_IOCTL_ERR on failure
EMAC_IOCTL_HOST_TX_RATE_LIMITER_CTRL **For future deliverable, subject to change.**	TBD	TBD	TBD
EMAC_IOCTL_HOST_RX_RATE_LIMTER_CTLR	TBD	TBD	TBD

For future deliverable, subject to change.			
EMAC_IOCTL_PKT_TO_FLOW_CLASSI_CTRL **For future deliverable, subject to change.**	TBD	TBD	TBD

Table 3 Single Instance ICSS Dual MAC FW IOCTL Commands

7.4.11 Platform Specific functions/configuration

Emac_soc_v5.c contains AM65XX SOC specific configuration which includes register address mapping, interrupts, NAVSS/UDMAP receive and transmit UDMA channel configuration. The SOC configuration structure will be defined in emac_soc_v5.h.
For details of the UDMA subsystem, please refer to Migrating_Applications_from_EDMA_to_UDMA_using_TI-RTOS .pdf as listed in the reference section.

7.4.12 Interrupts

Interrupt registration for receive packet is done within the LLD at time of *emac_open* which uses UDMA event registration API. Once interrupt is received, application provided receive packet callback is invoked.

NOTE: Interrupt support is currently available for DUAL MAC use case, support for interrupts at UDMA ring events is required for SWITCH use case and is being tracked by PRSDK-3812.

7.4.13 Multi- Core Support

Still an open issue, most likely APIs will be provided to clone driver context (handles) from master core and deliver to secondary cores for use with common APIs to enqueue/dequeue packets. Being tracked by PRSDK-5052 (am65xx: UDMA LLD: How to run instance of LLD on multiple cores, share handles, etc)

7.5 **EMAC Polling Link Status**

The application should poll the EMAC periodically (for example every 100msec) to monitor the PHY link status change via the MDIO peripheral using the API *emac_poll*(). This should make sure any changes in the link status (link up or down) should be communicated to the other modules using the EMAC LLD. The application can disable the polling for link status in the *emac_open*() API by disabling the MDIO module. Note that this does not apply for Maxwell and MDIO module is always enabled in order to poll for link status.

Function Declaration
EMAC_DRV_ERR_E emac_poll(uint32_t port_num, EMAC_LINK_INFO_T p_info)*

Where
```
typedef struct EMAC_LinkInfo_s
{
  bool              link_status_change;
  /**< True: link status changed, False: link status is not changed      */
```

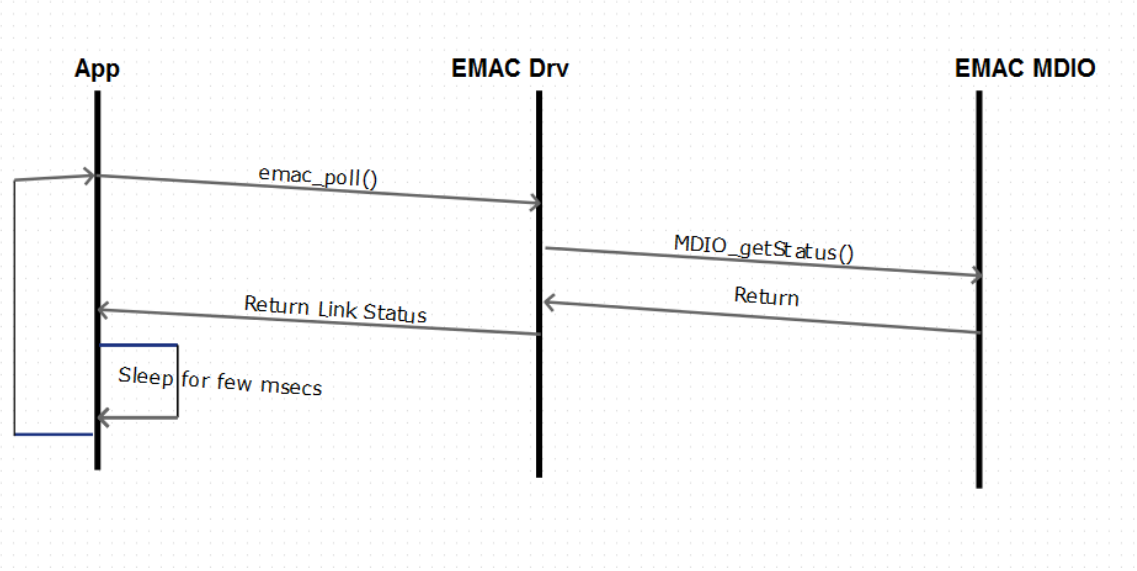
```
    EMAC_LINK_STATUS_T      link_status;
    /**< PHY link status, only valid when link_status_change is TRUE       */

} EMAC_LinkInfo;
```

For Maxwell EMAC driver use case with ICSSG Switch, when *link_status_change* is TRUE, the application will need to convey the link status change to the firmware using EMAC_IOCTL_PORT_STATE_CTRL as follows:

If *link_status* is zero (link down), issue EMAC_IOCTL_PORT_STATE_CTRL with sub-command EMAC_IOCTL_PORT_STATE_DISABLE.

If *link_status* is non-zero(link is up), issue EMAC_IOCTL_PORT_STATE_CTRL with sub-command EMAC_IOCTL_PORT_STATE_FORWARD or EMAC_PORT_BLOCK or EMAC_PORT_FORWARD_WO_LEARNING as appropriate.

An example for using *emac_poll* function is shown below



7.6 Error Handling

Error handling is done inside all the LLD APIs and returns following error codes as applicable.

Error status	Description
EMAC_DRV_RESULT_OK	Indicates successful API call.
EMAC_DRV_RESULT_GENERAL_ERR	Generic error status code returned or an unspecified error
EMAC_DRV_RESULT_INVALID_PORT	Invalid EMAC port number error returned from EMAC APIs
EMAC_DRV_RESULT_NO_CHAN_AVAIL	Error indicating that there are no channels are

	available. It is returned form EMAC_init() API
EMAC_DRV_RESULT_NO_MEM_AVAIL	Error indicating that there is no free memory available. Returned from EMAC_init APIs
EMAC_DRV_RESULT_OPEN_PORT_ERR	Error returned from *EMAC_open* API.
EMAC_DRV_RESULT_CLOSE_PORT_ERR	Error returned from *EMAC_close* API
EMAC_DRV_RESULT_CONFIG_PORT_ERR	Error returned from *EMAC_config* API
EMAC_DRV_RESULT_SEND_ERR	Error returned from *EMAC_send* API
EMAC_DRV_RESULT_POLL_ERR	Error returned form *EMAC_poll* API to indicate poll link status error
EMAC_DRV_RESULT_GET_STATS_ERR	Error returned from *emac_get_statistics* and emac_get_statistics_icssg APIs
EMAC_DRV_RESULT_ISR_ERR	Interrupt service error form emac_int_service
EMAC_DRV_RESULT_IOCTL_ERR	IOCTL command error
EMAC_DRV_RESULT_IOCTL_IN_PROGRESS	IOCTL command error, IOCLT command already in progress
EMAC_DRV_RESULT_IOCTL_ERR_IN_PROGRESS	VLAN ID is specified in IOCTL is invalid, out of range, valid rang is 0 to 4095
EMAC_DRV_RESULT_IOCTL_ERR_PORT_CLOSED	IOCTL command error, port is closed
EMAC_DRV_RESULT_IOCTL_ERR_SEND_MGMT_MSG	Error when sending MGMT message over Hardware Queue Manager to FW
EMAC_DRV_RESULT_IOCTL_IN_PROGRESS	Successful IOCTL API call and IOCTL command is in progress

8 Standards, Conventions and Procedures

8.1 Documentation Standards

Doxygen format is used for documentation in source code.

8.2 Naming conventions

Processor SDK standard naming conventions are used for file and module naming.

8.3 Programming Standards

- C99 standard data types are used in driver implementation.
- MISRA-C coding standards are followed wherever applicable.

8.4 Software development tools

- TI's Code Composure Studio for project build setup.
- Make files for source code compilation and Test Applications

- Doxygen for extracting documentation from source code
- Klocworks for static code analysis

9 IP Feature List Comparison

This section gives the details of feature comparison of different EMAC HW IPs and software support for those IP features.

NOTE: Table entries below marked with "*" are supported but currently not tested.

EMAC IP Features					
		OMAPL137		K2G	
		HW	SW	HW	SW
IP Driver Version		NA	0	NA	1
No. of hardware instance		1	NA	1	NA
Synchronous operations.	10 Mbps	YES	YES	YES	YES *
	100 Mbps	YES	YES	YES	YES *
	1000 Mbps	NO	NA	YES	YES
Standard Media Independent Interface (MII)		YES	YES	YES	YES *
Reduced Media Independent Interface (RMII)		YES	YES	YES	YES
GMII		NO	NA	NO	NA
RGMII		NO	NA	YES	YES
Support quality-of-service (QOS)		2	YES	8	YES *
Ether-Stats and 802.3-Stats statistics gathering.		YES	NA	With RMON Statistic gathering	NA
Transmit CRC generation		YES	NO	YES	NO
Broadcast and Multicast frames selection		YES	YES	YES	YES *
Promiscuous receive mode		YES	YES	YES	YES
Flow control Support		YES	YES	YES	YES
Programmable interrupt logic		YES	NA	YES	NA
CPPI buffer descriptor memory		8k	NA	2k	NA
MDIO module for PHY Management		YES	YES	YES	YES

Wire rate switching (802.1d)		NO	NA	NO	NA
Address Lookup Engine (ALE)	address entries plus VLANs	NO	NA	64	NA
	Wire rate lookup	NO	NA	YES	NO
	Host controlled Time-based aging	NO	NA	YES	NO
	Multiple spanning Tree support	NO	NA	YES	NO
	MAC authentication (802.1x)	NO	NA	YES	NO
	MAC address blocking	NO	NA	YES	NO
	Source port locking	NO	NA	YES	NO
	OUI host accept/deny Featu	NO	NA	YES	NO

	re				
VLAN support		NO	NA	YES	NO
Digital loopback and FIFO loopback modes supported		NO	NA	YES	NO
Emulation Support		NO	NA	YES	TBD
RAM Error Detection and Correction (SECDED)		NO	NA	YES	NO
Programmable transmit Inter-Packet Gap (IPG)		NO	NA	YES	TBD

EMAC IP Features									
		AM335x		AM437x		AM572x		AM6x	
		HW	SW	HW	SW	HW	SW	HW	SW
IP Driver Version		NA	4	NA	4	NA	4	NA	5
No. of hardware instance		2	NA	2	NA	2	NA	1	NA
Synchronous operations.	10 Mbps	YES	YES	YES	YES	YES	YES	YES	YES *
	100 Mbps	YES	YES	YES	YES	YES	YES	YES	YES *
	1000 Mbps	YES	YES	YES	YES	YES	YES	YES	YES
Standard Media Independent Interface (MII)		NO	NA	NO	YES	NO	YES	NO	NO
Reduced Media Independent Interface (RMII)		YES	YES	YES	YES	YES	YES	YES	NO
GMII		YES	YES	YES	YES	YES	YES	NO	NO
RGMII		YES	YES	YES	YES	YES	YES	YES	YES
Support quality-of-service (QOS)		4	YES	4	YES	4	YES	8	YES *
Ether-Stats and 802.3-Stats statistics gathering.		With RMON Statistic gathering	NA	With RMON Statistic gathering	NA	With RMON Statistic gathering	NA	With RMON Statistic gathering	NA
Transmit CRC generation		NO	NA	NO	NA	NO	NA	YES	NO
Broadcast and Multicast frames selection		YES	YES	YES	YES	YES	YES	YES	YES
Promiscuous receive mode		YES	YES	YES	YES	YES	YES	YES	YES
Flow control Support		YES	YES	YES	YES	YES	YES	YES	NO
Programmable interrupt logic		YES	NA	YES	NA	YES	NA	YES	NA
CPPI buffer descriptor memory		8k	NA	8k	NA	8k	NA	NA	NA
MDIO module for PHY Management		YES	YES	YES	YES	YES	YES	YES	YES
Wire rate switching (802.1d)		YES	NO	YES	NO	YES	NO	NO	NA
Address Lookup Engine (ALE)	address entries plus VLANs	1024	NA	1024	NA	1024	NA	64	NA

	Wire rate lookup	YES	NO	YES	NO	YES	NO	YES	NO
	Host controlled Time-based aging	YES	NO	YES	NO	YES	NO	YES	NO
	Multiple spanning Tree support	YES	NO	YES	NO	YES	NO	YES	NO
	MAC authentication (802.1x)	YES	NO	YES	NO	YES	NO	YES	NO
	MAC address blocking	YES	NO	YES	NO	YES	NO	YES	NO
	Source port locking	YES	NO	YES	NO	YES	NO	YES	NO
	OUI host accept/deny Feature	YES	NO	YES	NO	YES	NO	YES	NO
VLAN support		YES	NO	YES	NO	YES	NO	YES	NO
Digital loopback and FIFO loopback modes supported		YES	NO	YES	NO	YES	NO	YES	NO
RAM Error Detection and Correction (SECDED)		NO	NA	NO	NA	NO	NA	NO	NA
Programmable transmit Inter-Packet Gap (IPG)		YES	NO	YES	NO	YES	NO	YES	NO

10 System Design

10.1 Design Approach

The EMAC driver provides a well-defined API layer which allows applications to use the EMAC peripheral to control the flow of packet data from the processor to the PHY and the MDIO module to control PHY configuration and status monitoring.
The EMAC driver is designed to meet the following requirements:

- Support multiple EMAC ports (if available on the device) per core (i.e. A53/R5)
- Support multiple channels per core.
- Support multiple cores to use different channels on the same EMAC port.
- The driver is OS independent and exposes all the operating system callouts via the OSAL layer.
- EMAC example test application provides standard configurations and demonstrates measurable benchmarks.

Platform specific functions are mapped to the platform independent APIs using function table which is given below

```
  /*! Function to open the specified EMAC port */
  EMAC_OpenFxn          openFxn;
 /*! Function to config the specified EMAC port for RX filtering, multicast addresses */
  EMAC_ConfigFxn          configFxn;
  /*! Function to close the specified peripheral */
  EMAC_CloseFxn          closeFxn;
  /*! Function to send packet to network on specified EMAC port */
  EMAC_SendFxn          sendFxn;
  /*! Function to poll link status for specified EMAC port*/
  EMAC_PollFxn     pollFxn;
  /*! Function to get EMAC CPSW port statistics*/
  EMAC_GetStatsFxn     getStatsFxn;
  /*! Function to poll for receive packets  specified EMAC port*/
  EMAC_PollPktFxn     pollPktFxn;
  /*! Function to get EMAC ICSSG port statistics,  IP version 5 only*/
  EMAC_GetStatsIcssgFxn     getStatsIcssgFxn;
  /*! Function to send IOCTL command for specified port, IP version 5 only*/
  EMAC_IoctlFxn       ioctlFxn;
  /*! Function to Poll the driver for specified flow/rings, IP version 5 only */
     EMAC_PollCtrlFxn       pollCtrl;
```

10.2 Dependencies

None

10.3 Decomposition of System

The following is an architecture figure which showcases the EMAC driver architecture:-

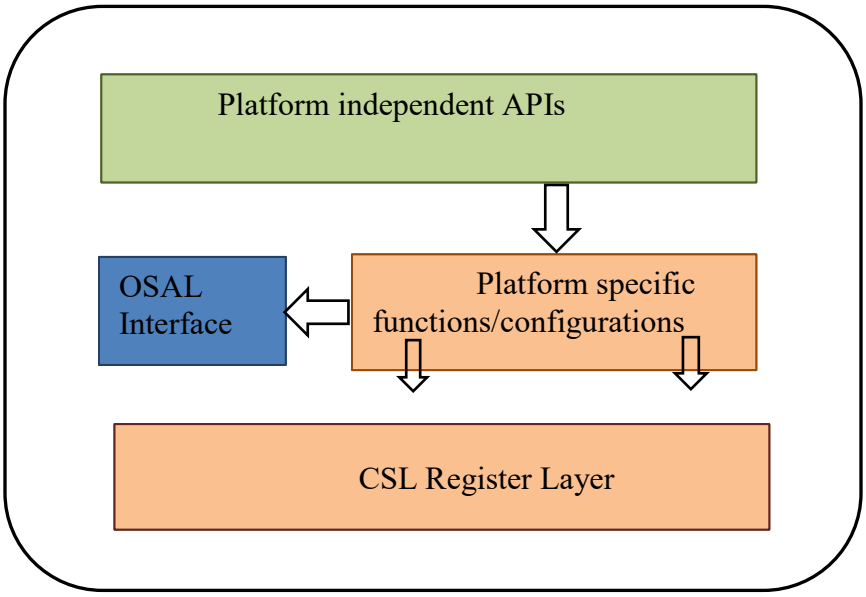


Figure 8 : EMAC LLD Subsystem Block Diagram

The figure illustrates the following key components:-

10.3.1 Platform Independent APIs

EMAC LLD exposes a set of well-defined APIs which are platform independent and common across the platforms. These are the functions which are exposed to application programs.

10.3.2 Platform specific functions/configurations

Platform specific functions implement actual functionality of EMAC LLD for a given platform. These functions can be specific to one or set of platforms. There will be multiple versions of platform specific functions based on the number of platforms supported.

Platform specific configurations will define high-level configurations specific to each platform. This includes register address mapping, interrupts, function table initialization etc. These configurations are included in soc file.

10.3.3 Operating System Abstraction Layer (OSAL)

The EMAC LLD is OS independent and exposes all the operating system callouts via this OSAL layer.

10.3.4 CSL Functional Layer

The EMAC driver uses the CSL EMAC functional layer to program the device IP by accessing the MMR.

10.3.5 CSL Register Layer

The register layer is the IP block memory mapped registers which are generated by the IP owner. The EMAC driver does not directly access the MMR registers but uses the EMAC CSL Functional layer for this purpose.

11 OMAPL13x Integration

This section describes the changes required for adding OMAPL13x platform support to EMAC LLD.

v0 version of EMAC LLD supported on C6657 platform will be used as reference for the OMAPL13x integration.

11.1 Platform Independent API

There will be no change to platform independent APIs during OMAPL13x integration.

11.2 Platform Specific functions/configuration

EMAC_soc.c file will be added which defines platform specific configurations for OMAPL13x.

Gigabit support

There is gigabit speed support for OMAPL13x platform but existing v0 EMAC driver supports gigabit mode through SGMII. Need to create new version of driver based on v0 for OMAPL13x if we need to avoid SOC specific defines in the driver.

DNUM dependencies

DNUM register is used in the EMAC LLD to decide the core number. OMAPL13x platform DNUM register returns a value 1 even though there is only one DSP core which is different from other platforms. LLD changes are needed to handle this case.

11.3 OSAL

No changes are expected for EMAC LLD OSAL for integration of OMAPL13x platform.

11.4 CSL

New version of CSL-RL file is added for OMAPL13x platform.

11.5 Build Setup

Update make files to add support for OMAPL13x platform.