20450 Century Boulevard
Germantown, MD 20874

EMAC Low Level Driver 1.0.2
Software Design Document

Revision B

November 10, 2010

Document License

Contributors to this document

Revision Record	
Document Title: **Software Design Specification**	
Revision	Description of Change
A	Initial Release
B	Added document license for Creative Commons

TABLE OF CONTENTS

1 Scope

This document describes the functionality, architecture, and operation of the EMAC Low Level Driver.

2 References

The following references are related to the feature described in this document and shall be consulted as necessary.

No	Referenced Document	Control Number	Description
1	EMAC PRD		EMAC Component Product Requirements Document
2	MC BIOS C64x SDK PDK		Multi-Core BIOS C64x SDK Product Requirements Document

Table 1. Referenced Materials

3 Definitions

Acronym	Description
API	Application Programming Interface
CSL	Chip Support Library
EMAC	Ethernet Media Access Controller
LLD	Low Level Driver Design
MC-SDK	Multi-Core Software Development Kit
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

Table 2. Definitions

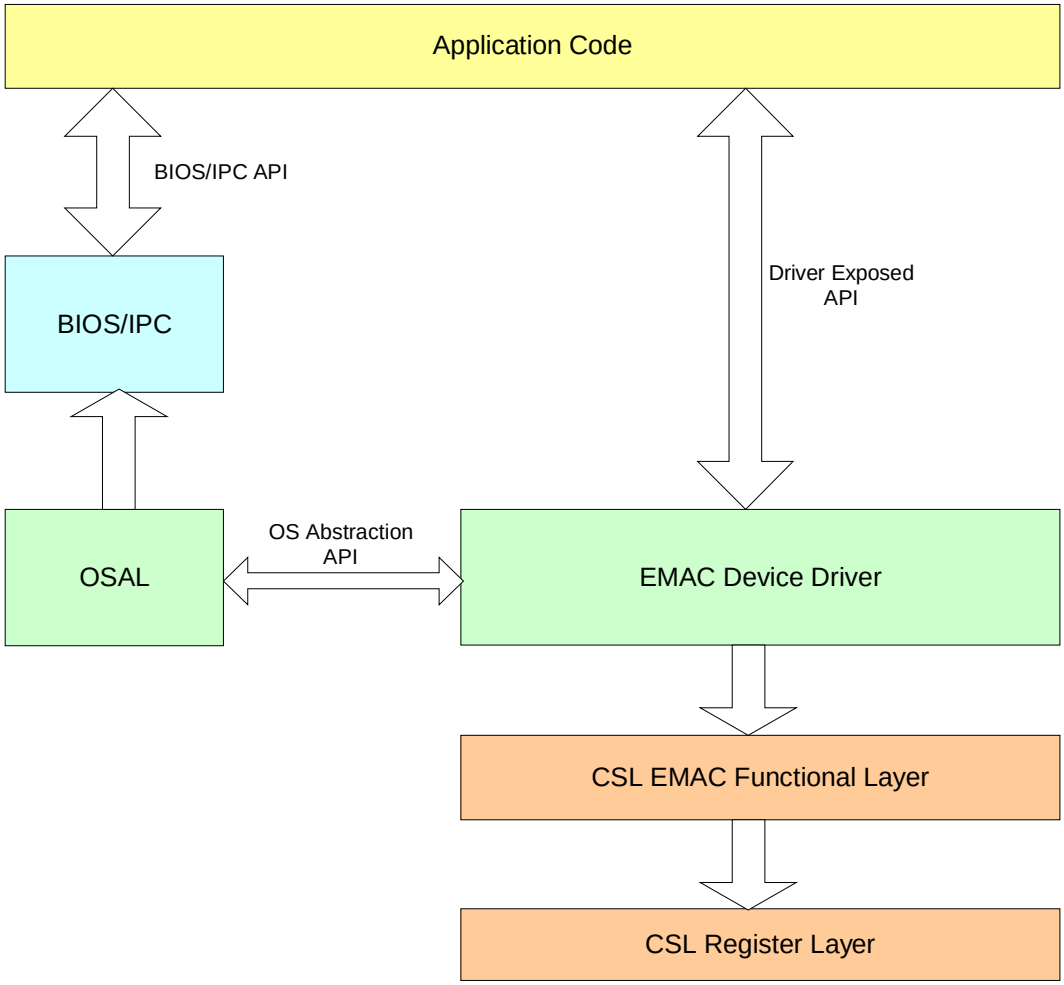
4 Overview

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.
- Support multiple channels/MAC addresses 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 measureable benchmarks.

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



The figure illustrates the following key components:-

1. **EMAC Device Driver**

   The device driver exposes a set of well defined API which is used by the application layer to send and receive data packets via the EMAC peripheral, and configure and monitor the PHY via the MDIO peripheral.

   The driver also exposes a set of well defined OS abstraction API which is used to ensure that the driver is OS independent and portable. The EMAC driver uses the CSL EMAC functional layer for all EMAC MMR accesses.

2. **Application Code**

   This is the user of the EMAC driver and its interface with the driver is through the well defined API set. Applications users use the EMAC driver API's to send and receive data packets via the EMAC peripheral.

3. **Operating System Abstraction Layer (OSAL)**
   The EMAC LLD is OS independent and exposes all the operating system callouts via this OSAL layer.

4. **CSL Functional Layer**

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

5. **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.

5 Design

The EMAC driver is responsible for the following:-

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

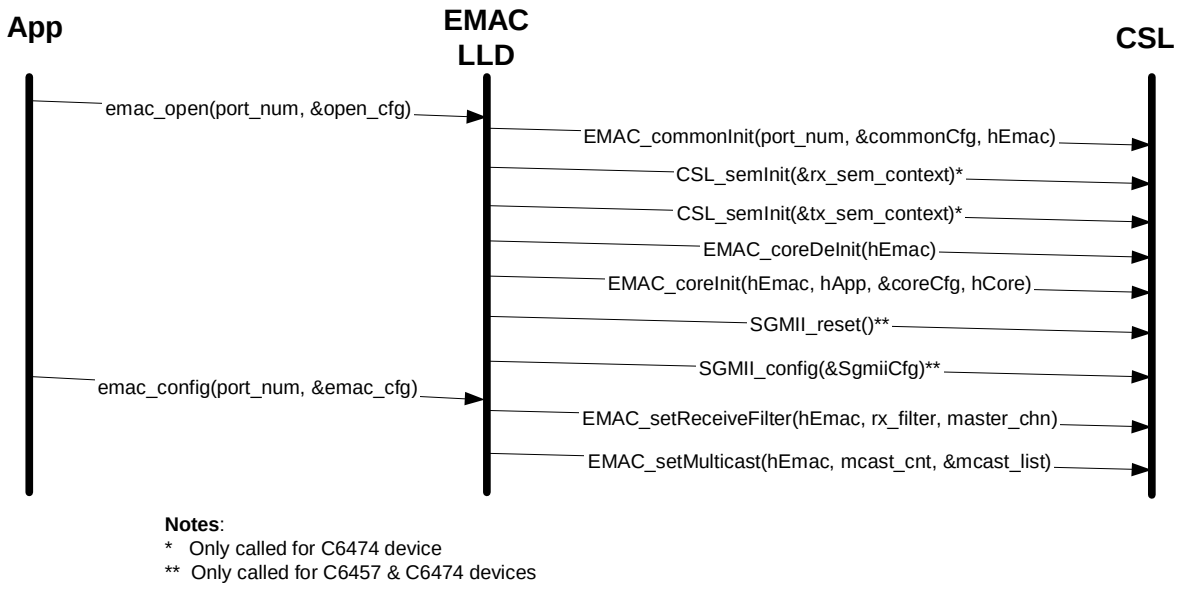
The next couple of sections document each of the above mentioned responsibilities in greater detail:

5.1 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:-



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

- EMAC port number
- EMAC TX/RX packet descriptor queue size
- Maximum packet size
- Number of channels on this port to be used by the core
- MAC address configured for each channel
- Master core flag
- Loopback test flag
- MDIO enable flag
- PHY address to be used
- Call back functions for receive/allocate/free a packet

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 need 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

5.2 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.

5.3 CPPI Packet Descriptor

By default, the EMAC driver selects CPPI RAM for EMAC IP managed Packet Descriptor memory. CSL allocates total 64 packet descriptors for both TX and RX channels. The number of packet descriptors allocated for a RX channel is also configurable by the EMAC driver.

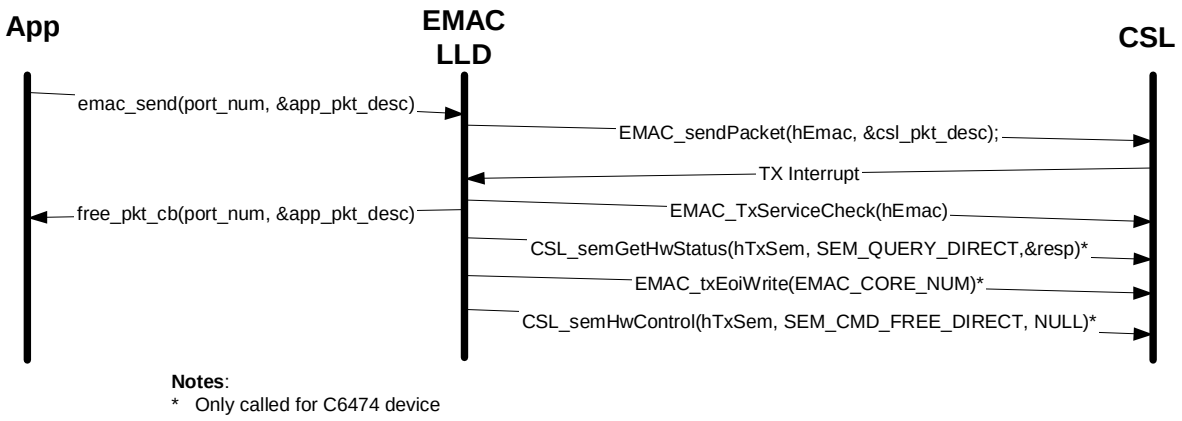
5.4 EMAC Polling

The application polls the EMAC periodically (every 100msec) to monitor the PHY link status change via the MDIO peripheral. The application can disable the polling in the emac_open() API by disabling the MDIO module.

5.5 Packet TX/RX

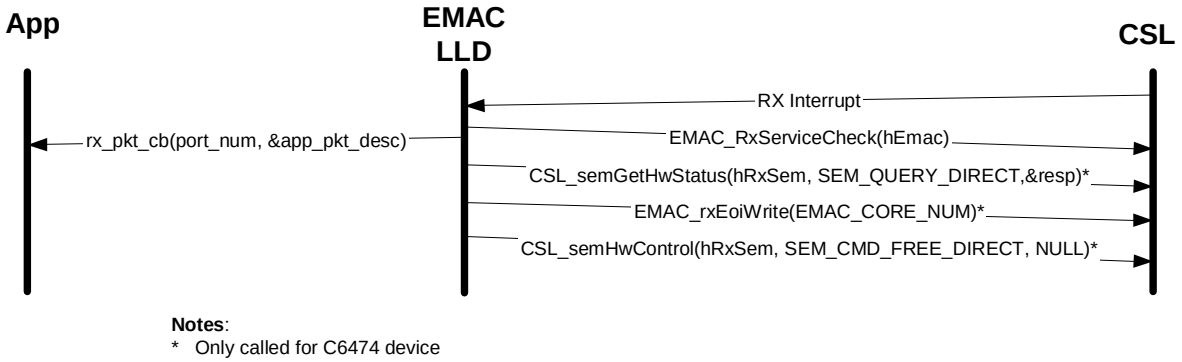
The application can send a packet by calling emac_send () API, the application needs to allocate an application managed packet descriptor from the application queue, copy the packet data and convert it to the EMAC driver managed packet descriptor format.

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



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:-



5.6 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.

5.7 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.

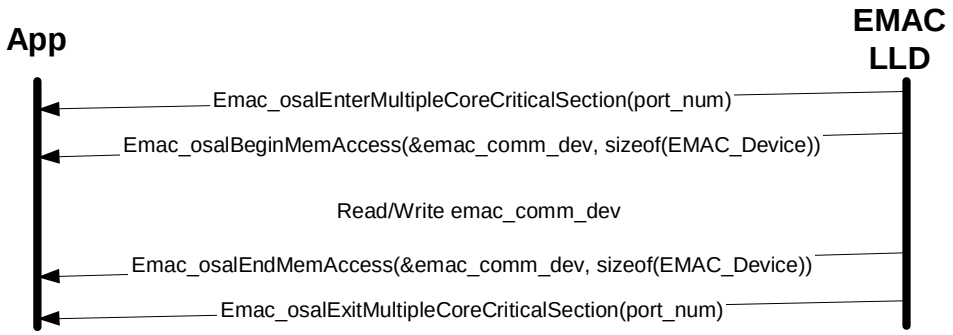
emac_comm_pcb contains common port control block information that is managed by the EMAC driver.

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:-



5.8 Packet Data Cache Coherence Protection

If the application places the packet data buffer to external memory, the application needs to do write back cache operation for the packet data before sending a packet and do an invalidate cache

operation after receiving a packet. The start address of each packet buffer should be aligned to the cache line size of the device by the application.

6 Future Extensions

6.1.1 Local switching of multicast/broadcast packets

The EMAC drive will need to support local switching of multicasting/broadcasting packets received from the master channel to all the channels on all the cores due to a hardware RX filter limitation.

6.1.2 Local switching of packets destined to the device

The EMAC driver will need to support local switching of packets destined to the channels on the same device.