RT58xZigbee

Application Guide

V1.6

Table of Contents

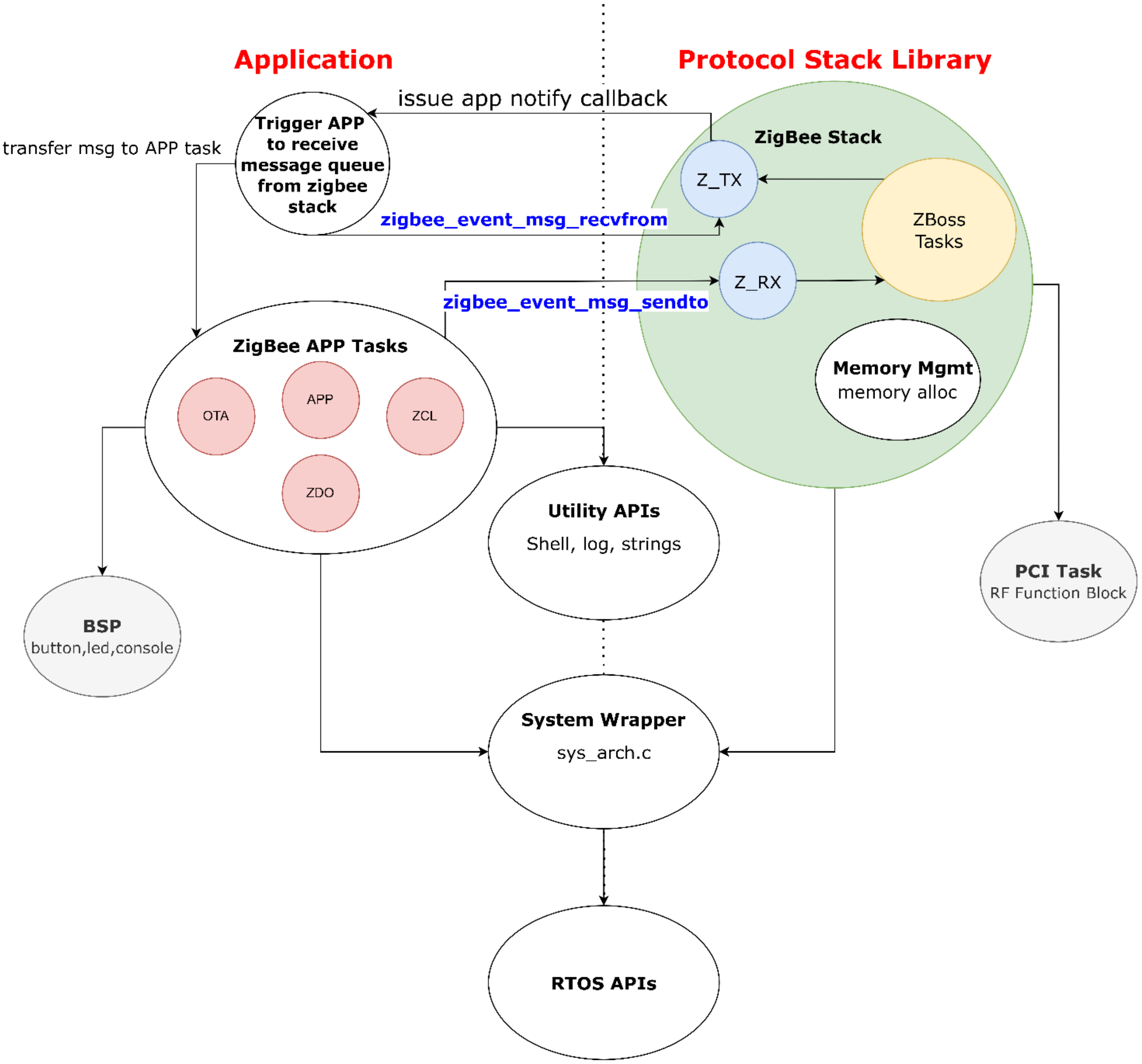
1. Introduction

This document is mainly about the instructions related to the ZigBee example demonstration. The content includes Gateway, Swtich(OnOff/Dimmer), Light(OnOff/Dimmer) and OTA.

2. System Architecture

This section provides an overview of the zigbee stack library operation and layered system architecture.

2.1 Task architecture



2.1.1 Zigbee Stack

Rafael ‘s Zigbee stack sub-system.

2.1.2 PCI Task

Rafael’s RF function block controller.

2.1.3 System wrapper

OS wrapper for real time OS.

2.1.4 Utility APIs

Useful functions and general macro definitions.
Ex.: debug log, queue, list, FSM.

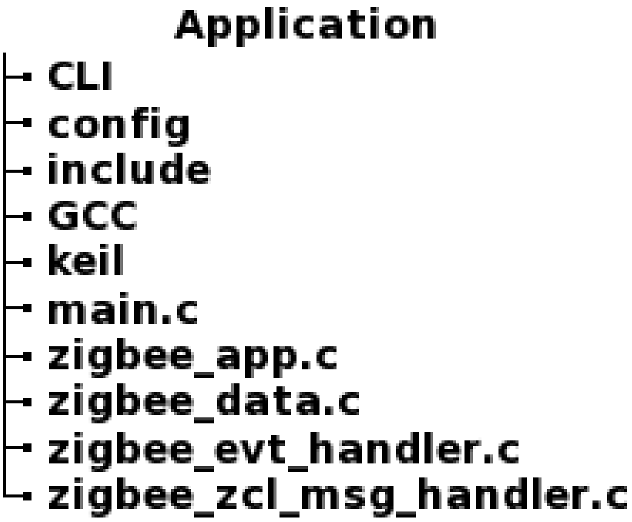
2.1.5 Zigbee APP Tasks

Handle BSP, Appliction, ZCL and product behavior.

2.1.6 RTOS

Zigbee SDK is based on FreeRTOS.

2.2 File architecture



```
Application
├ CLI
├ config
├ include
├ GCC
├ keil
├ main.c
├ zigbee_app.c
├ zigbee_data.c
├ zigbee_evt_handler.c
└ zigbee_zcl_msg_handler.c
```

2.2.1 CLI

Source files of CLI commands.

2.2.2 config

Congiguration of project and FreeRTOS.

2.2.3 Include

Header files of application.

2.2.4 Keil

Keil project files.

2.2.5 GCC

GCC project files.

2.2.6 main.c

Main function of application.

2.2.7 zigbee_app.c

Task function of Zigbee application.

2.2.8 zigbee_data.c

Device context of application (Attribute, Cluster, Endpoint, Simple description).

2.2.9 zigbee_evt_handler.c

Process event message from ZigBee stack library.

2.2.10 zigbee_zcl_msg_handler.c

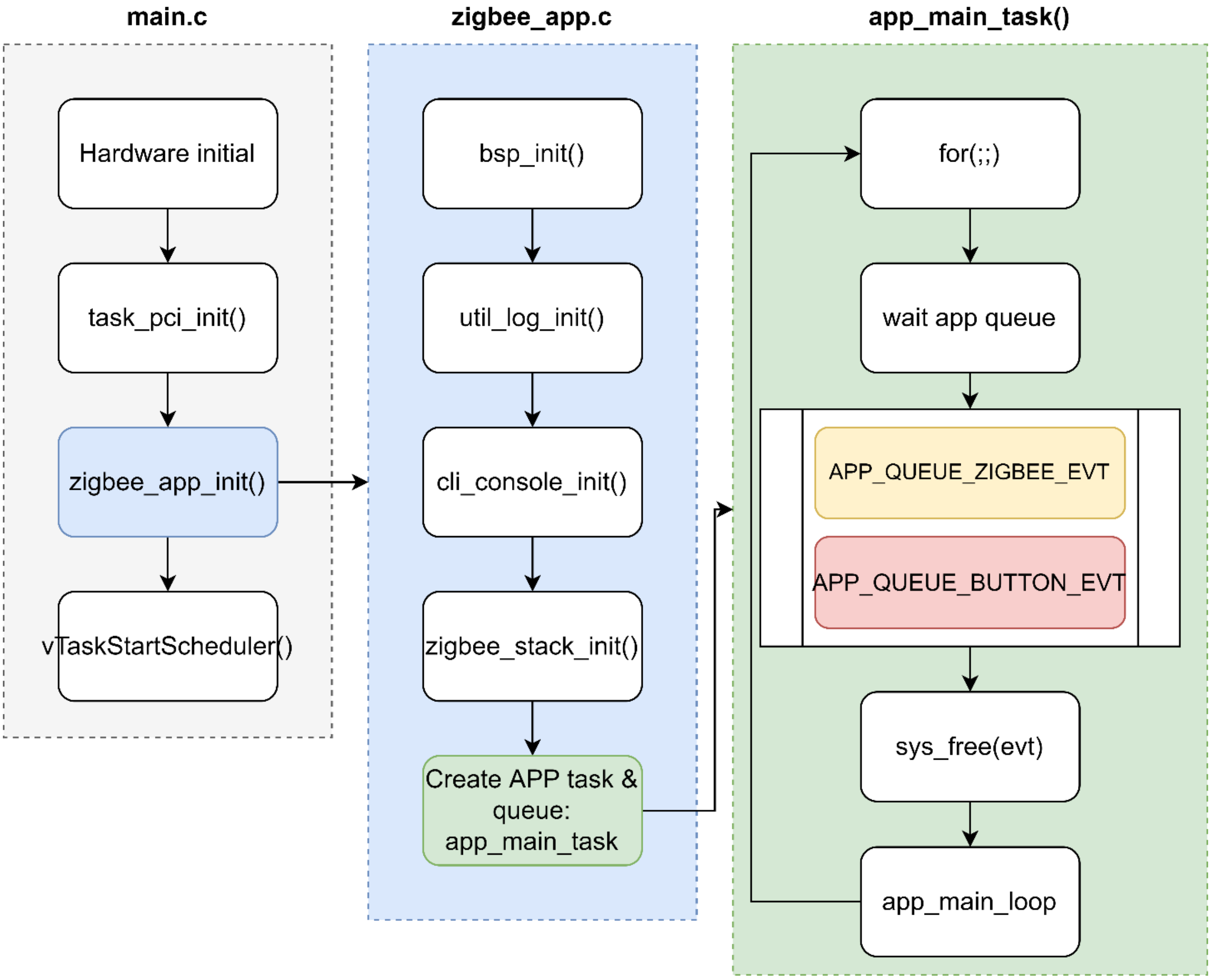
Process ZCL event message from ZigBee stack library.

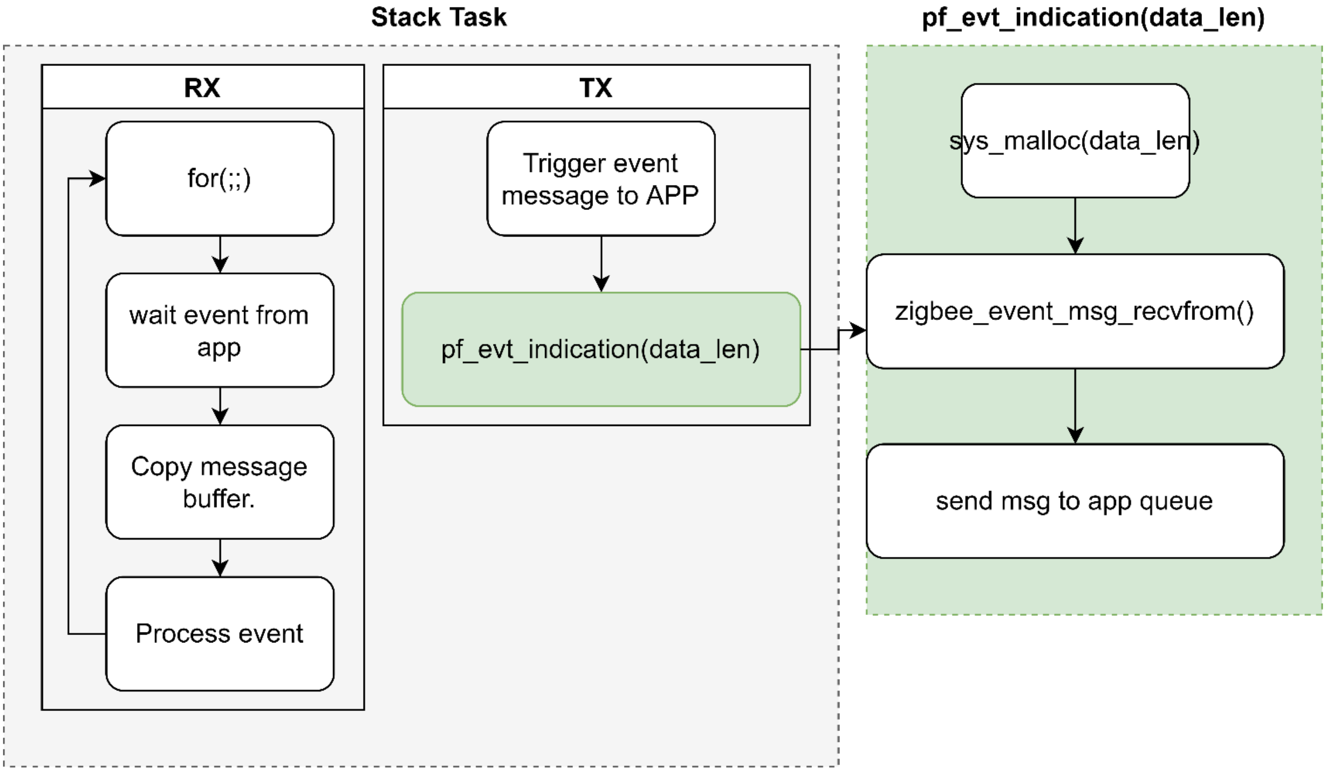
3. Application development

This section introduces the ZigBee application flow which is implemented in Rafael RT58x ZigBee SDK. User could follow the instruction in this section to develop a customized ZigBee application.

3.1 Application Flow

The following figure shows the ZigBee application flow in Rafael RT58x ZigBee SDK.





If ZigBee application task receives the event “APP_QUEUE_ZIGBEE_EVT” will issue “zigbee_zcl_msg_handler()” or “zigbee_evt_handler()” function.

3.2 Initialization

RT58x ZigBee application general initialization process is shown as below:

1. Initial 802.15.4 driver (task_pci_init())
2. Initial BSP(Optional).
3. Initial utility logging (Optional)
4. Initial CLI Console(Optional)
5. Registered ZigBee device context and event indication callback.
6. Initial ZigBee stack.
7. Creat application task and application message queue.

```
int32_t main(void)
{
    /*we should set pinmux here or in SystemInit */
    init_default_pin_mux();
    sys_set_random_seed(get_random_number());
    task_pci_init();
    zigbee_app_init();
    /* Start the scheduler. */
    vTaskStartScheduler();
    while (1) {
    }
}
```

```
void zigbee_app_init(void)
{
    /* Initil LED, Button, Console or UART */
    bsp_init(BSP_INIT_DEBUG_CONSOLE, NULL);
    bsp_init((BSP_INIT_LEDS | BSP_INIT_BUTTONS), app_bsp_event_handle);

    /* Retarget stdout for utility & initial utility logging */
    utility_register_stdout(bsp_console_stdout_char,
                            bsp_console_stdout_string);
    util_log_init();

    util_log_on(UTIL_LOG_PROTOCOL);

    gt_app_cfg.p_zigbee_device_contex_t = &simple_desc_switch_ctx;
    gt_app_cfg.pf_evt_indication = app_evt_indication_cb;

    info_color(LOG_BLUE, "Initial ZigBee stack\n");
    zigbee_stack_init(&gt_app_cfg);

    sys_queue_new(&app_msg_q, 16, sizeof(app_queue_t));

    info("Create app task\n");
    sys_task_new("app", app_main_task, NULL, 128, TASK_PRIORITY_APP);
}
```

3.3 Main loop

Implement ZigBee main loop “app_main_task ()” in zigbee_app.c file which is running the main application and waiting event message from zigbee stasck.

Process application event status in “app_main_loop()” in zigbee_app.c files which is controling the application flow.

```
staticvoid app_main_task(void *arg)
{
    app_queue_t t_app_q;
    app_event_state = APP_INIT_EVT;
    for(;;)
    {
        app_main_loop(app_event_state);
        if(sys_queue_recv(&app_msg_q, &t_app_q, 20) != SYS_ARCH_TIMEOUT)
        {
            if(t_app_q.event == APP_QUEUE_ISR_BUTTON_EVT)
            {
                if(t_app_q.pin == BSP_EVENT_BUTTONS_0)
                    send_toggle();
                if(t_app_q.pin == BSP_EVENT_BUTTONS_1)
                    send_level_step(0);
                if(t_app_q.pin == BSP_EVENT_BUTTONS_2)
                    send_level_step(1);
                if(t_app_q.pin == BSP_EVENT_BUTTONS_3)
                    send_move_color();
            }
            elseif(t_app_q.event == APP_QUEUE_ZIGBEE_EVT)
            {
                switch (t_app_q.pt_tlv->type)
                {
                case ZIGBEE_EVT_TYPE_ZCL_DATA_IDC:
                    zigbee_zcl_msg_handler(t_app_q.pt_tlv);
                    break;
                default:
                    zigbee_evt_handler(t_app_q.pt_tlv);
                    break;
                }
                if(t_app_q.pt_tlv)
                    sys_free(t_app_q.pt_tlv);
            }
        }
    }
}
```

```
staticvoid app_main_loop(uint32_t event)
{
    switch (event)
    {
    caseAPP_INIT_EVT:
        if(zigbee_ed_nwk_start_request(ZIGBEE_CHANNEL_ALL_MASK(), false, 3000,
                                       !bsp_button_state_get(BSP_BUTTON_0))
                                       == 0)
            app_event_state = APP_IDLE_EVT;
        break;
    caseAPP_NOT_JOINED_EVT:
        zigbee_join_request();
        app_event_state = APP_IDLE_EVT;
        break;
    default:
        break;
    }
}
```

3.4 ZigBee Event Handle

Implement ZigBee event handler “zigbee_evt_handler ()” in zigbee_evt_handler.c file which is processing the zigbee stack event.(ex: network start, device announce)

```
staticvoid _zdo_evt_start(sys_tlv_t *pt_tlv);
staticvoid(*zdo_evt_idc_func_list[])(sys_tlv_t *) = {
    [ZIGBEE_EVT_TYPE_START_IDC - ZIGBEE_EVT_TYPE_START_IDC] = _zdo_evt_start,
    [ZIGBEE_EVT_TYPE_DEVICE_ANNCE_IDC - ZIGBEE_EVT_TYPE_START_IDC] = NULL,
    [ZIGBEE_EVT_TYPE_LEAVE_IDC - ZIGBEE_EVT_TYPE_START_IDC] = NULL,
    [ZIGBEE_EVT_TYPE_DEVICE_ASSOCIATED_IDC - ZIGBEE_EVT_TYPE_START_IDC] =
NULL,
    [ZIGBEE_EVT_TYPE_PANID_CONFLICT_IDC - ZIGBEE_EVT_TYPE_START_IDC] = NULL,
};

staticvoid _zdo_evt_start(sys_tlv_t *pt_tlv)
{
    zigbee_nwk_start_idc_t *pt_start_idc =
(zigbee_nwk_start_idc_t*)pt_tlv->value;
    if(pt_start_idc->status != 0)
    {
        info_color(LOG_RED, "Device do rejoin\n");
        zigbee_app_evt_change(APP_NOT_JOINED_EVT);
    }
    else
    {
        info_color(LOG_GREEN, "Device join success\n");
        info_color(LOG_GREEN, "PAN: %04X, ShortAddr: %04X,
               MAC: %02X:%02X:%02X:%02X:%02X:%02X:%02X:%02X\n",
               pt_start_idc->panID, pt_start_idc->nwkAddr,
               pt_start_idc->ieee_addr[7], pt_start_idc->ieee_addr[6],
               pt_start_idc->ieee_addr[5], pt_start_idc->ieee_addr[4],
               pt_start_idc->ieee_addr[3], pt_start_idc->ieee_addr[2],
               pt_start_idc->ieee_addr[1], pt_start_idc->ieee_addr[0]);
    }
}
void zigbee_evt_handler(sys_tlv_t *pt_tlv)
{
    if((pt_tlv->type >= ZIGBEE_EVT_TYPE_ZDO_START_IDC) &&
       (pt_tlv->type <= ZIGBEE_EVT_TYPE_ZDO_FINISH_IDC))
    {
        if(zdo_evt_idc_func_list[pt_tlv->type - ZIGBEE_EVT_TYPE_START_IDC])
            zdo_evt_idc_func_list[pt_tlv->type - 
ZIGBEE_EVT_TYPE_START_IDC](pt_tlv);
    }
}
```

3.5 ZCL Message Handle

Implement ZigBee ZCLmessage handler “zigbee_zcl_msg_handler ()” in zigbee_zcl_msg_handler.c file which is processing the ZCL messages.

```
void zigbee_zcl_msg_handler(sys_tlv_t *pt_tlv)
{
    zigbee_zcl_data_idc_t *pt_zcl_msg = (zigbee_zcl_data_idc_t 
*)pt_tlv->value;
    do
    {
        if(!pt_zcl_msg)
            break;
        info("Recv ZCL message\n");
        info("Cluster %04x, cmd %d\n", pt_zcl_msg->clusterID, pt_zcl_msg->cmd);
        util_log_mem(UTIL_LOG_INFO, "  ", (uint8_t *)pt_zcl_msg->cmdFormat, 
pt_zcl_msg->cmdFormatLen, 0);
    } while(0);
}
```

3.6 Event Message Indication

Implement callback function of event message indication “app_evt_indication_cb()”in zigbee_app.c file which is sending event message to application task.

```
staticvoid app_evt_indication_cb(uint32_t data_len)
{
    int i32_err;
    uint8_t *pBuf = sys_malloc(data_len);
    app_queue_t t_app_q;
    do
    {
        if(!pBuf)
            break;
        t_app_q.event = 0;
        i32_err = zigbee_event_msg_recvfrom(pBuf, &data_len);
        t_app_q.pt_tlv = (sys_tlv_t *)pBuf;
        if (i32_err == 0)
        {
            sys_queue_send_with_timeout(&app_msg_q, &t_app_q, 0);
        }
        else
        {
            info_color(LOG_RED, "[%s] sys_err = %d !\n", __func__, i32_err);
            sys_free(pBuf);
        }
    } while (0);
}
```

4. Getting Started

The RT58x ZigBee SDK provides tools and instructions that allow you to start developing your own applications and use the ZigBee features.

4.1 ZigBee Demonstration

This is a quick demonstration of some basic concepts of the ZigBee network using Rafael's RT58x ZigBee SDK .

- Gateway_Module:This application is a Zigbee Coordinator device as the zigbee gateway module control by the Host MCU with specified Rafael Gateway Command or the user can simply use the CLI command for basic functions interact with other devices(e.g. Light, Switch, Custom_cluster).
- Light: This application is a Zigbee Router device for a light device.
- Switch: This application is a Zigbee End-device for a switch device.
- Custom_cluster: This application is a Zigbee Router device as an example for Custom Cluster.
- Wall_switch: This application is a Zigbee Router device of a wall-switch device.
- Thermostat: This application is a Zigbee Router device of a Thermostat device.
- Door_Sensor/Illuminance_Sensor/PIR_Sensor/Power_Meter/Temperature_Sensor: These applications are Zigbee end device of the corresponding sensor device.

Supported cluster of each application:

Application	Input Cluster	Output Cluster

Gateway	Basic(0x0000)
	Identify(0x0003)
	OTA Upgrade(0x0019)
	Custom cluster(0x1A0A)
		Basic(0x0000)
		Power Config(0x0001)
		Identify(0x0003)
		Groups(0x0004)
		Scenes(0x0005)
		Onoff(0x0006)
		Level Control(0x0008)
		Illuminance Measurement(0x0400)
		Temperature Measurement(0x0402)
		Relative humidity Measurement (0x0405)
		IAS Zone (0x0500)
		Metering(0x0702)
		Electrical Measurement(0x0b04)
		Custom cluster(0x1A0A)

Light	Basic(0x0000)
	Identify(0x0003)
	Groups(0x0004)
	Scenes(0x0005)
	Onoff(0x0006)
		OTA Upgrade(0x0019)

	Level Control(0x0008)	

Switch	Basic(0x0000)
	Identify(0x0003)
	Groups(0x0004)
	Scenes(0x0005)
	Onoff(0x0006)
		Onoff(0x0006)
		Identify(0x0003)
		OTA Upgrade(0x0019)

Wall switch	Basic(0x0000)
	Identify(0x0003)
	Groups(0x0004)
	Scenes(0x0005)
	Onoff(0x0006)
	Level Control(0x0008)
		Identify(0x0003)
		Onoff(0x0006)
		Level Control(0x0008)

Thermostat	Basic(0x0000)
	Identify(0x0003)
	Groups(0x0004)
	Scenes(0x0005)
	Thermostat(0x0201)	

Door sensor	Basic(0x0000)
	Power Config(0x0001)
	Identify(0x0003)
	Groups(0x0004)
	IAS Zone (0x0500)
		Identify(0x0003)

Illuminance sensor	Basic(0x0000)
	Power Config(0x0001)
	Identify(0x0003)
	Groups(0x0004)
	Illuminance Measurement(0x0400)
		Identify(0x0003)

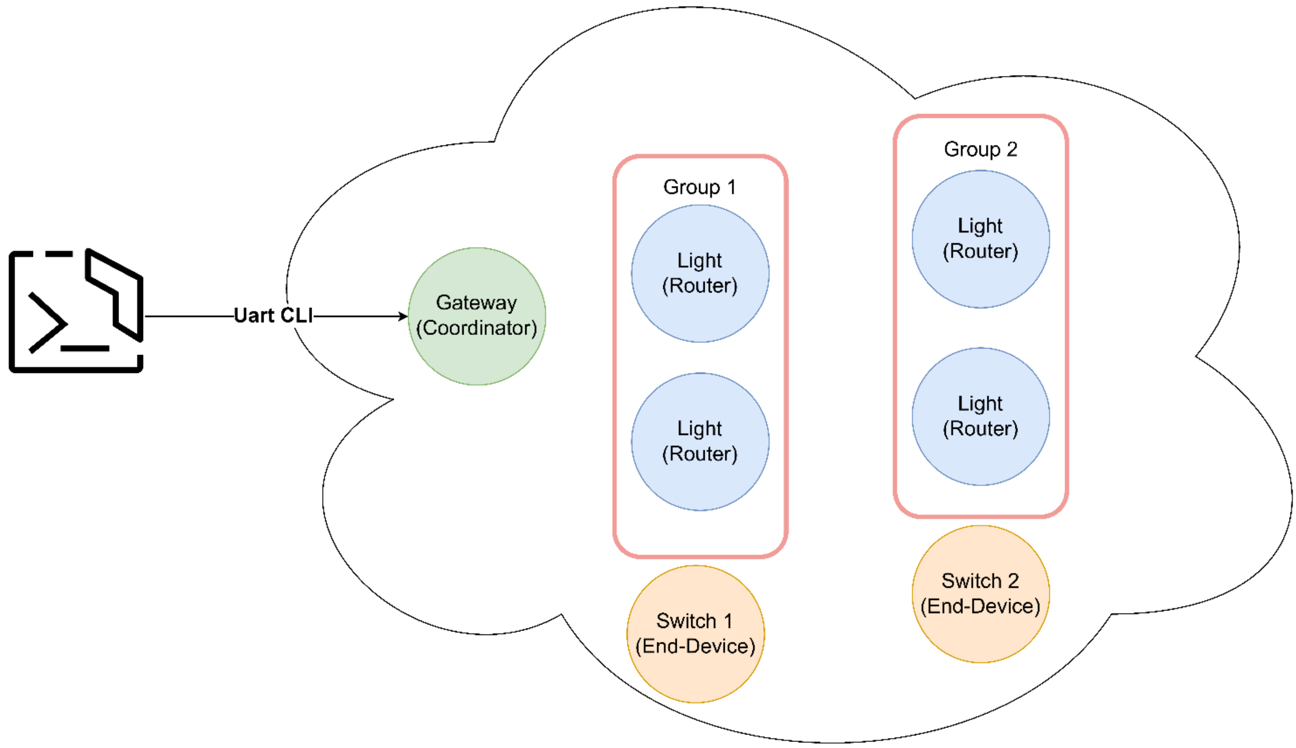
PIR sensor	Basic(0x0000)
	Power Config(0x0001)
	Identify(0x0003)
	Groups(0x0004)
	IAS Zone (0x0500)
		Identify(0x0003)

Power meter	Basic(0x0000)
	Identify(0x0003)
	Groups(0x0004)
	Onoff(0x0006)
	Metering(0x0702)
	Electrical Measurement(0x0b04)
		Identify(0x0003)

Temperature sensor	Basic(0x0000)
	Power Config(0x0001)
	Identify(0x0003)
	Groups(0x0004)
	Temperature Measurement(0x0402)
	Relative humidity Measurement (0x0405)
		Identify(0x0003)

Custom cluster	Custom cluster(0x1A0A)	Custom cluster(0x1A0A)

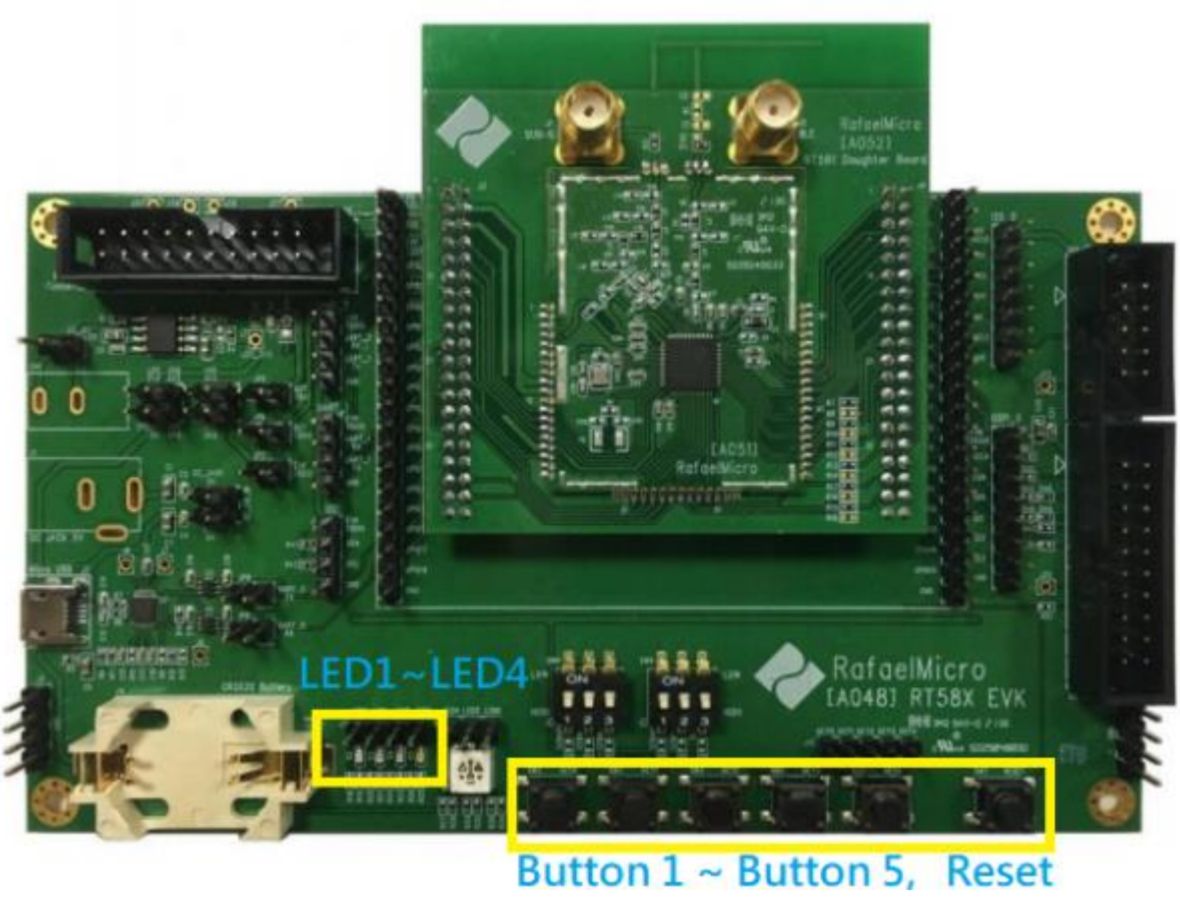
The following figure gives the overall view of the zigbee networking that will be set up in this example.



ZigBee network demonstration

4.1.1 Hardware

RT58x EVK board:



4.1.2 LED and button assignments

The buttons are used to initiate certain actions, and the LEDs (1 to 5) are used to reflect the status of action as follows:

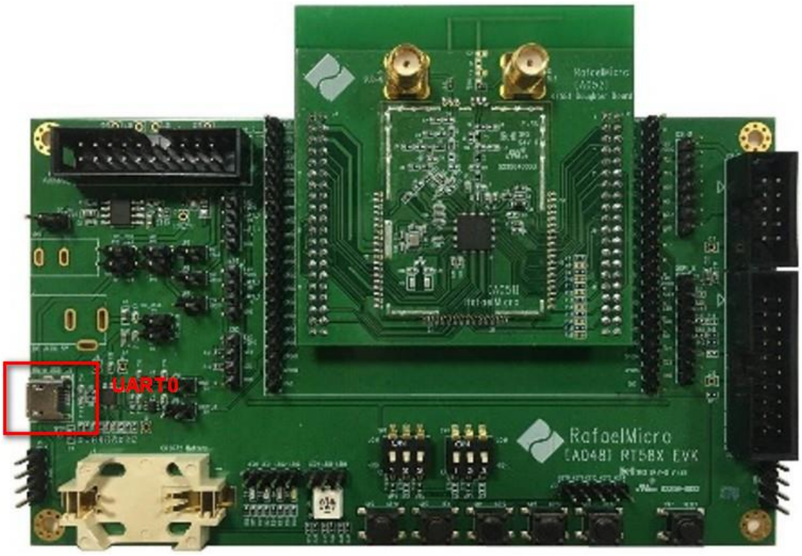
- During scan & join process:
  - LED2 blinking: Device identification active.
  - LED2 Off: Device identification success.
- After joining and configuration is over:
  - LED1: Reflects the value on OnOff state on the device
    - LED On: Value of the OnOff state is 1.
    - LED Off: Value of the OnOff state is 0.
    - LED Brightness: Value of the current level.
- Button behavior of light device:
  - Button4: enable network steering
  - Button5: start finding and binding process as target role
- Button behavior of switch and wall-switch device:

- Button1: Send ZCL OnOff toggle message to binding group.
- Button2&3(wall-switch): Send ZCL Light Level control step message to binding group
- Button3(switch): Broadcast network leave and reset device's network
- Button4(switch): enable network steering
- Button5(switch): start finding and binding process as initiator role

4.1.3 How to reset to factory new

- Press reset button for 5 times with each interval less than 0.5s.
  - For end devices, it would leave the network silently
  - For other device, it would broadcast network leave command then reset its network
- For switch device, press button 3 to broadcast network leave command

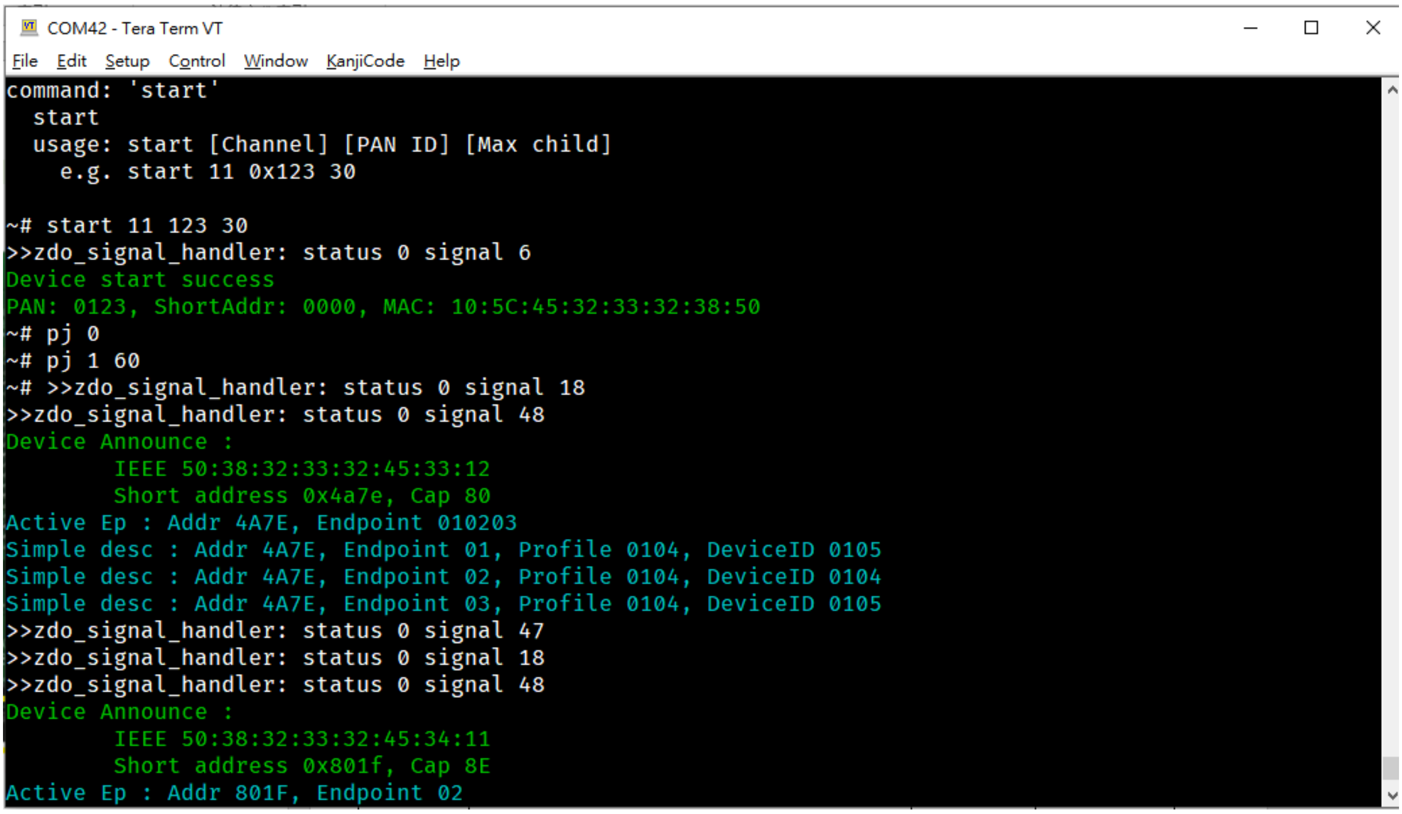
4.1.4 Running the demo with debug console(UART0)



If you want to see the debug logs printed during the provisioning and configuration process, complete the following steps:

- Connect the RT58x EVK boards to the USB ports.
- Start the serial tools (Putty, Tera Term)
- Config the baud rate to 115200, none, 1.

The following figure shows the logs printed, you also can enter the CLI command to control gateway_module and show debug messages.



```
command: 'start'
  start
  usage: start [Channel] [PAN ID] [Max child]
    e.g. start 11 0x123 30

~# start 11 123 30
>>zdo_signal_handler: status 0 signal 6
Device start success
PAN: 0123, ShortAddr: 0000, MAC: 10:5C:45:32:33:32:38:50
~# pj 0
~# pj 1 60
~# >>zdo_signal_handler: status 0 signal 18
>>zdo_signal_handler: status 0 signal 48
Device Announce :
        IEEE 50:38:32:33:32:45:33:12
        Short address 0x4a7e, Cap 80
Active Ep : Addr 4A7E, Endpoint 010203
Simple desc : Addr 4A7E, Endpoint 01, Profile 0104, DeviceID 0105
Simple desc : Addr 4A7E, Endpoint 02, Profile 0104, DeviceID 0104
Simple desc : Addr 4A7E, Endpoint 03, Profile 0104, DeviceID 0105
>>zdo_signal_handler: status 0 signal 47
>>zdo_signal_handler: status 0 signal 18
>>zdo_signal_handler: status 0 signal 48
Device Announce :
        IEEE 50:38:32:33:32:45:34:11
        Short address 0x801f, Cap 8E
Active Ep : Addr 801F, Endpoint 02
```

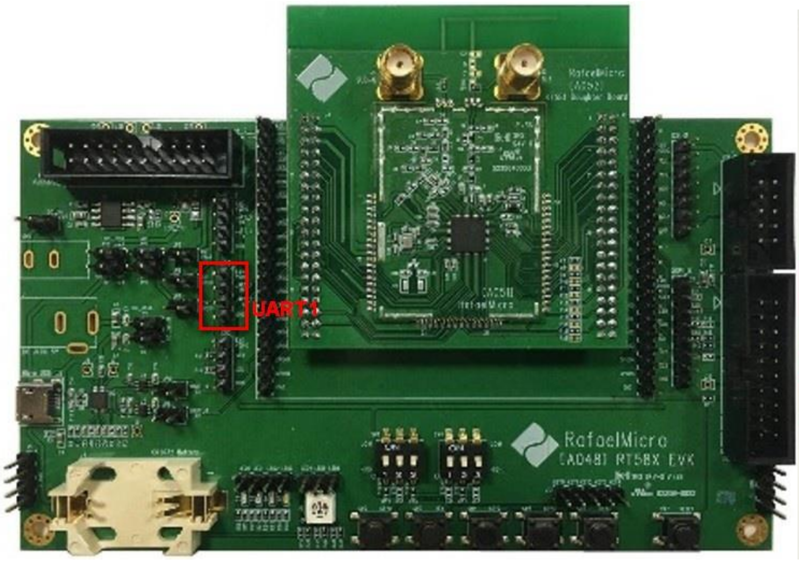
4.1.5 Gateway_moduleCLI Commands

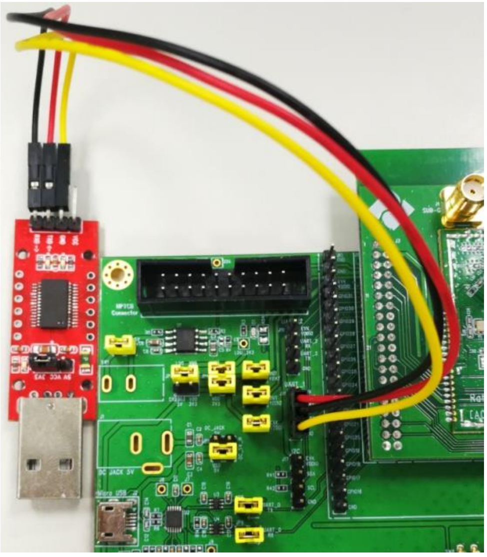
Command	Usage	example	Description
help	help	help	List commands
dw	dw[address] [length]	dw 0x20000000 32	Read memory data
ps	ps	ps	Show task counts
mem	mem	mem	Show the memory status
edscan	edscan	edscan	Energy detect scanning Must run before network started.
start	start [reset] [Channel] [PAN ID] [Max child]	start 1 11 0x123 30	Create and start a PAN.
pj	pj[enable] [timeout]	Enable : pj 1 60 Disabel : pj 0	Permit join
s2e	s2e	s2e	Show device table
ep	ep [address]	ep 0x0001	Get active endpoint
simple	simple[addr] [ep]	simple 0x0001 2	Get simple description
ra	ra [addr] [ep] [cluster id] [attr id]	ra 0x123 0x0006 0x0000	Read target’s attribute

wa	wa [addr][ep][cluster id][attr id][attr type][attr value]	wa 0x123 0x02 0x0003 0x0000 0x21 10	Write target’s attribute
cr	cr [addr] [ep] [cluster id] [attribute id] [data type] [min_interval][max interval]	cr 0x1234 2 0x0006 0x0000 0x10 0x1E 0x3D	Configure target’s reporting interval of specific attribute
ic	ic [IEEE addr 7 ~ 4 byte] [IEEE addr 3 ~ 0 byte] [install code with CRC]	ic 352E4532 33323850 85 FE D3 40 7A 93 97 23 a5 c6 39 b2 69 16 d5 05 90 89	Add install code
bind	bind [address mode] [src addr] [srcep] [dstaddr] [dst ep][cluster_id]	bind 0x1 0x1234 0x2 0x0000 0x2 0x0006	Request target’s endpoint to bind a group addreess.
unbind	unbind [address mode] [src addr] [srcep] [dstaddr] [dst ep][cluster_id]	unbind 0x1 0x1234 0x2 0x0000 0x2 0x0006	Request target’s endpoint to unbind a group addreess.
group	group [action] [addr][ep] [group id]	Add : group a 0x123 1 0x0001 Remove : group r 0x123 1 0x0001	Add/Remove a gruoup address
onoff	Onoff [addr] [ep] [cmd]	onfoff 0x0001 1 2	Send ZCLOnOff message to target.
scene	scene [action] [addr][ep] [groupid] [scene id]	Store: scene s 0x123 2 0x0001 1 Remove: scene re 0x123 2 0x0001 1 Recall: scene rc 0x123 2 0x0001 1 View: scene v 0x123 2 0x0001 1	Send ZCL scene message
level	level [dir] [addr] [ep]	level up 0x0001 2 level dn 0x0001 2	Send level control command to target
identify	id [addr] [ep]	id 0x0001 2	Send identify command to target
setpt	setpt [short address] [ep] [mode] [amount]	setpt 0x1234 2 0 30 setpt 0x1234 2 0 -20	adjust heat/cool setpoint by amount
ct	ct[short address] [end	ct 0x1234 02	Set the target address &

	point]		endpoint to send the UART transparent data with custom cluster
cs	cs [value]	cs 0xaa	Send string “0xaa” to the target address with the ct cli command

4.1.6 Gateway_module UART command(UART1)



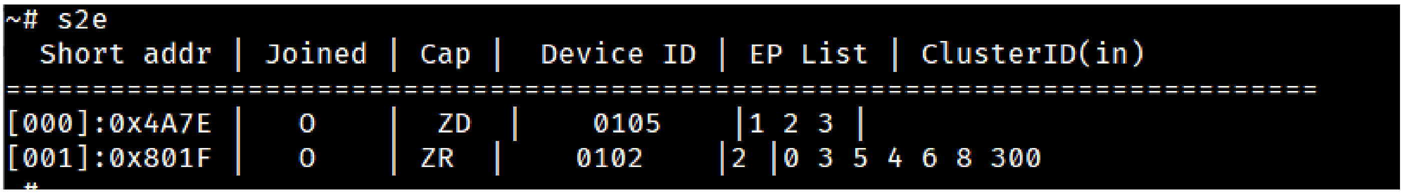


Besides the CLI command, the Gateway_module project also provide the UART command set (Hex format) from UART1, with these UART command set, the Gateway_module can work with a host MCU. The host can use the command set to control the Gateway_module to start a zigbee network and manage the zigbee network. For the details, please refer to the document, [SW_17]Gateway Command.pdf .

The user can use the UART to USB dongle and connect to EVK board UART1 port to send the hex format command with an appropriate PC application tool. The document [SW_16]RT58x_Zigbee_Gateway_Module_Porting_Guide_V0.1.pdf demonstrate how to control & get the response with the Gateway_module.

4.1.7 Control the light device

After light and switch join success. Use command “s2e” to show the device table to check address and device type.

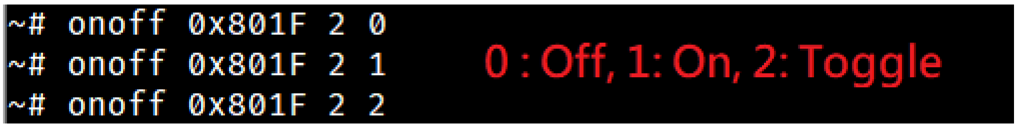


```
~# s2e
  Short addr | Joined | Cap |  Device ID | EP List | ClusterID(in)
=================================================================================
[000]:0x4A7E |   O    |  ZD |    0105    |1 2 3 |
[001]:0x801F |   O    | ZR |    0102    |2 |0 3 5 4 6 8 300
```

In this case, switch address is 0x4A7E and light address is 0x801F

4.1.7.1 Control light by gateway CLI command

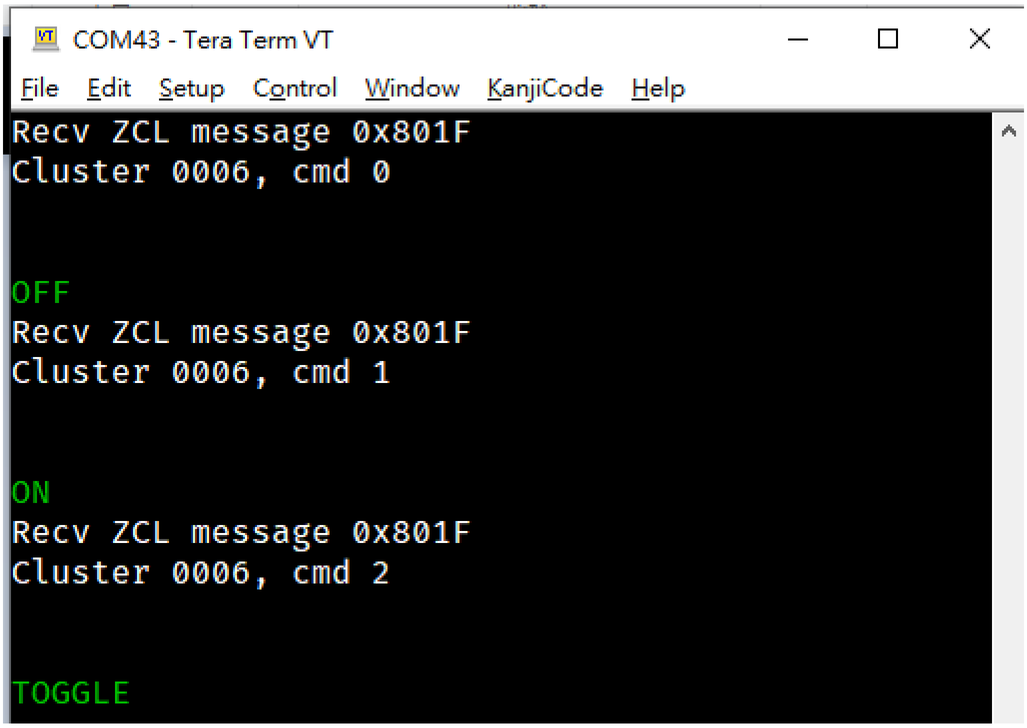
✧ Use command “onoff” to control light device.



```
~# onoff 0x801F 2 0
~# onoff 0x801F 2 1
~# onoff 0x801F 2 2
```

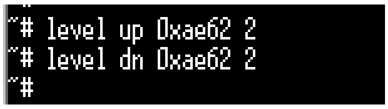
0 : Off, 1: On, 2: Toggle

OnOff status will show on light’s debug console.



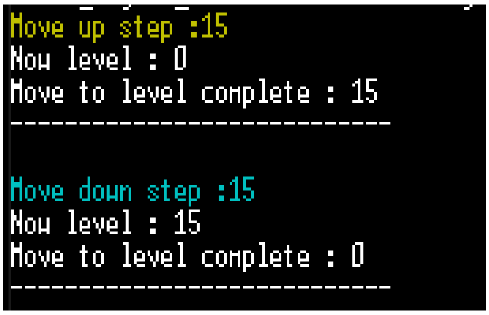
```
Recv ZCL message 0x801F
Cluster 0006, cmd 0


OFF
Recv ZCL message 0x801F
Cluster 0006, cmd 1


ON
Recv ZCL message 0x801F
Cluster 0006, cmd 2


TOGGLE
```

✧ Use level command to control light device



```
~# level up 0xae62 2
~# level dn 0xae62 2
~#
```

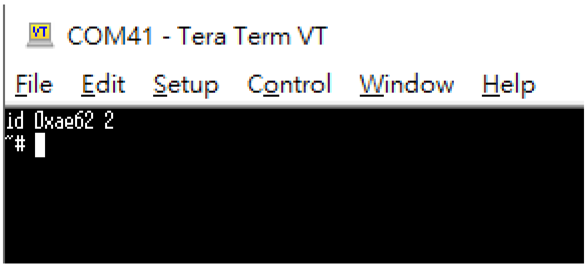
Level status will show at light’s debug console



```
Move up step :15
Now level : 0
Move to level complete : 15
----------------------------

Move down step :15
Now level : 15
Move to level complete : 0
----------------------------
```

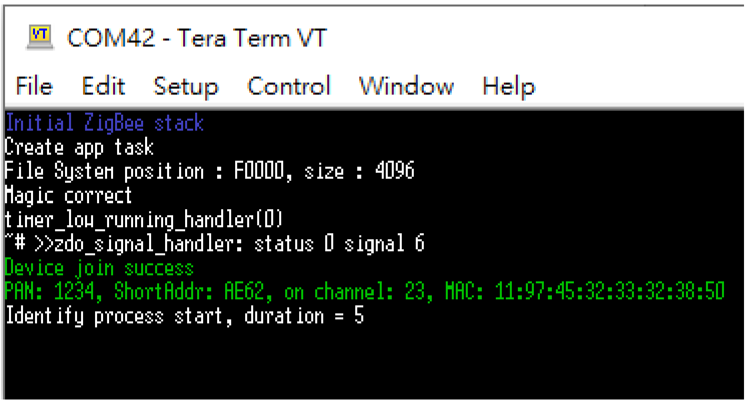
Brightness of LED1 also change with level and on/off state

✧ Use cli command to identify the device



```
id 0xae62 2
~#
```

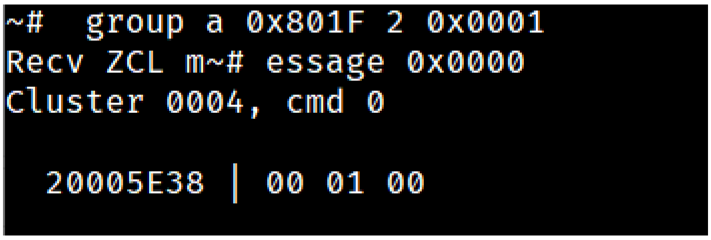
The message will show at light’s debug console and the LED1 will start blinking for 5 second



```
Initial ZigBee stack
Create app task
File System position : F0000, size : 4096
Magic correct
timer_low_running_handler(0)
~# >>zdo_signal_handler: status 0 signal 6
Device join success
PAN: 1234, ShortAddr: AE62, on channel: 23, MAC: 11:97:45:32:33:32:38:50
Identify process start, duration = 5
```

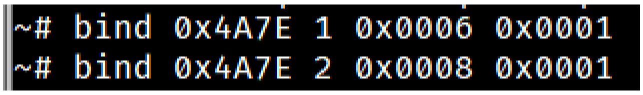
4.1.7.2 Control light by switch device

➢ Step1 : Add light device to a group address(0x0001).



```
~#  group a 0x801F 2 0x0001
Recv ZCL m~# essage 0x0000
Cluster 0004, cmd 0

  20005E38 | 00 01 00
```

➢ Step2 : Request switch’s endpoint to bind a group address(0x0001).



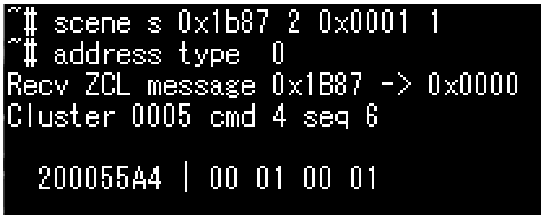
```
~# bind 0x4A7E 1 0x0006 0x0001
~# bind 0x4A7E 2 0x0008 0x0001
```

Now can control the light device by switch device.

4.1.7.3 Store scene by Gateway CLI command

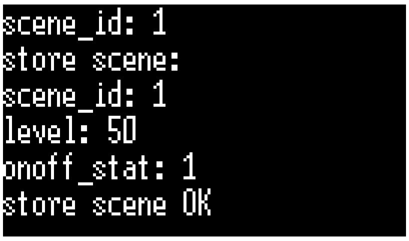
Use “scene s” command at gateway to store current level and on/off state to

specific group id and scene id.



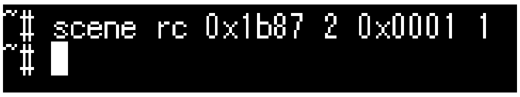
```
~# scene s 0x1b87 2 0x0001 1
~# address type  0
Recv ZCL message 0x1B87 -> 0x0000
Cluster 0005 cmd 4 seq 6

  200055A4 | 00 01 00 01
```

The status will show at light’s debug console



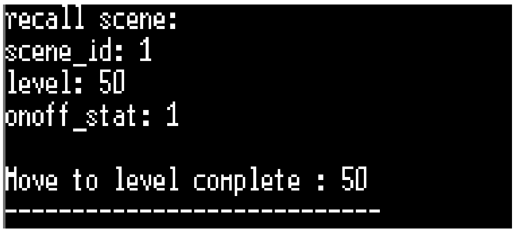
```
scene_id: 1
store scene:
scene_id: 1
level: 50
onoff_stat: 1
store scene OK
```

The stored scene can be recalled by “scene rc” command.



```
~# scene rc 0x1b87 2 0x0001 1
~# █
```

Recalled scene information will show at light’s console.



```
recall scene:
scene_id: 1
level: 50
onoff_stat: 1

Move to level complete : 50
----------------------------
```

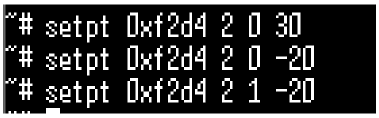
The brightness of LED1will change to the stored value

4.1.8 Control the Wall-switch

The control of wall-switch is the same as light device but it can work as a switch, which is able to send level control command using endpoint 2.

4.1.9 Control the Thermostat

Send ‘setpt’ command to the thermostat device at gateway side

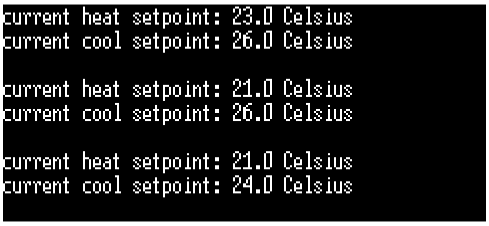


```
~# setpt 0xf2d4 2 0 30
~# setpt 0xf2d4 2 0 -20
~# setpt 0xf2d4 2 1 -20
```

- Mode 0=control the heat set-point
- Mode 1=control the cool set-point

○ Mode 2= control both set-point

Current value of both set-point will show at thermostat’s terminal



```
current heat setpoint: 23.0 Celsius
current cool setpoint: 26.0 Celsius

current heat setpoint: 21.0 Celsius
current cool setpoint: 26.0 Celsius

current heat setpoint: 21.0 Celsius
current cool setpoint: 24.0 Celsius
```

4.1.10 Sensor devices

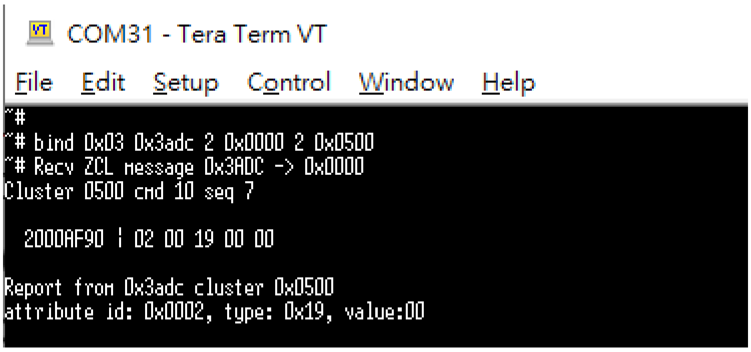
Door_Sensor, Illuminance_Sensor, PIR_Sensor, Power_Meter, Temperature_Sensor are sensor device for different clusters as shown below:

1. Door_Sensor: Power configuration, IAS Zone
2. Illuminance_Sensor: Power configuration, IAS Zone, Illuminance measurement
3. PIR_Sensor: Power configuration, IAS Zone
4. Power_Meter: Metering, Electrical measurement
5. Temperature_Sensor: Power configuration, Temperature measurement, Relative humidity

At gateway side, send following binding command to enable report attribute for specific cluster

bind 0x03 [sensor’s addr] [sensor ep] [gateway’s addr] [gateway’s ep] [cluster ID]

By default, sensor device would report its reportable attributes to gateway, which will show at gateway’s terminal.



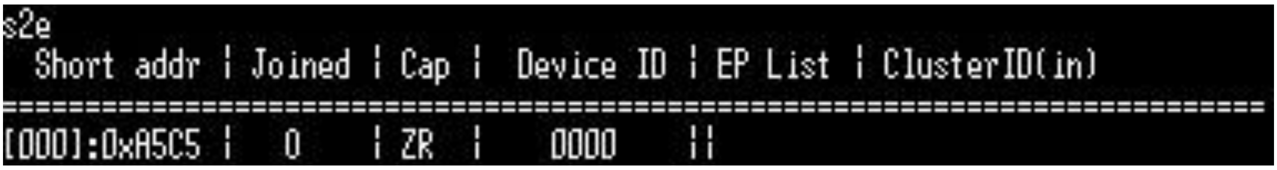
```
~#
~# bind 0x03 0x3adc 2 0x0000 2 0x0500
~# Recv ZCL message 0x3ADC -> 0x0000
Cluster 0500 cmd 10 seq 7

  2000AF90 | 02 00 19 00 00

Report from 0x3adc cluster 0x0500
attribute id: 0x0002, type: 0x19, value:00
```

Using following configure report command to change report interval:

cr [sensor’s addr] [sensor’s ep] [cluster id] [attribute id] [data type] [min_interval] [max interval]

4.1.11 UART Transparent Demo(using Custom Cluster)

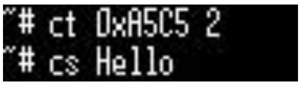
After custom cluster device joins success. Use command “s2e” to show the device table to check address and device type.



```
s2e
 Short addr ¦ Joined ¦ Cap ¦  Device ID ¦ EP List ¦ ClusterID(in)
==================================================================
[000]:0xA5C5 ¦   0    ¦ ZR  ¦    0000    ¦¦
```

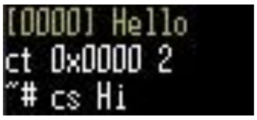
In this case, custom cluster device address is 0xA5C5.

Step1: Set target device address (0xA5C5) and endpoint (2) using “ct 0xA5C5 2” command.
Step2: Send message to the target device using “cs” command. For example:



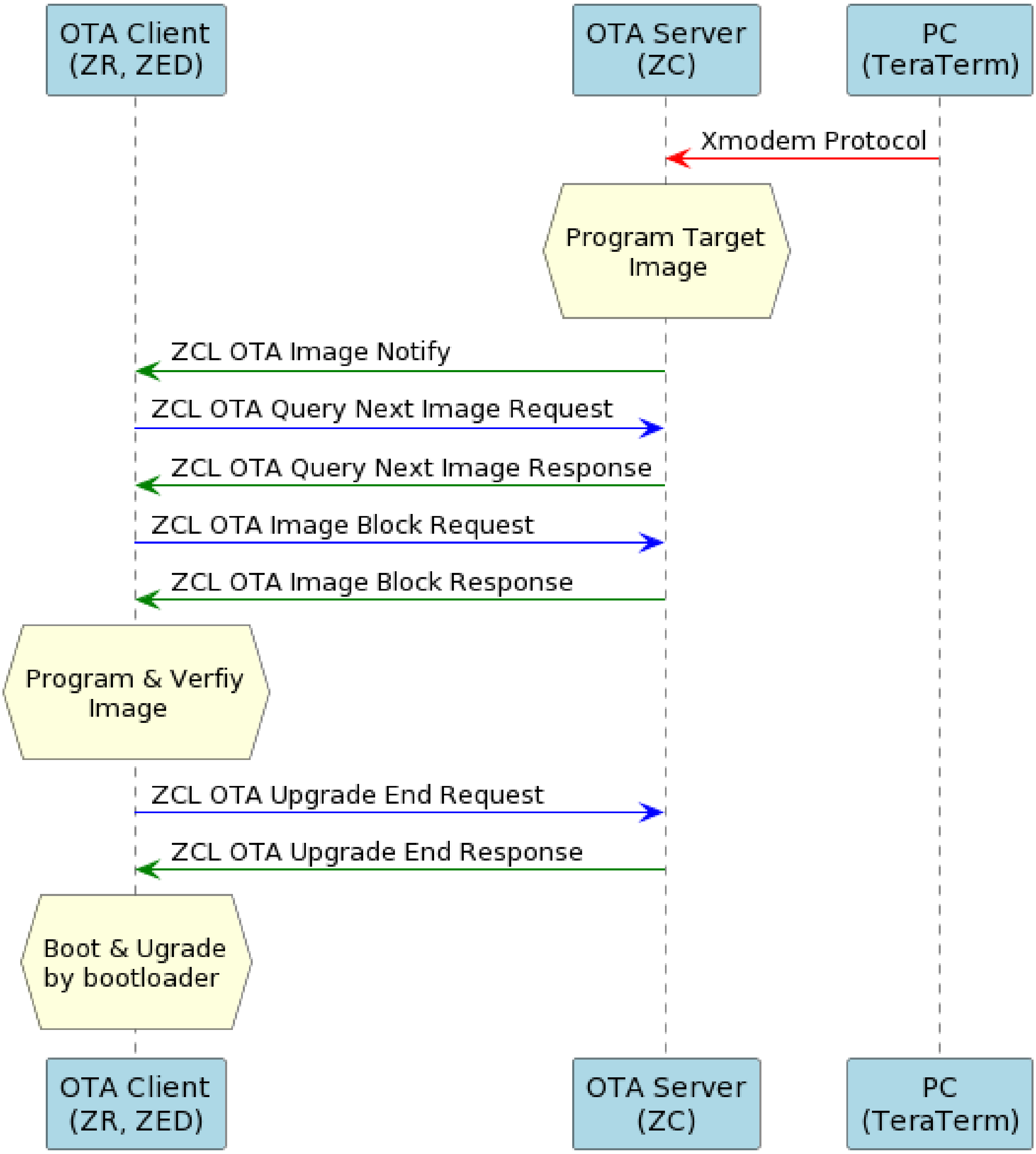
```
~# ct 0xA5C5 2
~# cs Hello
```

Then, the custom cluster device will show the received message on its console, and you can send some message back to gateway using the same method. For example:



```
[0000] Hello
ct 0x0000 2
~# cs Hi
```

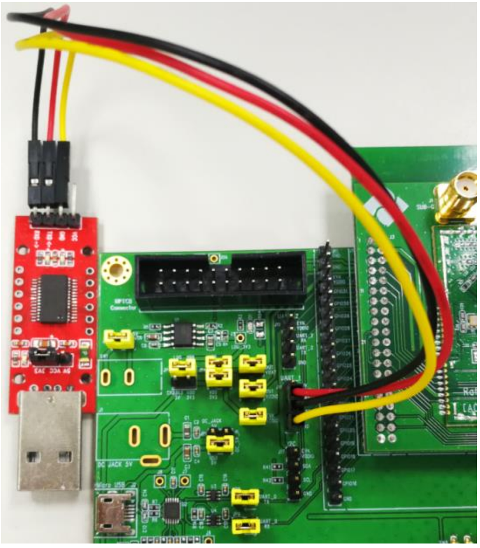
4.1.12 OTA


OTA Client (ZR, ZED)
OTA Server (ZC)
PC (TeraTerm)
Xmodem Protocol
Program Target Image
ZCL OTA Image Notify
ZCL OTA Query Next Image Request
ZCL OTA Query Next Image Response
ZCL OTA Image Block Request
ZCL OTA Image Block Response
Program & Verfiy Image
ZCL OTA Upgrade End Request
ZCL OTA Upgrade End Response
Boot & Ugrade by bootloader
OTA Client (ZR, ZED)
OTA Server (ZC)
PC (TeraTerm)

We can use the zigbee OTA(Over The Air) cluster to update the target device's firmware over the zigbee network. Please find the project "Gateway_Module" in the SDK example projects to be the OTA server and the Light or Switch project act as the OTA client. These projects demonstrate how the OTA firmware update works. In this section, the OTA server can provide the updated image to the target device. And the OTA client will use the OTA cluster to find the OTA server and if there is any image can be updated.

Setup on Gateway side(using light project for example):

1. Prepare the Light.bin file to update the OTA target device.
   The Light.bin file can be found in project folder after firmware build.
2. The "Gateway_Module" used the UART port(UART1) to download the .bin file into the OTA server storage space with the tool in the SDK folder .\Tools\Zigbee\ota_download_tool.
   a. Please use e.g. UART to USB dongle connect to EVK board UART_1 port.



   b. Connect the COM port and copy the target image(i.e. Light.bin) to the "ota_download_tool" folder.
   c. Edit and run the "start.bat" file in the folder to send the image to the OTA server via UART

create_image.exe Light.bin Light.ota v2222 0x007B 0x0141

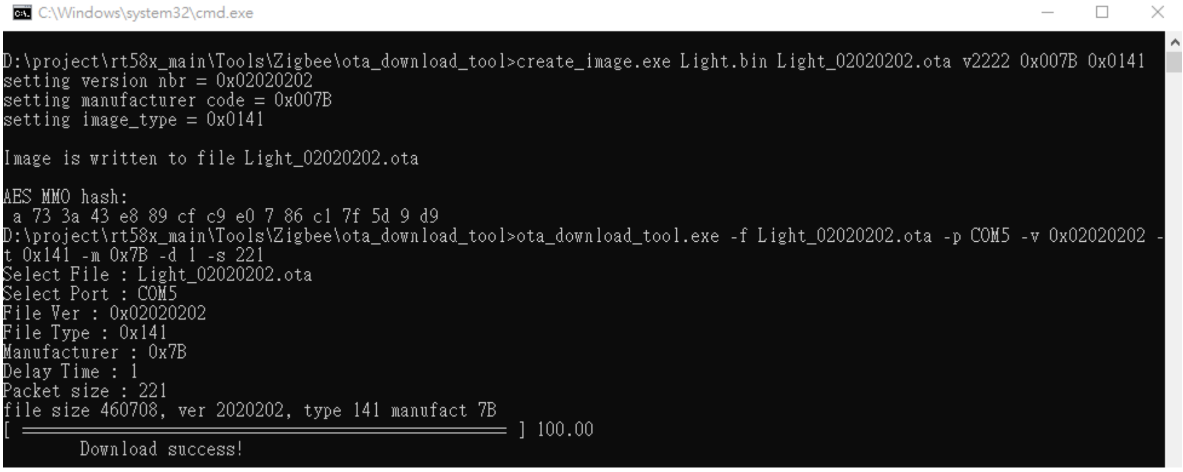
this would create a .ota file with specific file name, version, manufacture ID, image type

ota_download_tool.exe -f Light.ota -p COM5 -v 0x02020202 -t 0x141 -m 0x7B -d 1 -s 221

where
–f Light.ota is the assigned image file name
-p COM5 is com port number(COM5) on the computer

-v 0x02020202 is the image version(0x02020202)
-t 0x141is the image type, 0x141 is the default value
-m 0x7Bis the manufacture ID (0x7B)
-d 1 is the command delay time, default set to 1(ms)
-s 221 is the packet size, default set to 221

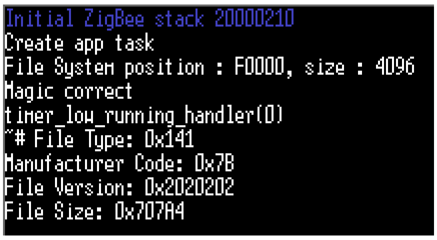
d. After running the file in step c, it will download the image into the OTA server automatically.



C:\Windows\system32\cmd.exe

```
D:\project\rt58x_main\Tools\Zigbee\ota_download_tool>create_image.exe Light.bin Light_02020202.ota v2222 0x007B 0x0141
setting version nbr = 0x02020202
setting manufacturer code = 0x007B
setting image_type = 0x0141

Image is written to file Light_02020202.ota

AES MMO hash:
 a 73 3a 43 e8 89 cf c9 e0 7 86 c1 7f 5d 9 d9
D:\project\rt58x_main\Tools\Zigbee\ota_download_tool>ota_download_tool.exe -f Light_02020202.ota -p COM5 -v 0x02020202 -
t 0x141 -m 0x7B -d 1 -s 221
Select File : Light_02020202.ota
Select Port : COM5
File Ver : 0x02020202
File Type : 0x141
Manufacturer : 0x7B
Delay Time : 1
Packet size : 221
file size 460708, ver 2020202, type 141 manufact 7B
[ ================================================== ] 100.00
        Download success!
```

e. On the Gateway CLI terminal, it will show the following information to insert the related OTA file after the download processing is done in step d.

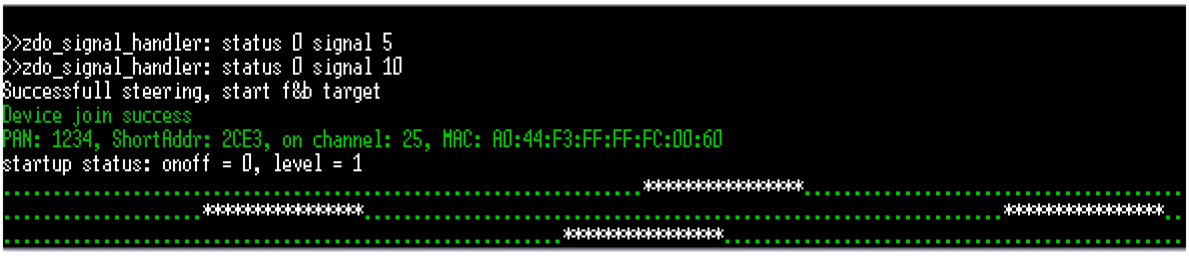


```
Initial ZigBee stack 20000210
Create app task
File System position : F0000, size : 4096
Magic correct
timer_low_running_handler(0)
~# File Type: 0x141
Manufacturer Code: 0x7B
File Version: 0x2020202
File Size: 0x707A4
```

After the image is load into the "Gateway_Module", we can start a zigbee network and let Light device join the network with following procedure:

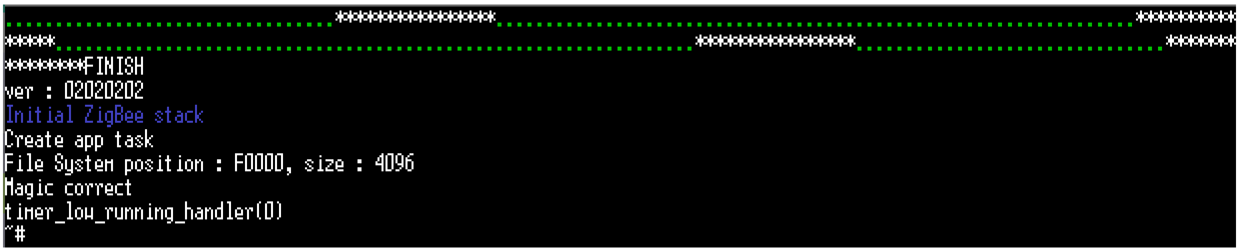
1. On the "Gateway_Module" side, use CLI command to start a zigbee network
   CLI command → start 1 18 0x1234 1
   (Start a zigbee network on channel 18 with PanID = 0x1234)
2. On the Light side, just reset & power on it, it will start the join procedure to join the zigbee network, please be sure that Light device is joined on the gateway's zigbee network(Check the "Gateway_Module" & "Light" terminal, it will show the join information after join procedure succeed)
3. On the Light side, it will send the Query Next Image Request packet to the OTA server every two minutes after join the network

4. If the image version is newer than the Light device firmware version, it will start to download the image block from Gateway to Light device over the air, the Light terminal will show as following during downloading



```
>>zdo_signal_handler: status 0 signal 5
>>zdo_signal_handler: status 0 signal 10
Successfull steering, start f&b target
Device join success
PAN: 1234, ShortAddr: 2CE3, on channel: 25, MAC: A0:44:F3:FF:FF:FC:00:60
startup status: onoff = 0, level = 1
...............................................................****************......................................
....................****************...............................................................****************..
........................................................****************.............................................
```

5. After successfully download, it will show following message then reboot to boot form downloaded image



```
.................................****************...............................................................**********
*****................................................................****************...............................*******
********FINISH
ver : 02020202
Initial ZigBee stack
Create app task
File System position : F0000, size : 4096
Magic correct
timer_low_running_handler(0)
^#
```

Revision History

Revision	Description	Owner	Date
1.0	Initial version	Rex	2022/01/21
1.1	Update OTA procedure	Justin	2022/03/30
1.2	Document update for SDK v1.1.0	Justin	2022/05/25
1.3	Update description for example projects	Justin	2022/07/28
1.4	Add cli commands and fix ota process	Randy	2022/09/22
1.5	Add sensor projects	Randy	2022/10/14
1.6	Add support cluster list of each application	Randy	2023/02/15

Information in this document is provided in connection with **Rafael Microelectronics, Inc.** (“**Rafael Micro**”) products. These materials are provided by **Rafael Micro** as a service to its customers and may be used for informational purposes only. **Rafael Micro** assumes no responsibility for errors or omissions in these materials. **Rafael Micro** may make changes to this document at any time, without notice. **Rafael Micro** advises all customers to ensure that they have the latest version of this document and to verify, before placing orders, that information being relied on is current and complete. **Rafael Micro** makes no commitment to update the information and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to its specifications and product descriptions.

THESE MATERIALS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, RELATING TO SALE AND/OR USE OF **RAFAEL MICRO** PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, CONSEQUENTIAL OR INCIDENTAL DAMAGES, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. **RAFAEL MICRO** FURTHER DOES NOT WARRANT THE ACCURACY OR COMPLETENESS OF THE INFORMATION, TEXT, GRAPHICS OR OTHER ITEMS CONTAINED WITHIN THESE MATERIALS. **RAFAEL MICRO** SHALL NOT BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUES OR LOST PROFITS, WHICH MAY RESULT FROM THE USE OF THESE MATERIALS.

Rafael Micro products are not intended for use in medical, lifesaving or life sustaining applications. **Rafael Micro** customers using or selling **Rafael Micro** products for use in such applications do so at their own risk and agree to fully indemnify **Rafael Micro** for any damages resulting from such improper use or sale. **Rafael Micro, logos** and **RT568** are **Trademarks** of **Rafael Microelectronics, Inc**. Product names or services listed in this publication are for identification purposes only, and may be trademarks of third parties. Third-party brands and names are the property of their respective owners.