USER'S GUIDE

Apollo510 Lite Series Software Porting

Ultra-Low Power Apollo SoC Family

A-SOCAP5-UGGA05EN v1.0

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. "TYPICAL" PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING "TYPICALS" MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER'S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Revision	Date	Description
1.0	January 2026	Initial Release.

Reference Documents

Document ID	Description

Table of Contents

List of Tables

SECTION

1

Overview

This document is a guide for porting applications from the Apollo510/Apollo510B SDK to the Apollo510 Lite SDK which is the version of the SDK supporting the Apollo510 Lite Series of SoCs. The structures between the Apollo510/Apollo510B SDK and the Apollo510 Lite SDK are similar, with similar standard CMSIS register defines and HAL API structure. This document covers the changes in the HAL between the two device families to provide user with software guidelines when porting applications from the Apollo510/Apollo510B to the Apollo510 Lite.

Some peripherals are similar between the Apollo510 and the Apollo510 Lite with minimal impact when porting. These peripherals are listed in Table 1-1 and will not be individually covered in the document:

Table 1-1: Peripheral Notes

Peripheral	Notes
IOM	Mostly identical; Maximum frequency changed from 48 MHz to 24 MHz on the Apollo510 Lite Series.
UART	New clock divider option introduced on the Apollo510 Lite. No other configuration changes.
GPDMA	No configuration changes in the HAL.
RTC	No configuration changes in the HAL.

SECTION

2

ADC HAL

The ADC HAL between the Apollo510 and Apollo510 Lite are mostly identical, with the configuration and operational API naming and functionality staying the same between the two families. There are a few changes in the clock selections that are introduced in the HAL.

2.1 Configuration

The Apollo510 Lite's ADC Hal accepts additional clock options for AM_HAL_ADC_CLKSEL_HFRC_48MHz and AM_HAL_ADC_CLKSEL_PLL_POSTDIV in addition to supporting the default HFRC_24MHz.

Table 2-1: Configuration

Apollo510	Apollo510 Lite
Supported clock: AM_HAL_ADC_CLKSEL_HFRC_24MHz	Supported clock: ▪ AM_HAL_ADC_CLKSEL_HFRC_24MHz ▪ AM_HAL_ADC_CLKSEL_HFRC_48MHz ▪ AM_HAL_ADC_CLKSEL_PLL_POSTDIV

A set of new macro defines is also introduced to help extract the clock source and clock divider:

```
// ****************************************************************
//
//! @brief Macros to extract clock source and clock divider ratio //
from am_hal_adc_clksel_e.
//
// ****************************************************************
#define AM_HAL_ADC_CRM_CLKSRC_POS    (0)
#define AM_HAL_ADC_CRM_CLKSRC_MSK    (0x000000FF)
#define AM_HAL_ADC_CRM_CLKDIV_POS    (8)
#define AM_HAL_ADC_CRM_CLKDIV_MSK    (0x0000FF00)
```

SECTION

3

CLKMGR HAL

The Clock Management in the Apollo510 Lite has several changes introduced by clock source changes as well as the integration of the Radio Subsystem.

3.1 Configurations

A new HAL API is introduced: **uint32_t am_hal_clkmgr_initialize(void)**

This API sets up the communication infrastructure between the MCU and the Radio Subsystem. There is a required clock initialization routine and it is typically called from within the **am_bsp_low_power_init()** API in the BSP layer.

There are no changes to other API parameters or names. The mapping is the same between the Apollo510 and the Apollo510 Lite.

There have also been some data structure updates to the Clock Manager HAL.

1. The **am_hal_clkmgr_user_id_e** enum group has enums added or removed based on the specific users available on the Apollo510 Lite device. A few notable additions and removals are as follows:
   a. Added AM_HAL_CLKMGR_USER_ID_I3C
   b. Added AM_HAL_CLKMGR_USER_ID_PLLVCO
   c. Removed AM_HAL_CLKMGR_USER_ID_AUDADC
   d. Removed AM_HAL_CLKMGR_USER_ID_CLKOUT_XTAL
   e. Removed AM_HAL_USER_ID_HFRC2
2. The **am_hal_clkmgr_clock_id_e**, which is the enum group that identifies the clocks to be managed, has the following changes:
   a. Added AM_HAL_CLKMGR_CLK_ID_PLLVCO
   b. Added AM_HAL_CLKMGR_CLK_ID_PLLPOSTDIV
   c. Removed AM_HAL_CLKMGR_CLK_ID_HFRC2
   d. Removed AM_HAL_CLKMGR_CLK_ID_SYSPLL

3. Added two enums: **am_hal_clkmgr_xtalhs_state_e** and **am_hal_clkmgr_xtalhs_op_e** for XTAL-HS configuration.
4. Removed all structures referencing the HFRC2 as it is not present on the Apollo510 Lite.
5. Added a new structure **am_hal_clkmgr_rfxtal_config_t** for RFXTAL Configuration.

Another change between the Apollo510 and the Apollo510 Lite is that the Apollo510 Lite's XTAL_HS clock source is sourced from the RFXTAL clock in the Radio Subsystem.

XTAL_HS – High frequency crystal on the MCU, which is leveraged from the Radio Subsystem's high-frequency crystal (RF XTAL).

In order to request the XTAL_HS clock, the following steps need to happen, which is mostly taken care of by the HAL:

1. The Radio Subsystem core must be enabled by calling the **am_hal_pwrctrl_rss_bootup()** API before requesting the **XTAL_HS**. This needs to be performed by the user.
2. The application or the underlying peripheral HAL will request the clock from clock manager.
3. The clock manager will request the clock source from the Radio Subsystem through the IPC mailbox.
4. The Radio Subsystem receives the request and enables the clock source.
5. The Clock Manager receives the acknowledgment, and the **XTAL_HS** is available to the MCU.

SECTION

4

GPIO HAL

The GPIO HAL has no changes in the API names or parameters between the Apollo510/Apollo510B and the Apollo510 Lite. The API mapping is the same between the two device families. There are a few changes in the configuration structure of the GPIO, and some characteristic changes of the GPIO pin.

4.1 Configurations

The main change in the GPIO HAL between the two device families is in the **am_hal_gpio_pincfg_t** structure, which is used to configure a GPIO pin. In the Apollo510 Lite, two additional fields are added for GPIO configuration:

1. **eTimerSel** – this bit field is used to configure GPIO for a non-secure timer or the Radio Subsystem timer. The user can consult the Apollo510 Lite's GPIO register set and look at the register field description for the **TIMERSELn** field in the **PINCFGn** registers.
2. **eDeeperSleepCfg** – this bit field configures the GPIO's pull-up and pull-down direction when entering deeper sleep. Note this field only applies to GPIOs that are not always on. Always on GPIOs have the ability to wake up the device from deeper sleep and will have those fields reserved. Refer to the Apollo510 Lite Series datasheet for a list of GPIOs that are always on. For a detailed description of the fields, the user can consult the register set and look at the register field description for the **DSPULLCFGn** field in the **PINCFGn** registers.

The drive strength and slew rate settings also changed between the Apollo510 and the Apollo510 Lite.

For the Apollo510, there are up to 8 different drive strength options for high-speed GPIO pads with no slew rate option, and up to 4 drive strength options for normal-speed GPIO pads with a slew rate option.

For the Apollo510 Lite, both normal-speed and high-speed GPIO have up to 4 drive strength options, but with different characteristics. For both normal-speed and high-speed GPIO, the drive strength configuration bits are located in the **DSn** field in the **PINCFGn** registers. For normal-speed GPIO pads, the drive strength consists of:

0P1X = 0x0 - 0.1x output driver selected
0P5X = 0x1 - 0.5x output driver selected
0P75X = 0x2 - 0.75x output driver selected
1P0X = 0x3 - 1.0x output driver selected

The slew rate bit (SR) which is bit 12 of the **PINCFGn** register can be set to 0 or 1 for normal-speed GPIO pads to configure the slew rate.

For high-speed GPIO on Apollo510 Lite, the slew rate bit is not used to configure slew rate. Instead, it is used to set the speed grade, and when this bit is set, it is configured for faster performance. Since all high-speed GPIO must operate below 2.5V (by VDDHn specifications), this bit should be set to 1.

The drive strength control for high speed GPIO pads is updated to:

PVT1_ODT50OHM = 0x0 - PVT1 - ODT 50 Ohm[1]
PVT2_ODT50OHM = 0x1 - PVT2 - ODT 50 Ohm
PVT3_ODT50OHM = 0x2 - PVT3 - ODT 50 Ohm
ODT_0OHM = 0x3 - Drive ODT 0 Ohm

These different drive strength settings for high-speed pad work best under different voltage levels. The table below shows the best setting for each voltage level.

Table 4-1: Configuration

Drive Strength Setting	Best Voltage Level	Notes
PVT1_ODT50OHM	3.3V	These settings will not apply since the max voltage supported is 2.2V
PVT2_ODT50OHM	1.8V	
PVT3_ODT50OHM	1.2V	
0DT_0OHM	N/A	Lowest impedance across all PVT.

[1]PVT - process/voltage/temperature; ODT - on-die termination. On-die termination is controlled by drive strength control bits in the GPIO_PINCFGn_DSn field.

SECTION

5

I²S HAL

The I²S HAL stayed mostly identical between the Apollo510 and the Apollo510 Lite with minor changes on some enum types. The API naming is the same between the two device families.

5.1 Configuration

The Apollo510 Lite added an additional enum for the MCLKOUT configuration: **am_hal_i2s_mclkout_sel_e** whereas apollo510 only has the **am_hal_i2s_clksel_e** enum type. This new enum type is also included in the **am_hal_i2s_config_t**, which is used by the **am_hal_i2s_configure** API to configure the I²S peripheral. The Apollo510 Lite has a reduced set of clock selections compared to the Apollo510. HFRC2 clock selection is removed completely as it is not present on the Apollo510 Lite, and all but one variant of HFRC is removed from **am_hal_i2s_clksel_e** enum group as well.

The new **am_hal_i2s_mclkout_sel_e** has the following enum fields:

```
//*****************************************************************************
//
//! Clock source of I2S MCLKOUT, which is used to drive external devices.
//
//*****************************************************************************
typedef enum
{
    //! HFRC 48 MHz, the actual frequency may change if HFADJ is enabled.
    AM_HAL_I2S_MCLKOUT_SEL_HFRC48     =
CRM_I2S0MCLKOUTCRM_I2S0MCLKOUTCLKSEL_HFRC48,

    //! External crystal oscillator.
    AM_HAL_I2S_MCLKOUT_SEL_XTAL       =
CRM_I2S0MCLKOUTCRM_I2S0MCLKOUTCLKSEL_RF_XTAL_48MHz,

    //! POSTDIV, an output of PLL.
    AM_HAL_I2S_MCLKOUT_SEL_PLLPOSTDIV =
CRM_I2S0MCLKOUTCRM_I2S0MCLKOUTCLKSEL_PLLPOSTDIV,
```

```
    //! FOUT3, an output of PLL, its frequency is equal to POSTDIV / 6.
    AM_HAL_I2S_MCLKOUT_SEL_PLLFOUT3   =
CRM_I2S0MCLKOUTCRM_I2S0MCLKOUTCLKSEL_PLLFOUT3,

    //! FOUT4, an output of PLL, its frequency is equal to POSTDIV / 8.
    AM_HAL_I2S_MCLKOUT_SEL_PLLFOUT4   =
CRM_I2S0MCLKOUTCRM_I2S0MCLKOUTCLKSEL_PLLFOUT4,

    //! External clock from GPIO 15.
    AM_HAL_I2S_MCLKOUT_SEL_EXTREF     =
CRM_I2S0MCLKOUTCRM_I2S0MCLKOUTCLKSEL_EXTREF_CLK,

    //! A virtual clock for internal use only, shouldn't be set by
applications.
    AM_HAL_I2S_MCLKOUT_SEL_OFF,
} am_hal_i2s_mclkout_sel_e;
```

The **am_hal_i2s_config_t** structure is updated with a new field for MCLKOUT selection and the MCLKOUT divider.

```
//
//! MCLKOUT selection.
//
am_hal_i2s_mclkout_sel_e eMclkout;
uint32_t ui32MclkoutDiv;
} am_hal_i2s_config_t;
```

SECTION

6

IOS HAL

For the IOS peripheral, the main change is that the macro define for the IOS name is changed from IOSLAVE to IOSLAVEFD0 due to Apollo510 Lite only having full-duplex IOS. This affects some of the macro naming definitions.

For example,

```
#define AM_HAL_IOS_INT_FSIZE  IOSLAVE_INTEN_FSIZE_Msk
#define AM_HAL_IOS_INT_FOVFL  IOSLAVE_INTEN_FOVFL_Msk
#define AM_HAL_IOS_INT_FUNDFL IOSLAVE_INTEN_FUNDFL_Msk
#define AM_HAL_IOS_INT_FRDERR IOSLAVE_INTEN_FRDERR_Msk
#define AM_HAL_IOS_INT_GENAD  IOSLAVE_INTEN_GENAD_Msk
#define AM_HAL_IOS_INT_IOINTW IOSLAVE_INTEN_IOINTW_Msk
#define AM_HAL_IOS_INT_XCMPWR IOSLAVE_INTEN_XCMPWR_Msk
#define AM_HAL_IOS_INT_XCMPWF IOSLAVE_INTEN_XCMPWF_Msk
#define AM_HAL_IOS_INT_XCMPRR IOSLAVE_INTEN_XCMPRR_Msk
#define AM_HAL_IOS_INT_XCMPRF IOSLAVE_INTEN_XCMPRF_Msk
#define AM_HAL_IOS_INT_DCMP   IOSLAVE_INTEN_DCMP_Msk
#define AM_HAL_IOS_INT_DERR   IOSLAVE_INTEN_DERR_Msk
```

are changed to

```
#define AM_HAL_IOS_INT_FSIZE  IOSLAVEFD0_INTEN_FSIZE_Msk
#define AM_HAL_IOS_INT_FOVFL  IOSLAVEFD0_INTEN_FOVFL_Msk
#define AM_HAL_IOS_INT_FUNDFL IOSLAVEFD0_INTEN_FUNDFL_Msk
#define AM_HAL_IOS_INT_FRDERR IOSLAVEFD0_INTEN_FRDERR_Msk
#define AM_HAL_IOS_INT_GENAD  IOSLAVEFD0_INTEN_GENAD_Msk
#define AM_HAL_IOS_INT_IOINTW IOSLAVEFD0_INTEN_IOINTW_Msk
#define AM_HAL_IOS_INT_XCMPWR IOSLAVEFD0_INTEN_XCMPWR_Msk
#define AM_HAL_IOS_INT_XCMPWF IOSLAVEFD0_INTEN_XCMPWF_Msk
#define AM_HAL_IOS_INT_XCMPRR IOSLAVEFD0_INTEN_XCMPRR_Msk
#define AM_HAL_IOS_INT_XCMPRF IOSLAVEFD0_INTEN_XCMPRF_Msk
#define AM_HAL_IOS_INT_DCMP   IOSLAVEFD0_INTEN_DCMP_Msk
#define AM_HAL_IOS_INT_DERR   IOSLAVEFD0_INTEN_DERR_Msk
```

An additional API to configure the FIFOPTR offset is also added on the Apollo510 Lite IOS module:

```
//*****************************************************************************
//
//! @brief Set the FIFOPTR register to the offset value of the given IOS instance
//!
//! @param pHandle - handle for the IOS.
//! @param ui32Offset value to change the register to
//!
//! This function updates the value of the FIFOPTR register to the given offset
//!
//! @return success or error code
//
//*****************************************************************************
extern uint32_t am_hal_ios_fifo_ptr_set(void *pHandle, uint32_t ui32Offset);
```

If the user intends to use IOS in half duplex mode on Apollo510 Lite, switch to one of the full duplex modules first (IOSFD0 or IOSFD1), then use the half duplex mode feature on the full duplex mode instance. Be aware that only one full duplex module can operate at once when operating in the half duplex mode.

SECTION

7

MSPI HAL

The MSPI HAL remains mostly identical between the Apollo510 and the Apollo510 Lite. The API naming is the same and there are only a few minor changes on supported mode and clock rates.

The following two MSPI interface mode and chip selection are added to the **am_hal_mspi_device_e** enum

- •AM_HAL_MSPI_FLASH_QUAD_DDR_CE0
- •AM_HAL_MSPI_FLASH_QUAD_DDR_CE1

The following SPI clock speeds are added to the **am_hal_mspi_clock_e** enum

- AM_HAL_MSPI_CLK_750KHZ
- AM_HAL_MSPI_CLK_1MHZ
- AM_HAL_MSPI_CLK_2MHZ

The following new MSPI request types are added to the **am_hal_mspi_request_e** enum

- AM_HAL_MSPI_REQ_LINK_CM4
- AM_HAL_MSPI_HALF_WORD_REVERSE_EN
- AM_HAL_MSPI_HALF_WORD_REVERSE_DIS

The Apollo510 Lite MSPI clock rates that are generated from 250MHz clock source will use the PLLVCO instead of the HFRC2 which is used on the Apollo510. The clock rates that are generated from the HFRC will continue using the corresponding HFRC frequency as the source.

Note that when PLLVCO is not fully running and it is configured as the clock source for MSPI, the MSPI operation will need to wait for PLL to lock. This situation is common if MSPI is enabled shortly after waking from deep sleep, or if 250MHz MSPI is used while MCU is in LP or HP1 mode. The MSPI HAL configuration will check to make sure the PLL is fully locked before proceeding with the MSPI operation.

SECTION

8

PDM HAL

8.1 Configuration

The PDM HAL has minimal changes in the configuration APIs. All API names are identical between the Apollo510 and the Apollo510 Lite's PDM HAL, with some enum changes that should not affect general PDM configuration.

There are a few new features added to the PDM structure that can be configured. The following new fields are added to the **am_hal_pdm_config_t** structure.

```
//! The data flow direction after PDM conversion.
am_hal_pdm_data_flow_direction_e eDataFlowDirection;

//! The word width of PCM data if the data flows to memory.
am_hal_pdm_data_word_width_e eWordWidth;

//! I2S role if I2S is enabled.
bool bI2sMaster;
```

Below are the enum definitions for **am_hal_pdm_data_flow_direction_e** and **am_hal_pdm_data_word_width_e**.

```
//*****************************************************************************
//
//! The data flow direction after PDM conversion.
//!
//! The data can flow to memory via DMA, to external pads via I2S (a sub-
//! module of PDM), or to both destinations simultaneously.
//
//*****************************************************************************
typedef enum
{
AM_HAL_PDM_DATA_FLOW_TO_MEMORY = 0,
AM_HAL_PDM_DATA_FLOW_TO_I2S = 1,
AM_HAL_PDM_DATA_FLOW_TO_BOTH = 2,
}
```

```
am_hal_pdm_data_flow_direction_e;

//
*****************************************************************************
*****
//
//! The word width of PCM data if the data flows to memory.
//
//
*****************************************************************************
*****
typedef enum
{
AM_HAL_PDM_DATA_WORD_WIDTH_24BITS = 0,
AM_HAL_PDM_DATA_WORD_WIDTH_16BITS = 1,
}
am_hal_pdm_data_word_width_e;
```

SECTION

9

PWRCTRL HAL

The Apollo510 Lite PWRCTRL HAL adds power control support for the Radio Subsystem and support for HP1/HP2 (192 MHz / 250 MHz) mode.

It also removed support for GPU power mode configuration.

9.1 Configurations

The following APIs are added in the Apollo510 Lite PWRCTRL HAL to provide configuration support for the Radio Subsystem (RSS).

Table 9-1: APIs Added in the Apollo510 PWRCTRL

API Name	Description
uint32_t am_hal_pwrctrl_rss_bootup(void)	Powers up RSS
uint32_t am_hal_pwrctrl_cm4_wakeup_req(void)	Wakes up the CM4, used internally by the rss_bootup API
uint32_t am_hal_pwrctrl_get_cm4_pwrstate(am_hal_pwrctrl_cm4_pwr_state_e * pCm4pwrstate)	Gets the current CM4 power state
uint32_t am_hal_pwrctrl_rss_pwroff(void)	Powers off RSS

These APIs will be mostly used in the BSP library to configure the board for low-power state.

CPU HP2 mode (250MHz) uses a PLL generated clock, as there is no HFRC2. The PLL clock requires some time to startup and lock.

An API is added to configure if entry to HP2 (250MHz) execution will stall and wait for the PLL to lock, or if execution will proceed in HP1 (192MHz) mode until PLL locks and then switch to HP2.

```
//*****************************************************************************
//! @brief Decides CPU will enter HP1 and wait until PLL_LOCK and switch
//! to HP2.
//! @param bWaitPllockForHp2 - true: CPU will NOT enter HP1. It will Wait
//! for PLL_LOCK and then
//! directly enter HP2 mode.
//! false:CPU will enter HP1 until PLL lock and then switch to HP2.
//! @return None.
//*****************************************************************************
extern void am_hal_pwrctrl_wait_pll_lock_for_hp2(bool
bWaitPllockForHp2);
```

By default, the device is configured to wait for HP2, which means that upon waking from deep sleep, the device will need to wait for PLL to lock (which takes a few hundred microseconds). The user can set the flag to false to disable wait for HP2. This will allow the CPU to enter HP1 and operate while the hardware logic wait for PLL to lock before switching to HP2. This allows faster wakeup from deep sleep.

The Apollo510 Lite has a new power mode where the 192 MHz clock is supported. It also supports more fine-tuned NVM power down settings.

```
//! NVM power down modes settings.
//
typedef enum
{
AM_HAL_PWRCTRL_NVM_RETENTION_POWER_DOWN = 0,
AM_HAL_PWRCTRL_NVM_DEEP_POWER_DOWN = 1,
AM_HAL_PWRCTRL_NVM_FULL_POWER_DOWN = 2,
} am_hal_pwrctrl_nvm_power_down_select_e;
```

The NVM can only be powered on or off as a whole block in the Apollo510 Lite, whereas the Apollo510 NVM is divided into two blocks and can be powered off individually.

The option to enable TCM is now unified as DTCM instead of ITCM and DTCM. The option to retain no, 128 kB, or 256 kB DTCM is provided.

```
//
//! DTCM enable settings.
//
typedef enum
{
AM_HAL_PWRCTRL_DTCM_NONE = PWRCTRL_MEMPWREN_PWRENTCM_NONE,
AM_HAL_PWRCTRL_DTCM128K = PWRCTRL_MEMPWREN_PWRENTCM_DTCM128K,
AM_HAL_PWRCTRL_DTCM256K = PWRCTRL_MEMPWREN_PWRENTCM_DTCM256K,
} am_hal_pwrctrl_dtcm_select_e;
```

A new deeper sleep option is available for configuration on top of the already existing deep sleep option. This option is only available on the Apollo510 Lite device.

SECTION

10

USB HAL

The USB HAL is mostly identical between the Apollo510 and the Apollo510 Lite. There are no changes in the API conventions and there are minor changes in the enum group defines.

The **am_hal_usb_phyclksrc_e** enum for the Apollo510 Lite has clock sources added and removed based on the availability and also has the enum value matched with its corresponding values in the CRM module.

```
typedef enum
{
AM_HAL_USB_PHYCLKSRC_HFRC_48M = CRM_USBREFCLKCRM_USBREFCLKCLKSEL_HFRC48,
AM_HAL_USB_PHYCLKSRC_RF_XTAL_48M = CRM_USBREFCLKCRM_USBREFCLKCLKSEL_RF_XTAL_48MHz,
AM_HAL_USB_PHYCLKSRC_PLLPOSTDIV = CRM_USBREFCLKCRM_USBREFCLKCLKSEL_PLLPOSTDIV,
AM_HAL_USB_PHYCLKSRC_PLLFOUT2 = CRM_USBREFCLKCRM_USBREFCLKCLKSEL_PLLFOUT2,
AM_HAL_USB_PHYCLKSRC_EXTREF_CLK = CRM_USBREFCLKCRM_USBREFCLKCLKSEL_EXTREF_CLK,
AM_HAL_USB_PHYCLKSRC_DEFAULT = AM_HAL_USB_PHYCLKSRC_EXTREF_CLK + 1,
} am_hal_usb_phyclksrc_e;
```

A new enum **am_hal_usb_phyclksrc_div_e** is added for the USB PHY reference clock source divider ratio and it is used for setting the USB PHY clock source and divider in the API:

```
extern uint32_t am_hal_usb_set_phy_clk_source(void *pHandle,
am_hal_usb_phyclksrc_e eClkSrc, am_hal_usb_phyclksrc_div_e eClkDiv);
```

The **am_hal_usb_phyclksrc_div_e** has the following fields:

```
typedef enum
{
AM_HAL_USB_PHYCLKSRC_DIV_1 = 0,
AM_HAL_USB_PHYCLKSRC_DIV_2 = 1,
AM_HAL_USB_PHYCLKSRC_DIV_3 = 2,
AM_HAL_USB_PHYCLKSRC_DIV_4 = 3,
} am_hal_usb_phyclksrc_div_e;
```

SECTION

11

Watchdog HAL

The watchdog HAL has a reduced clock source. The **XTAL_HS** and **XTAL_HS_DIV2** clock sources are removed.

11.1 Configuration

The watchdog HAL added an additional configuration allowing watchdog IRQ to be sent to NMI, resulting in the generation of a NMI instead of watchdog IRQ when time out.

```
//*****************************************************************************
//
//! @brief Enable watchdog IRQ sending to NMI.
//!
//! Generate a NMI instead of watchdog IRQ when timeout.
//!
//! @return WDT status code.
//
//*****************************************************************************
extern uint32_t am_hal_wdt_enable_nmi(am_hal_wdt_select_e eTimer);

//*****************************************************************************
//
//! @brief Disable watchdog IRQ sending to NMI.
//!
//! Disable generating NMI when watchdog timeout.
//!
//! @return WDT status code.
//
//*****************************************************************************
extern uint32_t am_hal_wdt_disable_nmi(am_hal_wdt_select_e eTimer);
```

SECTION

12

Apollo510 Lite New Features

The Apollo510 Lite device has a few new features on top of the Apollo510 device's feature set.

New features:

- A new I^{3}C module
- A new deeper sleep mode

12.1 I^{3}C Module

12.1.1 Configuration

The I^{3}C HAL has a comprehensive list of configuration APIs for fine tuning the peripheral. The general APIs that would be used to configure the I^{3}C in a general application level would be:

```
//*****************************************************************************
//
//! @brief I3C initialization function
//!
//! @param ui32Module   - module instance.
//! @param ppHandle     - returns the handle for the module instance.
//! @param eXferMode    - Host Xfer Mode (PIO or DMA).
//! @param eSpeedMode   - Host Speed Mode.
//!
//! This function accepts a module instance, allocates the interface and
//! then returns a handle to be used by the remaining interface functions.
//!
//! @return status      - generic or interface specific status.
//
//*****************************************************************************
extern uint32_t am_hal_i3c_controller_init(uint32_t ui32Module,
                      void **ppHandle,
                      am_hal_i3c_xfer_mode_e eXferMode,
                      am_hal_i3c_speed_mode_e eSpeedMode);
```

This API is used to configure, initialize, and enable the I^{3}C peripheral while also providing a pointer to the module instance to be used for other interface functions.

The following parameters are required for this API's usage:

Table 12-1: Required Parameters for API Usage

Parameters	Description
uint32_t ui32Module	I^{3}C module instance used
void **ppHandle	Interface instance pointer to be returned by this APIs. The interface allocated would be a struct type **am_hal_i3c_register_state_t**
am_hal_i3c_xfer_mode_e eXferMode	enum for I^{3}C xfer mode selection. Either PIO or DMA mode can be selected
am_hal_i3c_speed_mode_e eSpeedMode	enum for I^{3}C speed selection: **//** **//! I3C Speed Mode** **//** **typedef enum** **{** **AM_HAL_I3C_SDR0 = 0x0, // 12MHz** **AM_HAL_I3C_SDR1 = 0x1, // 8MHz** **AM_HAL_I3C_SDR2 = 0x2, // 6MHz** **AM_HAL_I3C_SDR3 = 0x3, // 4MHz** **AM_HAL_I3C_SDR4 = 0x4, // 2MHz** **AM_HAL_I3C_HDR_TS = 0x5,** **AM_HAL_I3C_HDR_DDR = 0x6,** **AM_HAL_I3C_I2C_MODE0 = 0x0, // 400KHz** **AM_HAL_I3C_I2C_MODE1 = 0x1, // 1MHz** **AM_HAL_I3C_I2C_MODE2 = 0x2, // User defined speed** **AM_HAL_I3C_I2C_MODE3 = 0x3, // User defined speed** **AM_HAL_I3C_I2C_MODE4 = 0x4, // User defined speed** **} am_hal_i3c_speed_mode_e;**

Other APIs for customizing the I3C configuration:

```
//*****************************************************************
//
//! @brief Set user defined I2C mode speed
//!
//! @param pHandle       - handle for the interface.
//! @param eI2CSpeedMode - user defined I2C speed mode
//! @param ui8LowCnt     - count for SCL Low
//! @param ui8HighCnt    - count for SCL High
//!
//! This function sets the I2C speed via setting duty cycles of both
//! low and high period of SCL.
```

```
//!
//! @return status      - generic or interface specific status.
//
//*****************************************************************
extern uint32_t am_hal_i3c_set_i2c_speed(void *pHandle, am_hal_i3c_-
speed_mode_e eI2CSpeedMode, uint8_t ui8LowCnt, uint8_t ui8HighCnt);


//*****************************************************************
//
//! @brief I3C IBI enable function
//!
//! @param pHandle      - handle for the interface.
//!
//! @param psTransaction - pointer to the transaction control
//! structure.
//!
//! This function enables I3C device IBI.
//!
//! @return status      - generic or interface specific status.
//
//*****************************************************************
extern uint32_t am_hal_i3c_ibi_enable(void *pHandle,
am_hal_i3c_transfer_t *psTransaction);
```

The I^3C peripheral does not require SCL/SDA pin configuration as there are dedicated SCL and SDA pins on the Apollo510 Lite device. These pins are hardware configured for I^3C usage.

12.1.2 Data Movement

The following APIs are used for I^3C data transfers

Table 12-2: General Data Transfers

API Name	Description
uint32_t am_hal_i3c_blocking_transfer(void *pHandle, am_hal_i3c_transfer_t *psTransaction);	Performs a transaction on the I^3C in PIO mode
uint32_t am_hal_i3c_nonblocking_transfer(void *pHandle, am_hal_i3c_transfer_t *psTransaction);	Performs a transaction on the I^3C in DMA mode
uint32_t am_hal_i3c_send_ccc_cmd(void *pHandle, am_hal_i3c_transfer_t *psTransaction);	Sends common command code (CCC) to secondary device.
uint32_t am_hal_i3c_do_daa(void *pHandle, am_hal_i3c_transfer_t *psTransaction);	Performs I^3C Dynamic Address Assignment

Where ***pHandle** is a handle to the I^3C interface that is allocated from the configuration API, and the **am_hal_i3c_transfer_t** is a transaction control structure defined in **am_hal_i3c.h**.

```
typedef struct
{
    //
    //! Number of bytes to transfer
    //
    uint32_t ui32NumBytes;

    //
    //! CCC Cmd ID
    //
    uint8_t  ui8CCCID;

    //
    //! speed mode
    //
    am_hal_i3c_speed_mode_e eSpeedMode;

    //
    //! Transfer Direction (Transmit/Receive)
    //
    am_hal_i3c_dir_e eDirection;

    //
    //! Device Struct
    //
    am_hal_i3c_device_t Device;

    //
    //! Enable Combo Xfer
    //
    bool bComboXfer;

    //
    //! Enable 16bit Offset. Default 8bit.
    //
    bool bCombo16bitOfs;

    //
    //! Combo Xfer Offset
    //
    uint16_t ui16ComboOfs;

    //
    //! Data Buffer
    //
    uint32_t *pui32TxBuffer;
    uint32_t *pui32RxBuffer;

    //
    //! IBI Status & Data Buffer
```

```
        //
        uint32_t *pui32IbiStatBuf;
        uint32_t *pui32IbiDataBuf;

        //
        //! Command Ring Buffer
        //
        uint32_t *pui32CmdRingBuf;
        uint32_t ui32CmdRingBufLen;

        //
        //! Response Ring Buffer
        //
        uint32_t *pui32RespRingBuf;
        uint32_t ui32RespRingBufLen;

    } am_hal_i3c_transfer_t;
```

12.1.3 Interrupts and Status

The following APIs are used for I^{3}C interrupt and status servicing.

Table 12-3: Interrupt and Status Servicing

API Name	Description
uint32_t am_hal_i3c_intr_signal_enable(void *pHandle, uint32_t ui32IntMask);	Enables the interrupt signal flags to generate interrupts on the I^{3}C peripheral. The corresponding interrupt status flags also need to be enabled to send interrupt signal to the CPU.
uint32_t am_hal_i3c_intr_signal_disable(void *pHandle, uint32_t ui32IntMask);	Disables the corresponding interrupt signal flags.
uint32_t am_hal_i3c_intr_status_enable(void *pHandle, uint32_t ui32IntMask);	Enables the interrupt status flag. The corresponding interrupt signal flags also need to be enabled to send interrupt signal to the CPU.
uint32_t am_hal_i3c_intr_status_disable(void *pHandle, uint32_t ui32IntMask);	Disables interrupt status flag.
uint32_t am_hal_i3c_intr_status_get(void *pHandle, uint32_t *pui32Status, bool bEnabledOnly);	Gets interrupt status flag value; option to only report enabled interrupts.

Table 12-3: Interrupt and Status Servicing *(Continued)*

API Name	Description
uint32_t am_hal_i3c_intr_status_clear(void *pHandle, uint32_t ui32IntMask);	Clears interrupt status flag value with the given mask.
uint32_t am_hal_i3c_interrupt_service(void *pHandle, uint32_t ui32IntStatus);	Enables non-blocking interrupt service designed to be called within the user-defined ISR.

12.1.4 Software Implementation

To configure the I^3C peripheral, the user can follow a general implementation procedure as provided in the following pseudo code block.

- Initialize I^3C controller using **am_hal_i3c_controller_init()** API
- Configure I^3C interrupt using the signal and status interrupt API
- Configure I^3C interrupt service and callback using the interrupt service API
- Configure secondary I^3C device and custom I^3C controller speed if needed
- Perform data operation using data transfer API
- If interrupt triggers, acknowledge and service interrupt

12.2 Deeper Sleep Mode

The deeper sleep state (SYS Deep Sleep Mode 3 as shown in the Apollo510 Lite device datasheet) is a new sleep mode on Apollo510 Lite device where minimal SSRAM and TCM memory is retained as needed for software to resume. This mode is similar to the deep sleep mode with one additional condition: it will have all but 16 GPIOs enter a power down stat. The 16 GPIOs that are still active in deeper sleep modes are: GPIO 5, 6, 8 – 21. This state has longer latency to resume, but it is the lowest level of power state of the SoC with functional retention mode.

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiq.com
sales@ambiq.com
+1 512. 879.2850

A-SOCAP5-UGGA05EN v1.0
January 2026