UG10193

i.MX8MP-LA9310 BSP User Guide

Rev. 1.0 — 27 February 2025 User guide

Document information

Information	Content
Keywords	UG10193, LA9310, i.MX 8M Plus, PCIe endpoint, ITCM, DTCM
Abstract	This document explains how to install, build, and use i.MX8MP-LA9310 BSP software.

1 Introduction

This document describes the build, installation, and usage instructions for the i.MX8MP-LA9310 Reference Design Kit which includes i.MX 8M Plus and LA9310 processor support and board support package (BSP) software.

- LA9310 kernel module drivers running on i.MX 8M Plus Linux host.
- LA9310 FreeRTOS endpoint software.

The i.MX8MP-LA9310 board includes the following two NXP SoCs on the board:

- LA9310: NXP Layerscape Access LA9310 processor, which is based on Arm Cortex-M4 core and works as a PCIe endpoint.
- i.MX 8M Plus: Four A53 plus M7 core running at 1.6 GHz and acts as PCIe host (using i.mx8mp-sdr as test platform).

1.1 Scope of document

This document includes software installation steps and instructions for using BSP software with LF RTE v.2.5 release. It provides details about the software components that NXP adds to the PCIe host (i.MX 8M Plus) and PCIe endpoint (LA9310).

2 Release notes

The BSP 1.0 software release is for i.MX8MP-LA9310 (aka SDR board), which features i.MX 8M Plus as host CPU and LA9310 Cortex M4 CPU as PCIe endpoint. The release supports U-Boot, Linux, FreeRTOS, and additional features that are mentioned in the following subsections.

2.1 What is new in BSP v1.0 release

- Linux factory real-time edge v2.5 based release:
  - Updated meta layer and manifest file to build.
- Kernel v5.15.71:
  - Added i.MX TMU temperature callback mechanism.
  - Ethernet multiqueue support.
  - TSN feature enablement.
  - DTS update to support TTI interrupt:
    - SDR: GPT3 based using `sdr_tti` module.
    - Non-SDR board: Direct GPIO connections from LA9310.
- LA9310 drivers:
  - LA9310 - Shiva driver for PCI RC driver:
    - DCS channel: Rate configuration from module arguments.
    - Reset: Support reloading of kernel modules including `la9310shiva.ko` module.
      - `la9310_wdog` app: support added.
    - Support to get temperature of modem and host `la9310_tvd_app`.
    - Add TTI functionality for direct GPIO with LA9310.
    - Add TTI from GPT3 via M7 core (`sdr_tti.ko` module).
    - Supporting TTI app in both modes:
      - GPIO based TTI from LA9310.
      - Host generated GPT3-based TTI interrupt.
    - `la9310shiva.ko` module: Change SDR compiled time support to the module argument as `sdr_board=1` and enable it for SDR board (enable by default).
    - Si5510 DCO frequency correction support.
    - Support restart of GPT timer in M-7 core.
    - New ESPI test application and library in use-space.
    - Host UTILS - IQ streamer.
    - Application to check the PTP timing clock.
- `LA9310 SDR` driver:
  - `sdr_lime`: add support for Lime V2 RF card.
  - `sdr_lalb`: add library to handle LA9310 mailbox communication.
  - `sdr_usb`: driver to boost second USB port.
  - `sdr_daughterboard`: driver for breakout card.
- FreeRTOS:
  - Upgraded to v10.6.0.
  - DFE APP v0.3 integration.
  - SDR V2H and RF driver support.
  - TMU support.
  - Support for phy timer.
  - DCS channel configuration via host module.
  - Support TDD switching.

  - 1SEC PPS flag support.
  - Time offset support.
  - Add GPT TTI counter correction.
- Arm RAL library:
  - v24.01 added for high-phy development over i.MX.
- Kpage-cache module:
  - Added to support the cache invalidation on i.MX.
- DPDK:
  - Add DPDK and DPDK based BBDEV RAW IPC support.
  - Add `enet-qos` driver - DPDK support.
- Remote CLI support:
  - Linux: Support to use UART3/Modem logs to the host processor console.
- M7 build:
  - M7-based GPT timer support for TTI.
  - UART4 support added to debug M7 core.
- System:
  - Optimize the build size by disabling the multimedia components (reduced by one-third).
  - Support added for OP-TEE and Non-OP-TEE TFA boot (default OP-TEE).
  - Support to boot from TFTP using kernel ITB.
  - Support to update the SD card image remotely.
- RF support:
  - RF MT3812 card integration.
  - MT3812 - clock selection support.
  - MT3812 - add frequency plan support.
- RF Lime library updated to Lime SuiteNG:
  - Lime is compiled only when `IMX_LIME` is passed with host compilation.

2.2 How to obtain this release

The base LA9310 BSP build package is available via the GitHub based yocto layer. Refer to Section 2.2.2.

The pre-build binary is available on nxp.com for download.

- Link to obtain the full image: Full image
- Link to obtain Kernel ITB: Kernel ITB (if you want to change `wic` image remotely)

The complete package LA9310-BSP is distributed via Secure Files on NXP.com and it has access control. To get the access code, contact an FAE or sales representative. Enter the code at Software and Support Activation.

2.2.1 Yocto layer for Cortex-A core

The Cortex-A core allows you to run Linux. The corresponding Yocto layer description is as follows:

Linux and Rootfs: The Yocto layer `meta-real-time-edge` focuses on Linux building on Cortex-A cores. This layer is based on Linux factory and describes the process for building all applications for Linux and rootfs on Cortex-A core.

2.2.2 Git tags

The BSP Git repositories use Git tags to indicate component revisions that have been released and tested together. Use the `Git tag` command to examine them and chose a tag to check out.

Table 1. Git tags

Component	GitHub location	Branch/Commit-ID
Repo manifest	https://github.com/nxp-real-time-edge-sw/yocto-real-time-edge.git	la93xx/783603001e17514d84170c5e887db7f22d96b6b7
-	https://github.com/nxp-real-time-edge-sw/meta-real-time-edge	la93xx/ff3dfb858b36631a5b8a33e1850e711f2c1f2dac
Linux	https://github.com/nxp-qoriq/linux.git	la9310-linux-5.15-rt /imx-la93xx-1.0
U-Boot	https://github.com/nxp-qoriq/u-boot.git	imx8mp-la9310-v2022.04-2.5.0/imx-la93xx-1.0
DPDK	https://github.com/nxp/dpdk	22.11-qoriq/imx-la93xx-1.0
LA9310_HOST	https://github.com/nxp-qoriq/la93xx_host_sw	main/imx-la93xx-1.0
LA9310_FRTOS	https://github.com/nxp-qoriq/la93xx_freertos	main/imx-la93xx-1.0
LA9310_FW	https://github.com/nxp-qoriq/la93xx_firmware	main/imx-la93xx-1.0
DPDK_EXTRAS	https://github.com/nxp-qoriq/dpdk-extras.git	-
LMS7002M_URI	https://github.com/rfnm/lms7002m	-
LA9310_FREERTOS_VARISCITE	https://github.com/varigit/freertos-variscite.git	mcuxpresso_sdk_2.10.x-var01
LA9310_IMX8MP_M7	https://github.com/rfnm/imx8mp-m7	main

2.2.3 Additional software

The following components are not included as a part of standard BSP. They shall be obtained from NXP unless stated specifically about the third-party source.

- IQ-PLAYER firmware v1.0
  This firmware is a simple solution to exercise LA9310 baseband RX and TX path, using raw time domain I/Q sample flow. It should enable use cases, such as SDR applications, RF testing and calibration, where LA9310 does signal conditioning including TX/RX QEC and decimation. It intends to reach the maximum capacity allowed by LA9310 hardware, in term of sampling rate, PCI throughput, and number of channels.
  Feature list:
  – Raw continuous time domain RX/TX streaming
  – Sampling rates: 61.44 Msps, 122.88 Msps, and 160 Msps
  – Support 1T1R, 1T2R, 1T4R
  – Support QEC
  – The code for the IQ-PLAYER host application is available as a part of this BSP. However, the complete package including the VSPA source code is required for any modifications.
- LA9310 discrete RedCap 1.0.1
  DFE application demonstrates several concepts (FDD, TDD) running on LA9310 SoC. The purpose of the DFE application is to educate the user on how to use the LA9310 SoC to implement FDD and TDD concepts, host to modem mechanisms. It is also a test vehicle for VSPA kernels.
- M11 RF release for BSP 1.0
  M11 RF software release supporting SDR2.0 board with the MT3812-based RF card. It provides the necessary tool to use the SDR2.0 board with the MT3812-based RF.
- RF driver: Metanoia Diora SDK (MT3812 support Library)
  Compatible Metanoia SDK: v2.2.1-rc2

NXP customers/partners must obtain the above-mentioned version of the Metanoia Diora SDK directly from Metanoia. NXP release does not deliver Metanoia libraries. These libraries are mandatory for the MT3812 RF functionality. In the absence of these libraries, MT3812 RF does not work.

2.3 Open issues

Table 2 describes open issues.

Table 2. Known limitations, open issues

Serial number	Jira issue ID	Description
1	BSP5G-1829	PPS_IN interrupt behavior does not match PhyTimer timestamp capture behavior

3 LA9310 architecture

This section provides information about LA9310 overview, multi MSI support, and API.

3.1 Overview

In a typical system, one or more LA9310 devices are connected to the host via a PCIe interface. The host system (PCIe host) in the case of SDR board is i.MX 8M Plus SoC. However, host can be any device. For more details, see Figure 1.

- The host device runs on Linux OS.
- The baseband device (LA9310) runs on FreeRTOS on Arm-M4 cores.

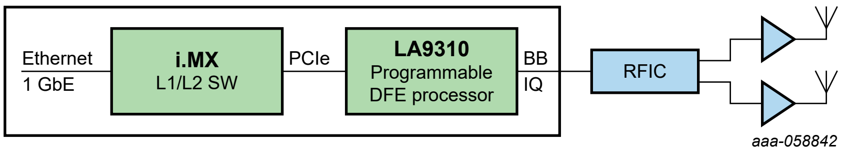

Figure 1. System high-level diagram

The overall system is i.MX8MP-LA9310 board containing i.MX 8M Plus host processor and NXP LA9310 SoC connected over PCIe interface, see Figure 2.

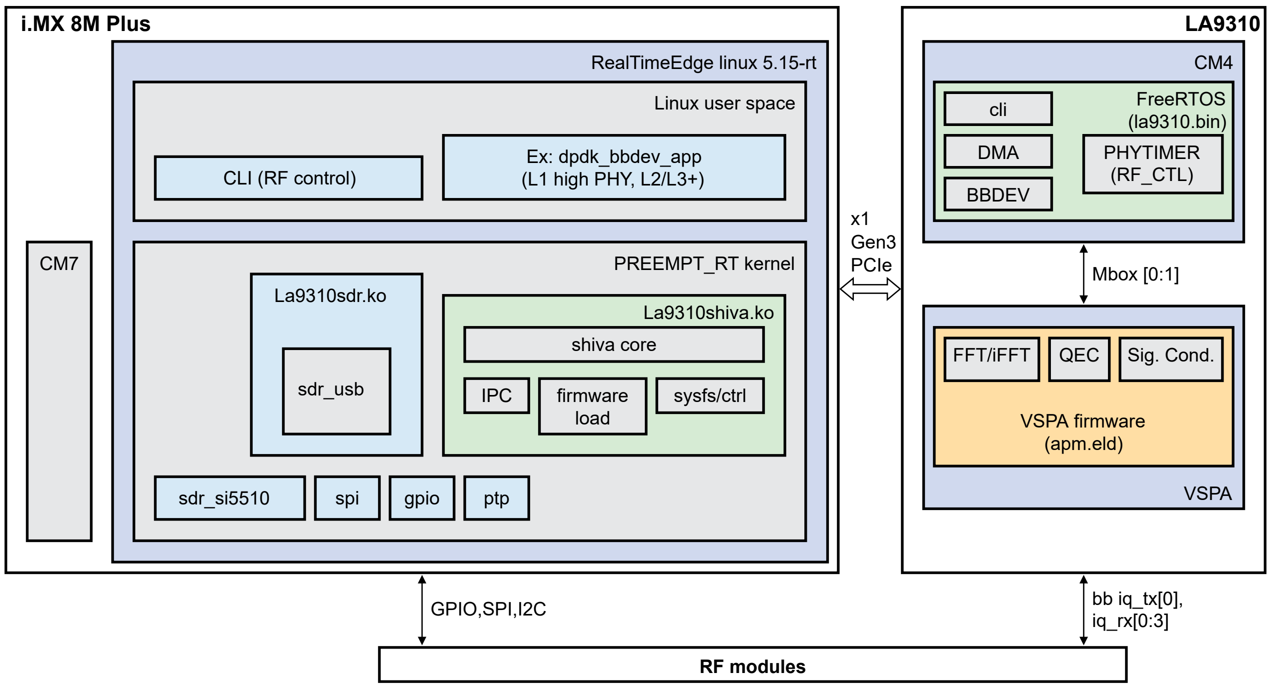

Figure 2. i.MX8MP-LA9310 overall system

3.1.1 Host-side i.MX 8M Plus software

The host side i.MX 8M Plus software includes the drivers required for the LA9310 device, the control/ configuration utilities, and the firmware files of various LA9310 components. The software on the host is for the Linux operating system. At high level, the host needs PCI-based drivers, the control/configuration utilities, the firmware files of various components on LA9310 SoC. All the software on the host is for the Linux operating system running on the i.MX processor. The components are developed in a split mode (functionality is split between host CPU and baseband processor). Host driver, libraries, and tools provide control interfaces which higher layer software (5G L2+ stack) uses to communicate/control LA9310 across PCIe.

The initialization of the baseband processor, including PCIe communication (inbound/outbound transfer windows), Arm M4 OS boot, VSPA image boot, RFIC initialization is performed by a host driver, that is, non-time-critical parts are run on host, and small and critical parts are run on LA9310.

The host side PCIe driver for LA9310 is also called Shiva driver. The i.MX8MP-LA9310 BSP software is structured in two components.

3.1.2 LA9310-side software

The software on LA9310 consists of the FreeRTOS operating system and drivers of various blocks in LA9310. Apart from drivers, it supports libraries, such as the communication library. The FreeRTOS APIs are provided for control/configuration of IP blocks (such as DMA), timer configuration, and so on. On the LA9310 side, a persistent memory is reserved for the modem host interface (MHIF). MHIF is a mechanism to share status and control information between Host and Baseband device.

SDK offers on the Baseband device side a set of libraries (drivers) for:

- Communication between Arm and VSPA (mailboxes)
- Communication between baseband and host (IPC)
- DMA engine (QDMA)
- RF control (example: TX/RX switch, frequency configuration, beam selection, and beam table update)
- GPIO and I2C

3.1.3 Boot flow

LA9310 is a PCIe end point (baseband device) on the host. The host boots the LA9310 baseband device and provides the FreeRTOS executable image over a PCIe interface. In PCIe boot, FreeRTOS binary images are loaded on host DDR via the Shiva host driver and an outbound PCIe window is created to get access of the host DDR. After the FreeRTOS executable file (bin) is loaded on to the host DDR, Boot Header and source (PCI address mapped to host DDR via Outbound window), and destination address (ITCM and DTCM) are programmed. Booting starts once the preamble is written after successful load of the image sections to their respective locations.

3.1.4 DDR memory allocation

LA9310 devices have a total of 1 GB address space available for PCI mapping. Some of the sections are fixed while others have variable size. By default, the scratch buffer size for an LA9310 instance is 64 MB (which can be changed). In such a case, Table 3 describes the mapping of baseband device PCI address space (1 GB) mapping with host memory.

Table 3. Mapping of baseband device

Region	Outbound window number	Modem memory address	Size	DDR memory address
SCRATCH BUFFER	0	0xA0000000-0xA40000000	0x4000000 (64 MB)	0x92400000
MSI	1	0xB0000000-0xB0001000	0x1000 (4 MB)	0x44059000
IQFLOOD	3	0xB0001000-0xBD001000	0xD000000 (208 MB)	0x96400000
IPC	2	0xC0000000-0xDFFFFFFF	Up to 512 MB	TBD

Table 4 shows the i.MX DDR memory allocation.

Table 4. i.MX 8M Plus DDR memory allocation

Name	Address	Size	DTS node	Attribute and description
ITCM	0x7e_0000 - 0x80_0000	0x2_0000 (128 kB)	m4@0x7E0000 { reg = <0x00 0x 7e0000 0x00 0x20000>;phandle = <0x87>;no-map;};	no-map
DTCM	0x80_0000 - 0x82_0000	0x2_0000 (128 kB)	m4@0x800000 {reg = <0x00 0x 800000 0x00 0x20000>;phandle = <0x8b>;no-map;};	no-map
OCRAM	0x90_0000	0x70000 (448 kB)	ocram@900000 {reg = <0x00 0x90 0000 0x00 0x70000>;no-map;};	no-map
DDR memory (all modules)	0x4000_0000 - 0x1_000 0_0000	0xc000_0000 (3 GB)	linux,cma {linux,cma-default;alloc-ranges = <0x00 0x40000000 0x00 0xc0000000>;compatible = "shared-dma-pool";size = <0x00 0x3c000000 >;reusable;};	reusable;
System RAM	0x4000_0000 - 0x5500_ 0000	0x1500_0000 (336 MB)	40000000-54ffffff: System RAM 40410000-41e3ffff: Kernel code 41e40000-421cffff: reserved 421d0000-4244ffff: Kernel data 43000000-4300efff: reserved	-
Vdev0vring0 It is virtual ring 0	0x5500_0000 - 0x5500_ 8000	0x8000 (32 kB)	vdev0vring0@55000000 {reg = <0x00 0x55000000 0x00 0x8000>; phandle = <0x85>;no-map;};	no-map
Vdev0vring1 It is virtual ring 1	0x5500_8000 - 0x5501_ 0000	0x8000 (32 kB)	vdev0vring1@55008000 {reg = <0x00 0x55008000 0x00 0x8000>; phandle = <0x86>;no-map;};	no-map
System RAM	0x5501_0000 - 0x550f_f0 00	0xe_f000 (956 kB)	55010000-550fefff: System RAM	-
rsc_table	0x550f_f000 - 0x5510_00 00	0x1000 (4 kB)	rsc_table@550ff000 {reg = <0x00 0x 550ff000 0x00 0x1000>;no-map;};	no-map
System RAM	0x5510_0000-0x5540_00 00	0x30_0000 (3 MB)	55100000-553fffff: System RAM	-
Vdevbuffer	0x5540_0000 - 0x5550_ 0000	0x10_0000 (1 MB)	vdevbuffer@55400000 {compatible = "shared-dma-pool";reg = <0x00 0x 55400000 0x00 0x100000>;phandle = <0x87>;no-map;};	shared-dma-pool, no-map
System RAM	0x5550_0000 - 0x5600_ 0000	0xb0_0000 (11 MB)	55500000-55ffffff: System RAM	-
optee_core	0x5600_0000 - 0x57e0_ 0000	0x1e0_0000 (30 MB)	optee_core@56000000 {reg = <0x00 0x56000000 0x00 0x1e0000 0>;no-map;};	no-map
optee_shm	0x57e0_0000 – 0x5800_ 0000	0x20_0000 (2 MB)	optee_shm@57e00000 {reg = <0x00 0x57e00000 0x00 0x200000 >;no-map;};	no-map
System RAM	0x5800_0000 - 0x8000_ 0000	0x2800_0000 (640 MB)	58000000-7fffffff: System RAM	-

Table 4. i.MX 8M Plus DDR memory allocation*...continued*

Name	Address	Size	DTS node	Attribute and description
M4	0x8000_0000 - 0x8100_0000	0x100_0000	m4@0x80000000 {reg = <0x00 0x80000000 0x00 0x1000000>;phandle = <0x88>;no-map;};	no-map
System RAM	0x8100_0000 -0x9240_0000	0x1140_0000 (10 MB)	81000000-923fffff : System RAM	-
La9310	0x9240_0000 - 0x9640_0000	0x400_0000 (64 MB)	la93@92400000 {compatible = "shared-dma-pool";reg = <0x00 0x92400000 0x00 0x4000000>;};	shared-dma-pool
dsp	0x9240_0000 - 0x9340_0000	0x100_0000 (16 MB)	dsp@92400000 {reg = <0x00 0x92400000 0x00 0x1000000>;phandle = <0x75>;no-map;};	no-map
dsp_reserved_heap	0x9340_0000 - 0x942f_0000	0xef_0000 (14 MB)	dsp_reserved_heap {reg = <0x00 0x93400000 0x00 0xef0000>;no-map;};	no-map
vdev0vring0	0x942f_0000 - 0x942f_8000	0x8000 (4 kB)	vdev0vring0@942f0000 {reg = <0x00 0x942f0000 0x00 0x8000>;phandle = <0x73>;no-map;};	no-map
vdev0vring1	0x942f_8000 - 0x9430_0000	0x8000 (4 kB)	vdev0vring1@942f8000 {reg = <0x00 0x942f8000 0x00 0x8000>;phandle = <0x74>;no-map;};	no-map
System RAM	0x9440_0000 - 0xbec0_0000	0x2a80_0000 (680 MB)	94400000-bebfffff: System RAM	-
iqflood	0x9640_0000 - 0xa340_0000	0xd00_0000 (208 MB)	iqflood@96400000 {reg = <0x00 0x96400000 0x00 0xd000000>;};	-
bootconfig	0xbec0_0000 - 0xbf00_0000	0x40_0000	bootconfig@bec00000 {reg = <0x00 0xbec00000 0x000x400000>;no-map;};	no-map
System RAM	0xbf00_00000x1_4000_0000	0x8100_0000 (2 GB)	bf000000-13fffffff: System RAM c0000000-ffffffff: reserved 13b000000-13f5fffff: reserved 13f6b3000-13f772fff: reserved 13f773000-13f773fff: reserved 13f774000-13f7bffff: reserved 13f7c2000-13f7c3fff: reserved 13f7c4000-13f7c4fff: reserved 13f7c5000-13f7d8fff: reserved 13f7d9000-13f7d9fff: reserved 13f7da000-13f829fff: reserved 13f82a000-13fffffff: reserved	
CM7 ALIAS CODE	0x10000000	0x1000000	m4@0x10000000 {reg = <0x00 0x10000000 0x00 0x1000000>;phandle = <0x89>;no-map;};	no-map

3.1.5 LA9310 Shiva driver

The host side PCIe driver (named Shiva) for LA9310 is the main LA9310 driver sitting in the Linux kernel, connects baseband device to Linux system. The driver comes in the form of a loadable module. This driver includes various sub-drivers for components on the LA9310 system, such as VSPA, RFIC, FreeRTOS, and IPC.

This driver is responsible for the initialization of the baseband device, including PCIe communication (inbound/outbound transfer windows), Cortex M4 OS boot, VSPA image boot, RFIC initialization, and various other channels establishment with the baseband device.

Some of the sub-drivers are:

- VSPA: Linux-based driver for VSPA booting. The VSPA driver loads VSPA image using the Linux firmware loading framework.
- IPC: Linux-based driver to map memory addresses for IPC usages. IPC implements communication pipe between software components on Arm host and components running on LA9310.
- RFIC: Responsible for initialization of RFIC using init scripts.
- WDOG: Detection of baseband device Hung state and Recovery mechanism.

3.1.6 Scratch buffer usages

Following are the current scratch buffer uses per baseband device instance. The scratch buffer has a size of 64 MB (`0x4000000`), which is used as listed in Table 5.

Table 5. Scratch buffer

Section name	Start-end address (modem address)	Size (section size + paint separator)	Start-end address (host address)
LA9310_VSPA_OVERLAY	0xa0000000 - 0xa0200040	0x200000 + 0x40	0x92400000 - 0x92600040
LA9310_MEM_REGION_VSPA	0xa0200040 - 0xa0200080	0x0 + 0x40	0x92600040 - 0x92600080
LA9310_MEM_REGION_FW	0xa0200080 - 0xa02180c0	0x18000 + 0x40	0x92600080 - 0x926180c0
LA9310_MEM_REGION_DBG_LOG	0xa02180c0 - 0xa0219100	0x1000 + 0x40	0x926180c0 - 0x92619100
LA9310_MEM_REGION_IQ_SAMPLES	0xa0219100 - 0xa1619140	0x1400000 + 0x40	0x92619100 - 0x93a19140
LA9310_MEM_REGION_NLM_OPS	0xa1619140 - 0xa2619180	0x1000000 + 0x40	0x93a19140 - 0x94a19180
LA9310_MEM_REGION_STD_FW	0xa2619180 - 0xa26391c0	0x20000 + 0x40	0x94a19180 - 0x94a391c0
LA9310_MEM_REGION_RF_CAL	0xa26391c0-0xa266b200	0x32000 + 0x40	0x94a391c0-0x94ab200

3.1.7 Baseband device sysfs interface

The sysfs file system is used for easy-to-use control, status, and debug logging interface for the baseband device. The Shiva driver creates multiple sysfs files, where each of these files serves as a control, status, or logging interface for exactly one module in the baseband device.

Each sysfs file is an easy interface in host Linux.

Module global configuration: displays the current high-level configuration parameters of all baseband device instances.

Table 6. Module global configuration commands

Commands	Description
`cat /sys/shiva/shiva_status`	Provides information about the number of baseband device instances and their respective configuration status
`cat /sys/shiva/shiva_version`	Displays Shiva module version

FreeRTOS and applications running on CM4 core uses macros (log_dbg, log_err, and log_info) to do message logging. FreeRTOS routes these log messages to a circular buffer (4K) maintained in ITCM/DTCM instead of sending them on LA9310 UART. There are sysfs files to read and clear the circular buffer. It gives easy access to the baseband device CM4 core logs from the host (Arm). Logging level of the logs (which one to show) can also be controlled through these sysfs files.

- **target_log**
  To dump LA9310 logs (FreeRTOS logs from LA9310) on Host:
  Default location: cat /sys/devices/platform/soc@0/<PCI address>/la9310sysfs/target_log
  Note: *The bus and device IDs mentioned in sysfs paths in this section are dependent on the host system and the number of PCIe devices connected to the host. Run* `lspci` *to check the correct IDs on the host and change the sysfs paths accordingly.*
  You can also find it in the /sys folder:

```
root@imx8mp-sdr:~# find /sys -name target_log
/sys/devices/platform/soc@0/33800000.pcie/pci0000:00/0000:00:00.0/0000:01:00.0/
la9310sysfs/target_log
```

  Example:

```
root@imx8mp-sdr:~# cat /sys/devices/platform/soc@0/33800000.pcie/
pci0000:00/0000:00:00.0/0000:01:00.0/la9310sysfs/target_log
prvInitLa9310Info: MSI init[0], a[  117.860700] NXP-LA9310-Driver 0000:01:00.0:
 LA9310 log buf dump, vaddr ffff8000342180c0, offset 0
ddr 0xb0000000, data 0x8
prvIni[  117.860757] NXP-LA9310-Driver 0000:01:00.0: log len: 1874, offset :
 1874
------------
```

  To reset the LA9310 log buffer on the host:

```
$ echo 0 > /sys/devices/platform/soc@0/33800000.pcie/
pci0000:00/0000:00:00.0/0000:01:00.0/la9310sysfs/target_log
```

- **target_log_level**

Change the current logging level of LA9310:

```
root@ua /$ echo $LOG_LEVEL > /sys/devices/platform/soc@0/33800000.pcie/
pci0000:00/0000:00:00.0/0000:01:00.0/la9310sysfs/target_log_level
```

After the log level is set, all the logs with equal or lower log level will be dumped in target_log (lower log_level number is for higher criticality. For example, 0 is for critical error).

Following are the supported log levels:

- LA9310_LOG_LEVEL_ERR 1
- LA9310_LOG_LEVEL_INFO 2
- LA9310_LOG_LEVEL_DBG 3
- LA9310_LOG_LEVEL_ISR 4
- LA9310_LOG_LEVEL_ALL 5

Check the current log level:

```
# cat /sys/devices/platform/soc@0/33800000.pcie/
pci0000:00/0000:00:00.0/0000:01:00.0/la9310sysfs/target_log_level
```

- **target_stats**

This section describes the commands to dump and clear host and LA9310 stats.

Dump stats:

```
# cat /sys/devices/platform/soc@0/33800000.pcie/
pci0000:00/0000:00:00.0/0000:01:00.0/la9310sysfs/target_stats
```

Clear stats:

```
root@ua /$ echo 0 > /sys/devices/platform/soc@0/33800000.pcie/
pci0000:00/0000:00:00.0/0000:01:00.0/la9310sysfs/target_stats
```

- **vspa_info**

  ```
  root@ua /$ cat /sys/devices/platform/soc@0/33800000.pcie/
  pci0000:00/0000:00:00.0/0000:01:00.0/la9310sysfs/vspa_info
  ```

  Output:

  ```
  VSPA Binary Software Version = 20002 VSPA state = 6
  ```

  This file lists the firmware version running on VSPA and the current state of VSPA.
  Use the state number to identify the VSPA state. Refer to the following information:
  - VSPA_STATE_UNKNOWN = 0
  - VSPA_STATE_POWER_DOWN = 1
  - VSPA_STATE_STARTUP_ERR = 2
  - VSPA_STATE_UNPROGRAMMED_IDLE = 3
  - VSPA_STATE_UNPROGRAMMED_BUSY = 4
  - VSPA_STATE_LOADING = 5
  - VSPA_STATE_RUNNING_IDLE = 6
  - VSPA_STATE_RUNNING_BUSY = 7

3.1.8 `la9310shiva` load time module parameters

The Shiva driver includes the following module parameters that can be given during the `insmod/modprobe` operation.

Table 7. Module parameters

S. No.	Parameter	Default value	Description
1	`scratch_buf_size`	-	(Mandatory) Scratch buffer size for all baseband devices. The default is 64 MB (0x4000000).
2	`scratch_buf_phys_addr`	-	(Mandatory) Scratch buffer start physical address for baseband device. The default is 0x92400000.
3	`adc_mask`	0xf	ADC channels enable mask - bitwise (max 0x4).
4	`adc_rate_mask`	0x0	ADC frequency mask for each channel (0 for full duplex [122.88 MHz], 1 for half-duplex [61.44 MHz]).
5	`dac_mask`	0x1	DAC channels enable mask - bitwise (max 0x2).
6	`dac_rate_mask`	0x0	DAC frequency for each channel (0 for full duplex [122.88 MHz], 1 for half-duplex [61.44 MHz]).
7	`iq_mem_addr`	0x96400000	SDR IQ flood memory address (reserved memory size = 208 MB).
8	`iq_mem_size`	(1024 * 1024 * 208)	SDR IQ flood memory size.

Table 7. Module parameters*...continued*

S. No.	Parameter	Default value	Description
9	`alt_vspa_fw_name`	apm.eld	Alternative VSPA firmware name prefix to be loaded to VSPA cores in the baseband device (apm.eld). For example: `alt_vspa_fw_name=”new_apm.eld”`.
10	`alt_firmware_name`	la9310.bin	Alternative firmware name to be loaded to LA9310 baseband device. For example: `alt_firmware_name=“new_la9310.bin”`. The firmware must be available in the /lib/firmware directory.
11	`sdr_board`	1	If the board is an SDR board.
12	`modem_host_uart`	0	This parameter routes modem UART to host when set `modem_host_uart=1` as a module parameter. You can access the modem log after `ssh` to the host and open a serial console session using `minicom -b 115200 -D /dev/ttymxc2`.
13	`modem_rf_data_size`	(1024 * 200)	RFIC buffer size for each LA9310 devices.

3.1.9 LA9310 memory resource allocation

Figure 3 shows the ITCM (aka TCML) allocation.

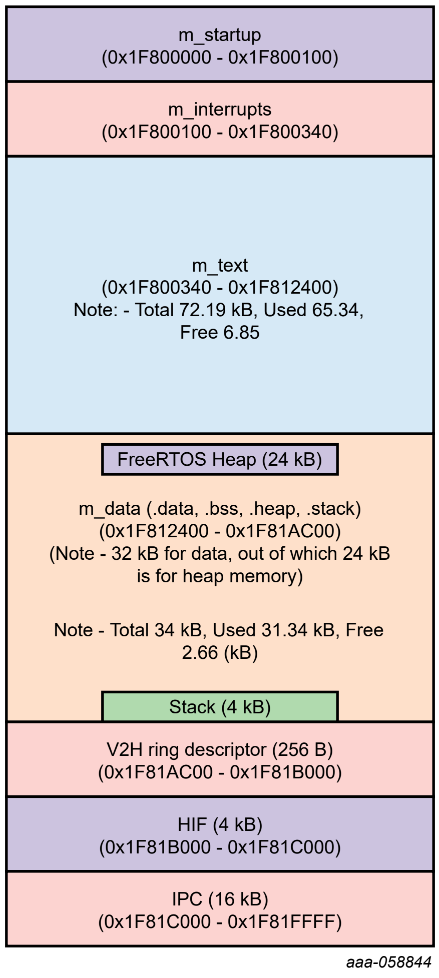
m_startup
(0x1F800000 - 0x1F800100)
m_interrupts
(0x1F800100 - 0x1F800340)
m_text
(0x1F800340 - 0x1F812400)
Note: - Total 72.19 kB, Used 65.34, Free 6.85
FreeRTOS Heap (24 kB)
m_data (.data, .bss, .heap, .stack)
(0x1F812400 - 0x1F81AC00)
(Note - 32 kB for data, out of which 24 kB is for heap memory)
Note - Total 34 kB, Used 31.34 kB, Free 2.66 (kB)
Stack (4 kB)
V2H ring descriptor (256 B)
(0x1F81AC00 - 0x1F81B000)
HIF (4 kB)
(0x1F81B000 - 0x1F81C000)
IPC (16 kB)
(0x1F81C000 - 0x1F81FFFF)
aaa-058844

Figure 3. ITCM (aka TCML) allocation

Figure 4 shows the DTCM (aka TCMU) allocation. The entire 64 kB of DTCM is allocated for VSPA DFE for IQ sample dump.

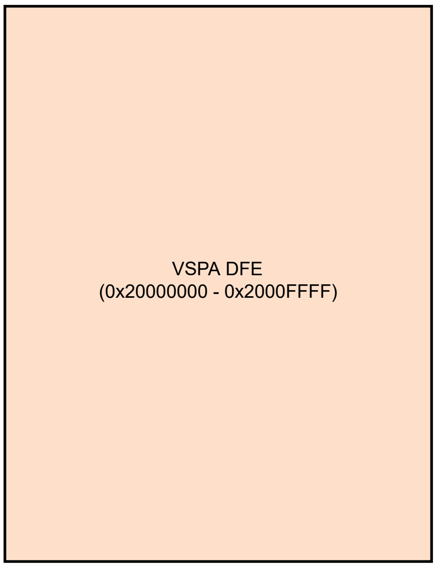

Figure 4. DTCM (aka TCMU) allocation

3.1.10 FreeRTOS memory calculator tool

A new tool has been developed in Python to find the current memory usage analysis of the FreeRTOS. It takes `la9310.map` as input and stores the results in a `.text` file for analysis. A dependency package must be installed on the host machine to run this command.

```
pip3 install tabulate
```

The following commands can be used to parse the input map file and generate the output:

```
/memory_calc.py -i <path>/release/la9310.map -o memory.dump
```

3.1.11 Data conversion subsystem (DCS)

In wireless baseband device implementation, RF transceivers convert baseband signals to RF radio signals.

The transmit processing steps include conversion of digitally modulated baseband signals to analog signals, followed by RF modulation and amplification, and afterward sending signals to physical antennas to radiate the RF signals.

The receive signal processing steps are the reverse of transmit steps, where at the end of RF signal and analog baseband signal processing, signals are converted to digital baseband signals for digital processing.

The LA9310 device supports a data conversion subsystem. It supports a programmable clock frequency (153.6 MHz) to each ADC pair and DAC pair. The data converter subsystem has four IQ ADC pair, one IQ DAC pair, and one auxiliary ADC pair.

3.1.12 TTI

The transmission time interval (TTI) represents the time duration for a certain block of information to be transmitted. It is the interval between the start of one transmission and the start of another transmission. The primary purpose of defining TTIs is to enable the efficient use of radio resources. By dividing time into intervals, the system can allocate resources, schedule transmissions, and manage interference more effectively.

TTI software (host side) consists of four main components:

- User space
- TTI Demo app
- TTI Lib
- Kernel space
- TTI kernel submodule
- TTI ISR

TTI Demo app registers itself for TTI interrupts coming from M-7 core or MSI-based from the baseband through TTI Lib APIs. The TTI registration API provides an `eventfd`, which L2 can use to block on TTI events. The API Lib forwards the request through IOCTL calls made to the TTI kernel submodule.

To exercise the TTI functionality provided by BSP, the following kernel module must be inserted.

```
insmod /lib/modules/$(uname -r)/extra/sdr_tti.ko
```

The TTI kernel side code is responsible for registering the interrupt handlers with the Linux kernel using standard Linux kernel APIs.

Once interrupt handlers (part of the TTI kernel submodule) are registered, the TTI Demo app quietly sleeps and listens for any activity on MSI interrupt (from LA9310 baseband device side). As the interrupt pins are asserted, ISR sends a wake-up signal to the TTI Demo app that was still sleeping for events to occur. After the wake-up signal is caught by the Demo app in user space, `TTI_ID` and its corresponding time of occurrence is stored in user space data structures.

Virtual IRQ lines are relinquished once the TTI Demo app reaches its completion.

3.1.13 TMU

The i.MX 8M Plus SDR includes i.MX 8M Plus and LA9310 that may get overheated during normal operations as the board is supposed to operate in outdoor environment.

The thermal variance detector mechanism helps the software (for example, L2 stack) to monitor the overheat condition.

Software is also able to detect the fall in temperature below the software programmable temperature threshold.

Major components in the board that are sources of heat are:

- i.MX 8M Plus (host)
- LA9310 (baseband processor)
- RF device
- TVD software module, which is a part of PCIe driver (Shiva) that is supposed to provide the thermal events (for CPU and baseband devices both) to userspace application. It also allows the software to set up the thermal thresholds for both CPU and baseband devices.
- TVD module, which communicates with the following drivers internally:
  - CPU thermal driver (CTD): Runs at CPU side (Linux) to monitor CPU temperature
  - Modem thermal driver (MTD): Runs at baseband side (FreeRTOS) to monitor baseband device and connected RF card

Testing TMU

```
$la9310_tvd_testapp modem_id -m -c
Options:
-h Help
-m Monitor Modem temperature
```

```
-c Monitor Host temperature
Modem current temp: VSPA : 56 *C DCS: 57 *C
Host current temp: Probe1: 58.00 *C
```

3.1.14 eDMA

The i.MX 8M Plus device contains an enhanced direct memory access (eDMA) controller that can perform complex data transfers with minimal intervention from a host processor. The hardware microarchitecture includes a DMA engine that performs:

- Source address and destination address calculations
- Data-movement operations
- Local memory containing transfer control descriptors for each of the 32 channels

eDMA software (host side) is a Userspace application, which consists of:

- eDMA demo app
- eDMA static lib

The eDMA demo app needs the following information:

- Transfer size
- Source address
- Destination address
- Channel number

The information is passed to the eDMA static lib to configure the PCI eDMA and start the DMA transfer.

Testing eDMA

```
root@imx8mp-sdr:~# la9310_edma_app -h
Usage : la9310_edma_app -c [val] -s [val] -d [val] -l [val] [option]
options: -r (read), -w (write), by default it is write
-c: Current channel number - default 1
-b: PCI EDMA base address
-n: PCI EDMA reg space size
-s: Source Address : default: 0x96400000(for write)
-d: Destination Address: default: 0x1f000000(for write)
-l: Tansfer Len - default 0x20
Read-> Dma transfer from Ep to RC
Write-> Dma transfer from Ep to RC
root@imx8mp-sdr:~# la9310_edma_app -l 0x40
/dev/mem opened.
Memory mapped at address 0xffff948c0000.
root@imx8mp-sdr:~#
root@imx8mp-sdr:~#
root@imx8mp-sdr:~# memtool -32 0x33b80200 40
E
Reading 0x40 count starting at address 0x33B80200
0x33B80200: 04000068 00000000 00000000 96400040
0x33B80210: 00000000 1F000040 00000000 00000000
Here 04000068 meaning completed, 04000048 meaning running otherwise there is
 some error and DMA did not complete.
```

3.1.15 Modem information

This is a Userspace application that is used to get the current modem information which includes the modem device ID, PCIe link status, memory-related information (different BAR mapping, PCIe mapped memory using outbound/inbound window), RF-related information, and so on.

How to use:

```
root@imx8mp-sdr:~# la9310_modem_info -h
Usage :  la9310_modem_info [optional]
        -h or -H    help and usages
        -m <dev id>
        -s  got printing stats
root@imx8mp-sdr:~# la9310_modem_info
Board Name - sdr imx8mp
LA93xx ID:0
Dev name - shiva0
PCI addr - 0000:01:00.0
PCI WIN start     0x0 Size:0x40000000
HIF start         0x1c01b000 Size:0x328
CCSR phys         0x18000000 Size:0x4000000
TCML phys         0x1c000000 Size:0x20000
TCMU phys         0x1f000000 Size:0x800000
Scratch buf phys 0x92400000 Size:0x4000000
Scrach Buf Regions
Region             |      Host Phy Addr    |      Modem Phy Addr   |      Size
VSPA OVERLAY phys |      0x92400000       |      0xa0000000       |
 0x200000
VSPA start         |      0x92600040       |      0xa0200040       |      0x0
FW start           |      0x92600080       |      0xa0200080       |
 0x18000
DBG LOG phys      |      0x926180c0       |      0xa02180c0       |      0x1000
IQ SAMPLES phys   |      0x92619100       |      0xa0219100       |
 0x1400000
IQ FLOOD phys     |      0x96400000       |      0xb0001000       |
 0xd000000
NLM OPS start     |      0x93a19140       |      0xa1619140       |
 0x1000000
STD FW phys       |      0x94a19180       |      0xa2619180       |
 0x20000
RF CAL phys        |      0x94a391c0       |      0xa26391c0       |
 0x32000
DAC Mask: 0x1  Rate: 122.88 MHz
ADC-0: ON  Rate : 122.88 MHz
ADC-1: ON  Rate : 122.88 MHz
ADC-2: ON  Rate : 122.88 MHz
ADC-3: ON  Rate : 122.88 MHz
```

3.1.16 Modem test framework

Integrated test status:

```
Test FreeRTOS IPs
(on FreeRTOS prompt of Core 0)
```

3.2 Multiple MSI support

The LA9310 device supports a maximum of eight different MSI interrupts. To enable this, make sure the host supports (hardware and Linux kernel) multiple MSI. The number of MSI interrupts supported by the host and LA9310 driver is defined by the below macro and can be changed. The number of MSI interrupts used depends upon overall system policy.

In a common header file: `LA9310_host_if.h`

```
#define LA9310_MSI_MAX_CNT
```

Each MSI interrupt is assigned to one subcomponent or module. The assignment of MSI interrupts to different subcomponents is done statically using the following `enum`:

```
enum la9310_msi_id {
        MSI_IRQ_MUX = 0,
        MSI_IRQ_V2H,
        MSI_IRQ_WDOG,
#ifndef LA9310_RESET_HANDSHAKE_POLLING_ENABLE
        MSI_IRQ_HOST_HANDSHAKE,
#else
        MSI_IRQ_UNUSED_1,
#endif
        MSI_IRQ_IPC_1,
        MSI_IRQ_IPC_2,
        MSI_IRQ_UNUSED_2,
        MSI_IRQ_UNUSED_3,
```

3.2.1 MSI usage API

The application programming interface (API) to use the MSI in LA9310 FreeRTOS software is as follows:

Raise MSI interrupt to host:

```
vRaiseMsi(struct LA9310_info *, enum LA9310_msi_id);
```

3.3 API

Table 8 provides a summary of the APIs supported by i.MX8MP-LA9310 BSP software.

Table 8. i.MX8MP-LA9310 API summary

S. No.	Module	Linux host API	FreeRTOS API	Comment
1	IPC	`common_headers/la93xx_bbdev_ipc.h` `common_headers/la93xx_ipc_ioctl.h`	`common_headers/la93xx_bbdev_ipc.h` `common_headers/la93xx_ipc_ioctl.h`	-
2	eDMA	`api/imx_edma_api.h`		-
3	TVD	`common_headers/la9310_tvd_ioctl.h`	`common_headers/la9310_tvd_ioctl.h`	-
4	SPI	`api/diora_ecspi_api.h` `api/imx_ecspi_api.h`	-	-
5	Watchdog	`api/la9310_wdog_api.h`	`common_headers/la9310_wdog_ioctl.h`	-

Table 8. i.MX8MP-LA9310 API summary*...continued*

S. No.	Module	Linux host API	FreeRTOS API	Comment
		`common_headers/la9310_wdog_ioctl.h`		
6	VSPA – AVI	-	-	-
7	GPIO	-	`include/la9310_gpio.h`	-
8	I2C	-	`include/la9310_i2cAPI.h`	-
9	Modem info	`common_headers/la9310_modinfo.h`	`common_headers/la9310_modinfo.h`	-
10	TTI	`common_headers/la9310_tti_ioctl.h`	`common_headers/la9310_tti_ioctl.h`	-
11	V2H	`common_headers/la9310_v2h_if.h`	`Find . -name common_headers/la9310_v2h_if.h`	-

4 Set up Yocto build environment at host and build Yocto images

This section describes how to set up Yocto build environment at host and build Yocto images.

4.1 Building images using Yocto

Yocto bitbake is the preferred build method to build the entire package which includes, Linux kernel image, FreeRTOS image for LA9310, host Linux kernel modules (Shiva, RFIC modules), and `root-fs` in an integrated `wic.zst` format package.

```
nxp-image-real-time-edge-imx8mp-sdr-<TimeStamp>.rootfs.wic.zst
```

Both internal and external users can build Yocto-based packages.

Internal NXP users who have access to bitbucket can build images using the development repository. However, external users can build Yocto based packages using the released GitHub external repository (see Section 2.2.2).

4.2 Steps to create a build environment (both external and internal users)

The following steps are used to create a build environment.

1. Create repository:

```
mkdir ~/bin
(This step may not be needed if the bin folder already exists)
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

2. Add the following line to the `.bashrc` file and ensure that the `~/bin` folder is in your PATH variable:

```
export PATH=~/bin:$PATH
Note-  change python to python3 in file  ~/bin/repo
vim ~/bin/repo
#!/usr/bin/env python3
```

3. Get the repository:

```
mkdir yocto-real-time-edge
cd yocto-real-time-edge
repo init -u https://github.com/nxp-real-time-edge-sw/yocto-real-time-
edge.git  -b la93xx  -m rte_la93xx_release.xml
(Please use -m rte_la93xx_devel.xml for NXP internal builds to use
 development repos)
NOTEIf you get error on repo init - it may be due to python version. Try
https://community.nxp.com/t5/i-MX-Processors-Knowledge-Base/repo-issue-quot-
def-print-self-args-kwargs-quot/ta-p/1756521
```

4. Do repository synchronization:

```
repo sync
 OR
repo sync --no-clone-bundle --force-sync --force-remove-dirty  --no-current-
branch -j4
```

4.3 Build images

This section describes how to build images using an external GitHub repository or internal bitbucket repository depending upon the user profile. It also describes how to start a build and how to build individual components using Yocto bitbake.

4.3.1 Build images using external GitHub repository (for external user)

Prepare a build directory for the board:

```
EULA=1 DISTRO=nxp-real-time-edge MACHINE=<machine>  source real-time-edge-setup-
env.sh -b <build-dir>
The current supported boards are:
imx8mp-sdr
imx8dxlb0-lpddr4-evk
imx8mp-lpddr4-evk
imx8mm-lpddr4-evk
imx8mp-seeve

e.g.
EULA=1 DISTRO=nxp-real-time-edge MACHINE=imx8mp-sdr source real-time-edge-setup-
env.sh -b build-imx8mpsdr
```

4.3.2 Build images using internal bitbucket repository (for internal NXP user)

Prepare the build directory for the board.

Note:

These steps require access to internal repository and uses top of the tree)

```
EULA=1 DISTRO=nxp-real-time-edge MACHINE=<machine> source real-time-edge-setup-
env.sh -b build-imx8mpsdr -i internal
Example:
EULA=1 DISTRO=nxp-real-time-edge MACHINE=imx8mp-sdr source real-time-edge-setup-
env.sh -b build-imx8mpsdr -i internal
```

4.3.3 Start build

```
-----------------------------------
bitbake nxp-image-real-time-edge
Note - Above command would build all the components and put it at location
 build-imx8mpsdr/tmp/deploy/images/imx8mp-sdr
```

NXP Internal users: User can also build kernel ITB the Linux kernel, `root-fs`, and `dtb` using the following Yocto bitbake command.

Generated kernel ITB is located in a deploy directory named as `kernel-imx8mp-sdr.itb`:

```
bitbake sdr-rootfs-minimal -c image_cpio
bitbake kernel-imx8mp-sdr
```

4.3.4 Build individual components using Yocto bitbake

Once a full build is complete, you can build individual components as well with their changes using the Bitbake commands listed in Table 9.

These steps are useful during development and debug.

Table 9. Bitbake commands

Module	Command
FreeRTOS image	`bitbake freertos-la9310`
User application	`bitbake userapp-la9310`
M7 binary	`bitbake imx-m7-la9310`
kernel-module-kpage-ncache	`bitbake kernel-module-kpage-ncache`
FR1-FR2-test-tool	`bitbake fr1-fr2-test-tool-la9310`
Granita and other kernel modules including Shiva	`bitbake kernel-module-sdr`
kernel-module-arm-pmu	`bitbake kernel-module-arm-pmu`
U-Boot	`bitbake u-boot-imx`
Linux	`bitbake linux-imx`

Following are the typical steps to debug an image by incremental build steps.

Example:

```
bitbake -c cleanall freertos-la9310
bitbake -c do_unpack -f freertos-la9310
bitbake -c do_patch -f freertos-la9310
bitbake -c do_compile -f freertos-la9310
```

5 Standalone build environment

Apart from Yocto build, individual components can be built during the development stage to debug and test. The following subsections provide some of the repositories that can be built using standalone mode.

5.1 Repository details

- External repository
- NXP internal repository

5.1.1 External repository

Note: *Check Branch/Commit-ID from the latest BSP software release, see Section 2.2.*

5.1.2 NXP internal repository

Note: *Check the Commit-ID tag from the latest BSP software release or use top of the trunk.*

Table 10. Internal repository

Software	Link	Tag
Linux	https://bitbucket.sw.nxp.com/projects/DN5G/repos/5g-linux/browse	la9310-linux-5.15-rt
U-Boot	https://bitbucket.sw.nxp.com/projects/DN5G/repos/5g-uboot/browse	imx8mp-la9310-v2022.04-2.5.0
la931x_freertos	https://bitbucket.sw.nxp.com/projects/DNNPI/repos/la931x_freertos	la12xx
la931x_host_sw	https://bitbucket.sw.nxp.com/projects/DNNPI/repos/la931x_host_sw/	la12xx
la931x_firmware	https://bitbucket.sw.nxp.com/projects/DN5G/repos/la931x_firmware/browse	master
dpdk	https://bitbucket.sw.nxp.com/projects/GITAM/repos/dpdk	22.11-qoriq
fr1_fr2_test_tool	https://bitbucket.sw.nxp.com/scm/dn5g/fr1_fr2_test_tool.git	master
imx8mp-m7	https://bitbucket.sw.nxp.com/projects/DN5G/repos/imx8mp-m7/browse	main
geul_rf_util	https://bitbucket.sw.nxp.com/projects/DNNPI/repos/geul_rf_util/browse	sdr_mt3812_dev

5.2 Build images

- Build Linux image
- Build host Shiva kernel module
- Build FreeRTOS image
- Build DPDK

5.2.1 How to build Linux image

Checkout Linux source code from the above-mentioned repository and branch.

Perform the steps as follows:

```
export CROSS_COMPILE=<compiler-path>aarch64-linux-gnu-
export ARCH=arm64
make imx8mp_sdr_defconfig
make -j 28
```

5.2.2 How to build host Shiva kernel module

The steps to build host Shiva kernel module are as follows.

```
la931x_host_sw$ git submodule update --init    //Get firmware binary
export CROSS_COMPILE=<compiler-path>aarch64-linux-gnu-
export ARCH=arm64
export KERNEL_DIR=<kernel-path>/imx8mp-kernel
export LA9310_COMMON_HEADERS=<repository-path>/la931x_freertos/common_headers
export LA9310_DRIVER=< la931x_host_sw  repository-path>/
export LMS7002M_MODULE_DIR=../lms7002m
export ECSPI_LIB_PATH=<la931x_host_sw repository path>/lib/ecspi/
make clean
make IMX_SDR=1  IMX_RFLIME=1  IMX_RFMT3812=1
make install IMX_SDR=1 IMX_RFLIME=1 IMX_RFMT3812=1
```

5.2.3 How to build FreeRTOS image

The steps to build FreeRTOS image are as follows.

```
export CROSS_COMPILE=<compiler-path>aarch64-linux-gnu-
export ARCH=arm64
export KERNEL_DIR=<kernel-path>/imx8mp-kernel
export ARMGCC_DIR=<toolchain-path>/gcc-arm-none-eabi-6-2017-q2-update/
export LA9310_COMMON_HEADERS=<repository-path>/la931x_freertos/common_headers
cd Demo/CORTEX_M4_NXP_LA9310_GCC/
./clean.sh
./build_release.sh -m pcie -t sdr -b release

To Build la9310_dfe.bin
./clean.sh
./build_release.sh -m pcie -t sdr_dfe -f LA9310_DFE_APP=ON -f
 LA9310_TURN_ON_COMMAND_LINE=ON
#Copy only the la9310.bin
#remove elf
```

5.2.4 How to build DPDK

```
export LA9310_COMMON_HEADERS=<repository-path>/la931x_freertos/common_headers
meson arm64-build --cross-file config/arm/arm64_dpaa_linux_gcc -Dexamples=all
ninja -C arm64-build
```

6 Steps to get board up and running

To bring up the board, an SD card with `nxp-image-real-time-edge-imx8mp-sdr-<TimeStamp>.rootfs.wic.zst` image must be flashed.

6.1 Flashing images

Unzip the image and flash into an SD card:

Note: *Ensure that the correct device is being used.*

```
Build Image Location:
build-imx8mpsdr/tmp/deploy/images/imx8mp-sdr$ ls -la
lrwxrwxrwx 2 nxf31049 nxp        66 Feb 28 01:14 nxp-image-real-
time-edge-imx8mp-sdr.wic.zst -> nxp-image-real-time-edge-imx8mp-
sdr-20240227155929.rootfs.wic.zst
Uncompress zst file:
build-imx8mpsdr/tmp/deploy/images/imx8mp-sdr$  zstd -d nxp-image-real-time-edge-
imx8mp-sdr-20240227155929.rootfs.wic.zst
```

Flash image into SD card:

```
sudo dd if= nxp-image-real-time-edge-imx8mp-sdr-20240227155929.rootfs.wic of=/
dev/sdb bs=8M oflag=direct status=progress
```

6.2 Flashing images remotely

When the board is already up and running, a new SD card image can be flashed into the board remotely using TFTP boot.

Table 11. TFTP boot

Step	Description
Bring tiny Linux itb image into TFTP server	Bring `kernel-imx8mp-sdr.itb` into TFTP server Example: `wget https://<Image_location>/kernel-imx8mp-sdr.itb`
Bring wic image into TFTP server	1. Bring the `.wic.zst` image to the TFTP server. Example: `wget` [`https://%3CImage_location%3E/nxp-image-real-time-edge-imx8mp-sdr-XYZ.rootfs.wic.zst`](https://%3CImage_location%3E/nxp-image-real-time-edge-imx8mp-sdr-XYZ.rootfs.wic.zst) 2. Uncompress `.zst file: zstd -d nxp-image-real-time-edge-imx8mp-sdr-XYZ.rootfs.wic.zst`

Boot board stop at U-Boot, and set unique eth1 address	Example:
	`-----------`
	`u-boot=> editenv eth1addr`
	`edit: ee:a0:94:0f:9b:91`
	`u-boot=> saveenv`

Set serverip	Example:
	`-----------`
	`u-boot=> edit serverip`
	`edit: 10.232.20.249`
	`u-boot=> saveenv`

Table 11. TFTP boot*...continued*

Step	Description

Get IP on the board (static/ dynamic)	Example:
	`-----------`
	`u-boot=> dhcp`

Boot board with tiny Linux	Example:
	`-----------`
	`u-boot=> tftp 0xa0000000 <tftp_server> /kernel-imx8mp-sdr.itb;bootm 0xa0000000`

Get IP on the board (static/ dynamic)	`root@TinyLinux:/mnt# udhcpc -i eth0`

List SD/eMMC devices	
	```
	root@TinyLinux:~# lsblk
	NAME          MAJ:MIN  RM   SIZE  RO TYPE MOUNTPOINT
	mmcblk2       179:0     0  58.3G  0  disk
	mmcblk2boot0  179:32    0     4M  1  disk
	mmcblk2boot1  179:64    0     4M  1  disk
	mmcblk1       179:96    0  29.7G  0  disk
	|-mmcblk1p1   179:97    0  83.2M  0  part
	`-mmcblk1p2   179:98    0   2.1G  0  part
	```
	Note: *`mmcblk1` is SD and `mmcblk2` is eMMC.*

Flash SD card with .wic image	1. If the switch setting to the boot board is an SD card, bring .wic image into eMMC as a backing store. If needed, format eMMC as follows: Optional step: `root@TinyLinux:~# mkfs.ext2 /dev/mmcblk2` 2. Mount eMMC to /mnt as follows:`root@TinyLinux:~# mount /dev/mmcblk2 /mnt/`

Bring .wic image on the board	Example:
	`-------------`
	`cd  /mnt`
	`root@TinyLinux:/mnt# scp  nxf31049@10.232.20.249:/tftpboot/nxf31049/*.wic .`

Flash .wic image	`root@TinyLinux:/mnt# dd if=nxp-image-real-time-edge-imx8mp-sdr-20240615220855.rootfs.wic of=/dev/mmcblk1 bs=8M oflag=direct status=progress`
	`root@TinyLinux:/mnt#sync`

Reboot board to boot from new image	`root@TinyLinux:/mnt# reboot`

7 LA9310 driver initialization

This section covers the following:

- Loading the Shiva kernel module
- SDR board-related execution scripts
- DPDK based IPC (BBDEV-RAW)

7.1 How to load Shiva kernel module at Linux prompt

1. Invoke the following commands before inserting Shiva kernel module:

```
echo "1" > /sys/bus/pci/rescan
echo 7 > /proc/sys/kernel/printk
< additional steps for SDR board >
gpioget 2 9
gpioget 2 14
```

2. Insert Shiva Linux kernel module:

```
insmod /lib/modules/$(uname -r)/extra/la9310shiva.ko
 scratch_buf_size=0x4000000 scratch_buf_phys_addr=0x92400000
```

Note: *When a user wants to test with dfe app, below is the sample* `insmod` *command that can be invoked with proper values of various module parameters.*

```
insmod  /lib/modules/$(uname -r)/extra/la9310shiva.ko scratch_buf_size=0x4000000
 scratch_buf_phys_addr=0x92400000 adc_mask=0xf adc_rate_mask=0xf dac_mask=0x1
 dac_rate_mask=0x1 alt_firmware_name=la9310_dfe.bin
```

7.1.1 Shiva kernel module log

```
root@imx8mp-sdr:~# echo "1" > /sys/bus/pci/rescan
root@imx8mp-sdr:~# echo 7 > /proc/sys/kernel/printk
root@imx8mp-sdr:~# gpioget 2 9
1
root@imx8mp-sdr:~# gpioget 2 14
1
root@imx8mp-sdr:~# insmod /lib/modules/$(uname -r)/extra/la9310shiva.ko
 scratch_buf_size=0x4000000 scratch_buf_phys_addr=0x92400000
[   49.322844] la9310shiva: loading out-of-tree module taints kernel.
[   49.324814] NXP PCIe LA9310 Driver: Init.
[   49.325125] la9310-reset-gpio 0 not found
[   49.325349] LA9310 IPC driver: major_nr 507, minor 0
[   49.325553] NXP-LA9310-Driver 0000:01:00.0: max payload size    rc:128 ep:256
[   49.325574] NXP-LA9310-Driver 0000:01:00.0: Init - shiva0 !
......
[   51.322662] NXP-LA9310-Driver 0000:01:00.0: Boot Ok Msg Verified: msb =
 F1000000, lsb = 00000000
[   51.322669] NXP-LA9310-Driver 0000:01:00.0: SW Version: vspa = 00020002, ippu
 = 00000000
[   51.322677] NXP-LA9310-Driver 0000:01:00.0: SPM Ack Msg: msb = F0700000, lsb
 = 00000000
[   51.322981] Created sysfs group (null)
root@imx8mp-sdr:~#
root@imx8mp-sdr:~#
root@imx8mp-sdr:~#
root@imx8mp-sdr:~# find /sys -name target_log
```

```
/sys/devices/platform/soc@0/33800000.pcie/pci0000:00/0000:00:00.0/0000:01:00.0/
la9310sysfs/target_log
root@imx8mp-sdr:~# cat /sys/devices/platform/soc@0/33800000.pcie/
pci0000:00/0000:00:00.0/0000:01:00.0/la9310sysfs/target_log
prvInitLa9310Info: MSI init[0], a[  117.860700] NXP-LA9310-Driver 0000:01:00.0:
 LA9310 log buf dump, vaddr ffff8000342180c0, offset 0
ddr 0xb0000000, data 0x8
prvIni[  117.860757] NXP-LA9310-Driver 0000:01:00.0: log len: 1874, offset :
 1874
-------------
NLM Board Rev#1
vLa9310_do_handshake: LA9310 -> HOST: LA9310_HOST_START_DRIVER_INIT
iRficInit: Started
DSPI Clock Set: Expected speed:7000000  Bus clock:245760000
-------
DCS Init completed
FreeRTOS command server.
Type Help to view a list of registered commands.
>root@imx8mp-sdr:~#
```

7.1.2 LA9310 FreeRTOS log

The following is the output from the LA9310 CM4 console.

```
>ucFdr is::::43
in INIT, value of ibsr is :: 12
in INIT,I2C initialised
STARTING NLM.. Boot Source (PCIe)
FreeRTOS Kernel vesrion V10.4.6
prvInitLa9310Info: MSI init[0], addr 0xb0000000, data 0x8
prvInitLa9310Info: MSI init[1], addr 0xb0000000, data 0x9
prvInitLa9310Info: MSI init[2], addr 0xb0000000, data 0xa
prvInitLa9310Info: MSI init[3], addr 0xb0000000, data 0xb
prvInitLa9310Info: MSI init[4], addr 0xb0000000, data 0xc
prvInitLa9310Info: MSI init[5], addr 0xb0000000, data 0xd
prvInitLa9310Info: MSI init[6], addr 0xb0000000, data 0xe
prvInitLa9310Info: MSI init[7], addr 0xb0000000, data 0xf
iLa9310IRQInit:Initialized IRQ EVT mux, irq_evt_cfg 0x601
vLa9310UpdateEvtHif: HIF update evtcnt 1, type 3, irq_evt_cfg 0x601
iLa9310RegisterEvt: Registered evt 5, mask 0x1f803f05, unmask 0x1f803ee9
prvSyncTimingDeviceWaitForCTS: Data read error
sync timing device GetDeviceInfo failed
vLa9310_do_handshake: LA9310 -> Host: LA9310_HOST START_CLOCK_CONFIG
vLa9310_do_handshake: Host -> LA9310: LA9310_HOST_COMPLETE_CLOCK_CONFIG
ucFdr is::::45
NLM Board Rev#1
vLa9310_do_handshake: LA9310 -> HOST: LA9310_HOST_START_DRIVER_INIT
iRficInit: Started
DSPI Clock Set: Expected speed:7000000  Bus clock:245760000
MCR    80090008
TCR    0
CTAR0  78020003
SR     42010000
RSER   0
CTARE0 10001
iRficInit: Completed
iLa9310HostPostInit: eDMA init DONE
IPC on modem ready
Waiting for HOST ready 0x0
```

```
INFO:iLa9310AviInit: AVI Init Starting
INFO:iLa9310AviConfig: AVI Config Starting
vLa9310UpdateEvtHif: HIF update evtcnt 2, type 1, irq_evt_cfg 0x601
iLa9310RegisterEvt: Registered evt 4, mask 0x1f80513d, unmask 0x1f80514d
INFO:iLa9310AviConfig: Enabling IRQ_VSPA
INFO:iLa9310AviConfig: AVI Init Done
CHECK_BIT(adc_freq_mask, (dcs -1) = 0
CHECK_BIT(adc_freq_mask, (dcs -1) = 0
CHECK_BIT(adc_freq_mask, (dcs -1) = 0
CHECK_BIT(adc_freq_mask, (dcs -1) = 0
vDcsInit: DAC Init completed
DCS Init completed
FreeRTOS command server.
Type Help to view a list of registered commands.
>
```

7.1.3 VSPA loading

Check for the following message during `insmod la9310shiva.ko`

```
[ 37.410708] NXP-LA9310-Driver 0000:01:00.0: Boot Ok Msg Verified: msb =
 F1000000, lsb = 00000000
[ 37.410725] NXP-LA9310-Driver 0000:01:00.0: SW Version: vspa = 00020002, ippu =
 00000000
[ 37.410740] NXP-LA9310-Driver 0000:01:00.0: SPM Ack Msg: msb = F0700000, lsb =
 00000000
```

Note: *VSPA binary is loaded as part of* `insmod la9310shiva.ko`*. If the above messages are displayed as highlighted, then it indicates that the VSPA image is loaded successfully.*

When the VSPA firmware is loaded successfully, the VSPA Linux host driver sends a mailbox message to VSPA core with the base address of the VSPA firmware location in DDR.

When the VSPA receives the mailbox message, it should send ACK/NACK. The VSPA Linux host driver waits for a timeout. However, if it fails to receive the mailbox message, then it displays a message as highlighted below and continues booting:

```
[ 369.288280] NXP-LA9310-Driver 0001:01:00.0: Base address=a0600000 Passed to
 vspa
```

7.2 SDR board-related execution scripts

Table 12 describes the SDR board-related execution scripts.

Table 12. SDR board-related execution scripts

S. No.	Script name	Description
1	sdr_enable_la9310_uart_ext_port	Enable serial UART Access.
2	sdr_load_la9310_drivers	Script to load Shiva and SDR drivers/modules
3	sdr_load_lime.sh	Script to load the sdr_lime module and to load the elf to M7 core
4	sdr_demo	Configure SDR demo
5	rmmod_la9310shiva.sh	To remove the Shiva kernel module.
6	sdr_en_daughterboard_power	Kernel module to enable SDR daughter board

Table 12. SDR board-related execution scripts*...continued*

S. No.	Script name	Description
7	sdr_init_rba_gpio	Script to set GPIO vales
8	sdr_en_lms	Script to enable lms device
9	enable_rf.sh	Script to test tdd and fdd

7.2.1 Loading kernel modules

```
//Load All kernel modules
root@imx8mp-sdr:~# ./sdr_load_la9310_drivers
root@imx8mp-sdr:~# ./sdr_load_lime.sh
```

7.2.2 Reloading modem

```
root@imx8mp-sdr:~# ./rmmod_la9310shiva.sh
root@imx8mp-sdr:~# ./sdr_load_la9310_drivers
root@imx8mp-sdr:~# ./sdr_load_lime.sh
```

7.2.3 Running TTI test

Make sure all the kernel modules are loaded before executing below listed commands.

```
root@imx8mp-sdr:~# ./sdr_demo
root@imx8mp-sdr:~# insmod /lib/modules/$(uname -r)/extra/sdr_tti.ko
root@imx8mp-sdr:~# echo 8 > /proc/irq/69/smp_affinity
root@imx8mp-sdr:~# la9310_tti_app 0 0 -e

On FreeRTOS console
>test 20
Switch RF (mode = 0xaaaaaaaa, issued = 0x96c8998d, target = 0x96c9118d --->
 1F80F740
//Verify tti interrupts (Optional steps  )

root@imx8mp-sdr:~# cat /proc/interrupts
```

7.3 DPDK based IPC (BBDEV-RAW)

This section describes how to configure DPDK-based IPC using the BBDEV-RAW interface, which covers creating hugepage memory, running applications on both FreeRTOS and host sides."

7.3.1 Create hugepage memory

```
mount -t hugetlbfs none /dev/hugepages
echo 448 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

7.3.2 Running application on FreeRTOS side

```
>test 9 latency
```

7.3.3 Running application on host side

```
dpdk-bbdev_raw_app --vdev=bbdev_la93xx -c 0x1 -n 1 -- --latency c

EAL: Detected CPU lcores: 4
EAL: Detected NUMA nodes: 1
EAL: Detected static linkage of DPDK
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Scan for (fslmc) bus failed.
EAL: Scan for (fslmc) bus failed.
EAL: lsx_pciep:SoC(0x5f580000[  112.056620] NXP-LA9310-Driver 0000:01:00.0: Huge
 Page Buff:0x4d600000[H]-0xc0000000[M],size 2097152
) not supported
EAL: Scan for (l[  112.056635] la9310_dev->hif->ipc_regs.ipc_mdata_size: 352
sx_pciep) bus failed.
EAL: Selected IOVA mode 'PA'
EAL: VFIO support initialized
TELEMETRY: No legacy callbacks, legacy socket not created
==================
HOST->MODEM operation latency:
        avg: 18.2157 cycles, 2.27696 us
        min: 15 cycles, 1.875 us
        max: 292 cycles, 36.5 us
HOST->MODEM latency test successful with 10000 test cases
==================

==================
MODEM->HOST test completed for confirmation mode. Check Geul console logs for
 results.
==================
root@imx8mp-sdr:~#
```

```
COMMAND for LATENCY NO CONF
./dpdk-bbdev_raw_app --vdev=bbdev_la93xx -c 0x1 -n 1 -- --latency n
```

```
COMMAND for VALIDATION CONF
root@imx8mp-sdr:~# ./dpdk-bbdev_raw_app --vdev=bbdev_la93xx -c 0x1 -n 1 -- --
validation c
```

```
COMMAND for VALIDATON NO CONF
root@imx8mp-sdr:~# ./dpdk-bbdev_raw_app --vdev=bbdev_la93xx -c 0x1 -n 1 -- --
validation n
```

8 Userspace application example

Userspace applications in embedded systems serve a critical role by providing an interface for interacting with the system, executing specific tasks, and enabling user interactions. They provide a structured way to implement high-level functionalities, manage hardware interactions, ensure security, and offer user interfaces.

This layer of abstraction allows for more accessible development, testing, and maintenance while ensuring system stability and security.

This section explains all the Userspace applications that are available in the system. The binaries for these userspace applications are present in /usr/bin/ in release images.

8.1 Baseband processor recovery

The baseband process recovery is based on implementing watchdog-based reset.

8.1.1 Introduction

The watchdog (Wdog) module is designed to enhance system reliability by monitoring the system operation and ensuring it functions correctly. If the system becomes unresponsive or encounters a critical fault, the watchdog module initiates a corrective action, typically resetting the system to restore normal operation.

The module is based around a 32-bit down counter that is initialized from the watchdog Load Register. It can be enabled or disabled as required.

The watchdog and restart mechanism of the baseband processor is called Baseband(LA9310)/modem recovery. It detects if a baseband processor is hung and initiates the recovery action once the hung condition is detected.

In i.MX8MP-LA9310, a Userspace application is provided to reset the modem manually and reload it. Using this application, you can reload images on the modem without rebooting the whole system.

8.1.2 Recovery sequence

Following is the sequence of watchdog and baseband processor restart:

- Hardware watchdog [baseband processor]: LA9310 SoC has an IP watchdog. When software enables it, a countdown starts. On the first expiry of the counter, an interrupt is raised to CM4 core and baseband processor watchdog ISR is executed.
- Baseband processor health check (watchdog ISR) [baseband processor]: This ISR is executed on the first expiry of watchdog timer. The job of this ISR is as follows:
  – Check that all components (registered tasks, VSPAs) are functioning and are not hung.
  – If all components are working, the watchdog counter is reloaded, and the baseband processor continues to work as usual.
  – If either one or more components are found not working or the ISR itself does not execute, the watchdog hardware raises MSI to the Linux host.
  – Linux host rescan Baseband (LA9310) end-point device and reenumerate.
  – Reload freeRTOS image on baseband (LA9310).
  – The baseband (LA9310) processor is synchronized with the Linux host and start working as expected.

8.1.3 Configuring baseband processor side watchdog

8.1.3.1 Recovery from the host

Host Linux needs to start watchdog application and listen for watchdog MSI from modem side. The following is how the watchdog application should be started.

```
$ la9310_wdog_testapp -w 0 &
```

8.1.3.2 Recovery testing commands with examples

Use the below steps to test LA9310 recovery:

1. Test baseband device force reboot:
   ```
   $la9310_wdog_testapp -f 0
   ```
   The above command resets the baseband device and reloads the images.
2. Test with baseband watchdog event
   a. Open the host Linux console and run the following command: `#la9310_wdog_testapp -w 0`
   b. Go to the baseband device console and start watchdog test: `>test 7`

9 RFIC subsystem

In a wireless baseband device implementation, RF transceivers convert baseband signals to RF radio signals. The transmit processing steps include conversion of digitally modulated baseband signals to analog signals, followed by RF modulation and amplification, and afterward sending signals to physical antennas to radiate the RF signals. The receive signal processing steps are the reverse of transmit steps, where at the end of RF signal and analog baseband signal processing, signals are converted to digital baseband signals for digital processing.

The LA9310 device implements the ADCs and DACs required for the above-mentioned signal processing steps, so that RF transceivers operate only on analog baseband signals. The Data Conversion Subsystems in the Layerscape Access SoCs deal with digital-to-analog and analog-to-digital conversion of I/Q samples.

- LS-DCS supports up to two low-speed I/Q channels. The ADCs support up to 245.76 MHz and the DACs support up to 491.52 MHz sampling clocks.

The LA9310-SDK provides some example RF cards integrations. Refer to these examples to integrate any new RF driver into the system.

9.1 LS-DCS

The LS-DCS on LA9310 has two macros:

- LS-DCS1
- LS-DCS2

Each of these LS-DCS macros has two ADC/DAC conversion pairs: CP1 (DAC1, ADC1) and CP2 (DAC2, ADC2). The ADC_DAC subsystem includes the following features:

- Supports four I/Q ADCs and four I/Q DACs.
- One I/Q ADC and one I/Q DAC are logically considered as one channel.
- All ADC/DACs support 12b resolution.
- Interfaces with the AXIQ interface with four transmit channels and four receive channels. The databus width of the transmit channel interface is 24-bit while it is 12-bit on the receive interface.
- All data converter clocks are sourced from a single clock input.
- Supports programmable clock frequency to ADCs and DACs.
- Additionally, a programmable clock output (referred to as `TBGEN_Clk` or `soc_clk_Isdcs`) is generated for use external to the subsystem.

9.1.1 LS-DCS driver

The LS-DCS driver is implemented in FreeRTOS and its source code is available in `<FreeRTOS folder>/drivers/dcs`.

The DCS driver in FreeRTOS starts with DCS PLL status check. If the PLL is locked it moves further ahead with the configuration, else it returns an error. Depending on the SoC version, `vLSDcsInit_a0 ( )` or `vLSDcsInit_b0 ()` function is called.

9.2 TTI design

- TTI duration is driven using the PHY timer, which uses the Si55 clock.
- RF control TX/RX switching with sample precision by using PHY timer rfctl associated GPIOs for FEM control
- Two MAC TTIs to host CPU associated with PHY timer rfctl GPIOs for notifying host application for slot/frame boundary, data preparation, and so on. Also, to generate an interrupt to configure the GPT timer which drives the host TTI.
- It allows synchronizing the TX/RX switch with AXIQ DMAs.

For more details, see Figure 5, Figure 6, Figure 7, and Figure 8

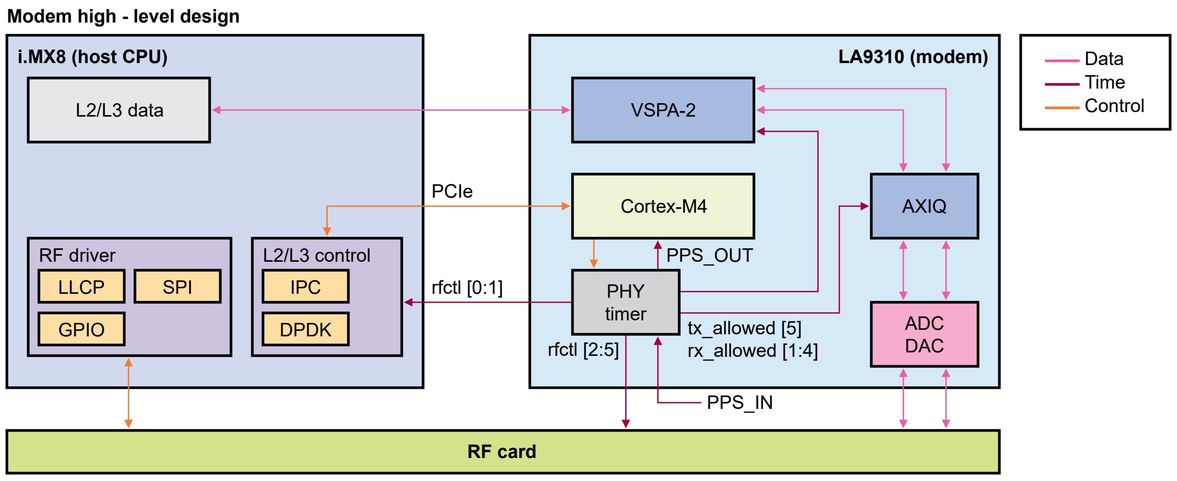

Figure 5. Modem high-level design

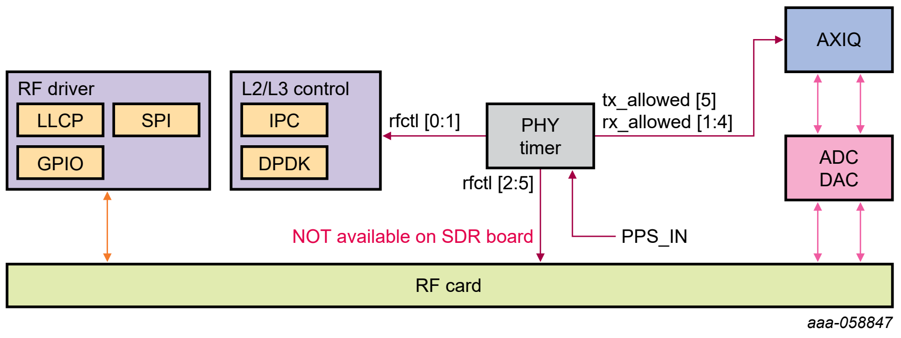

Figure 6. TTI design

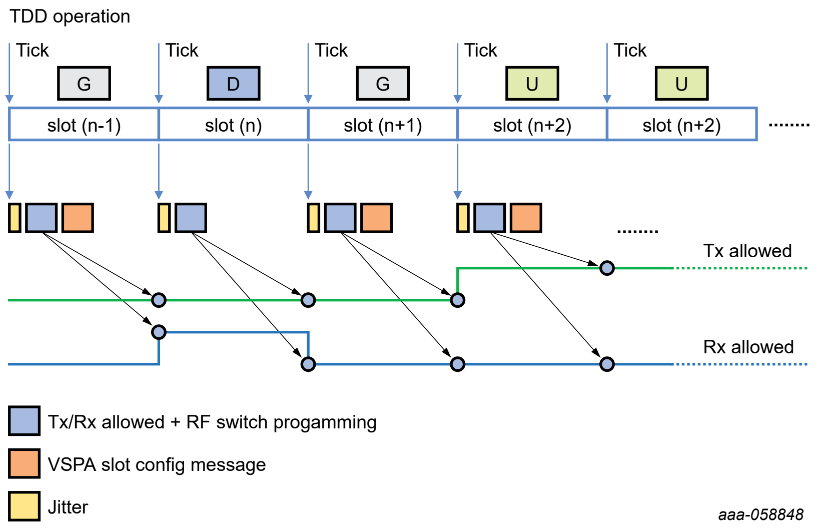

Figure 7. TDD operation

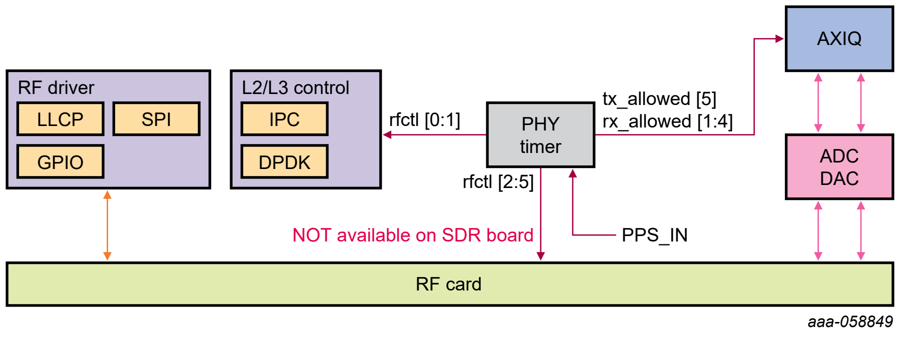

Figure 8. SDR RF TTI control solution

1. RF card is in default operating mode (RX off, TX off)
2. For TX/RX switch: set a variable at a known location in M4:
   - PhyTimer timestamp used for `GPIO_12` triggering
   - Command (TX/RX/LB mode)
3. Give the "Go" signal and M-7 captures it and configure the RF. The RF switches to TX/RX at the desired timestamp + small jitter.
4. M-7 also captures the TTI start event. It configures the GPT-3 capture and `SDR_TTI` module in the kernel and registers IRQ to generate TTI.
   This is used to set the number of counts after which an interrupt is generated by the GPT timer and to start the GPT timer. Send regular interrupts to A53 core (host) based on the count set by the PHY timer.
5. Once the GPT timer is configured it generates an interrupt after every COUNT (set using PHY timer) number of counts by the GPT timer.
6. This interrupt is handled by the sdr_tti driver on the host (A53 core) in which can be used to carry out any functionality, such as data transmission.

9.3 RF card support options

The current list of supported hardware:

- SDR board + LimeRF v2
  Available as a part of the host driver with an external library from LIME.
- SDR board + Metanoia RF (MT3812)
  Binary is available, for detail, refer to the RF library from Metanoia.

10 DFE application

DFE application demonstrates several concepts (FDD, TDD) running on LA9310 SoC. The purpose of the DFE application is to educate the user on how to use the LA9310 SoC to implement FDD and TDD concepts, host to modem mechanisms. It is also a test vehicle for VSPA kernels.

For details, refer to DFE_LA9310_v1.0.1.pdf (see Section 14).

10.1 Supported configurations and dependencies

The DFE application aims to provide a reference for demonstrating and understanding L1 functionality on the LA9310 platform and a starting point toward supporting custom designs. To achieve this, it builds on top of NXP BSP, targeting operation on the LA9310 SoC, see Figure 9:

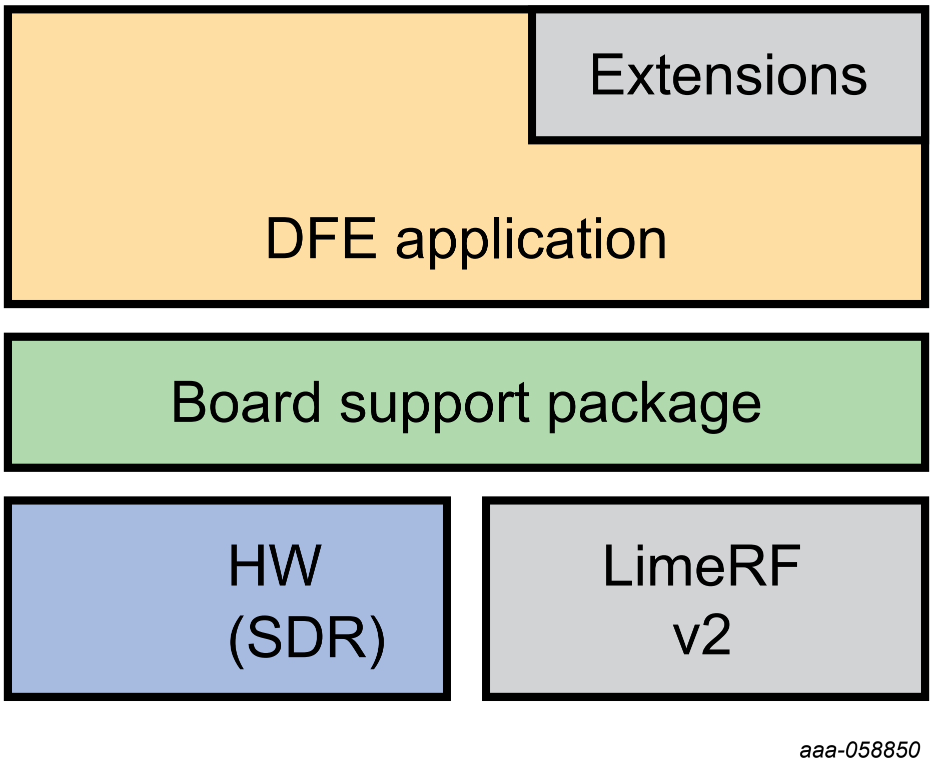

Figure 9. DFE application ecosystem

11 Exception handlers

Exception handling in FreeRTOS has been implemented to catch the faults generated at the runtime and dump the necessary information which can be helpful to debug the fault.

- Bus fault: Detects memory access errors on instruction fetch, data read/write, interrupt vector fetch, and register stacking (save/restore) on interrupt (entry/exit).
- Memory management fault: Detects memory access violations to regions that are defined in the memory management unit (MPU). For example, it can detect code execution from a memory region with read/write access only.
- Usage Fault: Detects execution of undefined instructions, unaligned memory access for load/store multiple. When enabled, divide-by-zero, and other unaligned memory accesses are also detected.
- Hard Fault: This fault arises if the handler of Bus faults, Memory Management faults, or Usage faults cannot be executed.

Expected output on console in case of fault (for example, bus fault):

```
========STACK FRAME=========
r0 : 1f803d9b r1 : a
r2 : 1000
r3 : 2020004
r12 : 0
lr : 1f8026bf pc : 1f80694
psr : 1000000
```

```
=====END of STACK FRAME======
Fault Address : 1f802694
SCB->CFSR = 0x0000040
SCB->HFSR = 0x40000000
Forced Hard Fault
Bus fault: Imprecise data access error
```

Identify the location where the fault occurred.

1. Take the value of “Fault Address”, which is the PC value in the dumped stack frame.
2. Take the objdump of the `LA9310.elf` file and search for the location found in step 1. You get the exact instruction and function where the fault occurred.

12 Arm Cortex-M4 errata workaround

12.1 Errata description

The Cortex-M4 includes a write buffer that permits execution to continue while a store is waiting on the bus. Under specific timing conditions, during an exception return while this buffer is still in use by a store instruction, there can be a late change in selection of the next interrupt to be taken. This can cause a mismatch between the interrupt acknowledged by the interrupt controller and the vector fetched by the processor.

12.2 Software workaround

Each interrupt handler must call the data synchronization barrier `dsb()` before exiting the interrupt handler function to ensure that all instructions are complete before returning from the interrupt handler.

Example:

```
void LA9310MSG_3_IRQHandler(void)
{
log_info("%s: MSG3 IRQ\n\r",   func  ); NVIC_ClearPendingIRQ(IRQ_MSG3);
#if ARM_ERRATUM_838869
dsb(); #endif
}
```

13 Appendix

13.1 How to benchmark Linux host code using PMU counter:

1. Insert `pmu_el0_cycle_counter` kernel module from Linux prompt as shown below.

```
root@imx8mp-sdr:~# insmod /lib/modules/$(uname -r)/extra/
pmu_el0_cycle_counter.ko
```

2. Example code to test PMU counter.

```
volatile  uint64_t prev,curr,delta;
        /*Get pmu counter cpu cycles before test start */
        asm volatile("isb;mrs %0, pmccntr_el0" : "=r"(prev));

        #if  1 // Test code for cpu cycle test
        volatile uint64_t n = 160000000;
        while(n > 0) n--;
        #endif // Test code end

        /*Get pmu counter cpu cycles after test  */
        asm volatile("isb;mrs %0, pmccntr_el0" : "=r"(curr));

        /*Calculate delta and print*/
        delta = curr-prev;
        if(delta)
                printf("cpu cycles (hex 0x%lx dec %lu) time taken %ld(nano second)  \r
\n", delta,delta,(delta * 10 )/16);
        else
                printf("No diff , something wrong\r\n");
```

13.2 Method to rebuild bitbake with small changes:

The following method describes how to rebuild bitbake with small updates.

```
=== to rebuild (fast) on existing repo
repo init -u ssh://git@bitbucket.sw.nxp.com/dn5g/yocto-real-time-edge.git  -b la93xx -m
 rte_la93xx_devel.xml
rm -rf sources/meta-real-time-edge/
cd sources/meta-imx/
git am --abort
cd -
repo sync --no-clone-bundle --force-sync --force-remove-dirty  --no-current-branch -j4
EULA=1 DISTRO=nxp-real-time-edge MACHINE=imx8mp-sdr source real-time-edge-setup-env.sh -b
 build-imx8mpsdr -i internal
bitbake -c cleanall nxp-image-real-time-edge; bitbake -c cleanall linux-imx -f ; bitbake
 -c cleanall kernel-module-la9310 -f ; bitbake -c cleanall userapp-la9310 -f ;bitbake -c
 cleanall freertos-la9310 -f ; bitbake -c cleanall dpdk -f

bitbake -c do_rootfs nxp-image-real-time-edge;bitbake nxp-image-real-time-edge -f
```

14 References

Table 13 lists and explains the additional documents and resources that you can refer to. Some of the documents listed below may be available only under a nondisclosure agreement (NDA).

To request access to these documents, contact your local field applications engineer (FAE) or sales representative.

Table 13. Related documentation/resources

Document	Description	Link / how to access
Real-time Edge Software User Guide (document REALTIMEEDGEUG)	This document describes the features and implementation of Real-time Edge Software on NXP hardware platforms. The key technology components include Real-time Networking, Real-time System, and Protocols.	REALTIMEEDGEUG
Real-time Edge Yocto Project User Guide (document RTEDGEYOCTOUG)	This document describes Real-time Edge software Yocto layer and its usage. It includes steps to build a Real-time Edge image for both i.MX and Layerscape boards by using a Yocto project build environment.	RTEDGEYOCTOUG
i.MX 8M Plus Reference Manual	Provides a detailed description about the iMX8MP multicore processor and its features. It includes details of memory map, serial interfaces, power supply, chip features, and clock information.	IMX8MPRM
LA9310 Reference Manual	Provides a detailed description about the LA9310 processor and its features, such as memory map, serial interfaces, power supply, chip features, and clock information.	Contact FAE or sales representative.
LA9310 Data Sheet	Provides information about LA9310 SoC electrical characteristics, hardware design considerations, and ordering information.	Contact FAE or sales representative.
iMX8MP-LA9310 Reference Manual	Provides a detailed functional description of the iMX8MP-LA9310 Reference Design Board.	Contact FAE or sales representative.
LA9310 Chip Errata	Lists the details of all known silicon errata for the LA9310 SoC.	Contact FAE or sales representative for access.
LA9310 Design Checklist	This document provides recommendations for new designs based on the LA9310 SoC. This document can also be used to debug newly designed systems by highlighting those aspects of a design that merit special attention during initial system startup.	Contact FAE or sales representative.

Table 13. Related documentation/resources*...continued*

Document	Description	Link / how to access
CodeWarrior Development Studio for VSPA Architectures Getting Started Guide	This document explains how to install the CodeWarrior software, prepare the boards supported by the current release, and then create, build, and debug a simple project.	Contact FAE or sales representative.
LEUE-MT3812 RF User Guide	Provides detailed information on build and test procedure for RF characterization on SDR+MT3812.	Contact FAE or sales representative.
DFE Application (Document DFE_ LA9310_v1.0.1)	DFE Application is meant to provide a starting reference for developing L1 software for the LA9310 SoC.	Contact FAE or sales representative.
iq-player_sw_reference_manual	Sample code for non-5G usages of VSPA L1 programming.	Contact FAE or sales representative.

15 Acronyms

Table 14 defines the acronyms used in this document.

Table 14. Acronyms

Acronym	Definition
ADC	Analog-to-digital converter
DAC	Digital-to-analog converter
RF	Radio frequency
CLI	Command line interface
CM4	Arm Cortex-M4 core
CP	Control plane communication
DTCM	Data tightly coupled memory
FW	Firmware
Host	I.MX 8M Plus Linux host
IPC	Inter-processor communication. There are two types of IPC libraries: • Control plane communication (CP) • Data plane communication
IRQ	Interrupt request
ISR	Interrupt service routine
ITCM	Instruction tightly coupled memory
MSI	Message signaled interrupts
RFIC	Radio-frequency integrated circuit transceiver
RX	Receive
SDR	Software defined radio
Shiva	Kernel driver used to boot FreeRTOS and VSPA
TTI	Transmission time interval time
TVD	Temperature variance detector
TX	Transmit

16 Revision history

Table 15 summarizes revisions to this document.

Table 15. Revision history

Document ID	Release date	Description
UG10193 v.1.0	27 February 2025	Initial public release

17 Note about the source code in the document

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

Legal information

Contents

Date of release: 27 February 2025
Document identifier: UG10193