RW-WLAN-nX-SW-DG

Developer Guide
RW-WLAN-nX-SW-DG
Version 1.12
2019-06-27

RivieraWaves
http://www.rivierawaves.com/

Revision History

Version	Date	Revision Description	Author
1.0	2012-10-26	Initial version	Steven L'Her
1.01	2014-03-10	Added the new options available on build command line	Steven L'Her
1.02	2014-11-07	New folder tree	Steven L'Her
1.03	2015-02-10	Few updates in the tools	Steven L'Her
1.04	2015-06-24	Updates in build options	Steven L'Her
1.05	2015-09-30	Updates in build options	Steven L'Her
1.06	2016-02-22	Updates in build options. Added example of build options for typical use cases.	Steven L'Her
1.07	2016-03-31	Added BFMER build option	Steven L'Her
1.08	2016-08-02	Added Mesh, MU TX, TDLS options	Steven L'Her
1.09	2016-12-14	Added some more description on MDM option Added PLATFORM build option	Steven L'Her
1.10	2017-09-08	Updated FW options	Steven L'Her
1.11	2018-09-27	Updated FW options. Added RISC-V platform.	Steven L'Her
1.12	2019-06-27	Added HE options.	Steven L'Her

Table of Contents

1 Overview

1.1 Document Overview

This document is the RW-WLAN-nX Software IP Developer's Guide (DG). This is intended for developers and programmers, who are integrating and porting the RW-WLAN-nX Software IP into their own platform and system.

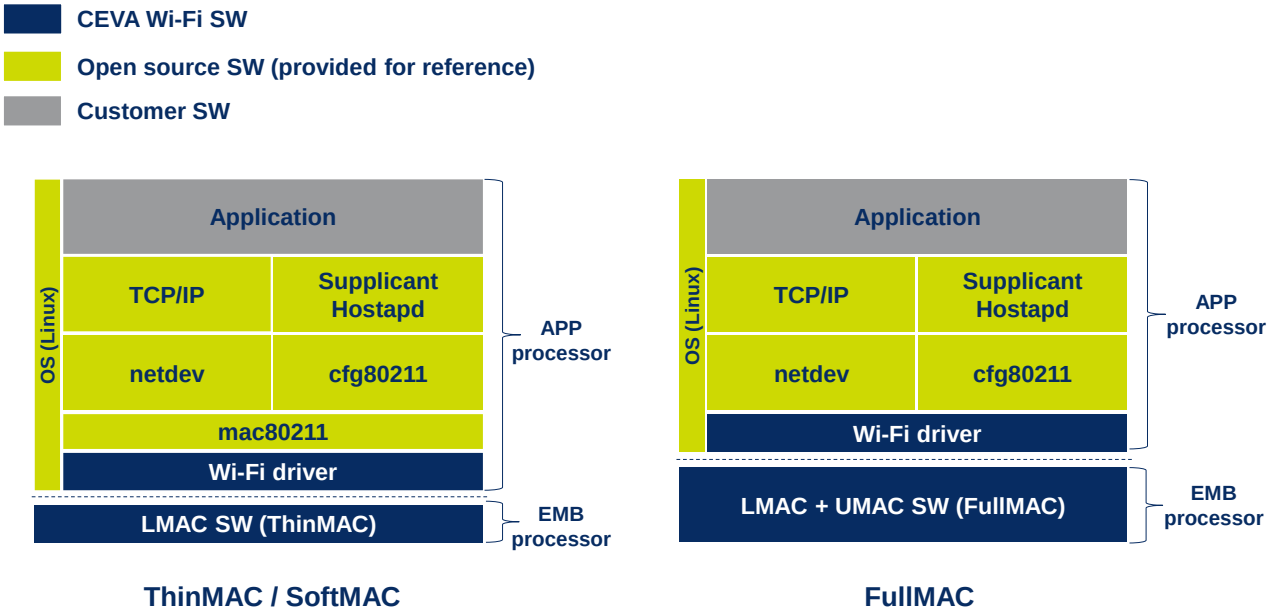
The purpose of this documentation is to provide basic understanding of the architecture and the build process of the Software IP solution of Ceva RivieraWaves.

1.2 Product Overview

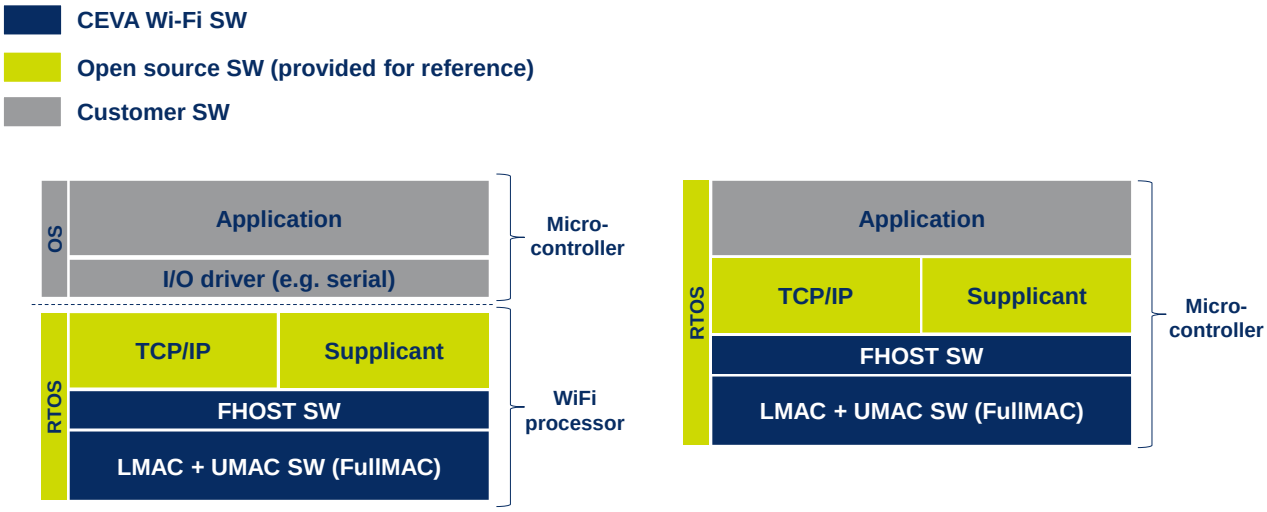
The RW-WLAN-nX SW IP is extremely portable. Written in C, the SW is very readable and organized. It is highly configurable and flexible for any type of application and role. The code size can be trimmed (according to requirements and configuration) and desired behavior can be achieved per specific role.

The recommended IDE is Eclipse with parts of the SW IP customized for specific compilers and microcontrollers.

The MAC SW and drivers can be built for various partitioning, depending on the customer use case. SoftMAC and FullMAC partitioning are based on two CPUs, with a Host running a Linux-like OS.



The two partitioning above are typically used in high performance applications, e.g. for Mobile or Access Point type of products. For IoT products, more integrated partitioning can be optionally proposed:



Micro-controller + WiFi companion chip **Micro-controller including WiFi connectivity**

These two partitioning allow customer running the full WiFi and networking SW (including TCP/IP) on a single CPU. The customer application can then run alone in an external micro-controller, or be even running on the same CPU as the WiFi and networking SW. This allows getting a low-cost and extremely power efficient IoT platform.

2 Content of the Package

The LMAC SW package is located in the MAC_SW folder of the delivered archive. It contains the following folders:

- DOC: All the documents provided in the package
- HOST: The tools code, patches and build scripts for the Linux Host running on Ceva board
- SW: The FW and driver source code and build scripts

The SW folder is composed of several sub-folders used for the MAC SW and Linux driver compilations:

- macsw: This folder contains the source code and tools specific to the MAC SW IP and the different platforms for which the MAC SW can be built.
- rwnx_drv: This folder contains the source code related to the Linux driver and the associated makefile.
- Optionally, additional folders might be present for the Fully Hosted SW partitioning (depends on customer options)

The “macsw” folder is itself subdivided into several subfolders:

- config: This folder contains the build script used for FW compilation.
- ip: This folder contains the protocol stack code.
- modules: This folder contains the modules that could be common to several IPs (kernel, list management, etc.).
- plf: This folder contains the code specific to the different CPU architectures and platform for which the MAC SW can be built. The MAC SW can be built for various CPU architectures:
  - RISC-V
  - Ceva TL4/X1
  - ARM/Cortex
  - Cortus APS3
- tools: This folder contains the different tools used by the build scripts.

This split allows the integration of the RW-WLAN-nX IP to a new platform without the need to modify any part of the IP code. A new platform code has simply to be added to the “plf” folder, based on the different platforms provided as example of the RW-WLAN-nX integration into a platform.

3 Installing the Build Tools

The following tools have to be installed in order to build the RW-WLAN-nX SW IP:

- Python v2.7.10 or newer
- RISC-V toolchain, Cortus toolchain, ARM toolchain or Ceva TL4/X1 toolchain

3.1 Environment Setup

3.1.1 Python Tool

Python can be downloaded from: https://www.python.org/downloads/. The Python version shall be a v2.x.x, not a v3.x.x.

Once downloaded, launch the Python installer.

4 Building the RW-WLAN-NX SW IP for the Platform

4.1 Build commands

The scons script file is responsible for building the binary. In a Windows command shell, go to the **SW/*macsw*/config** folder and input the command below:

python ..\tools\scons.py . ARCH=tl4xx

This will build all modules using Ceva TL4 toolchain into the ***SW/macsw/build/lmac_tl4xx_karst/*** folder.

In case there is a need to tweak the compilation, there are other compile options which the user may use to come up with a desired binary specific to the platform. Below are some of these command line options. The default value for the options is written in **bold**. Some of the options below are available whatever the built FW is (SoftMAC or FullMAC), others are dedicated to one of these flavors.

Options available for both SoftMAC, FullMAC and FullHOST:

- O Optimization level {0, 1, **2**, 3}
- V Verbose build {**0**, 1}
- PRODUCT Type of firmware that is built, i.e LowerMAC only, FullMAC or FullHOST {**lmac**, fmac, fhost}
  - The FullMAC configuration requires the optional RivieraWaves UpperMAC SW
  - The FullHOST configuration requires the FullMAC plus optional FullHOST components
- ARCH CPU architecture to build for {tl4xx, arm, aps3, cortex, **risc-v**}
- CPU CPU to build for. The possible values for this parameter depends on the chosen ARCH
  - aps3 {**aps**}
  - arm {**arm7**, arm9}
  - cortex {**cortex-m0**, cortex-m4}
  - tl4xx {**tl410**, tl411, tl420, tl421}
  - risc-v {**zeroriscy**, ri5cy}
- PLATFORM Platform the FW is built for {virtex6, **virtex7**}
  - virtex6 corresponds to the old reference platform on Dini FPGA board
  - virtex7 corresponds to the reference FPGA platform based on Virtex7 FPGA
- STA Maximum number of peer devices supported {1-40, **10}**
- DBG Debug enable {0, 1 or **2**}
  - 0: No debug enabled
  - 1: Simple assertion message
  - 2: Full assertion message
- BCN Beaconing modes {off, **on**} – Enables the AP mode
- AGG A-MPDU TX support {off, **on**}
- AMSDURX Maximum A-MSDU size supported in RX (**4k**, 8k, 12k}
- SPC Minimum A-MPDU spacing in TX, in µs (1-**16**)
- TXDESC0 Background queue number of TX descriptors {4, **8**, 16, 32, 64}
- TXDESC1 Best-effort queue number of TX descriptors {4, 8, 16, 32, **64**}
- TXDESC2 Video queue number of TX descriptors {4, 8, 16, 32, **64**}
- TXDESC3 Voice queue number of TX descriptors {4, 8, 16, **32**, 64}
- TXDESC4 Beacon queue number of TX descriptors {4, **8**, 16, 32, 64}
- PROF LMAC profiling {off, **on**}
- REC Recovery mechanism {off, **on**}
- DBGDUMP Debug dump forwarded to host in case of error {off, **on**}
- TRACE Trace buffer {off, **on**}
- RADAR Radar detection {**off**, on}
- PS Legacy power-save mode {off, **on**}
- UAPSD Station U-APSD support {off, **on**}
- DPSM Dynamic Power-Save mode {off, **on**}

- P2P Number of P2P connections that can be created in parallel {0-**2**}
- P2P_DBG P2P Debug information support {**off**, on}
- P2P_NOA_GO Notice Of Absence support in P2P GO mode {off, **on**}
- P2P_OPPPS_GO Opportunistic PS mode support in P2P GO node {off, **on**}
- MAC Version of the MAC HW which is used {**v10**, v20, v21}
  - v10 is used with 802.11n/ac MAC HW
  - v20 is a deprecated 802.11ax MAC HW version
  - v21 is used with the 802.11ax MAC HW
- MDM Version of the MODEM which is used {v10, v11, v20, v21, **v22**, v30}
  - v10/v11 correspond to old versions of the MODEM, used with Trident PHY
  - v20 is used with Karst and Aetnensis RF, when modem version (read from VERSION field of HDMVERSION register) is 0
  - v21 is used with modem version 1
  - v22 is used with modem version 2
  - v30 is used with 802.11ax modem
- PHY PHY for which the LMAC is built {trident, elma, **karst**, etc.}
- WAPI WAPI security support {**off**, on}
  - When enabled, this option requires an optional MAC HW block for the WAPI encryption/decryption
- KEYCFG Indicate whether key RAM dynamic mapping shall be used or not {off, **on**} – requires dedicated HW.

The following features are useful only when the system supports 802.11ac:

- AMSDU A-MSDU transmission support {off, **on**}
- BFMEE Beamformee support {off, **on**}. Requires dedicated HW
- BWLEN A-MPDU length adaptation in case of BW drop support {off, **on**}
- BFMER Beamformer support {off, **on**}. Requires dedicated HW.
- MUCNT Maximum number of MU-MIMO users the system can support in TX {**1**-4} – Requires dedicated HW. Value 1 disables the MU-MIMO TX support.

Options available for SoftMAC only:

- HWSCAN Scan handled by LMAC instead of UMAC {off, **on**}
- CMON Connection monitoring support in LMAC {off, **on**}
- MROLE Multi-role support (AP+STA, STA+STA) support {off, **on**}
- AUTOBCN Autonomous beacon transmission by the LMAC {off, **on**}

Options available for FullMAC only:

- VHT VHT support {off, **on**}
- HE HE (802.11ax) support {off, **on**}
- BATX Maximum number of TX block ack agreements we can handle in parallel {**5**}
  - This option is valid only when AGG is on
- BARX Maximum number of RX block ack agreements we can handle in parallel {**5**}
  - Put to 0 to disable BA in RX
- REORDBUF Number of buffers per reordering instance {4-**64**}
- MFP Management frame protection support {**off**, on}
- TDLS TDLS support (off, **on**}
- HSU Support for HW TKIP MIC/BIP accelerator {0, **1**, 2}.
  - 0: Don't use HSU. TKIP MIC and BIP are computed by the FW. This option should be chosen on old MAC HW implementations that don't include the HSU HW module
  - 1: Auto-detection of HSU. If HSU is available, use HSU. Otherwise, use FW for the computation. This option can be useful if the same FW is used on platforms where the HSU might be present or not.

  - 2: Always use HSU. The best option for code size (as SW algo are not included), but requires the presence of the HSU in the platform.
- ANT_DIV Indicate whether SW antenna diversity algorithm is included or not {off, **on**}

The following options allow configuring the Mesh support:

- MESH_VIF Number of supported Mesh Point Interfaces {**0**-n} – Value 0 disables the mesh feature
- MESH_LINK Number of supported Mesh Links (shared between the Mesh VIFs) {**2**}
- MESH_PATH Number of supported Mesh Paths {**10**}
- MESH_PROXY Number of supported Mesh Proxy Information {**3**}

Options available for FullHOST only:

- RTOS RTOS for which the FW is built {**freertos**}
- NETS Network (TCP/IP) stack included {**lwip**}
- TG Integration of the Traffic Generator used for WiFi certification {off, **on**}
  - Not including allows saving Code and Buffer RAM

The options above can be modified by adding them to the build command line as shown below:

python ..\tools\scons.py . ARCH=tl4xx AGG=off

The command above will build the LMAC without the code for A-MPDU in TX.

4.2 Typical configuration examples

This chapter lists the compilation options for different typical configurations. The values are indicative and might have to be modified according to specific needs if required. The options not listed take their default value.

4.2.1 IoT device

Such a device is typically 11n/20MHz. The constraint will be put on the memory footprint and the power consumption. The performance in terms of throughput is not the priority. The configuration will therefore be STA only, no Aggregation in TX, no P2P, and reduce the amount of memory used for reordering.

- PRODUCT=fmac
- STA=1
- BCN=off
- AGG=off
- P2P=0
- AMSDU=off
- BARX=1 (only 1 A-MPDU RX stream at a time – enough for WiFi certification)
- REORDBUF=4
- TXDESC*x*=4 (where *x* is 0 to 3)

4.2.2 Smartphone/Tablet

This device will typically be 11ac 1SS or 2SS. The throughput and the power consumption are the main factors that need to be taken into account for such a configuration. The default values represent a good compromise. Depending

on the number spatial streams that are supported, the A-MSDU transmission feature could be enabled (2SS) in order to optimize the throughput or disabled (1SS) to reduce the memory footprint:

- AMSDU=on (2SS) or off (1SS)
- BFMEE=on
- BWLEN=on

4.2.3 Home/Small Enterprise Access Point

In this configuration the performance is the main factor to optimize. Power-save modes or P2P are not required. More peer devices might also be required:

- PRODUCT=fmac
- STA=32
- VHT=on
- BFMEE=on
- BFMER=on
- PS=off
- P2P=0
- BARX=16 (Allow 20 RX BlockAck agreements in parallel)
- BATX=16 (Allow 20 TX BlockAck agreements in parallel)
- TXDESC*x*=64 (where *x* is 0 to 3)
- BWLEN=on

4.3 Macro Generation

All the register accesses are done over macros generated with a python tool: **SW\tools\reg_xls2h** at compilation time.

The excel sheets which contain the registers and fields definitions for the **PLATFORM** are located in **SW\plf\refip\import** and for the **MAC HW** registers in SW**\ip\lmac\import.**

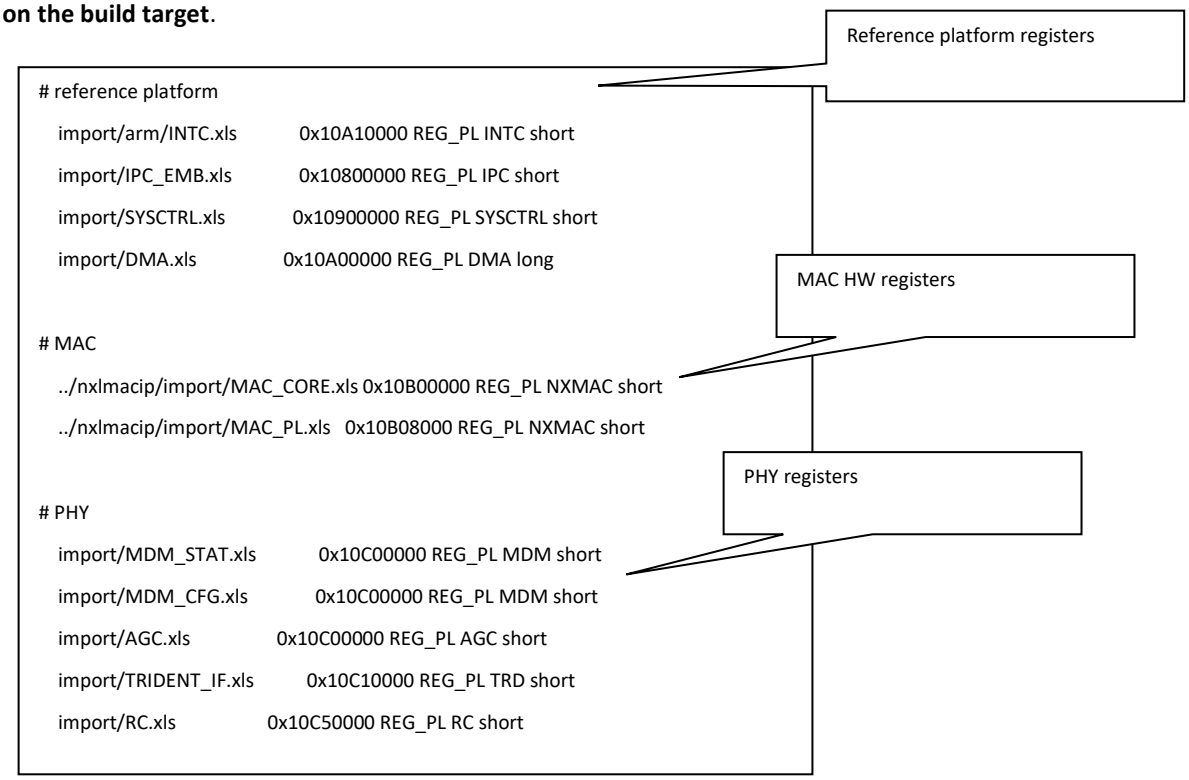
The macros take into account the offsets defined in the mapping section below.

4.4 Mapping

The mapping of our system is spread in two important files: reglist.txt and map.txt.

4.4.1 Registers

This is the register list: **SW\plf\refip\config\arm\reglist.txt where arm could be replaced by aps3 or tl4xx depending on the build target**.



4.4.2 Code and RAM

This linker script of refip platform is located in SW\plf**\refip\config\arm\gnuarm\map.txt** (ARM based platform), SW\plf**\refip\config\aps3\aps-gcc\map.txt** (Cortus based platform) or SW\plf**\refip\config\tl4xx\tl4\map_model.txt (TL4 based platform).**

5 CPU Dependencies

The arm platform is used as an example in this chapter.

5.1 Boot Vectors and Handlers

The boot vectors and handlers are located in the directory SW\plf**\refip\src\arch\arm\boot\gnuarm.**

5.2 Macro Definitions

The macros to start, to stop and to use critical sections are located in the directory SW\plf**\refip\src\arch\arm\ll\gnuarm.**

```
#define GLOBAL_INT_START()
do {
uint32_t __l_cpsr_tmp;
__asm("
        MRS     __l_cpsr_tmp, CPSR;
        BIC     __l_cpsr_tmp, __l_cpsr_tmp, #0x80;
        MSR     CPSR_cxsf, __l_cpsr_tmp;
    ");
} while(0)

#define GLOBAL_INT_STOP()
do {
    uint32_t __l_cpsr_tmp;
__asm("
        MRS     __l_cpsr_tmp, CPSR;
        ORR     __l_cpsr_tmp, __l_cpsr_tmp, #0x80;
        MSR     CPSR_cxsf, __l_cpsr_tmp;
    ");
} while(0)

#define GLOBAL_INT_DISABLE()
do {
    uint32_t __l_cpsr_tmp;
    uint32_t __l_irq_rest;
__asm("
        MRS     __l_cpsr_tmp, CPSR;
        AND     __l_irq_rest, __l_cpsr_tmp, #0x80;
        ORR     __l_cpsr_tmp, __l_cpsr_tmp, #0x80;
        MSR     CPSR_cxsf, __l_cpsr_tmp;
    ")

#define GLOBAL_INT_RESTORE()
__asm("
        MRS     __l_cpsr_tmp, CPSR;
        BIC     __l_cpsr_tmp, __l_cpsr_tmp, #0x80;
        ORR     __l_cpsr_tmp, __l_cpsr_tmp, __l_irq_rest;
        MSR     CPSR_cxsf, __l_cpsr_tmp;
    ");
} while(0)
```

Enable interrupts globally in the system.This macro must be used when the initialization phase is over and the interruptscan start is handled by the system.

Disable the interrupts globally in the system. This macro must be used when the system wants to disable all the interruptsthat it handles.

Disable interrupts globally in the system. This macro must be used in conjunction with the GLOBAL_INT_RESTORE macro since this last one will close the brace that the current macro opens. This means that both macros must be located at the same scope level.

Restore interrupts from the previous global disable.

5.3 Main Function

The entry point of the code is in the file SW\plf**\refip\src\arch\arm\arch_main.c.**

The SW\plf**\refip\src\arch\arm\arch.h** file contains the assert definitions.

6 IP Integration

6.1 Entry Points of the LMAC IP

The interrupt entry points for the RW-WLAN-nX SW IP can be found in file SW\plf**\refip\src\driver\intc\arm\intc.c** in the constant of function pointers called **intc_fiq_handlers[]**.

The other entry points of the IP are functions called from the main file (in SW\plf**\refip\src\arch\arm\arch_main.c**). These functions are **rwnxl_init()** (initialization function of the LMAC) called during startup phase, and the **ke_evt_schedule()** function that is called in the main loop.

6.2 Platform Drivers

To be able to start using the RW-WLAN-nX IP, the following essential blocks must be ported into the target platform – DMA, PHY and IPC (Interprocessor Communication Layer).

6.2.1 DMA

The LMAC Software requires a DMA driver that follows the API defined in SW\plf**\refip\src\driver\dma\dma.h**.

6.2.2 PHY

The LMAC Software requires a PHY driver that follows the API defined in SW\plf**\refip\src\driver\phy\phy.h**.

6.2.3 IPC

The LMAC Software requires an IPC driver that follows the API defined in SW\plf**\refip\src\driver\ipc\ipc_emb.h**.