

# The Architecture of SW

---

SEONGJI

September 10, 2019

<b>Introduction</b>	4
<b>Getting Started</b>	5
<b>The structure of the example files(tracker)</b>	6
<b>Mode</b>	7
<b>Introduction</b>	8
<b>Selecting Mode</b>	8
<b>Code review</b>	9
<b>Protocol with DBGI2C</b>	9
<b>User define Commands</b>	9
<b>Boot Mode</b>	10
<b>Introduction</b>	10
<b>Code review</b>	10
<b>Kind of boot mode</b>	10
<b>Scenario of Tracker</b>	11
<b>Introduction</b>	11
<b>Wake-up Device</b>	11
<b>Bluetooth</b>	12
<b>Determining location</b>	12
<b>Measurement frequency</b>	12
<b>Architecture</b>	13
<b>Introduction</b>	13
<b>Application – Profiles and Services</b>	13
<b>Create Scheduler</b>	14
<b>State machine</b>	15
<b>Inter-Communication</b>	15

<b>Configuration</b>	17
<b>NFC</b>	17
<b>BLE</b>	18
<b>MAIN</b>	18
<b>Accelerometer</b>	18
<b>SIGFOX</b>	19
<b>WIFI</b>	19
<b>BATTERY_CHECK</b>	20
<b>TMP</b>	21
<b>GPS</b>	21
<b>API</b>	23
<b>Bluetooth API</b>	23
<b>SIGFOX API(</b>	23
<b>WIFI API</b>	23
<b>ACCELEROMETER API</b>	24
<b>GPS API</b>	25
<b>TEMPERATURE API</b>	26
<b>Battery Check API (use ADC)</b>	27
<b>GPIO Keys API</b>	28
<b>Sleep</b>	28
<b>Deep Sleep</b>	28
<b>Flash binary</b>	28

## Introduction

### Device Model & Firmware Version

Model	Firmware
SFM20R	SFM20R_EVB_AV300 (SDK)
SRM200A	SRM200_EVB_AV300

The architecture of SW documentation includes descriptions to help you understand and develop SW in the module. Scenario mode and test mode are provided for development purposes only and should always be tested with your design.

## Getting Started

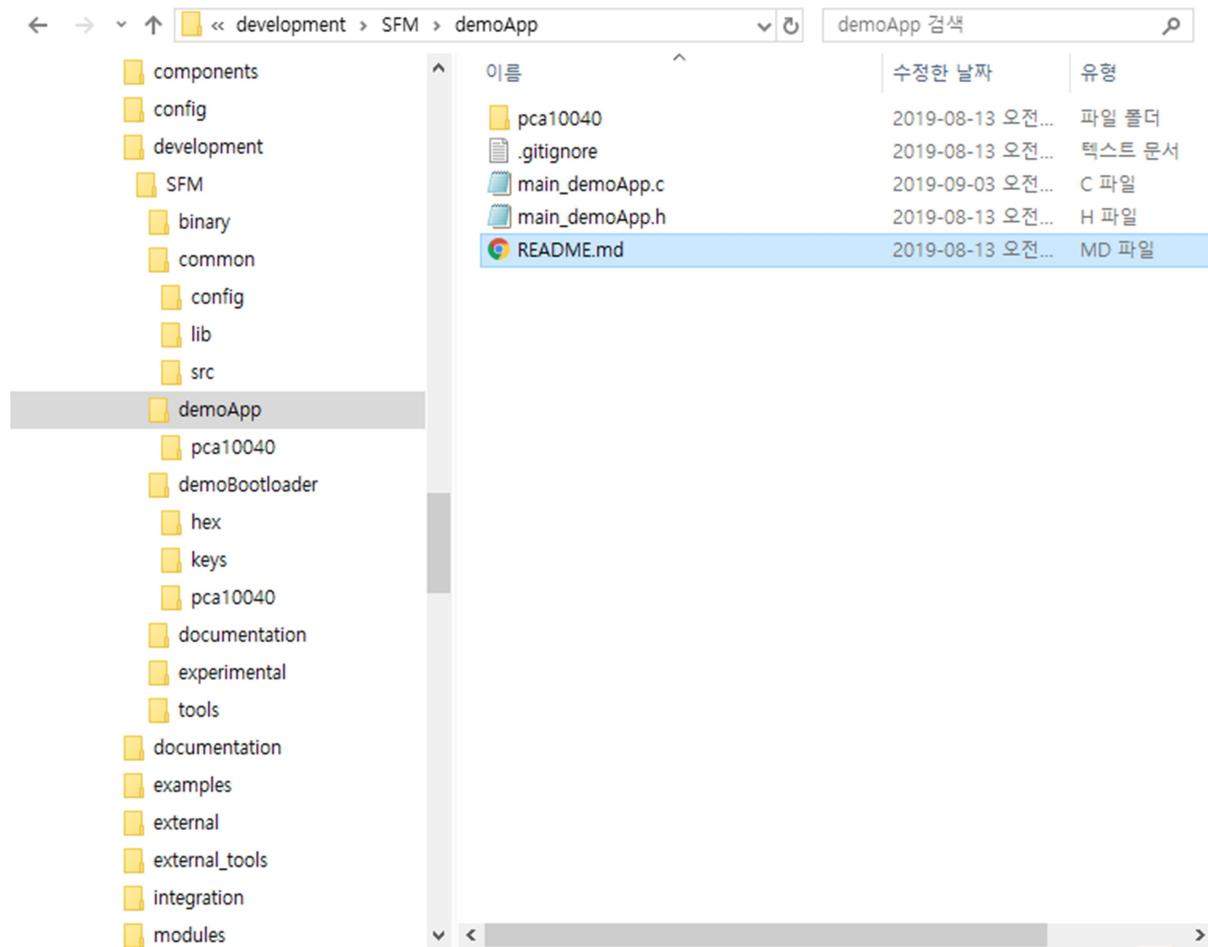
We provide the manual of setting up the development kit based on Nordic.

Refer to [SEONGJI]Development\_Environment\_Setup\_Guide\_V300.pdf

For more details, Refer to below

<https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v15.3.0%2Findex.html>

## The structure of the example files(tracker)



The tracker example is located in  
SF\_M\_15x\development\SFM\demoApp

The project files of the tracker are located in  
development\SFM\demoApp\pca10040\s132\ses\ SFM\_demoApp\_pca10040\_s132.emProject

Module files. (common file located in development\SFM\common\src)

- ✓ Main : main\_demoApp.c
- ✓ BLE : cfg\_ble\_ctrl.c, cfg\_nus\_cmd\_proc.c
- ✓ Temperature : cfg\_tmp102\_module.c
- ✓ Board support : cfg\_board.c, cfg\_nRF52\_peripherals.c
- ✓ Utility : cfg\_util.c
- ✓ Boot Mode ctl : cfg\_special\_boot\_mode.c
- ✓ User command With DBGI2C: cfg\_user\_cmd\_proc.c

- ✓ ADC for battery check : cfg\_adc\_battery\_check.c
- ✓ GPIO trigger(keys, magnetic) : cfg\_external\_sense\_gpio.c
- ✓ NVM Control : cfg\_nvm\_ctrl.c
- ✓ SIGFOX : cfg\_sigfox\_module.c, cfg\_sigfox\_module\_monarch.c
- ✓ GPS : cfg\_gps\_module.h
- ✓ WIFI : cfg\_wifi\_module.h
- ✓ Accelerometer : cfg\_bma250\_module.h
- ✓ Definition of scenario : cfg\_scenario.h
- ✓ DBG control over I2C, RTT : cfg\_twis\_board\_control.h

Library

## Mode

## Introduction

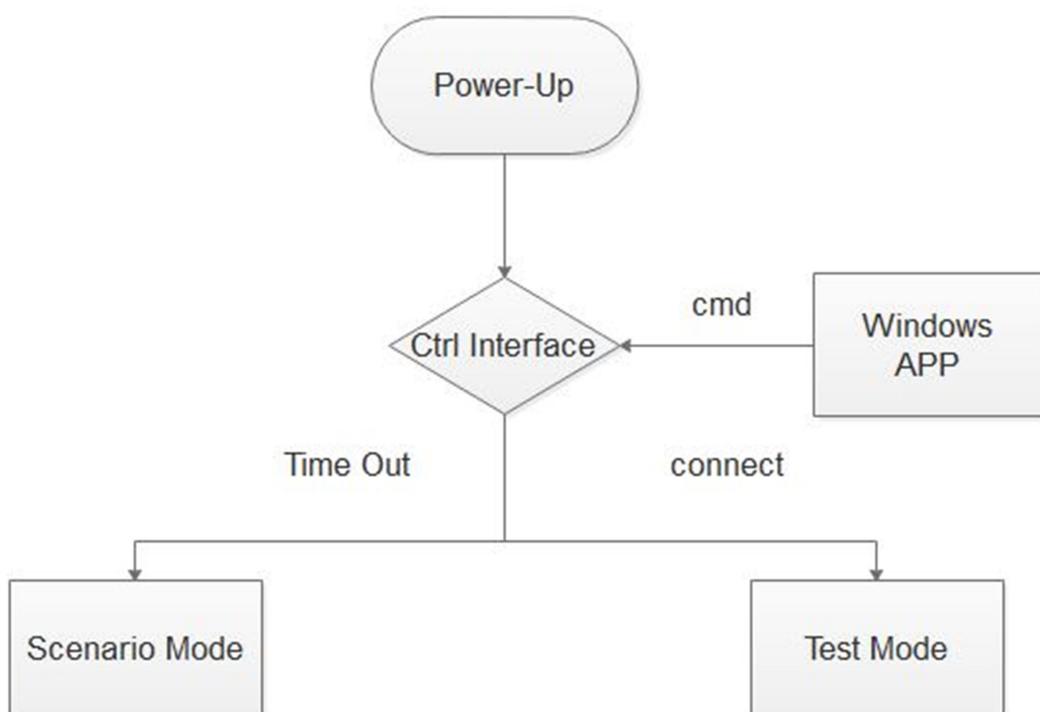
Examples are consist of two modes.

Scenario mode is based on Software Marketing Specification by SIGFOX.

Test mode enables user to test and debug each modules like a GPS module, a WIFI module, and so on.

## Selecting Mode

- Scenario mode is automatically selected in power-up.
- It can enter the test mode using the PC tool.  
PC tools only support windows. (development\sigfox\_cfg2\tools\EXE\_CONFIG2\_v013\_1.zip)
- Test mode needs pc tool and connecting to the used usb port. (use I2C)  
Run PC tool -> Connect with I2C in PC tool
- Instead of I2C, you can connect PC tool with J-Link (use SWD).  
Run RTT Viewer -> Run PC tool -> Connect with TCP in PC tool



## Code review

```
int main(void)
{
    ....
    cTBC_init(dbg_i2c_user_cmd_proc, true); //If i2c is not used, set the second argument to false
    cTBC_OVER_RTT_init(tbc_over_rtt_sec_tick_proc); //for use test mode over RTT
    ....
    cTBC_check_N_enter_bypassmode(200, main_bypass_enter_CB, main_bypass_exit_CB); //for enter test mode
    if(m_hitrun_test_flag)NVIC_SystemReset();
    ....
}
```

## Protocol with DBGI2C

development\SFM\documentation\Host\_FT20\_Protocal.txt

### User define Commands

development\SFM\common\src \cfg\_user\_cmd\_proc.c

dbg\_i2c\_user\_cmd\_proc()

## Boot Mode

### Introduction

Supports boot mode for RF regulatory certification.

For RF regulatory certification, see [RF\\_regulatory\\_certification\\_guide\\_REV00\\_eng\\_171021.doc](#).

See the cfg\_board\_check\_bootmode() function for more information.

It able to change the setting value with PC tool or RTT viewer.

Unless "normal mode", it can not use the PC tool.

And it can switch to "normal mode" by sending "CF" command with RTT.

When the setting value is saved, the setting is retained.

### Code review

see define FEATURE\_CFG\_CHECK\_NV\_BOOT\_MODE

```
int main(void)
{
    bool    erase_bonds = 0;

    cPrintLog(CDBG_MAIN_LOG, "\n===== %s Module Started Ver:%s bdtype:%d=====\\n", m_cfg_model_name, m_cfg_sw_ver, m_cfg_board_type);
    rPrintLog(CDBG_MAIN_LOG, "build date:%s, %s\\n", m_cfg_build_date, m_cfg_build_time);
    module_parameter_early_read(); //load setting value

    cfg_examples_check_enter();
    cfg_board_early_init();

    void cfg_board_early_init(void)
    {
        cfg_board_check_reset_reason();
        #ifdef FEATURE_CFG_CHECK_BOOTSTRAP_PIN
        cfg_board_check_bootstrap_pin();
        #else
        cfg_board_check_wifi_downloadmode();
        #endif
        #ifdef FEATURE_CFG_CHECK_NV_BOOT_MODE
        cfg_board_check_bootmode(); It is executed if boot mode is set.
        #endif
    }
}
```

### Kind of boot mode

Normal mode : 0 (default, The mode setting is possible with SWD)

Sigfox test mode : 6 (via SWD), 7(via UART)

WIFI test mode : 1 (via WIFI UART), \*WIFI download mode -> DL\_EN/INT\_WIFI to GND

BLE test mode : 3 (via SWD and via UART)

GPS test mode : 4 (via SWD and via UART)

#### \* Extended commands for RTT (uppercase, end of line is none)

"CC" -> Connect PC tool over RTT (Only available early in the boot)

"CR" -> Target reset

"CF" -> Change setting value to default

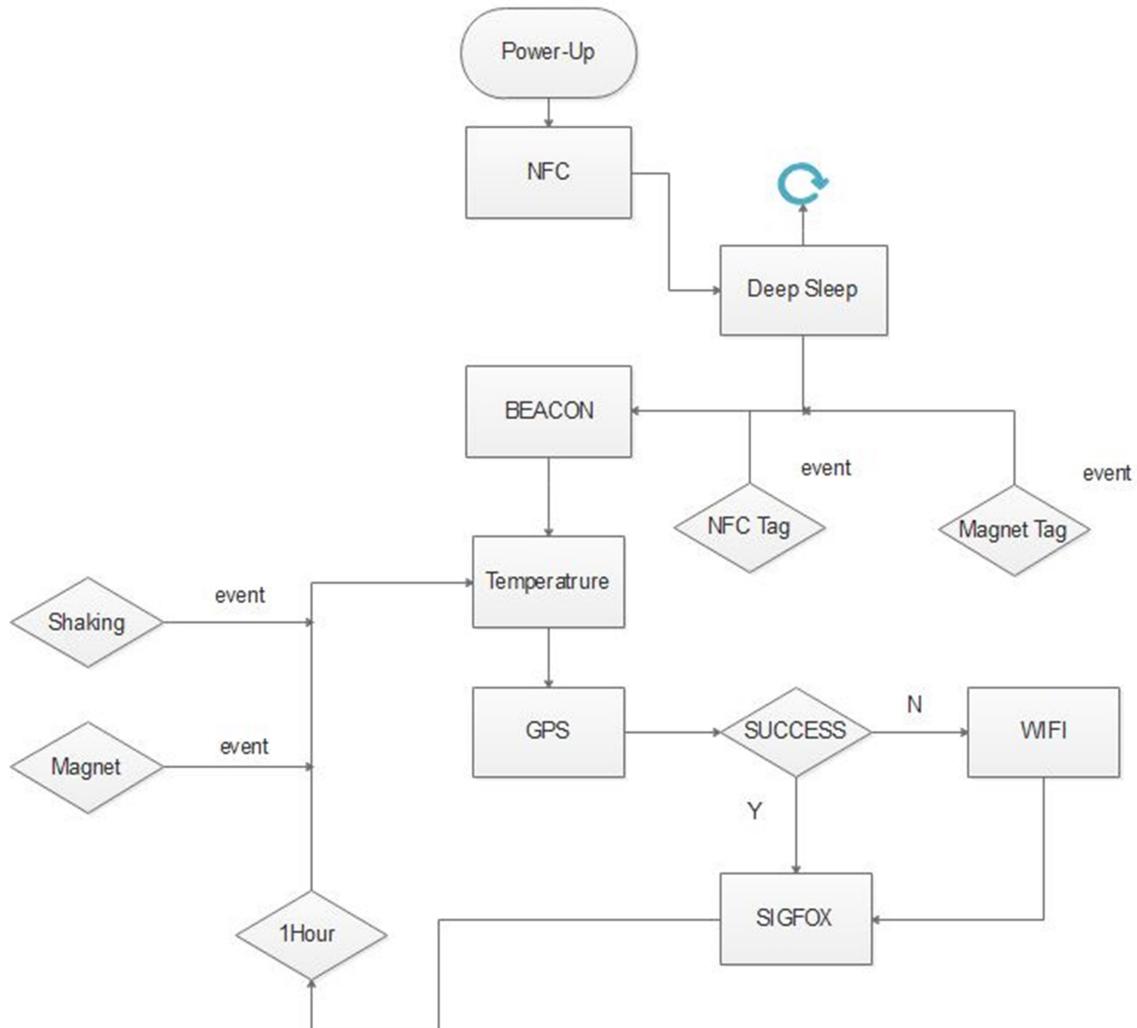
"CMx" -> Set boot mode (x is value of mode)

## Scenario of Tracker

### Introduction

The target application for the scenario mode is a low power tracking device.

The application use WIFI or GPS to determine location. It will then transmit the location information via SIGFOX. It also will transmit other information like temperature, accelerometer, and so on.



### Wake-up Device

- The device go to deep sleep state when power on
- Touching the NFC antenna with a smartphone or a tablet can wake up device.
- Touching magnetic sensor with magnetic tag can also wake up it.

## Bluetooth

- The default payload for the beacon advertisement will be the Model name(WISOL\_SFM21R).
- In Power-up or touching the NFC antenna with a smartphone, smartphone app will try to connect to the device with MAC of the device from NFC.
- Users can download firmware via FOTA, and get the information of device.

## Determining location

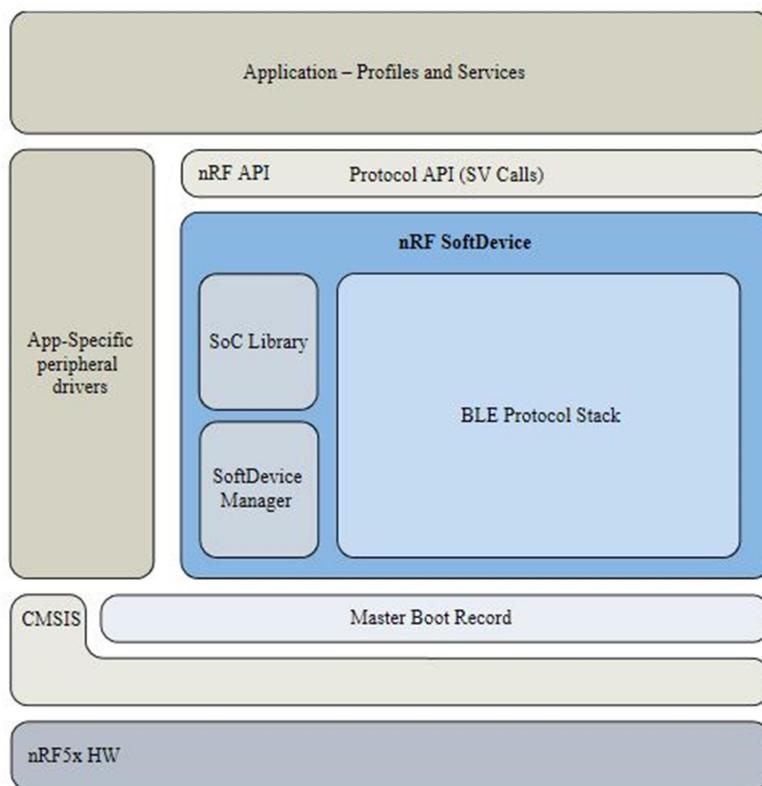
- The application can turn on the GPS radio and attempt to get a fix for 60s.
- The application can run a scan for access points for 10s with WIFI.

## Measurement frequency

- When power-up
- Every 10 minutes(editable)
- Whenever the accelerometer triggers an interrupt
- Whenever touching magnetic sensor with magnetic tag

## Architecture

### Introduction

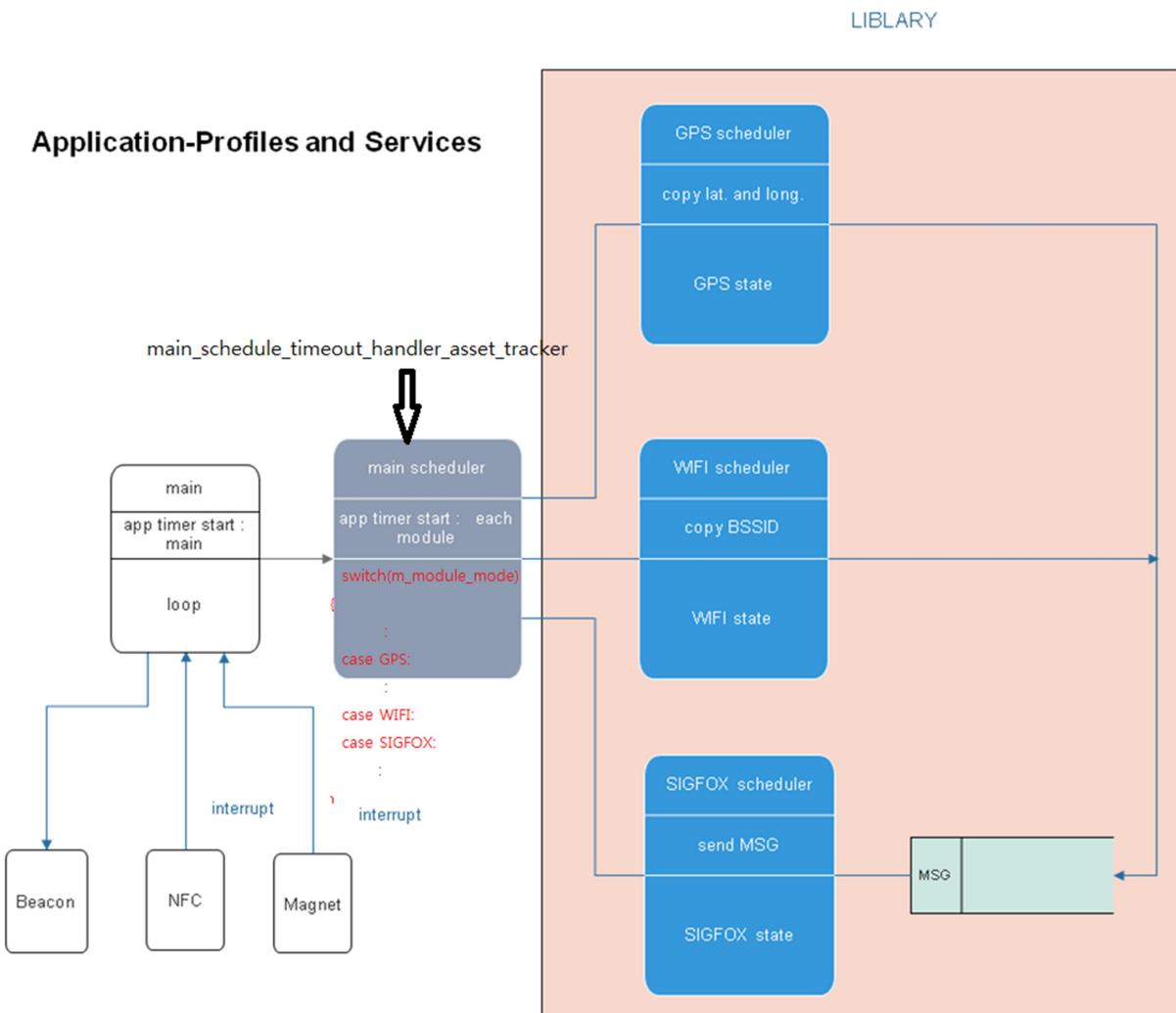


The application is based on nRF52832 platform. The application manages schedule using app timer.

The application has two kinds of scheduler, one is main scheduler and the other is module scheduler.

### Application – Profiles and Services

- The tracking-application runs on "application-Profiles and Services" layer.
- The developer can only use the API of Modules like create, start, stop.
- When "start" of API is called, each module automatically will runs and get the results.
- Each module is provided in the form of a library.



## Create Scheduler

- The timer library in nRF52832 platform enables the application to create multiple timer instances based on the RTC1 peripheral.
- Checking for time-outs and invoking the user time-out handlers is performed in the RTC1 interrupt handler.
- The main scheduler operates at a 20 ms cycle.
- Ex)

```
err_code = app_timer_create(&main_timer_id, mode, timeout_handler)
```

mode : either single shot or repeated

timeout\_handler : manage and execute each module

## State machine

- Scheduler uses state machine for managing each module.

- Ex)

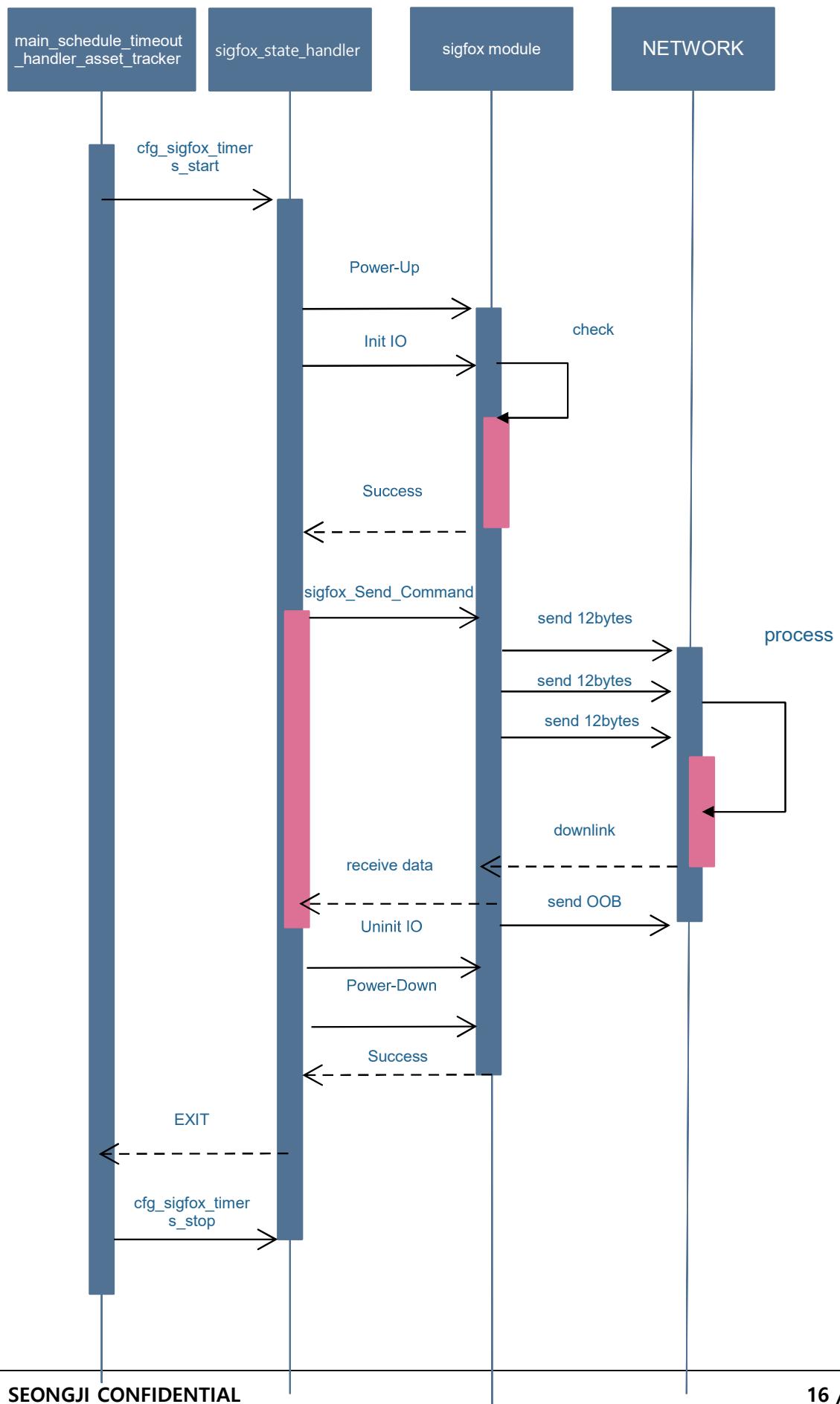
```
typedef enum
{
    NONE,
    ACC,
    MAIN_SCENARIO_LOOP,
    TMP,
    BLE,
    GPS,
    WIFI,
    SIGFOX,
    IDLE,
    BATTERY_CHECK
}module_mode_t
```

- When user time-out, scheduler determines whether to work in current state or move to another state.

## Inter-Communication

- Main scheduler calls create Function to execute any module.
- After each module work, the result is copied into sending buffer in SIGFOX.
- After main scheduler checking module's work, it move to another module.

- ex) between main scheduler and SIGFOX module



## Configuration

```


: /**
: * @brief setting items
: */
: #define module_parameter_t struct
: {
:     uint32_t magic_top;
:     uint32_t idle_time; //module_parameter_item_idle_time
:     uint32_t beacon_interval; //module_parameter_item_beacon_interval
:     uint32_t wifi_scan_retry_time_sec; //module_parameter_item_wifi_scan_retry_time_sec
:     uint32_t start_wait_time_for_board_control_attach_sec; //module_parameter_item_start_wait_time_for_board_control_attach_sec //use to sfm_boot_mode START_WAIT_TIME_FOR_BOARD_CONTROL_ATTACH_SEC
:     uint32_t start_wait_time_for_ctrl_mode_change_sec; //module_parameter_item_start_wait_time_for_ctrl_mode_change_sec //use to sigfox dl enable
:     uint32_t gps_acquire_tracking_time_sec; //module_parameter_item_gps_tracking_time_sec
:     bool boot_nfc_unlock; //module_parameter_item_boot_nfc_unlock
:     bool foto_enable; //module_parameter_item_foto_enable
:     uint16_t /*MODULE_DEFAULT_SCENARIO_TYPE*/ scenario_mode; //module_parameter_item_scenario_mode //MODULE_DEFAULT_SCENARIO_TYPE
:     bool magnetic_gpio_enable; //module_parameter_item_magnetic_gpio_enable
:     bool wklup_gpio_enable; //module_parameter_item_wklup_gpio_enable //MAIN_WKUP_GPIO_ENABLE_DEFAULT
:     bool wifi_teardown_enable; //module_parameter_item_wifi_teardown_enable
:     bool sigfox_snek_testmode_enable; //module_parameter_item_snek_testmode_enable
:     bool cgps_cn0_current_savetime_enable; //module_parameter_item_cgps_cn0_current_savetime_enable
:
:     //setting value here
:     uint32_t magic_bottom;
:     uint32_t guard_area_align4;
:
:     //id cache value here
:     uint8_t wifi_MAC_STA[6];
:     uint8_t sigfox_device_ID[4];
:     uint8_t sigfox_pac_code[8];
:     uint32_t boot_ID;
:     int32_t guard_area_align4_2;
: } ? end {anonmodule_parameter_t} : module_parameter_t;


```



- users can add the parameters they want
- static void module\_parameter\_init(void) : read the values of the parameters
- static void module\_parameter\_default\_init(void) : set the default values of the parameters

## NFC

- Initialize

```


void nfc_init()
{
    uint32_t err_code;
    if(!mnfc_init_flag)
    {
        /* Set up NFC */
        err_code = nfc_t2t_setup(nfc_callback, NULL);
        APP_ERROR_CHECK(err_code);

        /* @snippet [NFC URI usage_1] */
        /* Provide information about available buffer size to encoding function */
        uint32_t len = sizeof(m_ndef_msg_buf);

        welcome_msg_encode(m_ndef_msg_buf, &len);

        /* @snippet [NFC URI usage_1] */

        /* Set created message as the NFC payload */
        err_code = nfc_t2t_payload_set(m_ndef_msg_buf, len);
        APP_ERROR_CHECK(err_code);

        /* Start sensing NFC field */
        err_code = nfc_t2t_emulation_start();
        mnfc_init_flag = true;
    } ? end if !mnfc_init_flag ?
} ? end nfc_init ?


```

callback function for handling NFC events

- Set data

```


/**
 * @brief Function for encoding the welcome message.
 */

NFC_NDEF_MSG_DEF(welcome_msg, MAX_REC_COUNT);
static void welcome_msg_encode(uint8_t * p_buffer, uint32_t * p_len)
{
    sigfox_id_record_add(&NFC_NDEF_MSG(welcome_msg));
    sigfox_pac_record_add(&NFC_NDEF_MSG(welcome_msg));
    ble_mac_record_add(&NFC_NDEF_MSG(welcome_msg));

    /* @snippet [NFC text usage_2] */
    uint32_t err_code = nfc_ndef_msg_encode(&NFC_NDEF_MSG(welcome_msg),
                                            p_buffer,
                                            p_len);

    APP_ERROR_CHECK(err_code);
    /* @snippet [NFC text usage_2] */
}


```

set SIGFOX ID, PAC, and MAC to NFC MSG

## BLE

- m\_module\_parameter.beacon\_interval : advertising interval
- err\_code = sd\_ble\_gap\_device\_name\_set(&sec\_mode,  
`(const uint8_t *)DEVICE_NAME,`  
`strlen(DEVICE_NAME));`

you can change advertising payload using the function.

- static void services\_init(void) : add ble uart service and FOTA service
- refer to [http://infocenter.nordicsemi.com/index.jsp](http://infocenter.nordicssemi.com/index.jsp) for more details

## MAIN

- create main scheduler  
`static void main_timer_create()`
- start main scheduler  
`void main_timer_schedule_start(void)`  
`// call static void main_schedule_timeout_handler_asset_tracker(void * p_context) periodic`  
`200ms`
- go to sleep  
`void main_timer_idle_start(void)`  
`m_module_parameter.idle_time : report over sigfox every idle_time sec`
- stop main scheduler  
`void main_timer_schedule_stop(void)`
- callback function for handling main state from main schedule timer  
`static void main_schedule_timeout_handler_asset_tracker(void * p_context)`
- move to state  
`ex) main_set_module_state(MAIN_SCENARIO_LOOP); //go to MAIN_SCENARIO_LOOP`

## Accelerometer

- create accelerometer module  
`void cfg_bma250_timer_create()`
- start accelerometer module  
`void cfg_bma250_timers_start(void)`
- set interrupt  
`void cfg_bma250_interrupt_init(void)`
- callback function for handling accelerometer state from acc module timer  
`static void bma250_state_handler(void * p_context)`

```
case SET_S: call the function of configuration in accelerometer
    ex) error_code = bma250_slope_set();
```

## SIGFOX

- create SIGFOX module and get SIGFOX ID and PAC  
void cfg\_sigfox\_prepare\_start(void)
- create SIGFOX module  
void cfg\_sigfox\_timer\_create()  
: if call void cfg\_sigfox\_prepare\_start(void), don't call void cfg\_sigfox\_timer\_create()
- start SIGFOX module  
void cfg\_sigfox\_timers\_start(void)
- sending buffer  
: uint8\_t frame\_data[(SIGFOX\_SEND\_PAYLOAD\_SIZE\*2)+1]; //for hexadecimal
- downlink buffer  
: uint8\_t downlink\_data[30];
- check whether SIGFOX finishes sending.  
bool sigfox\_check\_exit\_execute(void)
- select SNEK or SIGFOX network  
m\_module\_parameter.sigfox\_snek\_testmode\_enable =  
MAIN\_SIGFOX\_SNEK\_TESTMODE\_ENABLE\_DEFAULT; true : snek
- enable/disable DOWNLINK  
cfg\_sigfox\_downlink\_on\_off(true); : call it before start SIGFOX module

```
case SIGFOX:
    if(sigfox_get_state() == SETUP_S)
    {
        if(m_init_execute)
        {
            cTBC_write_state_noti("Sigfox");
            cPrintLog(CDBG_FCTRL_INFO, "%s %d SIGFOX MODULE started\n", __func__, __LINE__);
            cfg_sigfox_timers_start();

            m_init_execute = false;
        }
    }
    else if(sigfox_check_exit_execute())
    {
        main_set_module_state(BLE);
        cfg_sigfox_timers_stop();
        sigfox_set_state(SETUP_S);
        nus_send_data('T');
        nus_send_data('B');
    }
break;
```

## WIFI

- WIFI initialization

```
void cfg_board_init(void)
{
    .....

#ifndef CDEV_WIFIT_MODULE
    cWifi_resource_init();
    cWifi_prepare_start();
#endif
    .....
}
```

- Run WIFI Scan

```

case WIFI:
{
    ...
    bool send_data_req = false;
    uint8_t send_data[SIGFOX_SEND_PAYLOAD_SIZE] = {0,};
#ifndef CDEV_WIFI_MODULE
    int get_bssid_result;
    uint8_t *bssidBuf;
    int wifi_result;
    +m_module_ready_wait_timeout_tick;
    if(m_init_execute)
    {
        ...
        wifi_result = cWifi_ap_scan_req();
        if(wifi_result == CWIFI_Result_OK)
        {
            ...
            m_init_execute = false;
        }
        else
        {
            cPrintLog(CDBG_FCTRL_INFO, "%s %d Not Available Wifi Module! send NULL data!\n", __func__, __LINE__);
            send_data_req = true;
        }
    }
    else
    {
        if(!cWifi_is_scan_state() && !cWifi_bus_busy_check()) //wait scan
        {
            ...
            get_bssid_result = cWifi_get_BSSIDS_bufPtr(&bssidBuf);
            cPrintLog(CDBG_FCTRL_INFO, "%s %d WiFi MODULE end! result:%d BSSID:", __func__, __LINE__, get_bssid_result);
            cDataDumpPrintOut(CDBG_FCTRL_INFO, bssidBuf, (CWIFI_BSSID_CNT*CWIFI_BSSID_SIZE));
            if(get_bssid_result == CWIFI_Result_OK)
            {
                ...
                memcpy(send_data, bssidBuf, sizeof(send_data));
            }
            else if(get_bssid_result == CWIFI_Result_NoData)
            {
                cPrintLog(CDBG_FCTRL_INFO, "%s %d WiFi MODULE NoData! send NULL data!\n", __func__, __LINE__);
            }
            else
            {
                cPrintLog(CDBG_FCTRL_INFO, "%s %d Not Available Wifi Module! send NULL data!\n", __func__, __LINE__);
                send_data_req = true;
            }
        } ? end if !cWifi_is_scan_state(...)??
    } ? end else ?
}
#endif
...
#endif
if(send_data_req)
{
    memcpy(m_module_peripheral_data.sigfixdata_wifi, send_data, sizeof(m_module_peripheral_data.sigfixdata_wifi));
    cfg_bin_2_hexadecimal(send_data, SIGFOX_SEND_PAYLOAD_SIZE, (char *)frame_data); //set sigfox payload
    cPrintLog(CDBG_FCTRL_INFO, "%s %d send request sigfox! data:%s\n", __func__, __LINE__, frame_data);
    main_set_module_state(SIGFOX);
}
}
break;
}

```

## BATTERY\_CHECK

- cfg\_bas\_timer\_create();
- Run BATTERY CHECK

```

#ifndef PIN_DEF_BATTERY_ADC_INPUT
+m_module_ready_wait_timeout_tick;
if(m_init_execute)
{
    adc_configure();
    cfg_bas_timer_start();
    m_init_execute = false;
}
else
{
    if(m_module_ready_wait_timeout_tick >= (APP_MAIN_SCHEDULE_HZ * 2)) //wait 2 sec
    {
        ...
        avg_batt_lvl_in_milli_volt = get_avg_batt_lvl_in_milli_volt();
        if(3000 <= avg_batt_lvl_in_milli_volt && 5200 >= avg_batt_lvl_in_milli_volt)
        {
            if(!m_module_parameter.wifi_testmode_enable /*disable_battery_power_down*/ )cPrintLog(CDBG_MAIN_LOG, "Battery Pwr Off Disabled\n");
            if(!m_module_parameter.wifi_testmode_enable /*disable_battery_power_down*/ && !cTBC_check_host_connected() && avg_batt_lvl_in_milli_volt < 3500) //low battery
            {
                ...
                main_powerdown_request = true;
            }
            else if(!m_module_parameter.wifi_testmode_enable /*disable_battery_power_down*/ && avg_batt_lvl_in_milli_volt < 3600) //battery warning
            {
                ...
            }
            else
            {
                main_set_module_state(TMP); //battery value is enough
            }
        }
        else
        {
            if(log_once_flag)cPrintLog(CDBG_MAIN_LOG, "ADC Not Available:%d\n", avg_batt_lvl_in_milli_volt);
            main_set_module_state(TMP); //battery value is not available
        }
    } ? end if m_module_ready_wait_t...??
} ? end else ?
#endif
main_set_module_state(TMP);
break;
}

```

## TMP

```
#define TMP102_I2C_ADDRESS 72 /* This is the I2C address for our chip – 0x48 */
```

- TMP102 timer create
- TMP102 timer start
- TMP102 timer stop

```
case TMP:
    if(tmp102_get_state() == NONE_TMP)
    {
        cTBC_write_state_noti("Temperature");
        cPrintLog(CDBG_FCTRL_INFO, "%s %d TMP MODULE started\n", __func__, __LINE__);
        tmp102_set_state(TMP_SET_S);
        cfg_tmp102_timers_start();
    }
    else if(tmp102_get_state() == EXIT_TMP)
    {
        cfg_tmp102_timers_stop();
        tmp102_set_state(NONE_TMP);
        main_set_module_state(GPS);
    }
break;
```

tmp102 time create

```
int tmp102_get_result(int *tmp102_int, int *tmp102_dec)
{
    int result = 0;
    *tmp102_int = tmp102a;
    *tmp102_dec = tmp102b;
    if((tmp102a != 0) || (tmp102b != 0))
    {
        result= true;
    }
    else
    {
        result = false;
    }
    return result;
}
```

tmp102 time stop

get temperature

## GPS

The description below is based on the sigfox module.

GPS hot start :

- You must use backup battery to enable hot start.
- The hot start works normally when retry tracking within 2 hours.
- In order to use backup battery, V\_BCKP\_GPS and VDD\_MAIN\_3P0 must be connected (see Schematics).

- Connecting backup battery adds 15uA of current consumption.
- Also, GPS should be turned off after about 1 ~ 2 seconds after GPS fixed. (In case of EVB, GPS power off after about 2 seconds)

- If GPS power off immediately after Fixed, no information such as minimum ephemeris, almanac, time is received.

- Initialize GPS

```
void cfg_board_init(void)
{
    ...
    gps_resource_init();
    gps_prepare_start();
    ...
}
```

GPIO & Power control

- GPS start

```

        break;
    case GPS:
    #ifdef CDEV_GPS_MODULE
        memset(gps_send_data, 0, sizeof(gps_send_data));
        if(work_mode == GPS_START)
        {
            if(cGps_status_available() == CGPS_Result_OK)
            {
                cTBC_write_state_noti("GPS");
                cPrintLog(CDBG_FCTRL_INFO, "$s %d GPS MODULE started \n", __func__, __LINE__);
                cGps_nmea_acquire_request();
                cfg_ble_led_control(false);
                work_mode = GPS_WORK; //wait scan
            }
            else
            {
                work_mode = GPS_END;
            }
        }
        ...
    
```

Request start GPS acquire

- GPS position fixed check

```

else if(work_mode == GPS_WORK)
{
    cfg_ble_led_control(true);
    get_nmea_result = cGps_acquire_tracking_check();
    if(get_nmea_result == CGPS_Result_OK)
    {
        cfg_ble_led_control(false);
        work_mode = GPS_END;
    }
    else if(get_nmea_result == CGPS_Result_Busy)
    {
        ;
    }
    else

```

GPS position fixed check

- Send data to sigfox

```

else if(work_mode == GPS_END)
{
    cfg_ble_led_control(true);
    get_nmea_result = cGps_nmea_get_bufPtr(&nmea_Buf);
    if(get_nmea_result == CGPS_Result_OK)
    {
        memcpy(gps_send_data, nmea_Buf, sizeof(gps_send_data));
        send_gps_data_req = true;
    }
    if(send_gps_data_req)
    {
        gps_send_data[9] = SIGFOX_MSG_SEND_REASON(main_wakeup_reason); /*10th byte Status filed modified*/
        cfg_bin_2_hexadecimal(gps_send_data, SIGFOX_SEND_PAYLOAD_SIZE, (char *)frame_data); //set sigfox payload
        cPrintLog(CDBG_FCTRL_INFO, "[GPS] %d send request sigfox! frame_data:[%s] \n", __LINE__, frame_data);
        send_gps_data_req = false;
        work_mode = GPS_START;
    }
}

```

GPS Latitude & longitude data check

send data to sigfox

## API

### Bluetooth API

- refer to documents of Nordic SDK

### SIGFOX API(

- Module mode
  - start SIGFOX module after calling cfg\_sigfox\_timer\_start()
  - ex) example of tracker when using statemachine
- API mode
  - Users have to wait for result of sending
- refer to #development#sigfox\_cfg2#documentation#application\_note#[WISOL]AppNote\_SFM20R\_example\_of\_Sigfox\_V200.pdf for more details

### WIFI API

- \* Can not run with GPS because it shares SPI resources
- Initialize wifi driver
 

```
int wifi_drv_init(void)
{
    cWifi_resource_init();
    cWifi_prepare_start();
    if(cWifi_is_detected())
    {
        return CWIFI_Result_OK;
    }
    else
    {
        return CWIFI_Result_NoDevice;
    }
}
```
- Scan for local WIFI access points
  - 1) Sort by RSSI. It can get up to 2

```
int start_AP_scan(void)
{
    return cWifi_ap_scan_req();
}

retval CWIFI_Result_OK in successc
```

2) BSSID filtered(prefix). It can get highest RSSI 1.

```
wifi_result = cWifi_ap_get_available_first_BSSID(prefixSSID);
retval CWIFI_Result_OK in success
```

- Set the duration of a scan from 1s to 60s

```
void set_scan_interval(int interval);
param interval 1~60 sec
```

- Report basestation IDs and associated RSSI readings

```
int get_AP_scanResult(uint32_t * get_cnt, uint8_t **ssid, int32_t **rss, uint8_t **bssid);
param[out] *get_cnt pointer to scan count
param[out] **ssid pointer to ssid data
param[out] **rss pointer to rss data
param[out] **bssid pointer to bssid data
```

- refer to #development#sigfox\_cfg2#documentation#application\_note#[WISOL]AppNote\_SFM20R\_example\_of\_WIFI\_V200.pdf for more details

\* Use the Bypass API, it can control directly with AT commands.

See the cfg\_examples\_wifi\_bypass() function. (cfg\_examples.c)

## ACCELEROMETER API

- Initialize I2C

```
void cfg_i2c_master_init(void);
```

- Uninitialize I2C

```
void cfg_i2c_master_uninit(void);
```

- Take an accelerometer reading(x,y,z)

```
bool cfg_bma250_read_xyz(struct bma_accel_data *accel);
param[out] struct bma_accel_data pointer to value of x,y,z
retval true in success
```

```
struct bma_accel_data {
    int16_t x; /*< accel x data */
    y; /*< accel y data */
    z; /*< accel z data */
};
```

- Read the value of register  

```
bool cfg_bma250_read_reg(uint8_t reg_addr, uint8_t * read);
param[in] uint8_t reg_addr address of register
param[out] uint8_t * read value of register
retval true in success
```
- Write the value in register  

```
bool cfg_bma250_write_reg(uint8_t reg_addr, uint8_t reg_data);
param[in] uint8_t reg_addr address of register
param[in] uint8_t reg_data value to write to register
```
- refer to BST-BMA250E-DS004-06.pdf for knowing registers in detail.

## GPS API

- Initialize GPS driver  

```
void gps_init(void)
{
    cGps_resource_init(); // initializing gpio of GPS
    cGps_prepare_start(); // GPS power control
}
```
- Start tracking  

```
int start_gps_tracking(void);
retval 0 in success
```
- GPS Tracking set/get interval time  

```
void gps_tracking_set_interval(module_parameter_item_e item, unsigned int val);
unsigned int gps_tracking_get_interval(module_parameter_item_e item);
```
- Get GPS NMEA data  

```
int get_NMEA_Location(char **latitude, char **longitude);
int get_NMEA_UTCTime(uint8_t *hour, uint8_t *minute, uint8_t *second);
int get_NMEA_HDOP(char **hdop);
int get_NMEA_Speed_knot(char **speed);
int get_NMEA_Direction(char **ns, char **ew);
int get_NMEA_UTCDDate(uint8_t *year, uint8_t *month, uint8_t *day);
```
- GPS position fixed check

```
int cGps_nmea_position_fix_check(void);
```

- Report the longitude and latitude

```
int get_lastLocation(char **ns, char**latitude, char ** ew, char**longitude);
param[out] char **ns pointer to the data of cardinal point ( N or S)
param[out] char **latitude pointer to the data of latitude
param[out] char **new pointer to the data of cardinal point ( E or W)
param[out] char **longitude pointer to the data of longitude
retval 0 in success
```

## TEMPERATURE API

```
#define TMP102_I2C_ADDRESS 72 /* This is the I2C address for our chip – 0x48 */
```

```
#define TMP102_TEMP_REG          0x00
#define TMP102_CONF_REG           0x01
#define TMP102_TLOW_REG           0x02
#define TMP102_THIGH_REG          0x03
```

- TMP102 timer create  
void cfg\_tmp102\_timer\_create(void);
- TMP102 timer start  
void cfg\_tmp102\_timers\_start(void);
- TMP102 timer stop  
void cfg\_tmp102\_timers\_stop(void);
- TMP102 shutdown  
uint32\_t tmp102\_req\_shutdown\_mode(void);
- Initialize I2C  
void cfg\_i2c\_master\_init(void);
- Uninitialize I2C  
void cfg\_i2c\_master\_uninit(void);
- TMP102 set/get state  
typedef enum
{
 NONE\_TMP,
 TMP\_SET\_S,
 TMP\_SET\_R,
 TMP\_READ\_DATA\_S,
 TMP\_READ\_DATA\_R,
 TMP\_SLEEP\_S, //5
 TMP\_SLEEP\_R,

```

    EXIT_TMP
} tmp102_state_s;

void tmp102_set_state(tmp102_state_s m_state);
tmp102_state_s tmp102_get_state(void);

```

- Read temperature

```

int get_temperature(int * tmp_int, int *tmp_dec);
param[out] int *tmp_int pointer to integer of temperature
param[out] int *tmp_dec pointer to decimals of temperature
retval 1 in success

```

- Set threshold(low, high)

```

uint32_t set_alert_threshold(u16 tmp_min, u16 tmp_max);
param[in] u16 tmp_min the low value of temperature
param[in] u16 tmp_max the high value of temperature
retval 0 in success

```

## Battery Check API (use ADC)

ADC Resolution is 10 bit

see define PIN\_DEF\_BATTERY\_ADC\_INPUT  
 (undefined or NRF\_SAADC\_INPUT\_AIN0 or NRF\_SAADC\_INPUT\_AIN1)  
`: #define PIN_DEF_BATTERY_ADC_INPUT NRF_SAADC_INPUT_AIN0`

Modify ADC\_RESULT\_IN\_MILLI\_VOLTS macro for resistance distribution

```

#define ADJUST_BATTERY_VALUE [50]
#define ADC_RESULT_IN_MILLI_VOLTS(ADC_VALUE) (((((ADC_VALUE) * ADC_REF_VOLTAGE_IN_MILLIVOLTS) / ADC_RES_10BIT) * ADC_PRE_SCALING_COMPENSATION) * [5/3])

```

- Initialize ADC driver

```
cfg_bas_timer_create();
```

- Read (ADC Resolution is 10 bit)

```
adc_configure();
cfg_bas_timer_start();
```

- Get value

```
get_avg_batt_lvl_in_milli_volts()
retval eg. 3.7V -> 3700
```

## GPIO Keys API

### 1) Magnetic

```
modify #define PIN_DEF_MAGNETIC_SIGNAL
cfg_magnetic_sensor_init(magnetic_attach_callback, magnetic_detach_callback)
```

### 2) WKUP key

```
modify #define PIN_DEF_WKUP
cfg_wkup_gpio_init(wkup_detect_callback)
```

### 3) Button

```
short press interval : GFG_BUTTON_SENSE_PRESS_TICK_SHORT
log press interval : GFG_BUTTON_SENSE_PRESS_TICK_LONG
cfg_button_init(is_active_high, pin, short_press_CB, long_press_CB)
```

## Sleep

The ARM core is in a sleep state. A transition from "if 0" to "if 1" interrupt occurs.

```
cfg_board_power_manage()
```

\*sleep test code (uncomment this function)  
`cfg_board_testmode_BLE_Advertising_LowPwr()`

## Deep Sleep

The ARM core and Ram are in a power-down state. A call to the test function.

It can wake up with GPIO or NFC.

```
·cfg_board_prepare_power_down();
·cPrintLog(CDBG_MAIN_LOG, "power off\n");
·cfg_board_goto_power_down();
```

In cfg\_scen\_main\_handler()

## Flash binary

The binary download uses J-Link equipment.

1. Build by SES
2. Run development\WIFM\binary\ download\_Factory.cmd  
 Download the firmware using J-Link.