Atmosic Application Development Guide

Revision History

Date	Version	Description
March 30, 2021	0.50	Initial version created.
July 20, 2021	0.60	Updated various sections for SDK 4.1.0
September 29, 2021	0.70	Renamed this document and added 3 System Utilities and 4 Hardware Drivers sections.

Table of Contents

List of Figures

List of Tables

1 Overview

1.1 Software Architecture

The software for Atmosic ATM2/ATM3 SoC includes 3 layers:

- Application
- Application framework
- Protocol stack and hardware registers

Figure 1 shows ATM2/3 software architecture.

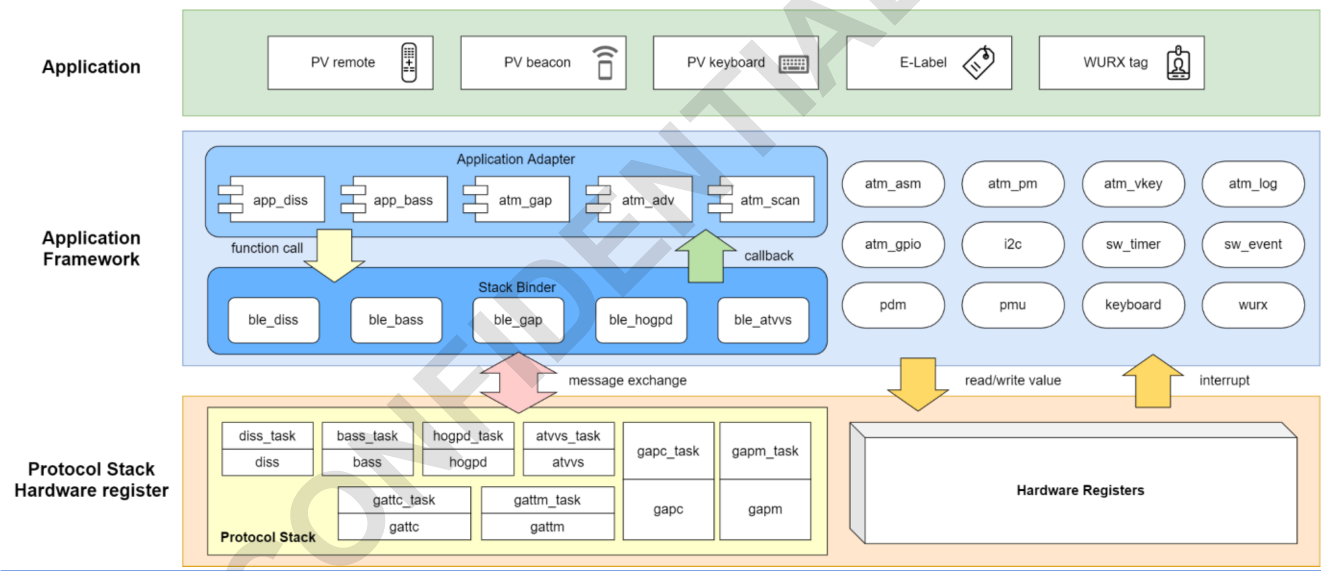


Figure 1 - ATM2/3 software architecture

The ATM2/ATM3 software is written in C using functions provided by the application framework. Currently some basic projects, such as broadcaster and observer, and some reference design projects, such as remote controller and Bluetooth LE UART bridge, are provided.

1.2 Application Framework

Atmosic Application Framework is a set of C library functions for application development. It was designed and developed based on CEVA RivieraWaves Bluetooth Low Energy software protocol stack and ATM2/ATM3 hardware registers.

The purpose of the application framework is to allow customers to develop their Bluetooth LE applications and products based on Atmosic ATM2/ATM3 SoC more easily and quickly.

The application framework consists of application adapters (Bluetooth LE activities and profiles managers), Atmosic profiles, stack binders, system utilities and hardware drivers. The stack binder transforms the function calls from application adapters into the sending messages to CEVA stack or the receiving messages from CEVA stack into the callback invokings to application adapters. The application adapter provides high level APIs for applications to interact with the profiles. The driver provides the functionality and configuration APIs to access ATM2/ATM3 hardware devices.

Table 1 lists all Atmosic application framework modules and their filenames and description.

Category	Path	Filename	Description
Application Adapter	lib/app_bass	app_bass.c app_bass.h	Battery Service (BAS) server application adapter
	lib/app-diss	app_diss.c app_diss.h	Device Information Service (DIS) server application adapter
	lib/app_gap	app_gap.c app_gap.h	LE Generic Attribute Profile (GAP) application adapter
	lib/app_hrps	app_hrps.c app_hrps.h	Heart Rate Profile (HRP) server application adapter
	lib/app_htpt	app_htpt.c app_htpt.h	Heath Thermometer Profile (HTP) server application adapter
	lib/app_otaps	app_otaps.c app_otaps.h	Atmosic firmware update Over-The-Air Profile (OTAP) server application adapter
	lib/atm_gap	atm_gap.c atm_gap_param.c atm_gap_param.h	LE device and link manager
	lib/atm_adv	atm_adv.c atm_adv_param.c atm_adv_param.h	LE advertising manager
	lib/atm_scan	atm_scan.c atm_scan_param.c atm_scan_param.h	LE scanning manager

	lib/atm_init	atm_init.c atm_init_param.c atm_init_param.h	LE Initiating manager
Atmosic Profile	lib/prf_bridge	prf_bridge.c prf_bridge.h	Atmosic bridge profile (based on Atmosic profile server)
	lib/atm_prfs	atm_prfs.c atm_prfs.h atm_prfs_task.c atm_prfs_task.h	Atmosic profile server
Stack Binder	lib/ble_task	ble_task.c ble_task.h	BLE application task message handler
	lib/ble_gap	ble_gap.c ble_gap.h	GAP API
	lib/ble_gap_sec	ble_gap_sec.c ble_gap_sec.h	GAP security part API
	lib/ble_bass	ble_bass.c ble_bass.h	BAS server API
	lib/ble_diss	ble_diss.c ble_diss.h	DIS server API
	lib/ble_hogpd	ble_hogpd.c ble_hogpd.h	HOGPD API
	lib/ble_atvvs	ble_atvvs.c ble_atvvs.h	Android TV voice service (ATVVS, aka Google Voice Service) server API
	lib/ble_atmprfs	ble_atmprfs.c ble_atmprfs.h	Atmosic profile server API
	lib/ble_gattc	ble_gattc.c ble_gattc.h	GATT message handler worked with ROM profile stack
	lib/ble_hrps	ble_hrps.c ble_hrps.h	HRP server API
	lib/ble_htpt	ble_htpt.c	HTP server API

		ble_htpt.h	
	lib/ble_lecb	ble_lecb.c ble_lecb.h	LE credit based connection procedures
	lib/ble_module	ble_module.c ble_module.h	Link time module
	lib/ble_otaps	ble_otaps.c.c ble_otaps.h	OTAP server API
System Utilities	lib/atm_asm	atm_asm.c atm_asm.h	Application state machine API
	lib/atm_log	atm_log.h	Log utility
	driver/atm_pm	atm_pm.c atm_pm.h	Power management API
	driver/sw_event	sw_event.c sw_event.h	Software event API
	driver/sw_timer	sw_timer.c sw_timer.h	Software timer API
	lib/at_cmd	at_cmd.c at_cmd.h at_cmd_pasr.c at_cmd_pasr.h at_cmd_sysreset.c	Atmosic AT command engine
	lib/at_cmd_set	at_cmd_event.h at_cmd_init.c at_cmd_init.h at_cmd_utils.c at_cmd_utils.h [at_cmd_handlers..]	AT command generic API and existing command handlers
Hardware Driver	driver/atm_ble	atm_ble.c atm_ble.h	BLE stack API extension
	driver/atm_button	atm_button.c	Button driver based on GPIO

		atm_button.h	
	driver/atm_gpio	atm_gpio.c atm_gpio.h	GPIO driver
	driver/atm_vkey	atm_vkey.c atm_vkey.h	Virtual key event model
	driver/brwnout	brwnout.c brwnout.h	Brownout support
	driver/ext_flash	ext_flash.c ext_flash.h	External flash driver
	driver/gadc	gadc.c gadc.h	ADC driver
	driver/hib_storage	hib_storage.c hib_storage.h	Hibernation storage driver
	driver/i2c	i2c.c i2c.h	I2C driver
	driver/interrupt	interrupt.c interrupt.h	Interrupt routing and handler
	driver/ir	ir.c ir.h nec_ir.c	IR driver
	driver/keyboard	keyboard.c keyboard.h keyboard_internal.h keyboard_param.h usb_hid_keys.h	Key scan matrix driver
	driver/led_blink	led_blink.c led_blink.h	LED driver
	driver/pdm	adpcm_enc.c adpcm_enc.h pdm_intp_data.h pdm_intp.c	PDM driver and ADPCM encoder

		pdm_intp.h pdm.c pdm.h	
	driver/pmu	pmu.c pmu.h	Energy harvesting management API
	driver/spi	qspi.h spi_flash.c spi_flash.h spi.c spi.h	SPI/QSPI driver
	driver/uart_flash	uart_flash.c	UART flash driver
	driver/uart0_raw	uart0_raw.c uart0_raw.h	UART driver
	driver/wurx	wurx.c wurx.h	Wakeup receiver driver

Table 1 - Atmosic application framework module list

2 Bluetooth LE Application

2.1 Bluetooth LE Connection Flow

Figure 2 depicts the overview of the connection flow to establish a Bluetooth LE connection via Atmosic application framework:

- Profile Registration
- Initialization
- Device discovery
  - Advertising
  - Scanning
- Connection establishment
- Connection mechanism
- Connection detachment

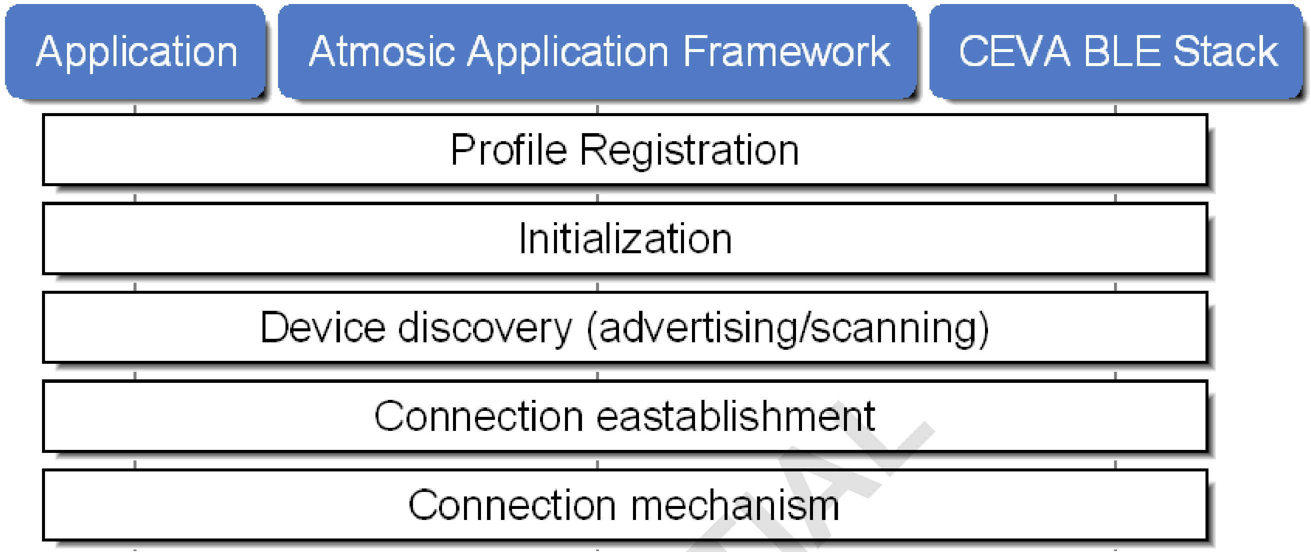


Figure 2 - Overview of the Atmosic Application Framework Connection Flow

2.1.1 Atmosic GAP Modules

Atmosic GAP modules are the basic modules of Atmosic Application Framework and required for all BLE applications. With **atm_gap_param** and **atm_gap** modules, applications could use their API to complete the BLE stack initialization and GAP messages exchange.

atm_gap_param Module

The **atm_gap_param** module provides an API to get a set of default parameters for GAP initialization and the header file **atm_gap_param_internal** defines the default values of GAP configurations in case these configurations are not overwritten in applications. **atm_gap_param_t** is the structure of GAP configuration. All parameters in **atm_gap_param_t** are shown in Table 2.

Member	Description
uint8_t* dev_name	dev_name is a pointer to a string array. This string will be the **Device Name (0x2A00)** characteristic value of GAP services . If NVDS tag 0x02 exists, the dev_name will be replaced.
uint8_t dev_name_len	Length of device name
uint8_t dev_name_max	Size of the device name array

uint16_t appearance	**Appearance (0x2A01)** characteristic value of GAP services.
struct gap_slv_pref slv_pref_params	**Peripheral(Slave) preferred connection parameters (0x2A04)** characteristic values of GAP services. This configuration includes connection interval minimum, connection interval maximum, slave latency and connection timeout parameters.
uint8_t* fix_irk	Force irk from the application for special usage. Null if not used.
struct gapm_set_dev_config_cmd dev_config	See Table 3
bd_addr_t addr	Bluetooth device address

Table 2 - atm_gap_param_t Atmosic GAP parameters

dev_config is the most important configuration in **atm_gap_param_t** for the initialization procedure to decide the device role, privacy, security and data exchange features. All configurations in **dev_config** are shown in Table 3.

Member	Descriptions
Generic Configuration	
uint8_t role	Bluetooth LE device role. It would be central, peripheral, observer, broadcaster or all roles.
Privacy Configuration	
uint16_t renew_dur	Address renew duration when controller privacy is enabled. Unit is second.
bd_addr_t addr	Device static private random address. If the NVDS tag 0x01 exists, it will be replaced.
struct gap_sec_key irk	Device IRK used for resolvable random BD address generation (LSB first).
uint8_t privacy_cfg	Privacy configuration. It is used to enable controller privacy.
Security Configuration	

pairing_mode	pairing mode is used to enable pairing feature, legacy or SC.
LE Data Length Extension Configuration	
sugg_max_tx_octets	The Controller's maximum transmitted number of payload octets to be use
sugg_max_tx_time	The Controller's maximum packet transmission time to be used.
L2CAP Configuration	
max_mtu	Maximum MTU acceptable for the device.

Table 3 - Device Configurations

atm_gap Module

The **atm_gap** module provides the required APIs to configure the device and initialize the BLE stack with variant configurations according to the application requirements. During the connection mechanism process, it provides APIs to accept the connection request, negotiate the connection parameters and detach the existing links .These API functions are shown in Table 4.

Function	Descriptions
Initialization	
atm_gap_prf_reg(char const *name, void const *parm)	Register profile with profile name and parameters.
atm_gap_start(atm_gap_param_t *init, atm_gap_cbs_t const *cbs)	Initialize BLE stack with init parameters and callbacks
Connection	
atm_gap_connect_accept(uint8_t conidx)	Accept connection request by connection index
atm_gap_connect_param_nego(uint8_t conidx, atm_gap_param_nego_t const *param)	Launch process of requesting new connection parameter
atm_gap_print_conn_param(atm_connect_info_t *info)	Print connection parameter

atm_gap_disconnect(uint8_t conidx, uint8_t reason)	Disconnect connection with reason by connection index

Table 4 - Frequently Used GAP API Functions

The **atm_gap** also provides a set of callback functions, **atm_gap_cbs_t** to handle the GAP messages from the BLE stack in the applications. The frequently used callback functions are shown in Table 5.

Callback	Descriptions
Initialization	
p_init_cfm	Confirmation of finish of atm_gap_start.
p_quick_start_op	Quick start operation. Called after bt reset but before atm_gap_start finished.
Scan	
p_ext_adv_ind	Indicate reception of advertising, scan response or periodic advertising data.
Connection	
p_conn_ind	Indicate that a connection has been established.
p_disc_ind	Indicate that a link has been disconnected
p_conn_param_updated_ind	Indication that connection parameters have been updated.
Pairing	
p_pair_req_ind	Indicate received pairing request from master.
p_sec_req_ind	Indicate received a security request from the slave.
p_pair_passkey_ind	Indication of passkey display or passkey input.
p_pair_numeric_ind	Indication of numeric comparison reception.
p_pair_ind	Indication of pairing result.

Table 5 - Frequently Used GAP Callbacks

2.2 Profile Registration

Profile registration phase is about registering profile API into the ***atm_gap*** module. After the profile is registered, the events of the registered profile could be received through its callback functions. Application calls ***atm_gap_prf_reg*** function with profile name and configuration parameters, as shown in Figure 3.

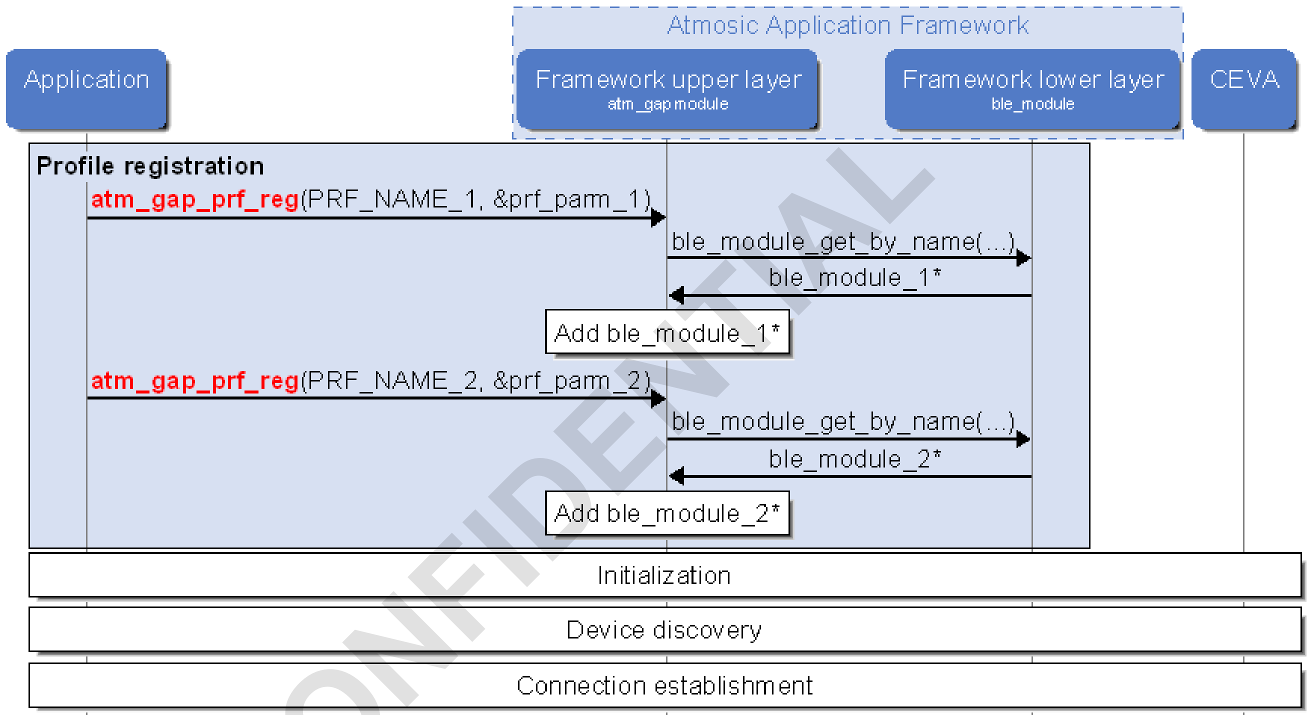


Figure 3 - Profile Registration

2.3 Initialization

After profile registration, the application calls the ***atm_gap_start*** with gap parameter and application callbacks. Once initialization is done, the callback ***p_init_cfm*** which is provided in ***atm_gap_start*** argument would be invoked to notify the application, as shown in Figure 4. Refer to the 2.7.1 Detailed LE Connection Flow section for details of this initialization sequence.

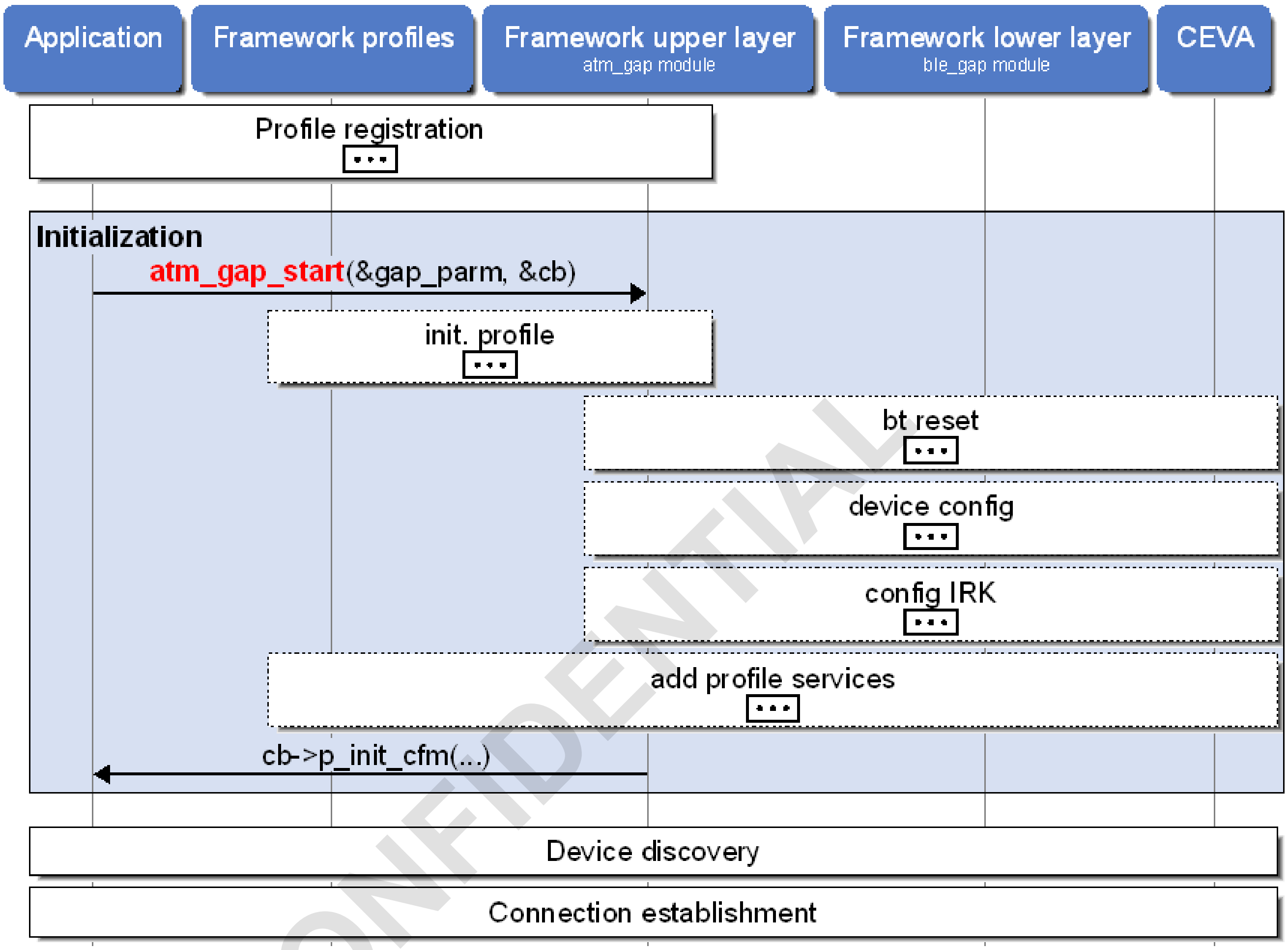


Figure 4 - Initialization Process

The following example shows how to configure the ***atm_gap_param_t*** and ***atm_gap_cbs_t*** in codes and call atm_gap_start for initialization in an application.

```
struct gapm_set_dev_config_cmd default_dev_conf = {
    .role = CFG_GAP_ROLE,
    .pairing_mode = CFG_GAP_PAIRING_MODE,
    .sugg_max_tx_octets = CFG_GAP_MAX_TX_OCTETS,
    .sugg_max_tx_time = CFG_GAP_MAX_TX_TIME,
    .max_mtu = CFG_GAP_MAX_LL_MTU,
    .att_cfg = CFG_GAP_ATT_CFG};

atm_gap_param_t gap_param= {
      .dev_name = dname,
```

```
        .dev_name_max = CFG_GAP_DNAME_MAX_LEN,
        .fix_irk = CFG_GAP_FIX_IRK,
        .appearance = CFG_GAP_APPEARANCE,
        .slv_pref_params = {
            .con_intv_min = CFG_GAP_CONN_INT_MIN,
            .con_intv_max = CFG_GAP_CONN_INT_MIN,
            .slave_latency = CFG_GAP_SLAVE_LATENCY,
            .conn_timeout = CFG_GAP_CONN_TIMEOUT,
        },
        .dev_config = (struct gapm_set_dev_config_cmd*)&default_dev_conf,
    };

const static atm_gap_cbs_t cb = {
    .p_conn_ind = _cb_conn_ind,
    .p_disc_ind = _cb_disc_ind,
    .p_pair_req_ind = _cb_pair_req_ind,
    .p_pair_passkey_ind = _cb_pair_passkey_ind,
    .p_pair_numeric_ind = _cb_pair_numeric_ind,
    .p_pair_ind = _cb_pair_ind,
    .p_conn_param_updated_ind = _cb_gap_conn_param_updated_ind,
    .p_init_cfm = _cb_gap_init_cfm,
    .p_quick_start_op = _cb_gap_quick_start_op,
};

...

static void _cb_gap_init_cfm(uint8_t status)
{
    if (status == GAP_ERR_NO_ERROR) {
            //Success
    } else {
        //Error
    }
}

static rep_vec_err_t _init(void)
{
    ...
    atm_gap_start(&gap_param, &cb);
    return (RV_DONE);
}
```

2.4 Device Discovery - Advertising

In the device discovery process, the device could enter the advertising or scanning state to make it able to be found by the other devices or to scan other nearby devices.

The Atmosic application framework provides advertise modules to manage the most common Bluetooth LE advertising behavior and provide the flexible APIs to let developers adapt the Bluetooth LE advertising modes through these modules easily.

2.4.1 Atmosic Advertisement Modules

The **atm_adv_param** and **atm_adv** are the modules used by the Atmosic application framework. The **atm_adv_param** module exports the API to prepare the advertisement parameter for the atm_adv API. The advertising parameter can come from Flash NVDS or predefined structure that can be overridden by using the define preprocessor. The parameters of advertisement are able to be changed and re-configured on the fly in application. See Figure 5.

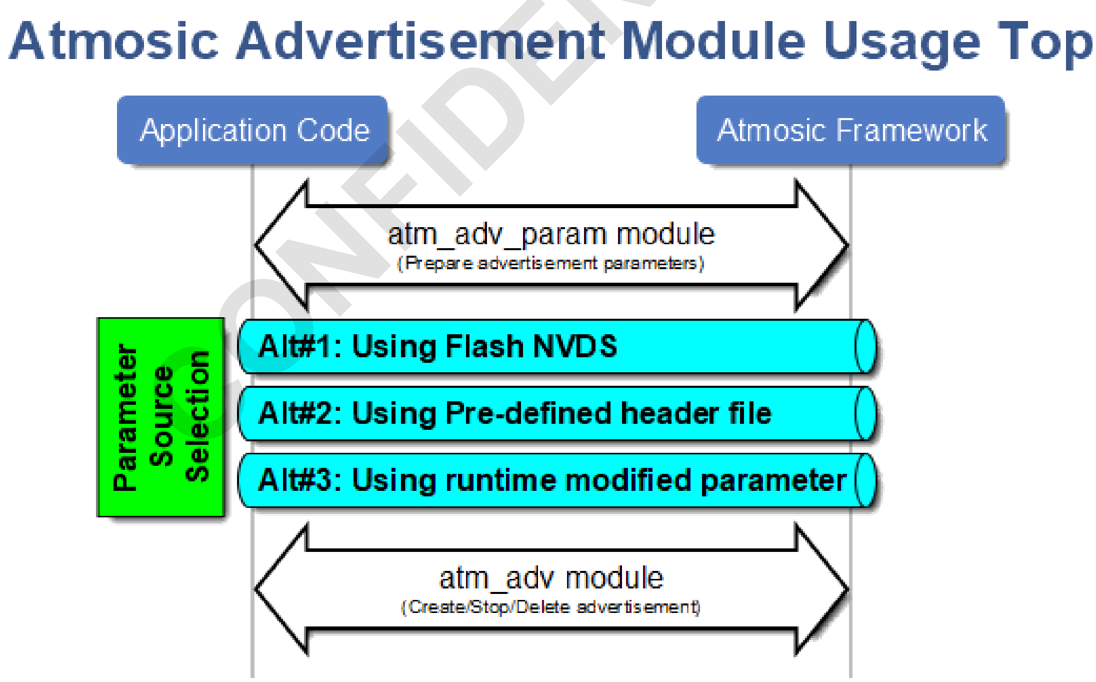


Figure 5 - Advertisement Modules

2.4.2 atm_adv_param Module

There are four advertisement parameters to create advertisements. The **atm_adv_param** module exports the APIs to retrieve the parameters from Flash NVDS or predefined structure.

Table 6 shows the parameters located in Flash NVDS.

API Name	Parameter Purpose	Description	Parameters NVDS tag ID
atm_adv_create_param_nvds	Create parameter	Advertisement type Discovery mode Tx power RF channel configuration...etc	0x06
atm_adv_advdata_param_nvds	Adv. payload parameter	Advertisement payload	0x0B
atm_adv_scandata_param_nvds	Scan response payload parameter	Scan response payload	0x0C
atm_adv_start_param_nvds	Start parameter	Advertisement duration Advertisement event counter	0x05

Table 6 - Flash NVDS Parameters

Table 7 shows the parameters located in predefined structure. Atmosic framework provides CFG_GAP_ADV_MAX_INST(2) instance number by default. Users can use makefile to overwrite the default instance number. The application code can use the instance number as input parameter when calling **atm_adv_xxx_param_get** API.

API Name	Parameter Purpose	Description	Parameters Location
atm_adv_create_param_get	Create parameter	Configure: advertisement type discovery mode tx power channel configuration...etc	default_adv_create_param of atm_adv_param.c
atm_adv_advdata_param_get	Adv. payload parameter	Configure advertisement payload	default_set_adv_data of atm_adv_param.c
atm_adv_scandata_param_get	Scan response payload parameter	Configure scan response payload	default_set_adv_data of atm_adv_param.c
atm_adv_start_param_get	Start parameter	Configure: advertisement duration advertisement event counter	default_adv_start_param of atm_adv_param.c

Table 7 - API Parameters Description

Get parameter from Flash NVDS

Flash NVDS has defined four tag identities for the parameter used by advertisement. The application code can use the makefile and toolchain to build the flash NVDS data then burn into an Atmosic chip. The application code can use ***atm_adv_xxx_parameter_nvds*** to retrieve those settings before using atm_adv module to create the advertisement.

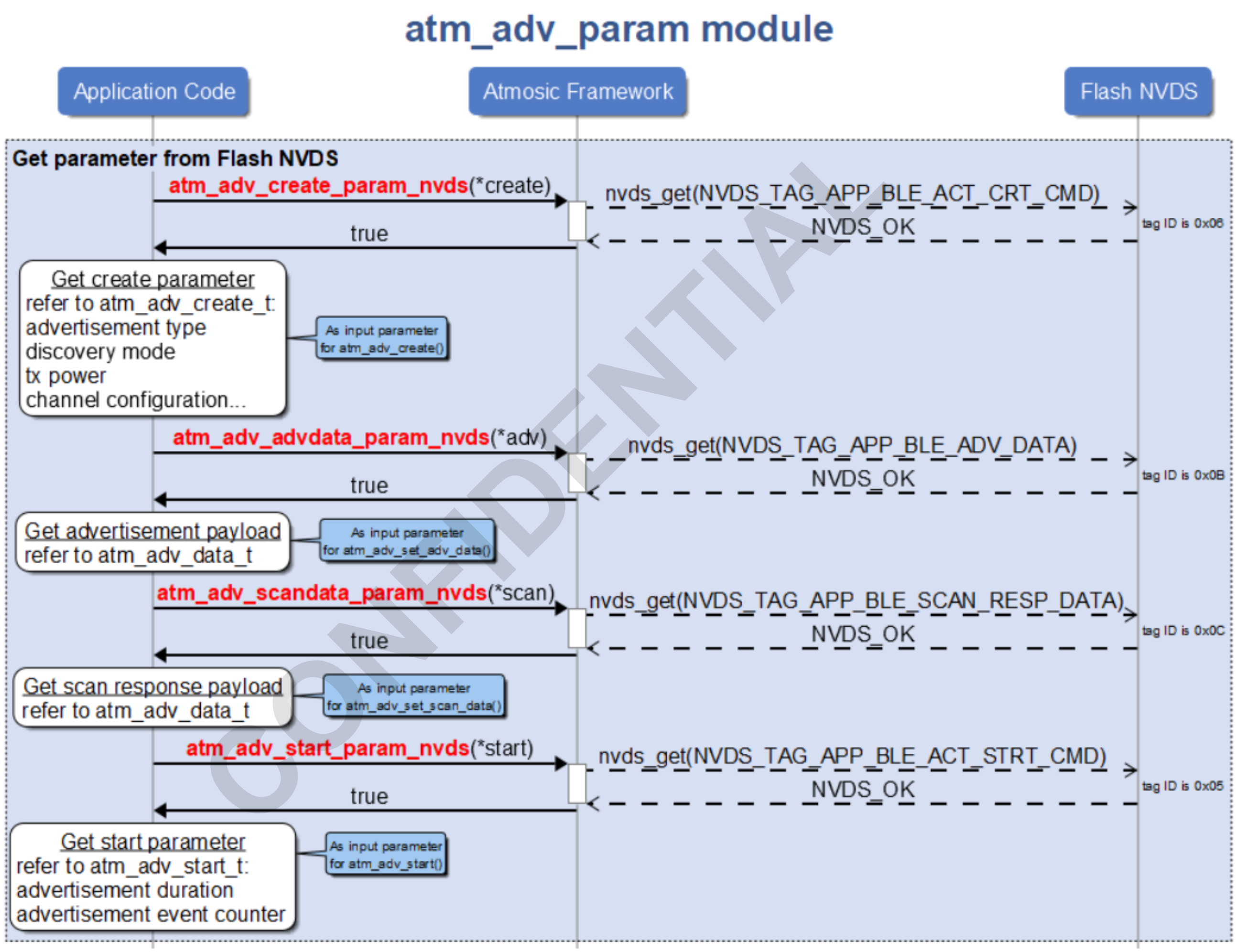


Figure 6 - atm_adv_param Flow

Get parameter from predefined structure

The predefined advertisement parameter instance is in the ***atm_adv_param*** module and also provides many overwrite fields to let the application code change the default value of predefined parameter instance. The parameters that can be overwritten are listed in `atm_adv_param_internal.h`. The application can apply the specific header file (-DGAP_PARM_NAME=”xxx.h” in makefile) to overwrite the parameter setting. See Figure 7.

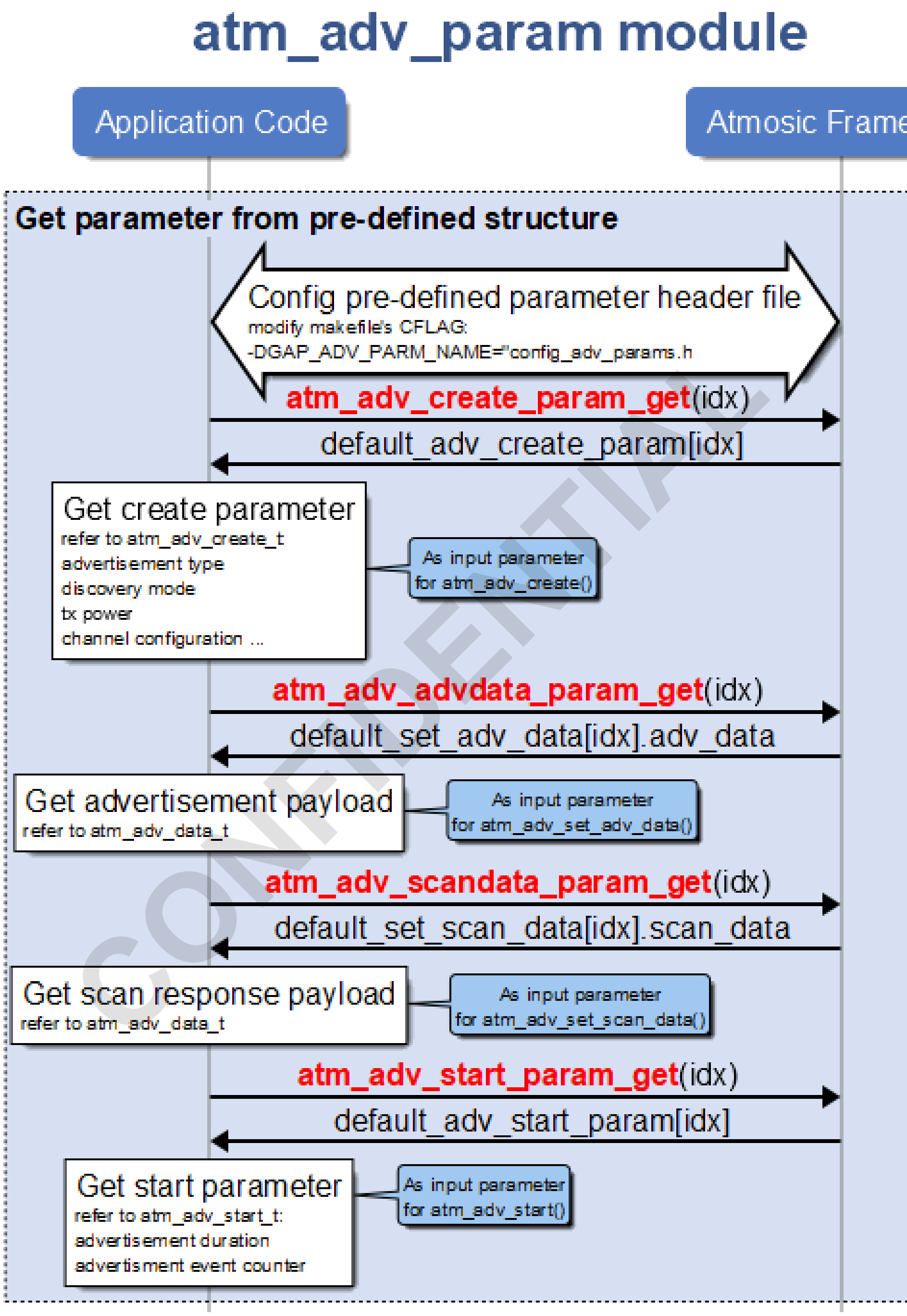


Figure 7 - Get Parameter from Predefined Structure

Using runtime modified parameter

The default data type is constant in the predefined advertisement parameter instance. If the application code will need to change the parameter in some cases in runtime. The application code needs to remove the constant data type using "-DCFG_ADV_xxx_PARAM_CONST=0". See Figure 8.

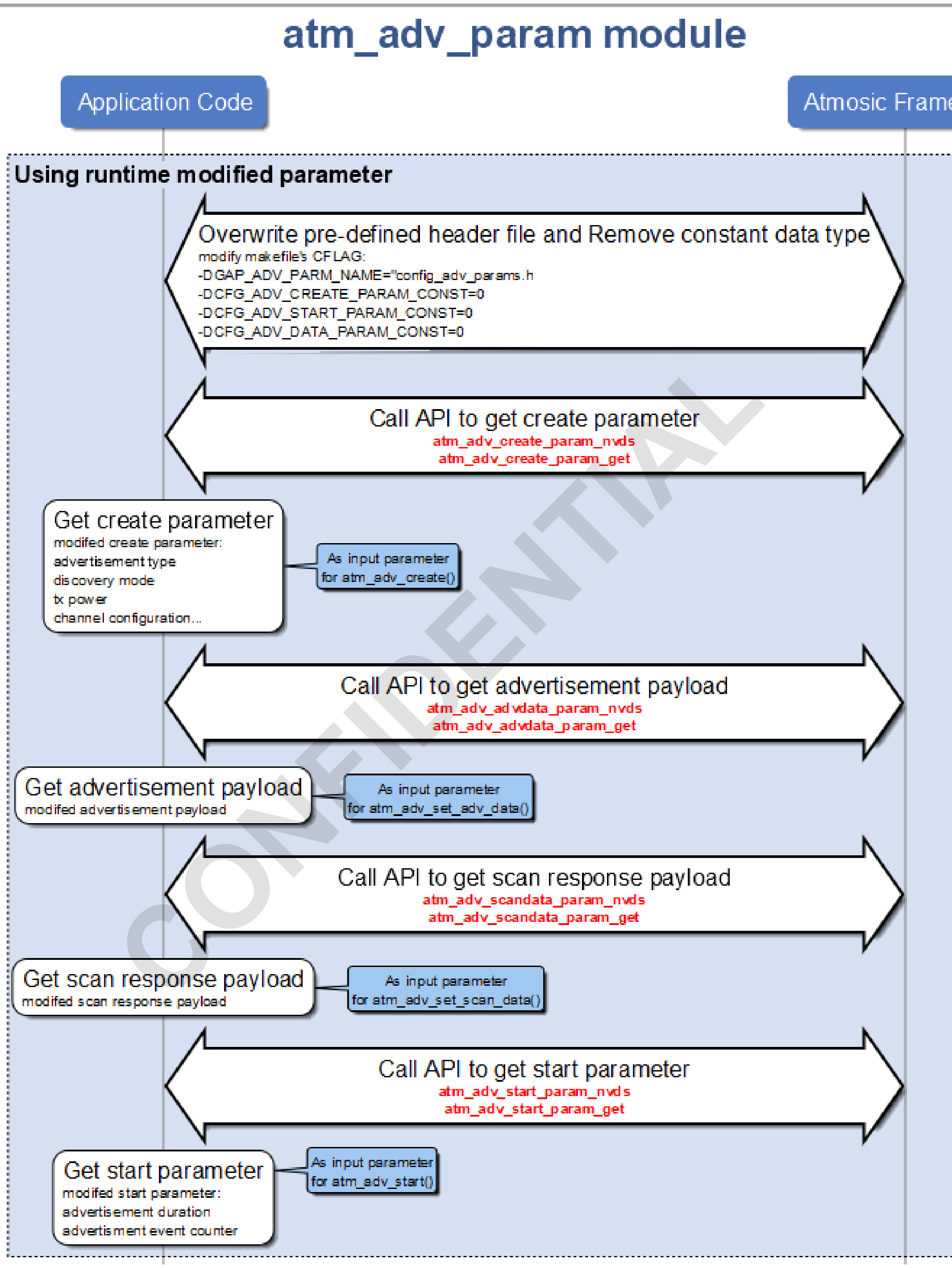


Figure 8 - Using Runtime Modified Parameter

2.4.3 atm_adv Module

After preparing the advertisement parameter datas via **atm_adv_param** module. The application will need to use the atm_adv module to control the advertisement. The application needs to provide the callback function that works with the atm_adv module. The atm_adv_reg is the first function call that provides the callback function. The **atm_adv** module will provide the event, activity index and status information for each **atm_adv exported** API. The application code uses **atm_adv_create** API to create advertisement activity. The callback function will receive the ATM_ADV_CTEATED event that includes the activity index. After getting the activity index value, the application code can use it as the input parameter for **atm_adv** exported API to control the advertisement. The application code can use **atm_adv_create** API to create multiple advertisement sets and uses the activity index to control specific advertisement sets.

The **atm_adv_stop** API will stop the advertisement transmission and the activity index is still valid for the atm_adv module. The application code will use **atm_adv_start** to re-enable the advertisement transmission again and doesn't need to call **atm_adv_create** and **atm_adv_set_xxx_data** again. The **atm_adv** module will destroy the activity instance when calling **atm_adv_delete** API. After this, the activity index will become invalid index for the atm_adv module.

Depending on advertising type, the advertisement and scan response payload will be used or not. The **atm_adv_set_data_sanity** API will perform sanity check before calling **atm_adv_start** API. See Figure 9 for the atm_adv flow.

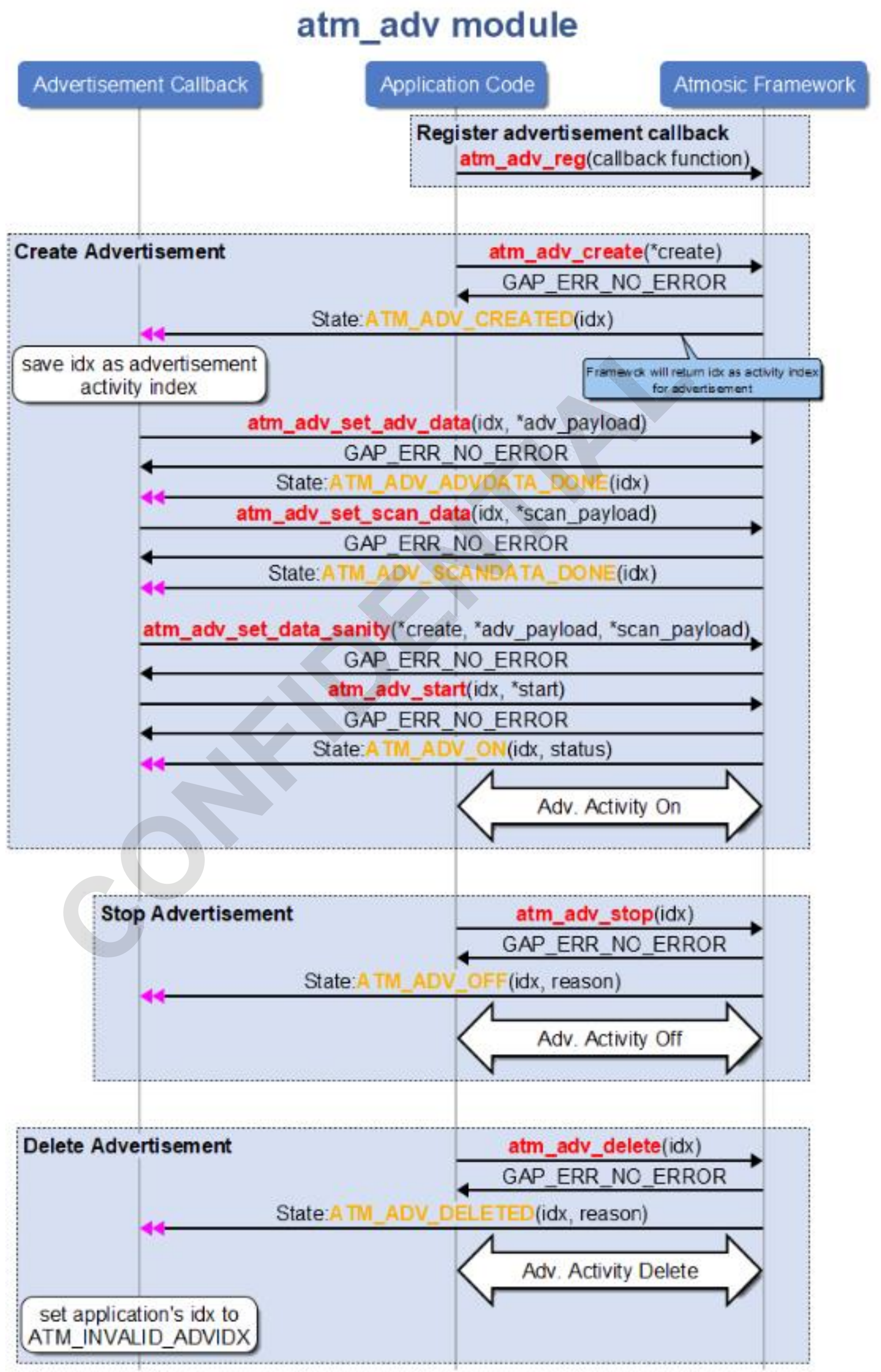


Figure 9 - atm_adv Flow

2.5 Device Discovery -Scanning

The Atmosic application framework provides scan modules to manage the Bluetooth LE scanning behaviors and provide the associated callback function to handle the scanning results.

2.5.1 Atmosic Scan Modules

The **atm_scan_param** and **atm_scan** are the modules used by the Atmosic application framework. The **atm_scan_param** module exports the API to prepare the scan parameter for the atm_scan API. The scanning parameter could be loaded from Flash NVDS or a predefined instance that can be overridden by makefile with some particular preprocessor MACROs. These parameters can also be changed and re-configured by application during runtime.

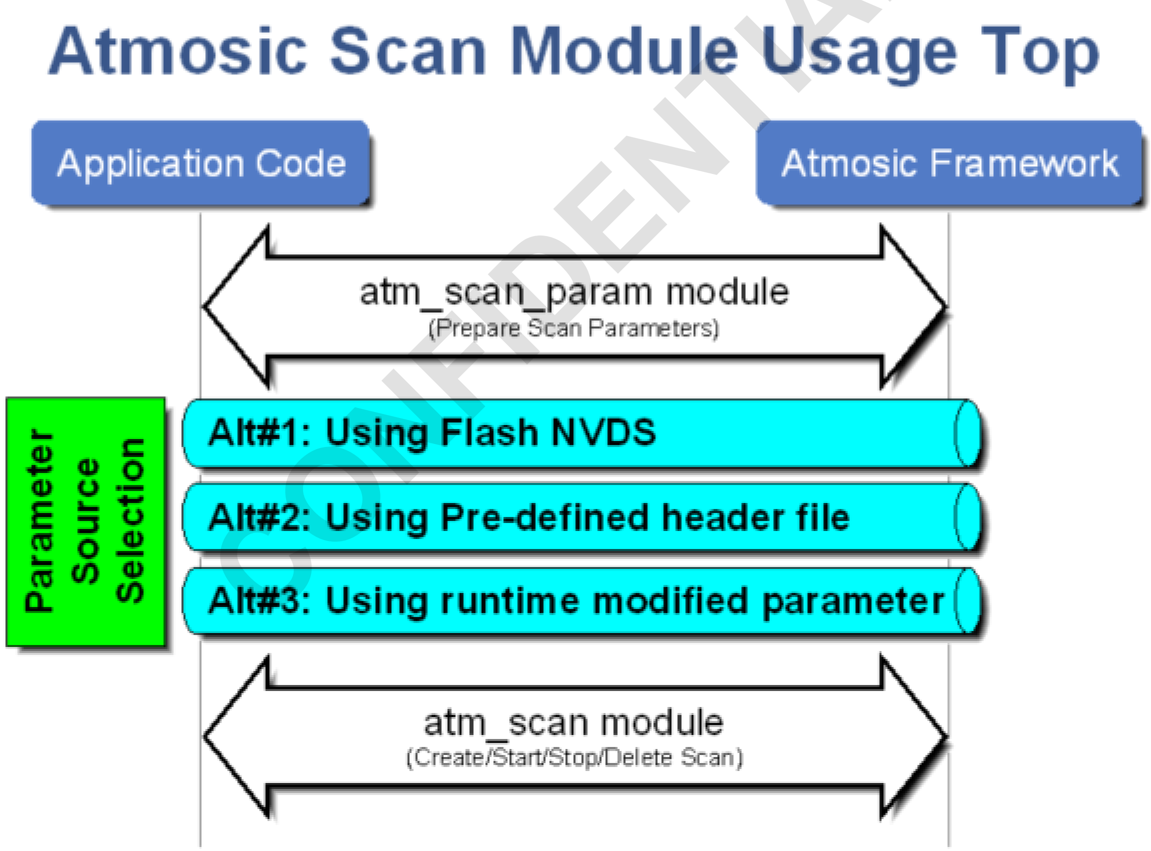


Figure 10 - Scan Module Usage Diagram

2.5.2 atm_scan_param Module

The overall behavior of the scanning activity is described by an aggregated parameter named **atm_scan_params_t**. This parameter includes both HW and SW settings such as scanning interval or the filtering policy for duplicated advertising packets. Since the SDK users would choose to use similar settings for most of their applications. The NVDS or predefined structure provides great reusability and flexibility for the SDK user to utilize.

Get parameter from Flash NVDS

Flash NVDS has defined one tag identity for the parameter used by scan activity. The application code can use the makefile and toolchain to build the flash NVDS data then burn into an Atmosic chip. The application code can use **atm_get_nvds_scan_params** to retrieve the setting before using the atm_scan module to create the scan activity.

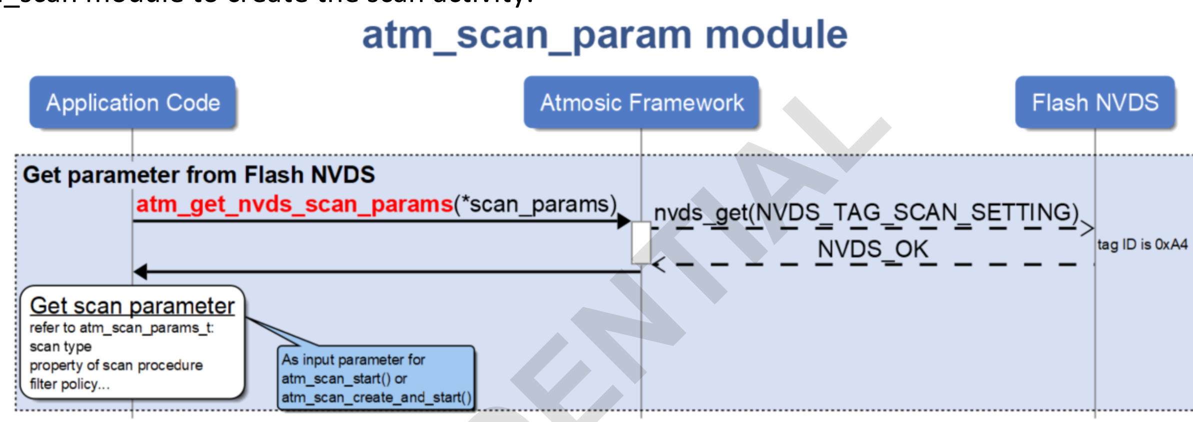


Figure 11 - atm_scan_param Module Diagram

Get parameter from predefined structure

The Atmosic framework provides **CFG_GAP_SCAN_MAX_INST** instances of a predefined scan parameter in the **atm_scan_param** module by default. Users can modify the number of instances in makefile and predefine the initial value of these instances for different scenarios as needed. The predefined scan parameter instance in the ***atm_scan_param*** module is initialized by several overwritable preprocessor macros. The SDK users may change these values by defining desired values in each application's makefile. The overwritable parameters are listed in atm_scan_param_internal.h. In addition to defining new values of those overwritable fields, the SDK users can also specify another header file in makefile (-DGAP_SCAN_PARM_NAME="xxx.h") to overwrite the entire setting for the parameter.

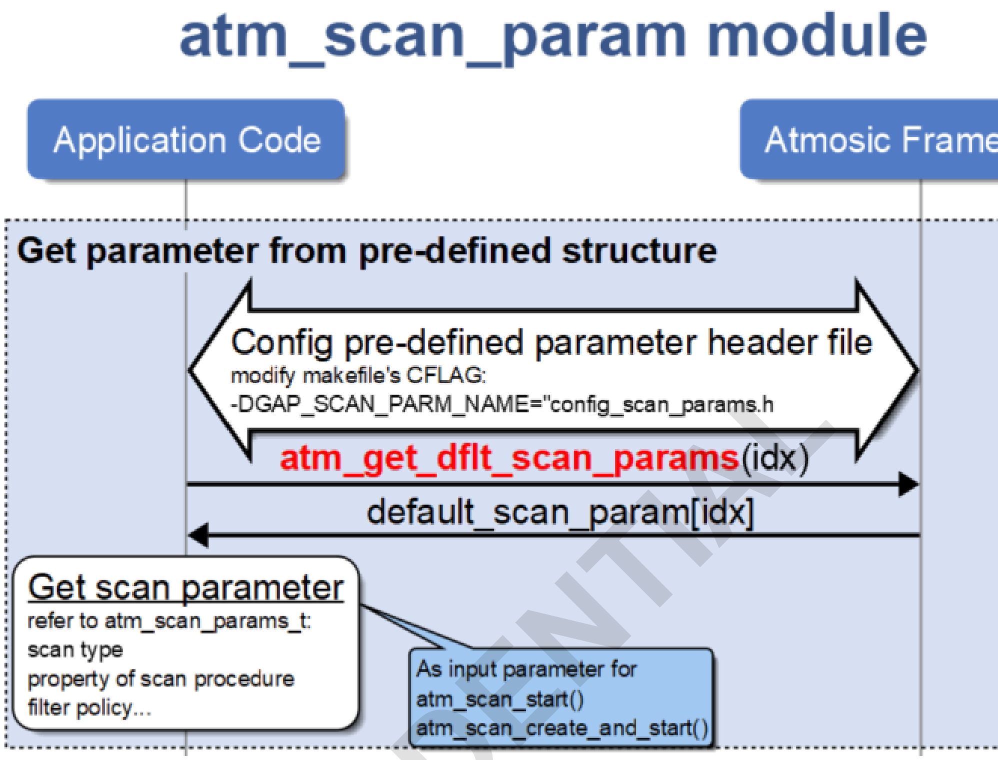


Figure 12 - atm_scan_param Module Pre-defined Structure

Using runtime modified parameter

In most scenarios the predefined parameters are loaded and directly sent into another API to start scanning activities without modifications. Thus the default data type is constant for the predefined scanning parameters. If the application code would need to change partial values of the parameter during runtime, the constant data type should be removed by adding "-DCFG_SCAN_PARAM_CONST=0" in the makefile of that application.

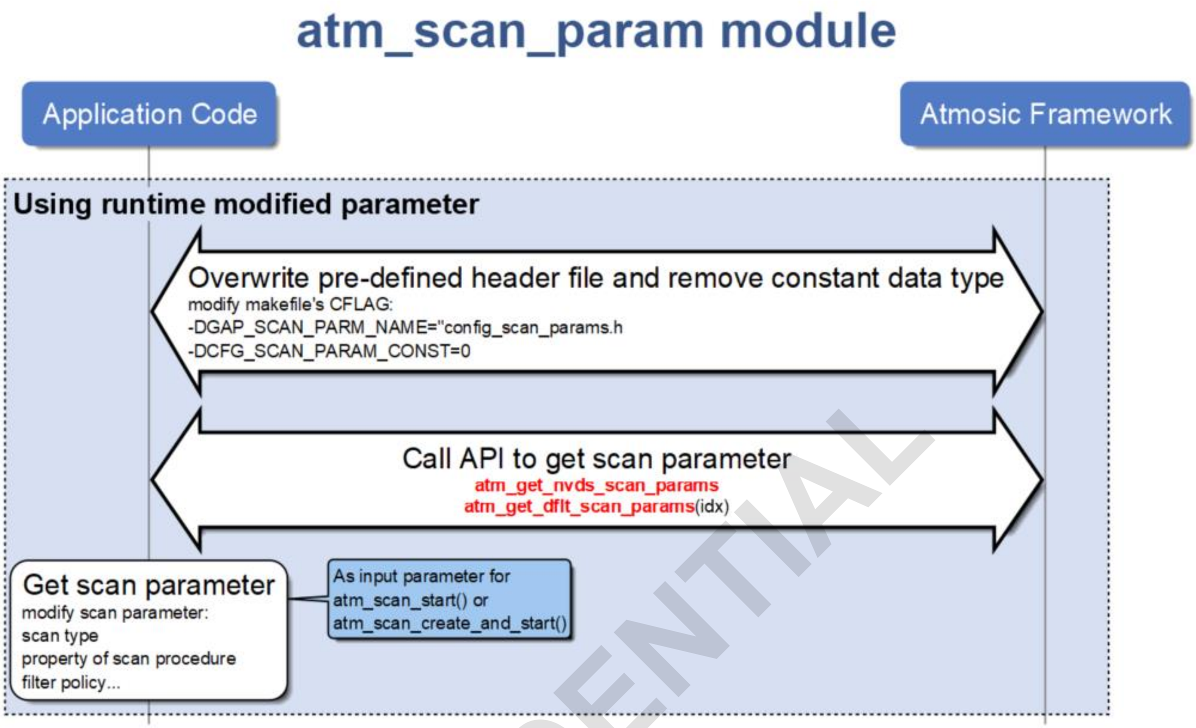


Figure 13 - atm_scan_param Module Runtime Modified Parameter

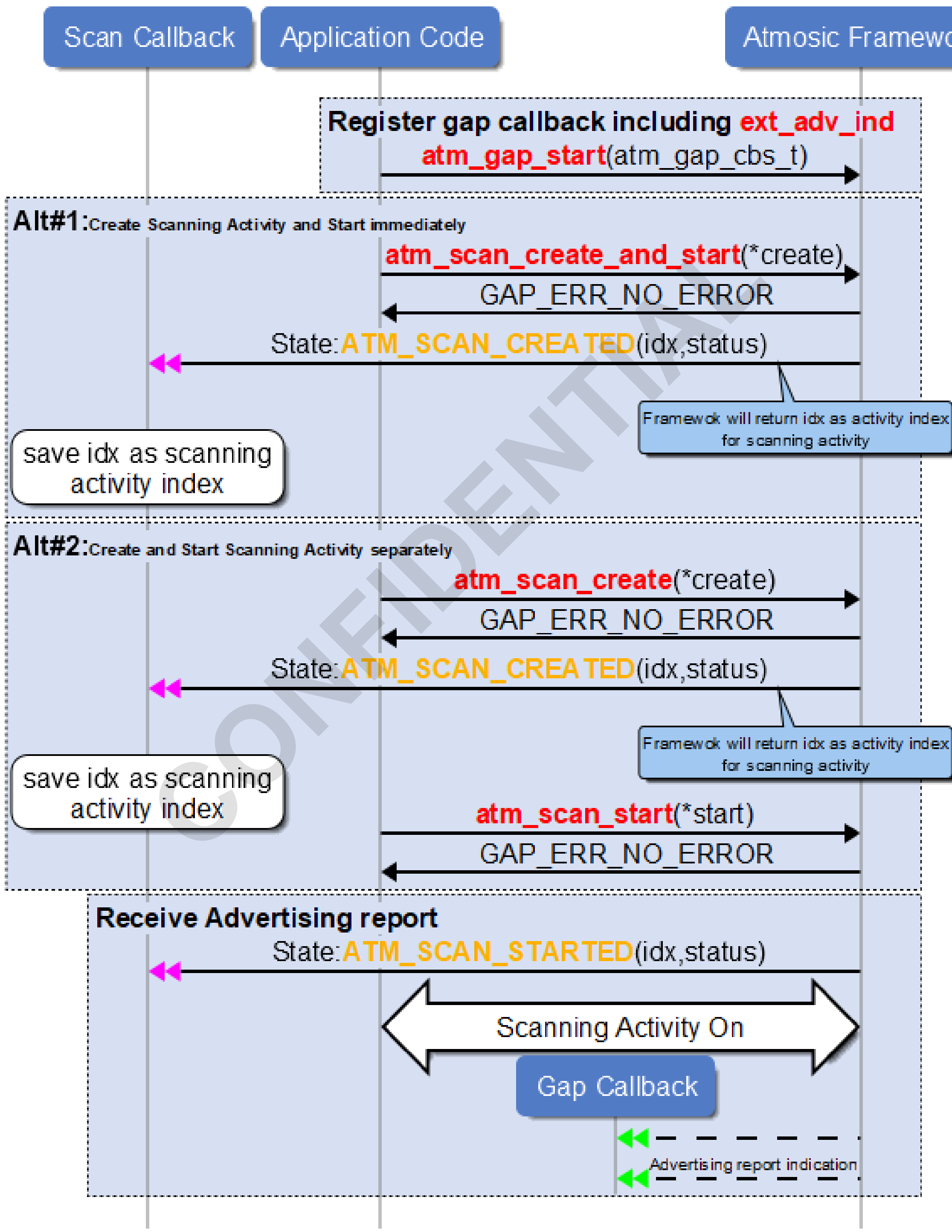
2.5.3 atm_scan Module

After preparing the scanning parameter with the methods introduced previously, the application can now utilize the **atm_scan** module to perform the scanning activity. The application needs to provide an extra callback function to handle the scanned advertising report in the function sets provided to the **atm_gap** module. Then the application code uses either **atm_scan_create_and_start** API to start scanning activity immediately or **atm_scan_create** to create scanning instance first then calls **atm_scan_start** to start scanning activity later. These two alternatives both expect the application code to provide a callback function set to handle the index of the scanning activity as well as the status change. Once the scanning activity is started successfully, the framework will start to send advertising reports to the gap callback function if there are BLE devices nearby.

If the duration of the scanning parameter is not zero, the scanning activity would stop automatically as specified. Otherwise the scanning activity would last permanently until the application code calls the **atm_scan_stop** API. The activity index is still valid for the **atm_scan** module even after the scanning activity is stopped. The application code could restart scanning by calling **atm_scan_start** again with the same activity index provided.

Once the scanning activity is not needed anymore, the application code should call **atm_scan_delete** API to destroy the scanning activity instance and release related resources in the Atmosic framework. The activity index will also become invalid thus the application code should clear the index cached locally.

atm_scan module



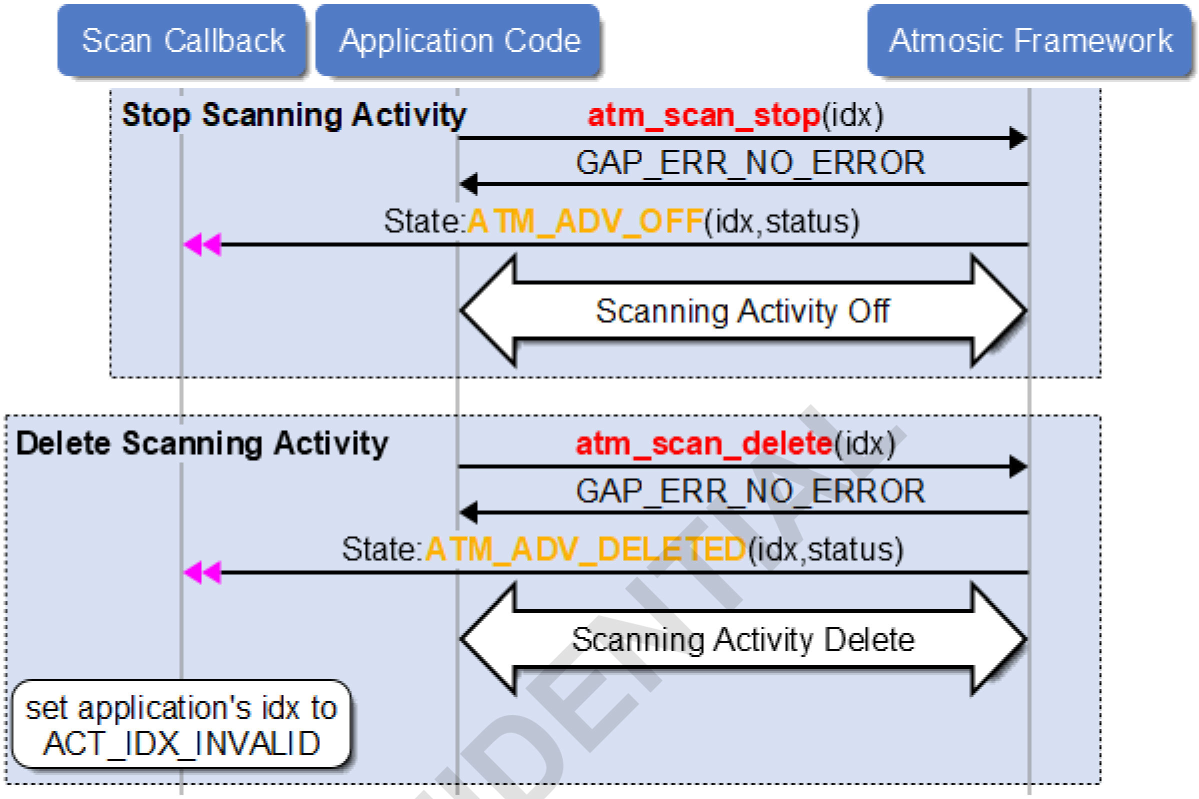


Figure 14 - atm_scan Module Flow

2.6 Connection Establishment

The Atmosic application framework provides an initiator module to manage the Bluetooth LE initiating behaviors and provide the associated callback functions to handle the related events.For the controller, only one initiating procedure can be run. In the other word, it can not be allowed to establish two connections at the same time. In addition, it is needed to support the central role. Please check the role of gap with GAP_ROLE_CENTRAL.

2.6.1 Atmosic Initiator Modules

The **atm_init_param** and **atm_init** are the modules used by Atmosic application framework. The **atm_init_param** module exports the API to prepare the initiator parameter for the atm_init API. The initiator parameter can come from Flash NVDS or predefined structure that can be overridden by using the define preprocessor. The parameters of the initiator are able to be changed and re-configured on the fly in application.

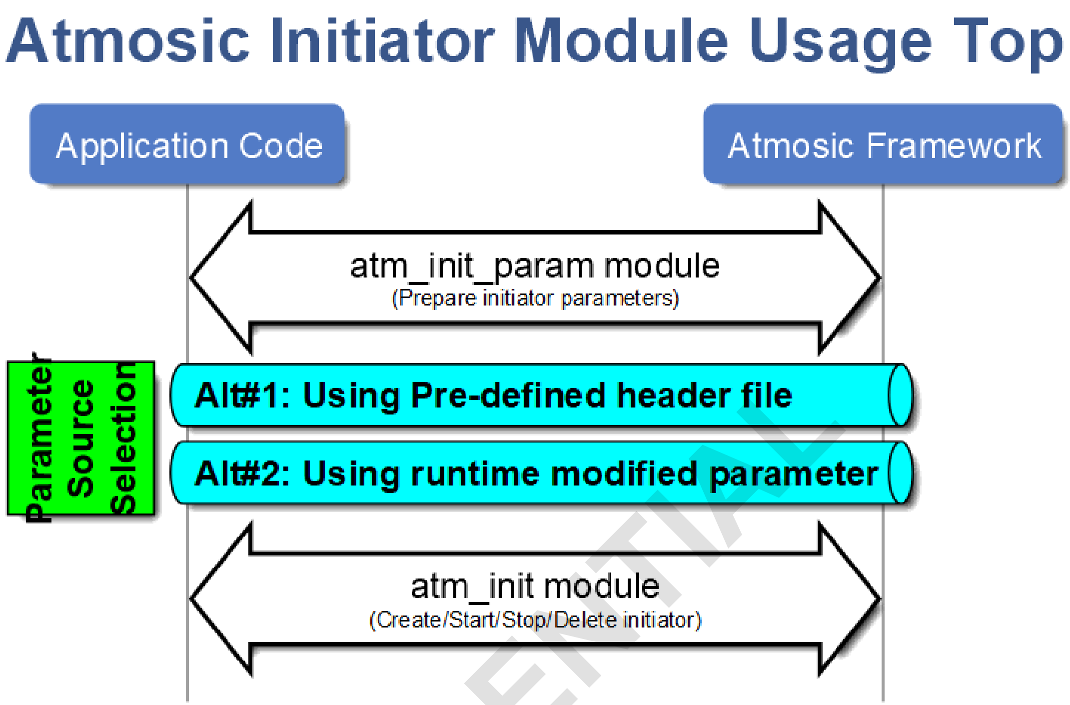


Figure 15 - Initiator Modules

2.6.2 atm_init_param Module

There is one kind of parameter to start the initiator. The **atm_init_param** module exports the APIs to retrieve the parameters from the predefined structure. See Table 8.

API Name	Parameter Purpose	Description	Parameters Location
atm_init_param_get	Start parameter	Configure: type properties connection timeout connection interval slave latency...etc	default_init_param of atm_init_param.c

Table 8 - Initiator Parameters API Description

Get parameter from predefined structure

The predefined initiator parameter instance is in ***atm_init_param*** module and also provides many overwrite fields to let the application code change the default value of the predefined parameter instance. The parameters that can be overwritten are listed into `atm_init_param_internal.h`. The application can apply the specific header file (-DGAP_INIT_PARM_NAME=”xxx.h” in makefile) to overwrite the parameter setting.

Using runtime modified parameter

The default data type is not constant in the predefined initiator parameter instance (CFG_INIT_PARAM_CONST is set to zero), so the application code can change the parameter in runtime.

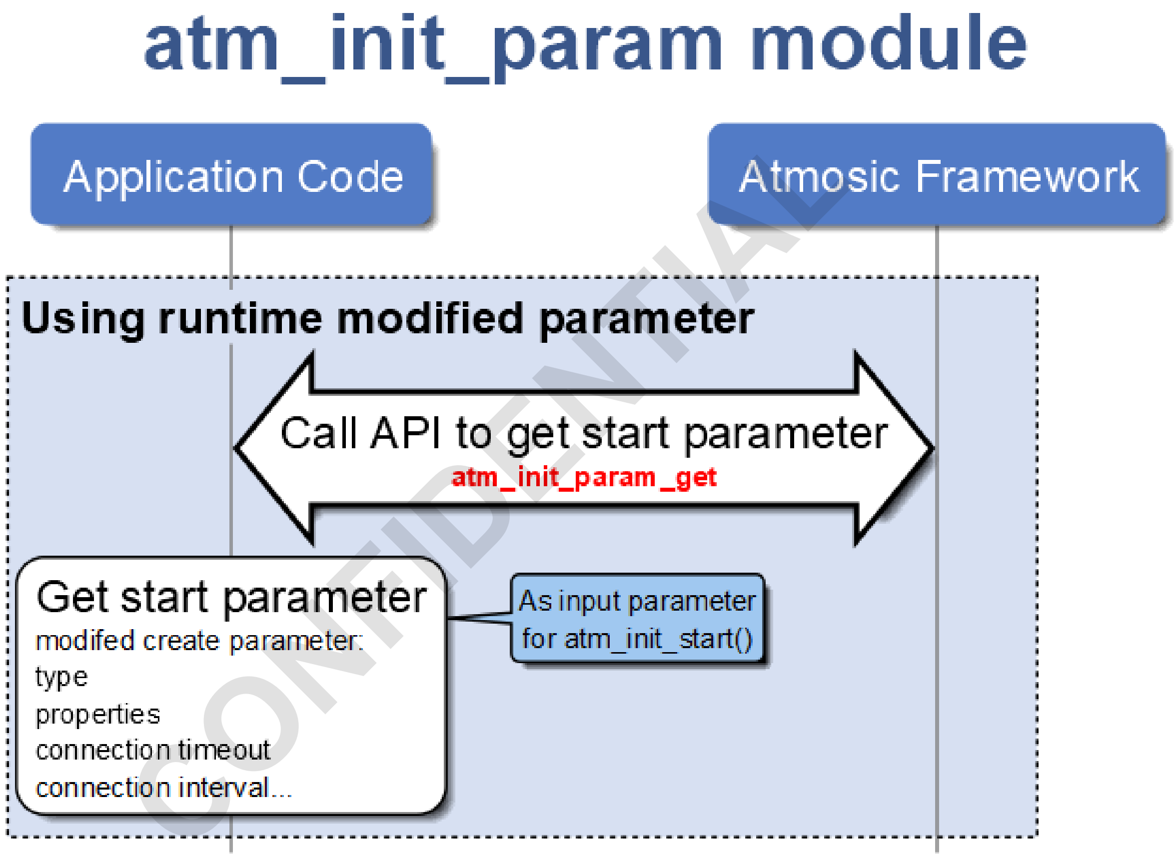


Figure 16 - Initiator Parameter Module

2.6.3 atm_init Module

After preparing the initiator parameter datas via **atm_init_param** module. The application will need to use **atm_init** module to control the initiator. For initiator, please set CFG_GAP_SCAN_MAX_INST value .The application needs to provide the callback function that works with **atm_init** module. The atm_init_reg is the first function call that provides the callback function. The **atm_init** module will provide the state, activity index and status information for each **atm_init** exported API.

The application code uses **atm_init_create** API to create initiator activity. The callback function will receive the ATM_INIT_CREATED state that includes the activity index. After getting the activity index value, the application code can use it as the input parameter for **atm_init** exported API to control the initiator.

For establishing connections, the application layer can call the **atm_init_start** API and input the related parameter from **atm_init_param** modified the related parameters, ex: peer address for connecting device. The most important is that only one initiator can be executed. Don't execute two initiators at the same time. If receiving ATM_INIT_STARTING_FAIL state, the initiator would not start. If connected, the state would change to ATM_INIT_OFF. The activity index is still valid for **atm_init** module until invoking the **atm_init_delete** API.

The **atm_init_stop** API will stop the initiator and the activity index is still valid for **atm_init** module. The application code will use **atm_init_start** with the activity index to establish a new connection again and doesn't need to call **atm_init_create** again.

The **atm_init** module will destroy the activity instance when calling **atm_init_delete** API. After this, the activity index will become invalid index for the atm_init module.

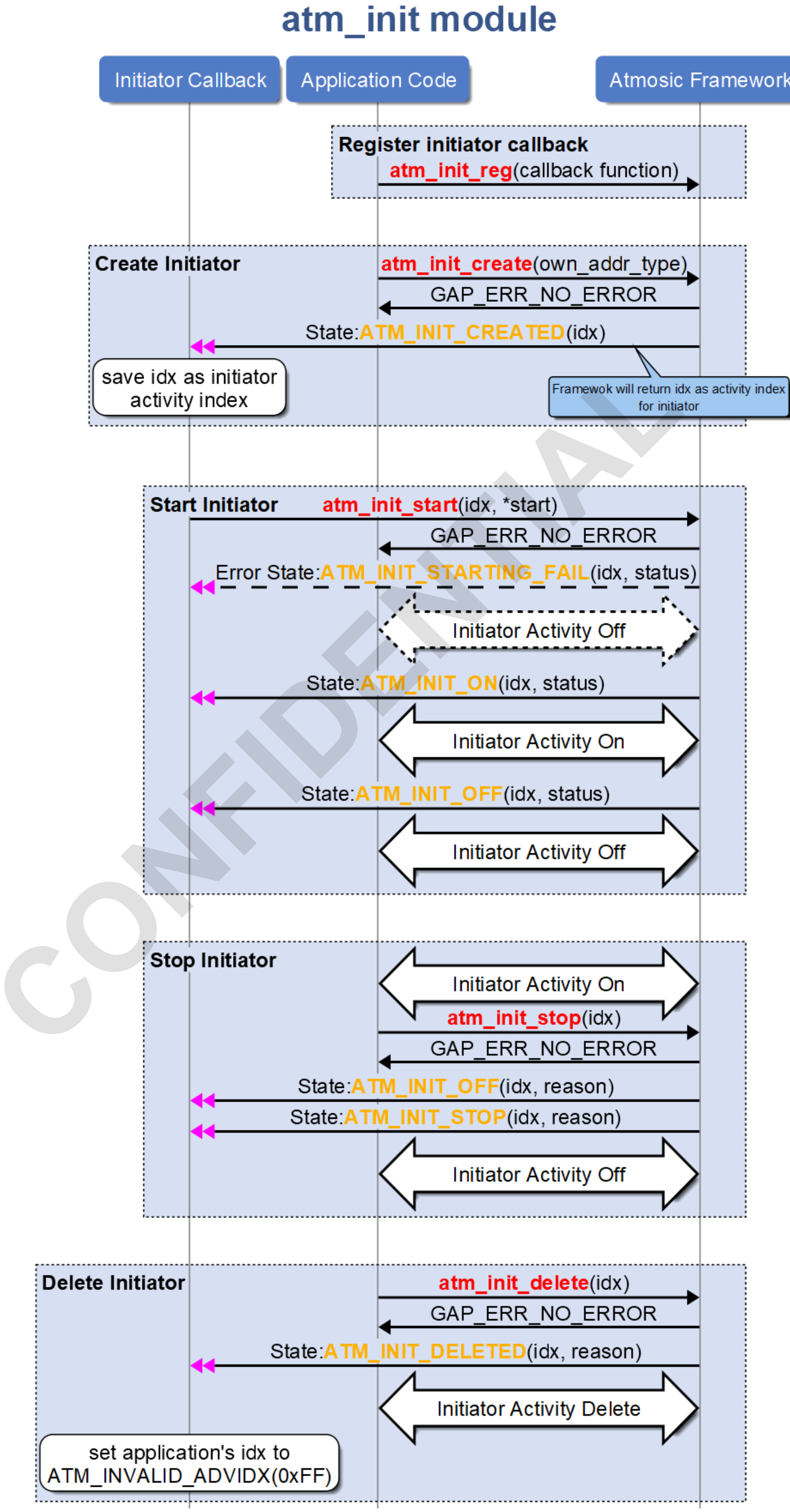


Figure 17 - atm_init Flow

2.7 Connection Mechanism

In connection mechanism process, the applications could use **atm_gap** API functions and callbacks to handle the GAP messages from CEVA BLE stack. When a connection request indication event is received in **atm_gap** and the **conn_ind** callback is invoked in the application, the callback handler may call **atm_gap_connect_accept** to accept this connection request and keep handling the further indications with **atm_gap** API functions and callbacks.

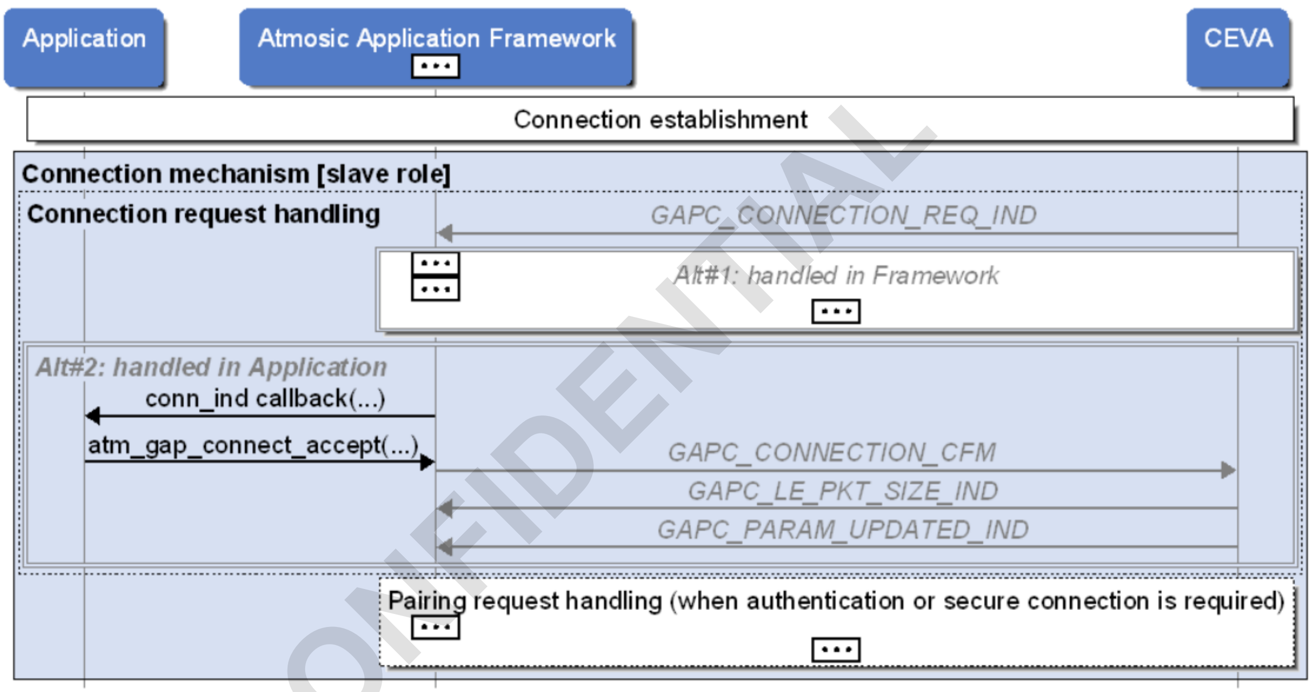


Figure 18 - Connection Mechanism Sequence Overview

2.7.1 Detailed LE Connection Flow

Initialization sequence of the Bluetooth LE framework is shown in Figure 19. Atmosic Application Framework Connection Mechanism Sequence is shown in Figure 20.

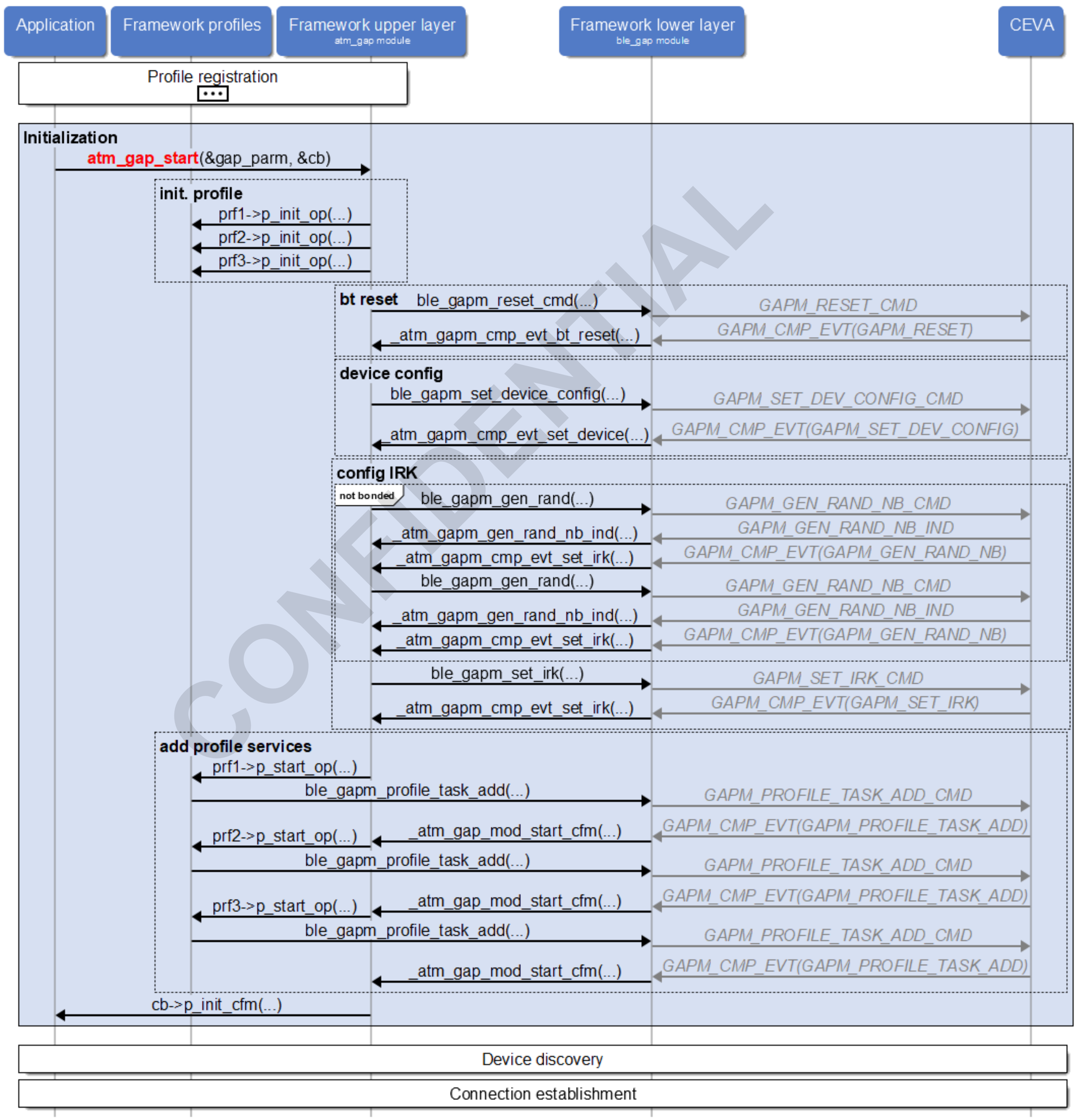


Figure 19 - Atmosic Application Framework Initialization Sequence

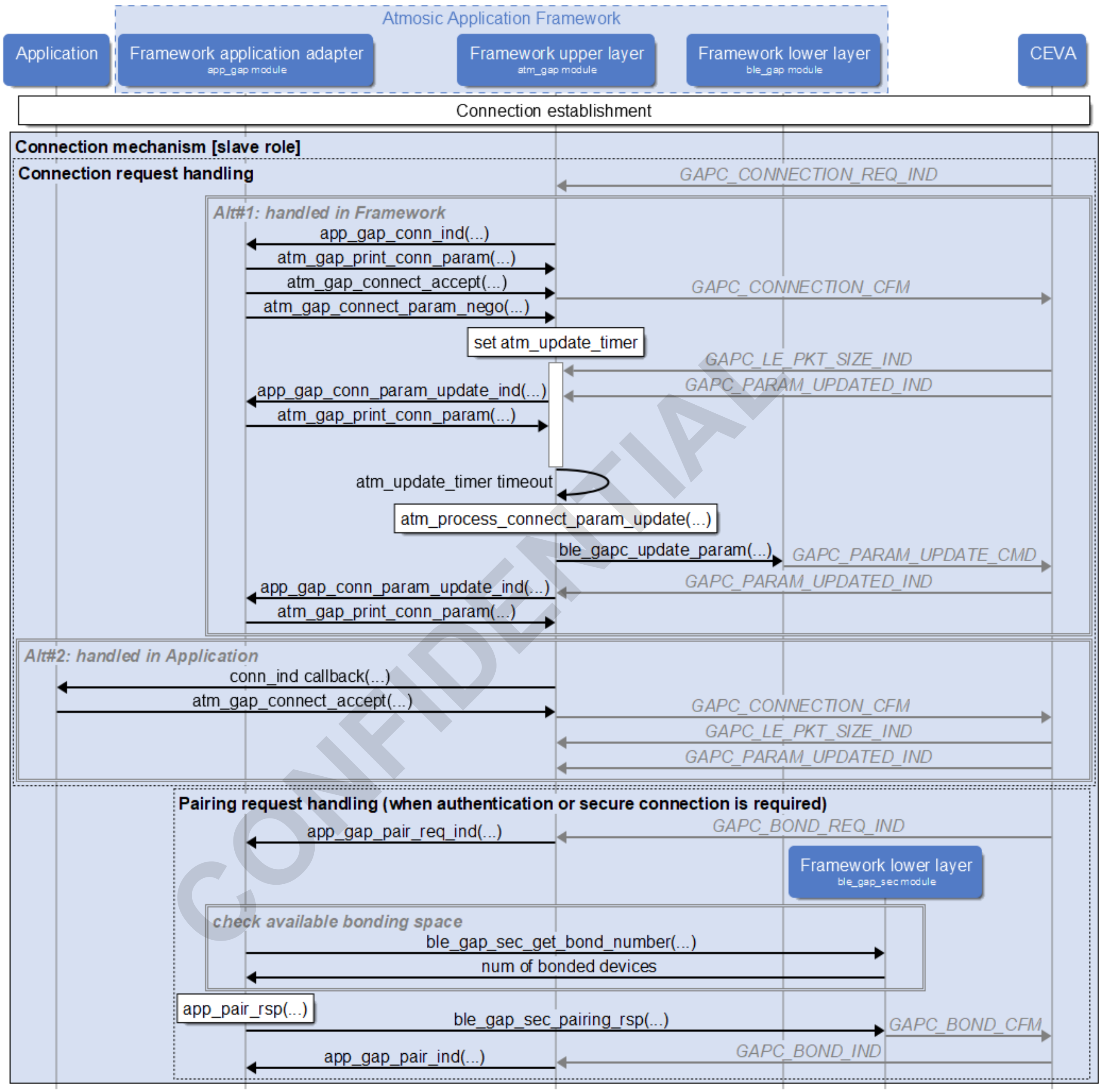


Figure 20 - Atmosic Application Framework Connection Mechanism Sequence

3 System Utilities

3.1 Application State Machine

Please see Application State Machine Application Note.

3.2 Log Utility

Application framework provides a log utility atm_log to manage and configure debug messages by each module (c source file).

3.2.1 ATM_LOG Macro

To use the atm_log utility, please include “atm_log.h” and call ATM_LOG_LOCAL_SETTING macro to configure the debug level and module name in each source file and then use ATM_LOG macro to print the different level debug messages.

- **ATM_LOG_LOCAL_SETTING**(*name*, *level*)
- **ATM_LOG**(*level*, *dbg_msg*, ...)

ATM_LOG debug message format:

@{*timestamp*} [{*name*}][{*level*}]: {*dbg_msg*}

3.2.2 Debug Log Level

Table 9 lists all debug log level masks which are used in the ATM_LOG macros.
Users could use -DATMLOG_GLOBAL_LEVEL={*debug_level_mask*} to configure the debug level of the whole application.

Debug Level Mask	Log Type	Description
V	Verbose log	When this debug level mask is set in the local setting, all log types will be output in the module.
D	Debug log	When this debug level mask is set in the local setting, the Debug, Notify, Warning and Error log types will be output in the module.
N	Notify log	When this debug level mask is set in the local setting, the Notify, Warning and Error log types will be output in the module.
W	Warning log	When this debug level mask is set in the local setting, the Warning and Error log types will be output in the module.

E	Error log	When this debug level mask is set in the local setting, only the Error log type will be output in the module.

Table 9 - Debug Log Level

Note: The length of the name display is up to 10. The oversized characters will be truncated.

3.2.3 How to Use

The following is an ATM_LOG macros use example:

C source file

```
#include "atm_log.h"

ATM_LOG_LOCAL_SETTING("modulename", D);

static void module_test(void)
{
    ATM_LOG(D, "enter %s", __func__);
    ATM_LOG(W, "exit %s", __func__);
}
```


The debug messages would be printed as below in the serial terminal software which supports ANSI escape code.

```
@0016f0e0 [modulename][D]: enter module_test
@0016f186 [modulename][W]: exit module_test
```

For the users who just need a simple debug function with timestamp only and ignore the debug level setting, please use the marco below:

DEBUG_TRACE(*dbg_msg*, ...)

Debug trace message format:

@{*timestamp*} {*dbg_msg*}

3.3 Power Management

Application framework provides a power manager module atm_pm to help users to manage the power mode of the device.

3.3.1 Power Saving Modes

ATM2/3 devices support 3 types of power saving mode as shown in Table 10:

Power Saving Mode	Description
Sleep Mode	All variables and registers would be kept. The power consumption is the highest in 3 power saving modes.
Retention Mode	Drop to retention voltage, retain the required system blocks only. The power consumption is in the middle of 3 power saving modes.
Hibernation Mode	Only specific registers would be kept. The power consumption is the lowest and close to SoC OFF mode. It looks like a reboot when the system wakes up from hibernation mode.

Table 10 - Power Saving Modes

3.3.2 Power Mode Lock

ATM2/3 devices will always try to enter the lowest power consumption mode when the system is idle. However, there are some activities that might still be on-going or wait for the response. To prevent the system unexpectedly going to power saving mode during the activities, atm_pm uses the locks for different power saving modes to control the system behavior. Figure 21 illustrates the power mode control with the atm_pm locks.

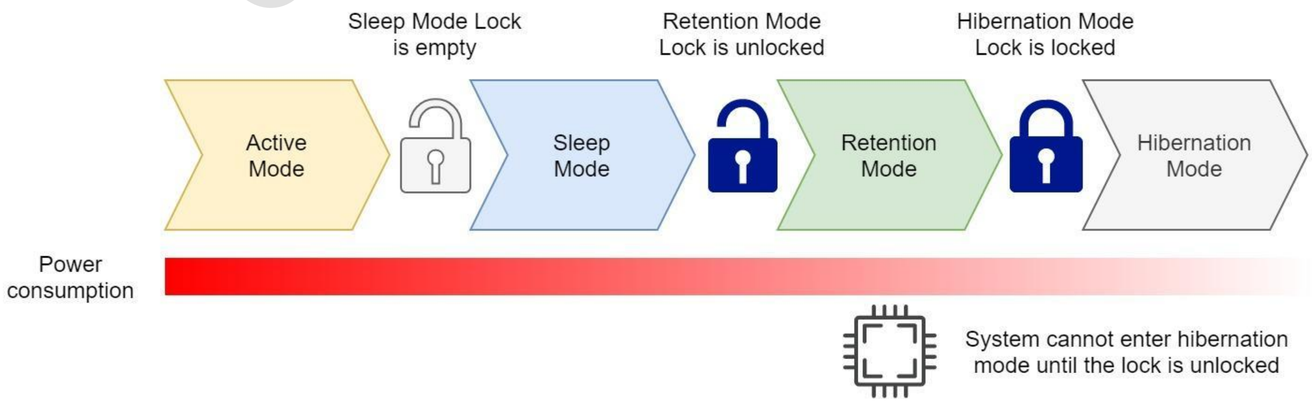


Figure 21 - Power Mode Lock

3.3.3 Power Manager API

Table 11 lists all power manager APIs.

Power Manager API	Description
pm_lock_id_t atm_pm_alloc(pm_lock_type_e type)	Allocate lock identifier with lock type
bool atm_pm_realloc(pm_lock_id_t id, pm_lock_type_e type)	Allocate lock identifier with specified id and type
void atm_pm_free(pm_lock_id_t id)	Free allocated lock
void atm_pm_lock(pm_lock_id_t index)	Lock specific power mode
void atm_pm_unlock(pm_lock_id_t index)	Unlock specific power mode
void atm_pm_lock_info(void)	Print all locks status
void atm_pm_set_hib_restart_time(uint32_t restart_time)	Set restart time from hibernate
void atm_pm_set_hibernate_cb (rep_vec_fn__ret_bool__int32_t__uint32_t__t cb)	Set function of atm_pm replacement vector of hibernate

Table 11 - Power Manager API

3.3.4 atm_pm lock example

The following is a simple atm_pm lock example:

makefile

```
DRIVERS := interrupt timer atm_pm sw_timer sw_event
```

C source file

```
#include "atm_pm.h"

static pm_lock_id_t lock_hiber;

static void user_init(void)
{
    ...
    lock_hiber = atm_pm_alloc(PM_LOCK_HIBERNATE);
    atm_pm_lock(lock_hiber);
    ATM_LOG(D, "Hibernation is locked.");
    // Prevent the system entering hibernation.
    ...
```

```
}

static void activity_stop(void)
{
    atm_pm_unlock(lock_hiber);
    ATM_LOG(D, "Hibernation is unlocked.");
    // System goes into hibernation.
}
```

Please refer to pm_demo example in the SDK for more atm_pm use examples.

3.4 Software Timer

Application framework provides software timer functionality through sw_timer module. Users can use sw_timer API to alloc, set and clear timers for their application.

3.4.1 Software Timer API

Table 12 lists all sw_timer APIs.

Software Timer API	Description
sw_timer_id_t sw_timer_alloc (sw_timer_func_t handler, const void *ctx)	Allocate and configure timer
void sw_timer_free(sw_timer_id_t timer_id)	Free allocated timer
void sw_timer_reconfig (sw_timer_id_t timer_id, sw_timer_func_t handler, const void *ctx)	Reconfigure timer handler and context
void sw_timer_set(sw_timer_id_t timer_id, uint32_t centisec)	Start one-shot timer running
void sw_timer_clear(sw_timer_id_t timer_id)	Stop/abort running timer
bool sw_timer_active(sw_timer_id_t timer_id)	Get timer status

Table 12 - Software Timer APIs

The time unit of sw_timer_set is cent-second (10 ms). A "SW_TIMER_ID_MAX too big for sw_timer_id_t" assertion might occur when the users try to allocate more than 8 software timers. By default, the number of software timers is limited to 8 by SW_TIMER_ID_MAX. Users could change the value in makefile with -DSW_TIMER_ID_MAX=*N* to enlarge the maximum number of software timers based on their application needs.

3.4.2 sw_timer example

The following is a simple sw_timer example:

makefile

```
DRIVERS := interrupt timer atm_pm sw_timer sw_event
```

C source file

```
#include "sw_timer.h"

static sw_timer_id_t tid_test;
static bool repeat;

static void timeout_handler(sw_timer_id_t tid, void const *ctx)
{
    ATM_LOG(D, "Timer expired.");
    if (repeat) {
        sw_timer_set(tid_test, 5 * SW_TIMER_1_SEC);
    }
}

static void user_init(void)
{
    ...
    tid_test = sw_timer_alloc(timeout_handler, NULL);
    sw_timer_set(tid_test, 5 * SW_TIMER_1_SEC);
    ATM_LOG(D, "Timer started.");
    ...
}
```

3.5 Software Event

Application framework provides software event functionality through sw_event module. Users could use sw_event APIs to allocate, set and clear software events for their application.

3.5.1 Software Event API

Table 13 lists all sw_event APIs.

Software Event API	Description
sw_event_id_t sw_event_alloc (sw_event_func_t handler, const void *ctx)	Allocate and configure event
void sw_event_free(sw_event_id_t event_id)	Free allocated event
void sw_event_reconfig (sw_event_id_t event_id, sw_event_func_t handler, const void *ctx)	Reconfigure event handler and context
void sw_event_set(sw_event_id_t event_id)	Trigger event
void sw_event_clear(sw_event_id_t event_id)	Clear event
bool sw_event_get(sw_event_id_t event_id)	Get event status

Table 13 - Software Event APIs

A "SW_EVENT_ID_MAX too big for sw_event_mask" assertion might occur when the users try to allocate more than 8 software events. By default, the number of software events is limited to 8 by SW_EVENT_ID_MAX. Users could change the value in makefile with -DSW_EVENT_ID_MAX=*N* to enlarge the maximum number of software events based on their application needs.

3.5.2 sw_event example

The following is a simple sw_event example:

makefile

```
DRIVERS := interrupt timer atm_pm sw_timer sw_event
```

C source file

```
#include "sw_event.h"

static sw_event_id_t eid_test;

static void event_handler(sw_event_id_t eid, void const *ctx)
{
    bool done = false;
    ATM_LOG(D, "Handling the event...");
```

```
    ...
    if (done) {
        sw_event_clear(eid);
        ATM_LOG(D, "Done. Clear the event.");
    }
    ...
}

static void user_init(void)
{
    ...
    eid_test = sw_event_alloc(event_handler, NULL);
    ...
}

static void trigger_event(void)
{
    sw_event_set(eid_test);
    ATM_LOG(D, "Event triggered.");
}
```

3.6 AT Command

Please see the Atmosic AT Command Application Note (available on the Atmosic support website).

4 Hardware Drivers

4.1 GPIO Driver

The Atmosic application framework provides atm_gpio modules to provide the associated API function to control the pins for GPIOs application.

4.1.1 atm_gpio Module

To enable the atm_gpio module, please add the module to DRIVERS in makefile.

```
DRIVERS := \
      atm_ble \
      atm_button  \
      atm_gpio \
      atm_pm \
```

As initializing the **atm_gpio** module, the **atm_gpio_init_constructor** would be executed and valid io would be marked according to the chip type to avoid setting invalid io.

For GPIO applications, the applications could use **atm_gpio** API functions to control or get the output and input status. For input pin application, it also can support the interrupt handling. The related setting flow and APIs are listed as below:

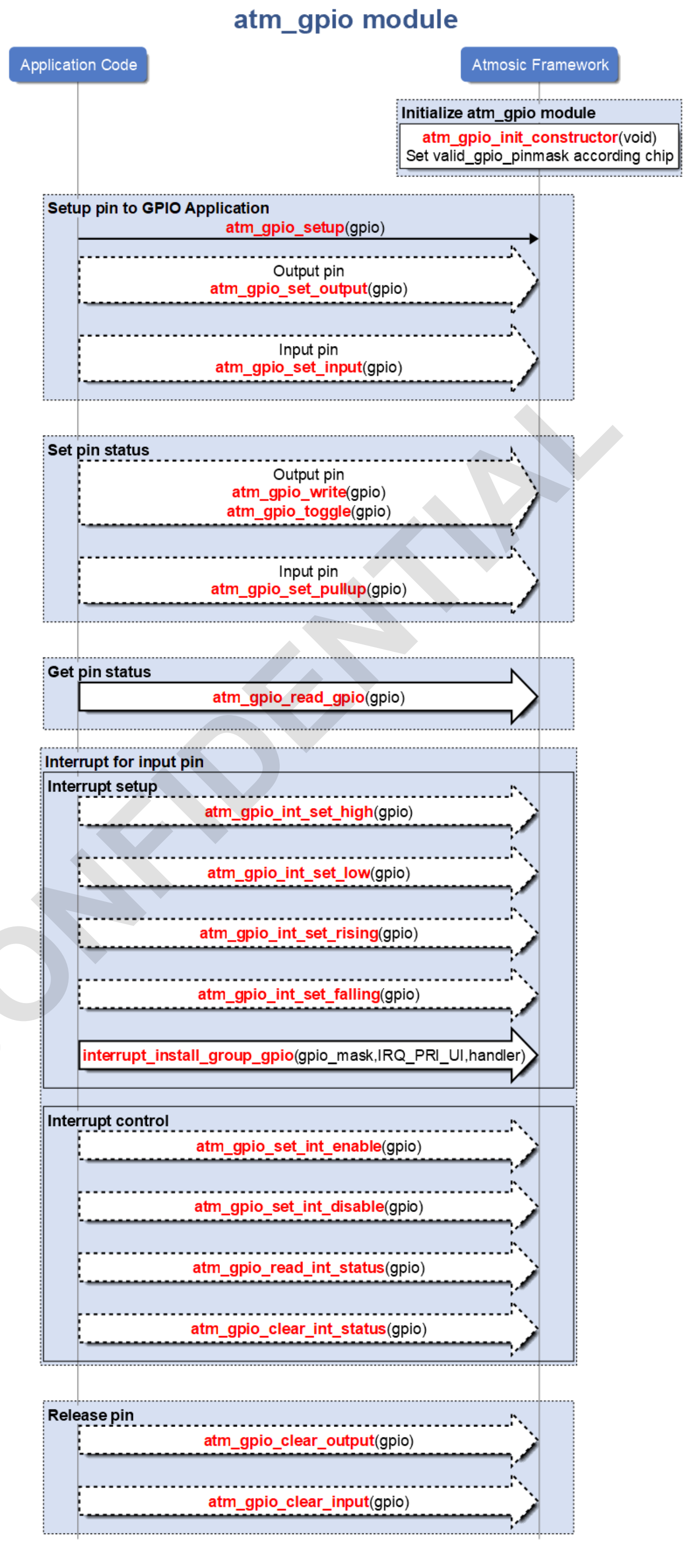


Figure 22 - atm_gpio Module Flow

The **atm_gpio** module exports the APIs to set pins as GPIOs and control/retrieve the status of GPIOs. See Table 14.

API Name	Purpose	Parameter Description
void atm_gpio_setup(uint8_t gpio)	Set pin as GPIO	gpio: The number of GPIO not pin. EX: For P20, the related GPIO is 16. atm_gpio_setup(16); The related mapping please refer to section 2.7 Pin Multiplexing in ATM3_ATM2 Reference Manual.
void atm_gpio_set_input(uint8_t gpio)/ void atm_gpio_clear_input(uint8_t gpio)	Set/Clear gpio as input io	gpio: The number of GPIO
void atm_gpio_set_output(uint8_t gpio)/ void atm_gpio_clear_output(uint8_t gpio)	Set/Clear gpio as output io	gpio: The number of GPIO
void atm_gpio_set_pullup(uint8_t gpio)/ void atm_gpio_clear_pullup(uint8_t gpio)	Set/Clear gpio with pullup	gpio: The number of GPIO
void atm_gpio_write(uint8_t gpio, bool value)	Set output high or low for output io	gpio: The number of GPIO value: 0-low 1:high
bool atm_gpio_read_gpio(uint8_t gpio)	Read the status for GPIO	gpio: The number of GPIO return: the status of the GPIO
bool atm_gpio_toggle(uint8_t gpio)	Toggle the output	gpio: The number of GPIO return: the status of the GPIO after toggle

Table 14 - Atmosic GPIO API Description

4.2 I2C Driver

There are two identical I2C master ports, and it supports clock frequencies between 3.9kHz~4MHz. The Atmosic application framework provides **i2c** modules with the associated APIs for i2c communication.

4.2.1 i2c Module

To enable the atm_gpio module, please add the module to DRIVERS in makefile.

```
DRIVERS := \
      atm_ble \
      atm_button   \
```

```
    i2c \
```

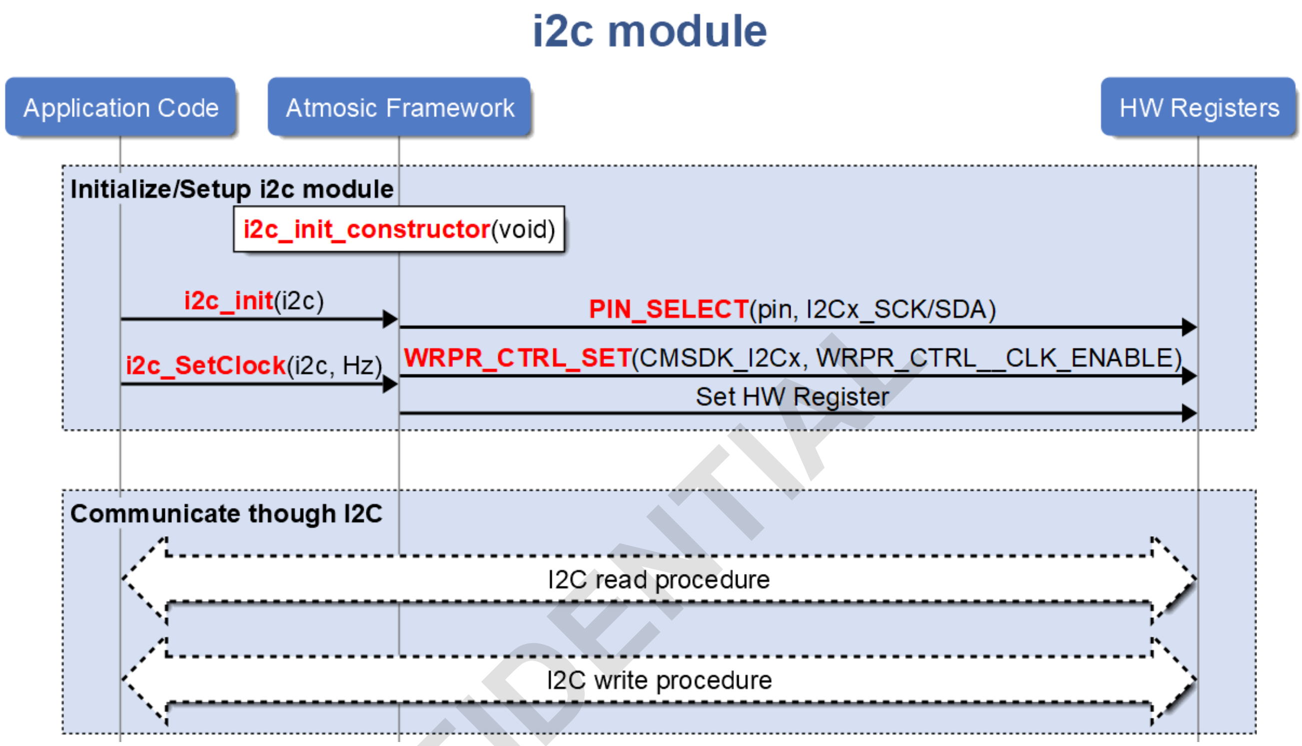


Figure 23 - i2c Module Flow

When initializing the **i2c** module, the **i2c_init_constructor** would be executed to register related callback function to control the related pin as enter/exit retention or hibernation mode.

The input **i2c_dev_t** for the I2C APIs can be configured according to the I2C set and default status of the data pin. The CMSDK_AT_APB_I2C_TypeDef can be selected to CMSDK_I2C0 or CMSDK_I2C1. **enable_data_pullup** is used to set the default status of the data pin.

```
typedef struct i2c_dev_s {
    CMSDK_AT_APB_I2C_TypeDef *base;
    bool enable_data_pullup;

    int rx_head, rx_tail;
    uint8_t rx_buf[I2C_BUF_SIZE];

    int tx_head, tx_tail;
    uint8_t tx_address;
    uint8_t tx_buf[I2C_BUF_SIZE];
} i2c_dev_t;
```

i2c_init is the initialization function for I2C. It would set the I2C pin assignment according to the chip and below define. The default setting is defined in i2c.c, and it can be changed by defining the setting in makefile. For the pin multiplexing, please refer to section 2.7 of the Pin multiplexing in ATM3_ATM2 Reference Manual (available on the Atmosic support website).

```
#ifndef CFG_5x5_I2C0_SCK
#define CFG_5x5_I2C0_SCK 9
#endif

#ifndef CFG_5x5_I2C0_SDA
#define CFG_5x5_I2C0_SDA 10
#endif

#ifndef CFG_5x5_I2C1_SCK
#define CFG_5x5_I2C1_SCK 11
#endif

#ifndef CFG_5x5_I2C1_SDA
#define CFG_5x5_I2C1_SDA 13
#endif

#ifndef CFG_6x6_I2C0_SCK
#define CFG_6x6_I2C0_SCK 29
#endif

#ifndef CFG_6x6_I2C0_SDA
#define CFG_6x6_I2C0_SDA 30
#endif

#ifndef CFG_6x6_I2C1_SCK
#define CFG_6x6_I2C1_SCK 12
#endif

#ifndef CFG_6x6_I2C1_SDA
#define CFG_6x6_I2C1_SDA 13
#endif
```


For I2C clock frequency, it can be set by the **i2c_SetClock**.

```
EX: I2C 0, pullup data pin, clock frequency: 100KHz
static i2c_dev_t i2c_dev = {
    .base = CMSDK_I2C0,
    .enable_data_pullup = true
};
#define I2C_CLK_100K 100000

i2c_init(&i2c_dev);
i2c_SetClock(&i2c_dev, I2C_CLK_100K);
```

For the **i2c** module, there are APIs for I2C communication steps and it can be composed to a procedure for a master to access a slave device.

API Name	Purpose	Parameter Description
void i2c_init(i2c_dev_t *i2c)	Initialize I2C device structure	i2c: device structure
void i2c_SetClock(i2c_dev_t *i2c, uint32_t Hertz)	Configure I2C clock frequency	i2c: device structure Hertz: clock frequency
int i2c_requestFrom(i2c_dev_t *i2c, uint8_t address, int quantity, bool stop)	Initiate read transaction from I2C slave device	i2c: device structure address: i2c slave address quantity: Number of bytes to read from slave. stop: Signal stop after transaction return: Success: number of bytes requested from bus. Failure: negative i2c_et_t failure code
int i2c_available(i2c_dev_t *i2c)	Poll status of current read transaction	i2c: device structure Number of bytes available to read via i2c_read()
uint8_t i2c_read(i2c_dev_t *i2c)	Read next byte of data.	i2c: device structure return: Next byte of data
void i2c_beginTransmission(i2c_dev_t *i2c, uint8_t address)	Initiate write transaction to I2C slave.	i2c: device structure address: i2c slave address
int i2c_write_byte(i2c_dev_t *i2c, uint8_t value)	Append single byte to current write transaction	i2c: device structure value: write data return:Success: 1 Failure: 0

int i2c_write_block(i2c_dev_t *i2c, const uint8_t *data, int length)	Append bytes to current write transaction	i2c: device structure data: Block of bytes to append. int: length return: Number of bytes actually added.
i2c_et_t i2c_endTransmission(i2c_dev_t *i2c, bool stop)	Finalize current write transaction and wait for completion.	i2c: device structure stop: Signal stop after transaction return: error codes

Table 15 - Atmosic I2C API Description

In the SDK, there are some sensor drivers using I2C, ex: adt7420/bme680/lis3dh. Those drivers can be reference code for I2C applications. In the section, use lis3dh as an example for I2C read / write procedure .

The lis3dh_read API is to read data from the lis3dh register. First, write one byte data for the register address and then readback the data.

```
static int lis3dh_read(uint8_t reg_addr, uint8_t *data, int length)
{
    i2c_beginTransmission(&i2c_dev, I2C_LIS3DH_ADDR);
    i2c_write_byte(&i2c_dev, reg_addr);
    int ret = i2c_endTransmission(&i2c_dev, false);
    if (ret != I2C_ET_SUCCESS) {
      DEBUG_TRACE("I2C eT=%d", ret);
      return 0;
    }
    ret = i2c_requestFrom(&i2c_dev, I2C_LIS3DH_ADDR, length, true);
    if (ret != length) {
      DEBUG_TRACE("I2C rF=%d", ret);
      return 0;
    }

    length = i2c_available(&i2c_dev);
    for (int i = 0; i < length; i++) {
      data[i] = i2c_read(&i2c_dev);
    }

    return length;
}
```

The lis3dh_write API is to write the data for the lis3dh register.

```
static int lis3dh_write(uint8_t reg_addr, uint8_t *data, int length)
{
    i2c_beginTransmission(&i2c_dev, I2C_LIS3DH_ADDR);
    i2c_write_byte(&i2c_dev, reg_addr);
    length = i2c_write_block(&i2c_dev, data, length);
    int ret = i2c_endTransmission(&i2c_dev, true);
    if (ret != I2C_ET_SUCCESS) {
      DEBUG_TRACE("I2C eT=%d", ret);
      return 0;
    }
    return length;
}
```

4.3 SPI Driver

The ATM2/3 can support general purpose four pins SPI master port with mode 0 (CPOL=0, CPHA=0), and the SPI port clock frequency is programmable and ranges from 7.8 KHz to 8 MHz. The Atmosic application framework provides SPI modules with the associated APIs for SPI read/write communication.

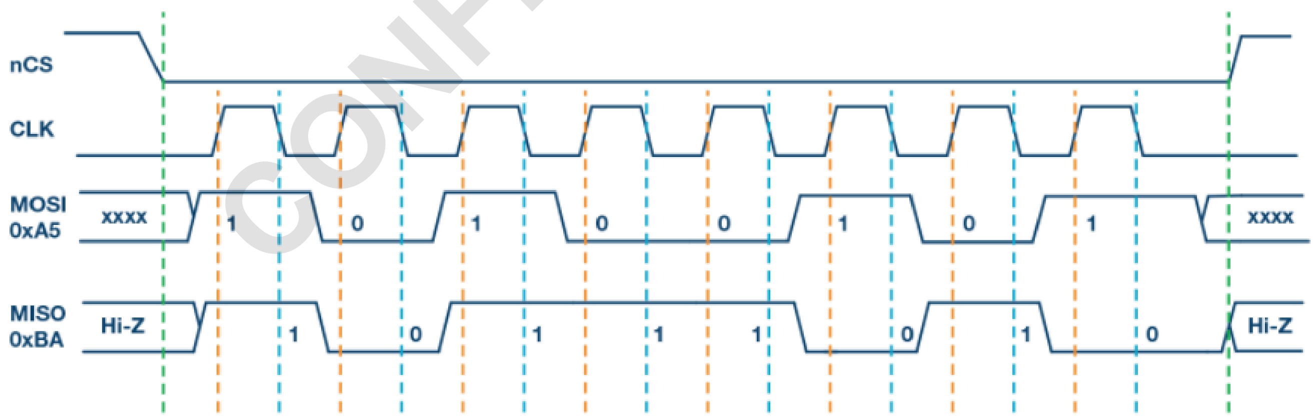


Figure 24- Timing Diagram For SPI Mode 0 - Sample At Rising Edge

4.3.1 Pin Assignment for SPI

For ATM2/3, it supports 2 set SPI and please refer to section 2.7 of the Pin multiplexing in ATM3_ATM2 Reference Manual for the related pin . There are four pins - CS, CLK, MISO, MOSI for SPI communication.

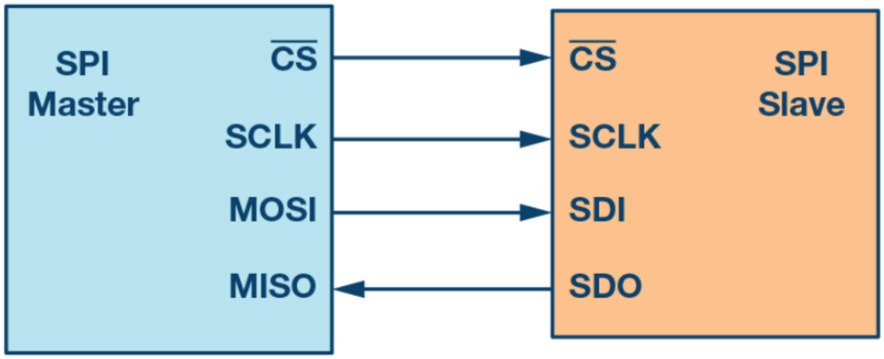


Figure 25- SPI Pin

Please do related pin assignment with **PIN_SELECT** API and enable it with **WRPR_CTRL_SET**.

```
EX: SPI0, CS: P10, CLK: P20, MOSI: P22,
PIN_SELECT(13, SPI0_MISO);
PIN_SELECT(22, SPI0_MOSI);
PIN_SELECT(10, SPI0_CS);
PIN_SELECT(20, SPI0_CLK);

WRPR_CTRL_SET(CMSDK_SPI0, WRPR_CTRL__CLK_ENABLE);
```

4.3.2 SPI Module

The SPI module is implemented in rom code. The **spi** module exports the APIs for application layer to read/write with SPI slave device.

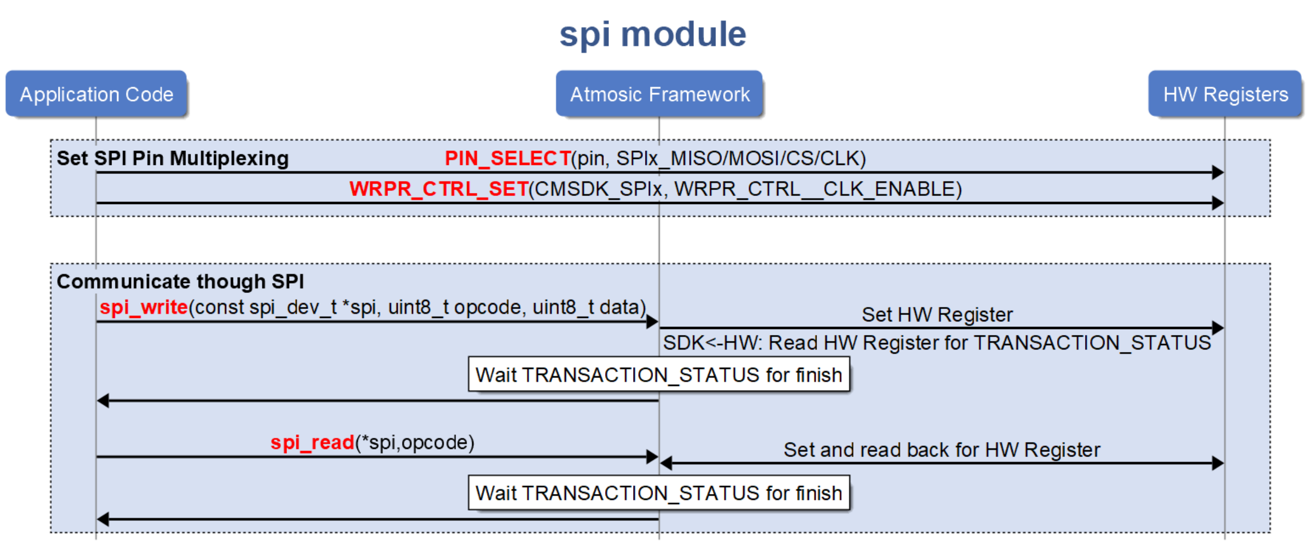


Figure 26 - SPI Module

The input **spi_dev_t** for the SPI APIs can be configured according to the SPI set and SPI clock. The **CMSDK_AT_APB_SPI_TypeDef** can be selected to CMSDK_SPI0 or CMSDK_SPI1. For SPI clock frequency, it can be set by the **clkdiv**. SPI clock would be system clock/(2*(clkdiv+1)), and the system clock is 16 MHz.

```
typedef struct spi_dev_s {
    CMSDK_AT_APB_SPI_TypeDef *base;
    uint16_t clkdiv;
    uint8_t dummy_cycles;
} spi_dev_t;

EX: SPI 0 with 4MHz clock frequency
static spi_dev_t const 4M_spi0 = { CMSDK_SPI0, 1, 0};
```

For SPI communication, it transits with 1 byte opcode. The opcode byte can not be removed. It would be the first byte data in MOSI, and the first byte data in MISO would be updated to the **OPCODE_STATUS** register.

```
#define SPI_TRANSACTION_STATUS__OPCODE_STATUS__SHIFT                    8
#define SPI_TRANSACTION_STATUS__OPCODE_STATUS__WIDTH                    8
#define SPI_TRANSACTION_STATUS__OPCODE_STATUS__MASK            0x0000ff00U
#define SPI_TRANSACTION_STATUS__OPCODE_STATUS__READ(src) \
                    (((uint32_t)(src)\
                    & 0x0000ff00U) >> 8)
```

For the SPI module, it provides read/write APIs for 1 to 4 bytes. See Table 16.

API Name	Purpose	Parameter Description
uint8_t spi_read(const spi_dev_t *spi, uint8_t opcode) uint16_t spi_read_2(const spi_dev_t *spi, uint8_t opcode) uint32_t spi_read_3(const spi_dev_t *spi, uint8_t opcode) uint32_t spi_read_4(const spi_dev_t *spi, uint8_t opcode)	Read 1~4 byte data	spi_dev_t : spi configuration opcode: the first byte data for MOSI return: the 2nd~5th byte data of MISO
void spi_write(const spi_dev_t *spi, uint8_t opcode, uint8_t data) void spi_write_2(const spi_dev_t *spi, uint8_t opcode, uint16_t data) void spi_write_3(const spi_dev_t *spi, uint8_t opcode, uint32_t data) void spi_write_4(const spi_dev_t *spi, uint8_t opcode, uint32_t data)	Write 1~4 byte data to slave	spi_dev_t : spi configuration opcode: the first byte data for MOSI data: the 2nd~5th byte data for MOSI

Table 16 - Atmosic SPI API Description

Atmosic™

Forever Connected, Anywhere.

ATMOSIC TECHNOLOGIES - DISCLAIMER

www.atmosic.com