LAB 2 REPORT
SETUP A BLE CONNECTION

Director: Dr. Ahmed Rafaey Hussein

Group Members: Mohammadamin Saburruhmonfared & Chaoao Shi

June 9, 2016

OBJECTIVES

1. Learn how to use the BLE Component
2. Implement a standard BLE Find Me Profile with the Immediate Alert Service (IAS)
3. Learn how to use the CySmart BLE Test and Debug Tool to debug BLE designs

PROCEDURE

1. Block diagram

Set up the project in PSoC Creator so that it follows the block diagram shown in Figure 1.

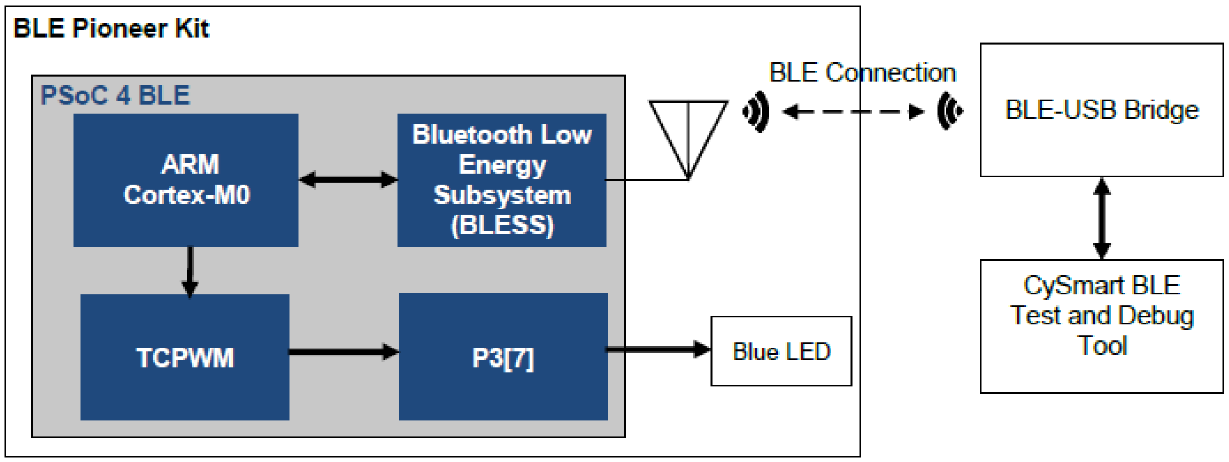


Figure 1: Block Diagram for Lab 2

2. Schematic

Open the BLE Lab 2 template provided by Cypress, and open TopDesign.cysch. In the Bluetooth Low Energy sheet of the schematic, place the BLE component, as shown in Figure 2.

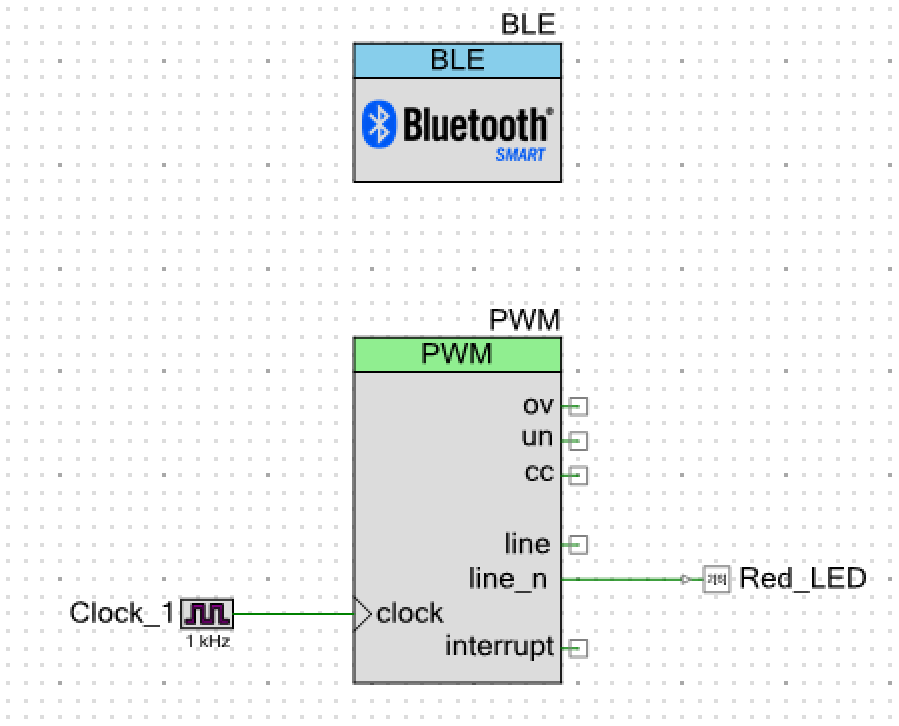
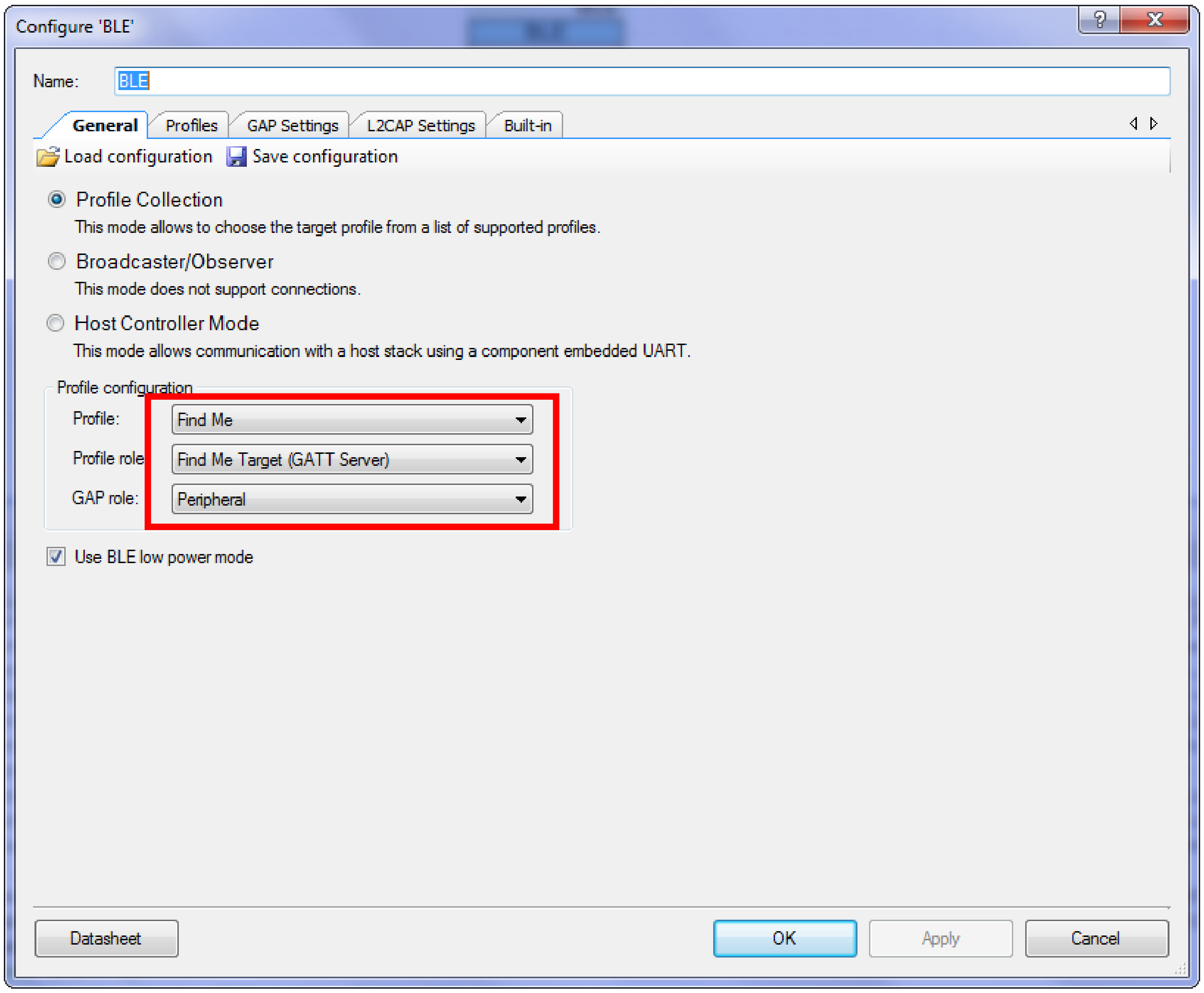


Figure 2: Schematic

3. BLE configuration – General Tab

By following the step below, the BLE component can be set to the Find Me Target (GATT Server and GAP Peripheral).


Configure 'BLE'

Name: BLE

Profile Collection [✓]
This mode allows to choose the target profile from a list of supported profiles.

Broadcaster/Observer
This mode does not support connections.

Host Controller Mode
This mode allows communication with a host stack using a component embedded UART.

Profile configuration

Profile: Find Me

Profile role: Find Me Target (GATT Server)

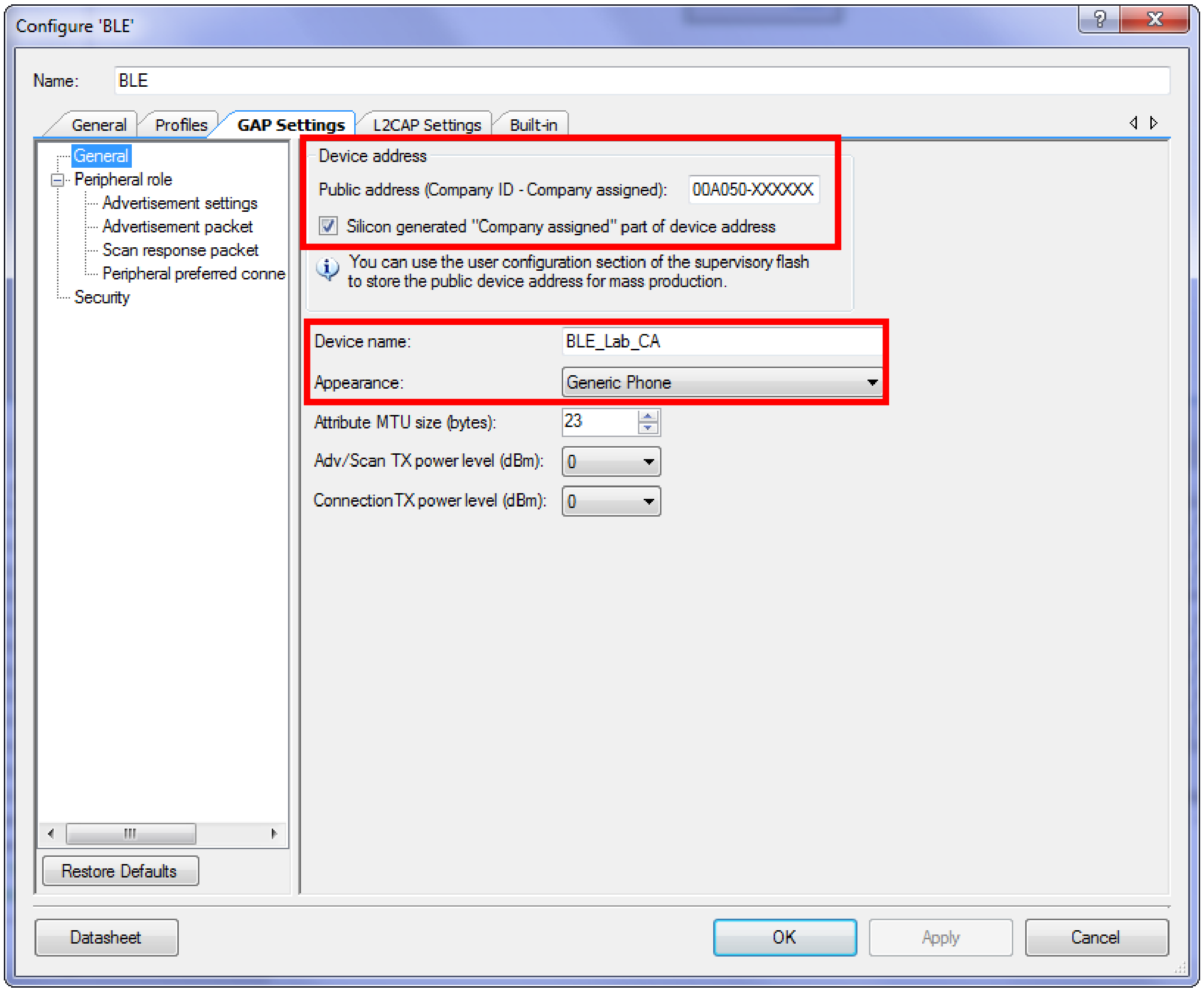
GAP role: Peripheral

[✓] Use BLE low power mode

Figure 3: BLE Component Configuration – General Tab

4. BLE configuration – GAP Settings Tab

In order to set the BLE component as a GAP Peripheral, the following configurations need to be completed.


Configure 'BLE'

Name: BLE

General | Profiles | **GAP Settings** | L2CAP Settings | Built-in

- General
- Peripheral role
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
  - Peripheral preferred conne
- Security

Restore Defaults

Device address

Public address (Company ID - Company assigned): 00A050-XXXXXX

[✓] Silicon generated "Company assigned" part of device address

You can use the user configuration section of the supervisory flash to store the public device address for mass production.

Device name: BLE_Lab_CA

Appearance: Generic Phone

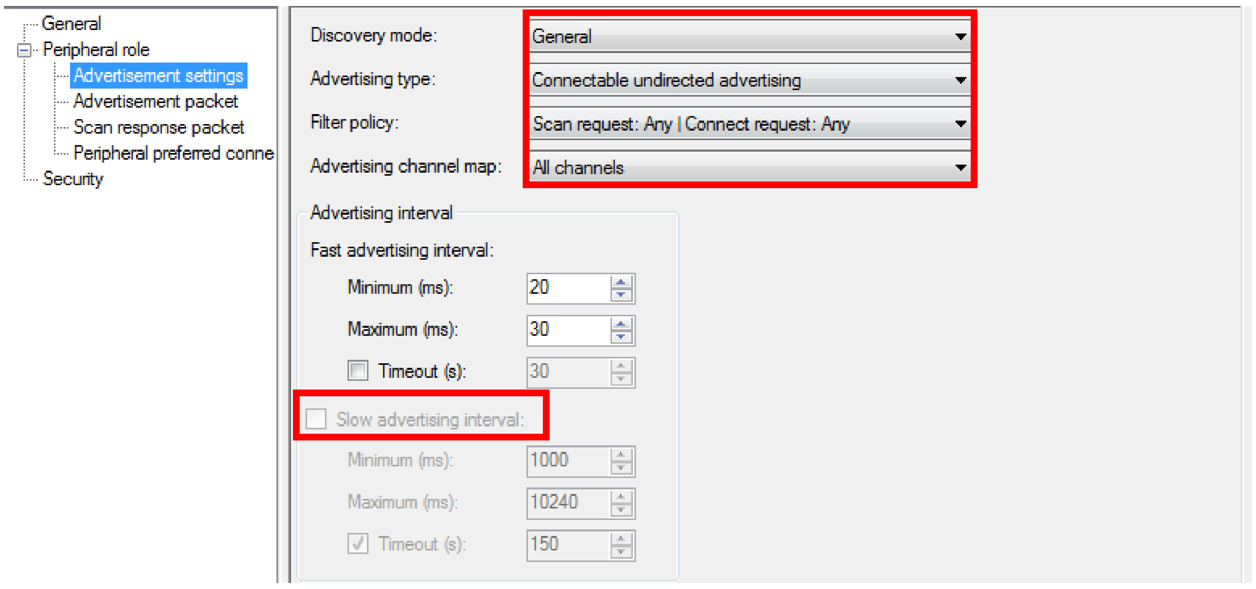
Attribute MTU size (bytes): 23

Adv/Scan TX power level (dBm): 0

Connection TX power level (dBm): 0

Datasheet | OK | Apply | Cancel

Figure 4: GAP Setting Tab - General



General
Peripheral role
Advertisement settings
Advertisement packet
Scan response packet
Peripheral preferred conne
Security

Discovery mode:	General
Advertising type:	Connectable undirected advertising
Filter policy:	Scan request: Any | Connect request: Any
Advertising channel map:	All channels

Advertising interval

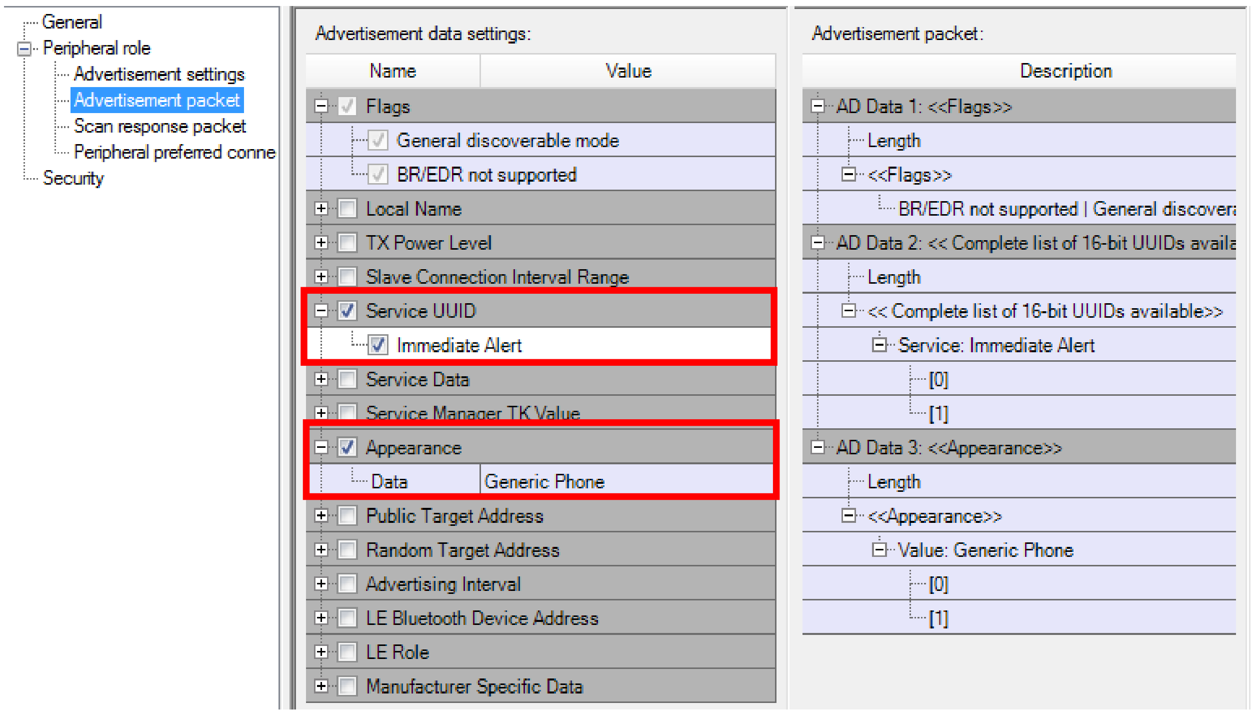
Fast advertising interval:

Minimum (ms):	20
Maximum (ms):	30
☐ Timeout (s):	30

☐ Slow advertising interval:

Minimum (ms):	1000
Maximum (ms):	10240
✓ Timeout (s):	150

Figure 5: GAP Setting Tab – Advertisement settings



General
Peripheral role
Advertisement settings
Advertisement packet
Scan response packet
Peripheral preferred conne
Security

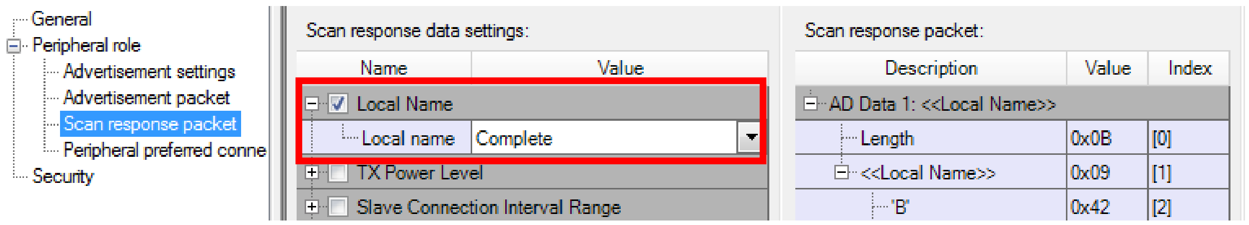
Advertisement data settings:

Name	Value
✓ Flags	
✓ General discoverable mode	
✓ BR/EDR not supported	
☐ Local Name	
☐ TX Power Level	
☐ Slave Connection Interval Range	
✓ Service UUID	
✓ Immediate Alert	
☐ Service Data	
☐ Service Manager TK Value	
✓ Appearance	
Data	Generic Phone
☐ Public Target Address	
☐ Random Target Address	
☐ Advertising Interval	
☐ LE Bluetooth Device Address	
☐ LE Role	
☐ Manufacturer Specific Data	

Advertisement packet:

Description
AD Data 1: <<Flags>>
Length
<<Flags>>
BR/EDR not supported | General discovera
AD Data 2: << Complete list of 16-bit UUIDs availa
Length
<< Complete list of 16-bit UUIDs available>>
Service: Immediate Alert
[0]
[1]
AD Data 3: <<Appearance>>
Length
<<Appearance>>
Value: Generic Phone
[0]
[1]

Figure 6: GAP Setting Tab – Advertisement Packet



General
Peripheral role
Advertisement settings
Advertisement packet
Scan response packet
Peripheral preferred conne
Security

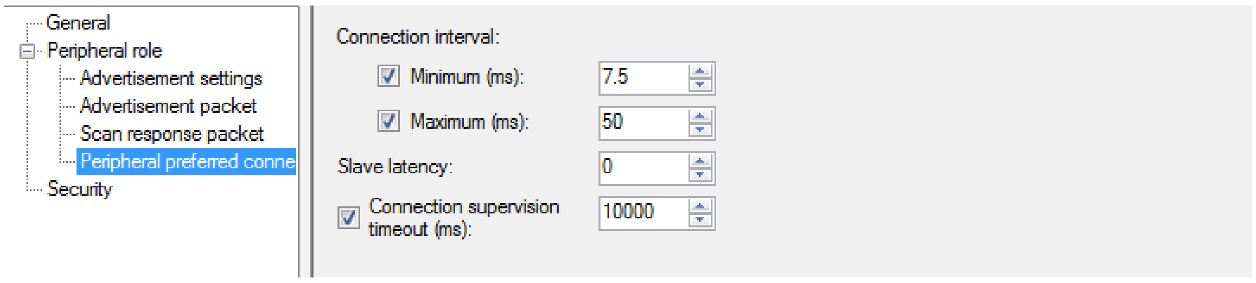
Scan response data settings:

Name	Value
Local Name	
Local name	Complete
TX Power Level	
Slave Connection Interval Range	

Scan response packet:

Description	Value	Index
AD Data 1: <<Local Name>>		
Length	0x0B	[0]
<<Local Name>>	0x09	[1]
'B'	0x42	[2]

Figure 7: GAP Setting Tab – Scan response packet

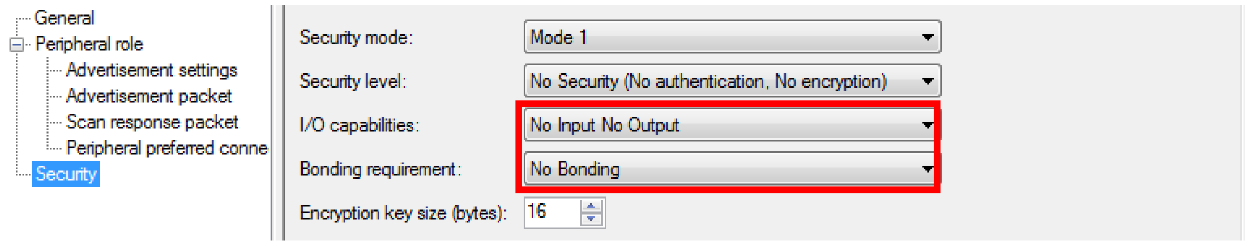


General
Peripheral role
Advertisement settings
Advertisement packet
Scan response packet
Peripheral preferred conne
Security

Connection interval:

Minimum (ms):	7.5
Maximum (ms):	50
Slave latency:	0
Connection supervision timeout (ms):	10000

Figure 8: GAP Setting Tab – Peripheral preferred connection



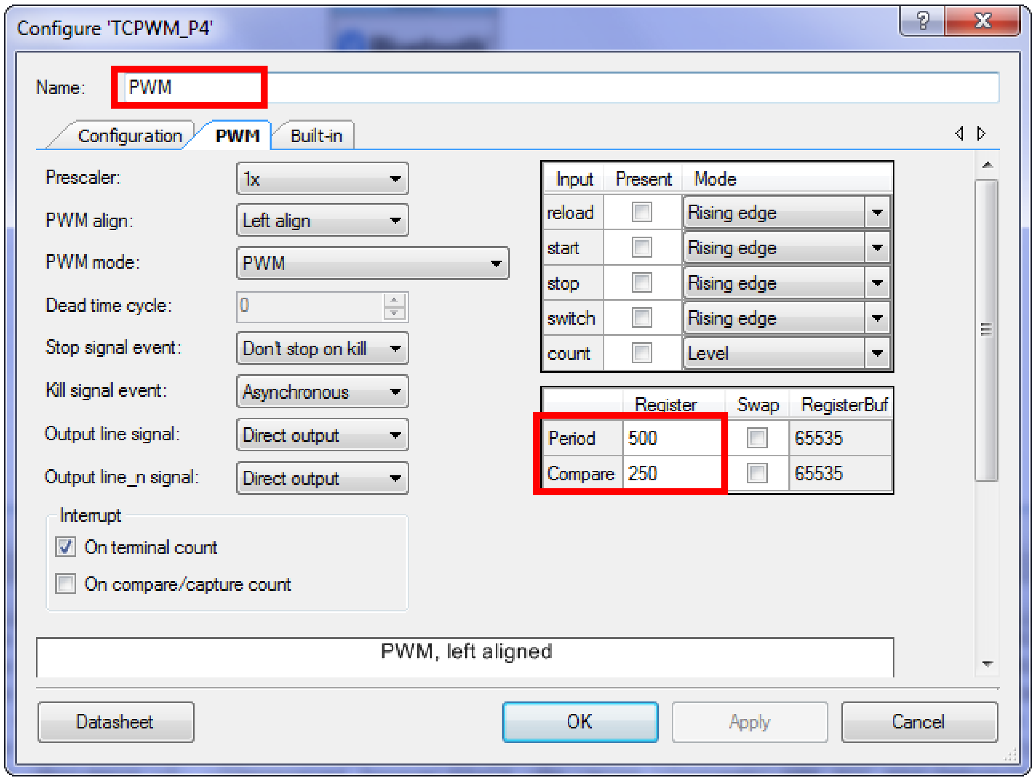
General
Peripheral role
Advertisement settings
Advertisement packet
Scan response packet
Peripheral preferred conne
Security

Security mode:	Mode 1
Security level:	No Security (No authentication, No encryption)
I/O capabilities:	No Input No Output
Bonding requirement:	No Bonding
Encryption key size (bytes):	16

Figure 9: GAP Setting Tab – Security

5. PWM component

The PWM component, the clock and the output pin need to be configured as a part of the project template.



Configure 'TCPWM_P4'

Name: PWM

Prescaler:	1x
PWM align:	Left align
PWM mode:	PWM
Dead time cycle:	0
Stop signal event:	Don't stop on kill
Kill signal event:	Asynchronous
Output line signal:	Direct output
Output line_n signal:	Direct output

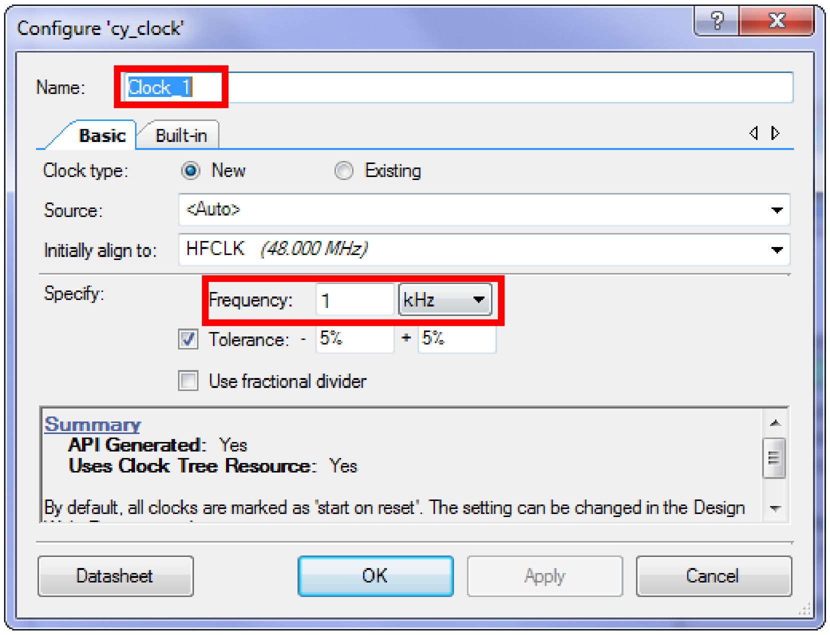
Interrupt
- [x] On terminal count
- [ ] On compare/capture count

Input	Present	Mode
reload	☐	Rising edge
start	☐	Rising edge
stop	☐	Rising edge
switch	☐	Rising edge
count	☐	Level

	Register	Swap	RegisterBuf
Period	500	☐	65535
Compare	250	☐	65535

PWM, left aligned

Figure 10: PWM Component Configuration



Configure 'cy_clock'

Name: Clock_1

Clock type: [✓ New] Existing

Source: <Auto>

Initially align to: HFCLK (48.000 MHz)

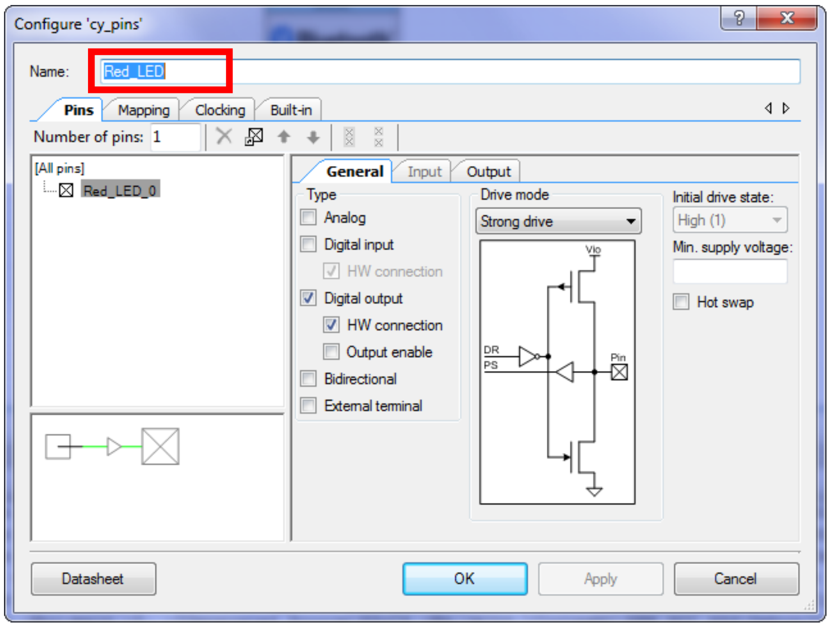
Specify: Frequency: 1 kHz

- [x] Tolerance: - 5% + 5%
- [ ] Use fractional divider

Summary
API Generated: Yes
Uses Clock Tree Resource: Yes

By default, all clocks are marked as 'start on reset'. The setting can be changed in the Design

Figure 11: PWM Component Configuration - Clock



Configure 'cy_pins'

Name: Red_LED

Pins | Mapping | Clocking | Built-in

Number of pins: 1

[All pins]
Red_LED_0

General | Input | Output

Type
- [ ] Analog
- [ ] Digital input
  - [✓] HW connection
- [✓] Digital output
  - [✓] HW connection
  - [ ] Output enable
- [ ] Bidirectional
- [ ] External terminal

Drive mode: Strong drive

Initial drive state: High (1)

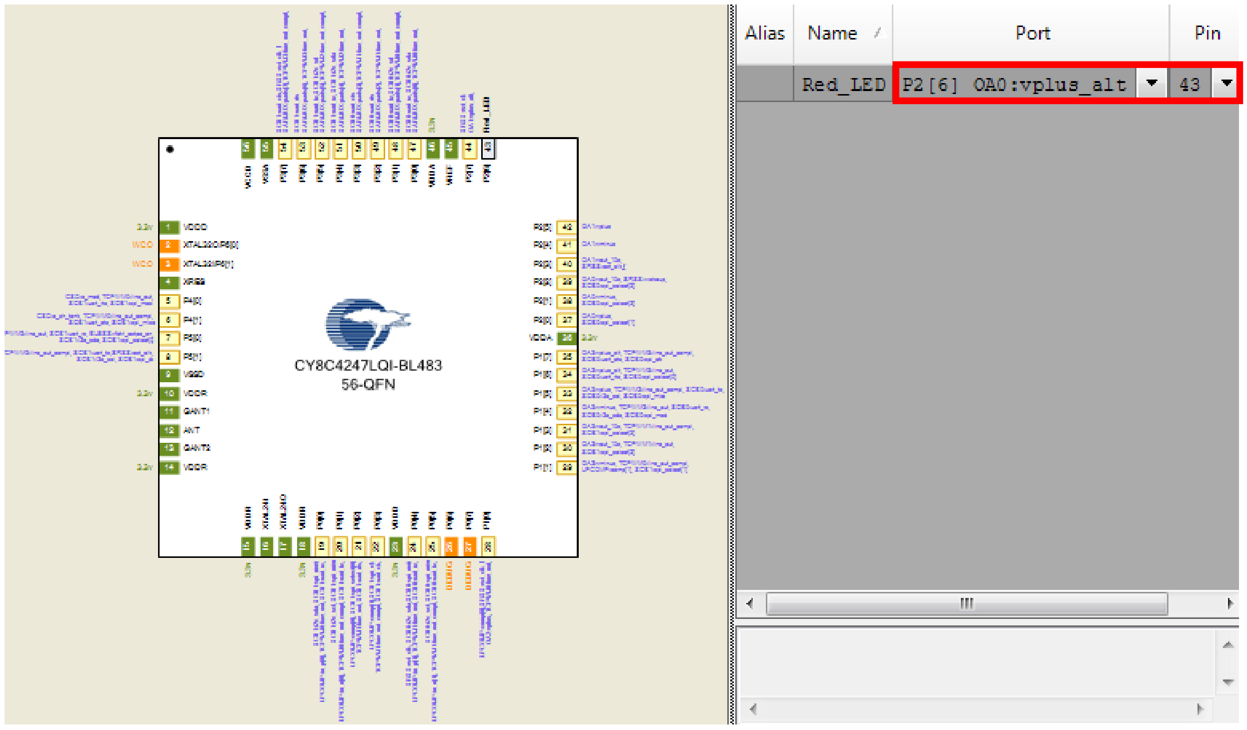
Min. supply voltage:

- [ ] Hot swap

Figure 12: PWM Component Configuration – Output Pin

6. DWR configuration

Now we're done with configuring the schematics of the BLE and the PWM component. We're going to configure the DWR file, which is a Design Wide Resources. So I open BLE Lab 2.cydwr, and set Red_LED to P2[6] as shown in Figure 13.

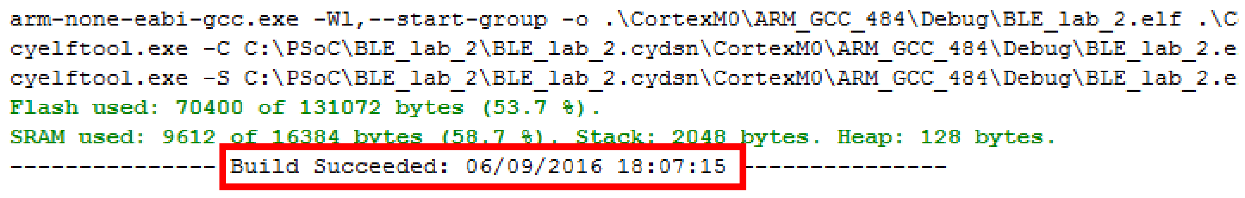


Alias	Name	Port	Pin
	Red_LED	P2[6] OA0:vplus_alt	43

Figure 13: DWR Configuration

7. Build & Program

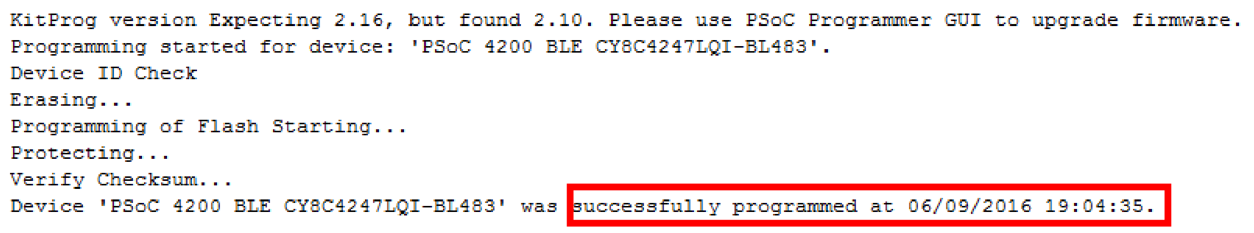
Then we build the application to generate source code files, which will auto-complete the API names, variables and macro. Figure 14 shows the build is successful.



```
arm-none-eabi-gcc.exe -Wl,--start-group -o .\CortexM0\ARM_GCC_484\Debug\BLE_lab_2.elf .\C
cyelftool.exe -C C:\PSoC\BLE_lab_2\BLE_lab_2.cydsn\CortexM0\ARM_GCC_484\Debug\BLE_lab_2.e
cyelftool.exe -S C:\PSoC\BLE_lab_2\BLE_lab_2.cydsn\CortexM0\ARM_GCC_484\Debug\BLE_lab_2.e
Flash used: 70400 of 131072 bytes (53.7 %).
SRAM used: 9612 of 16384 bytes (58.7 %). Stack: 2048 bytes. Heap: 128 bytes.
--------------- Build Succeeded: 06/09/2016 18:07:15 ---------------
```

Figure 14: Successful Project Build

The program log for this lab is shown in Figure 15.



```
KitProg version Expecting 2.16, but found 2.10. Please use PSoC Programmer GUI to upgrade firmware.
Programming started for device: 'PSoC 4200 BLE CY8C4247LQI-BL483'.
Device ID Check
Erasing...
Programming of Flash Starting...
Protecting...
Verify Checksum...
Device 'PSoC 4200 BLE CY8C4247LQI-BL483' was successfully programmed at 06/09/2016 19:04:35.
```

Figure 15: Successful Programmed to Kit

8. Firmware Flow

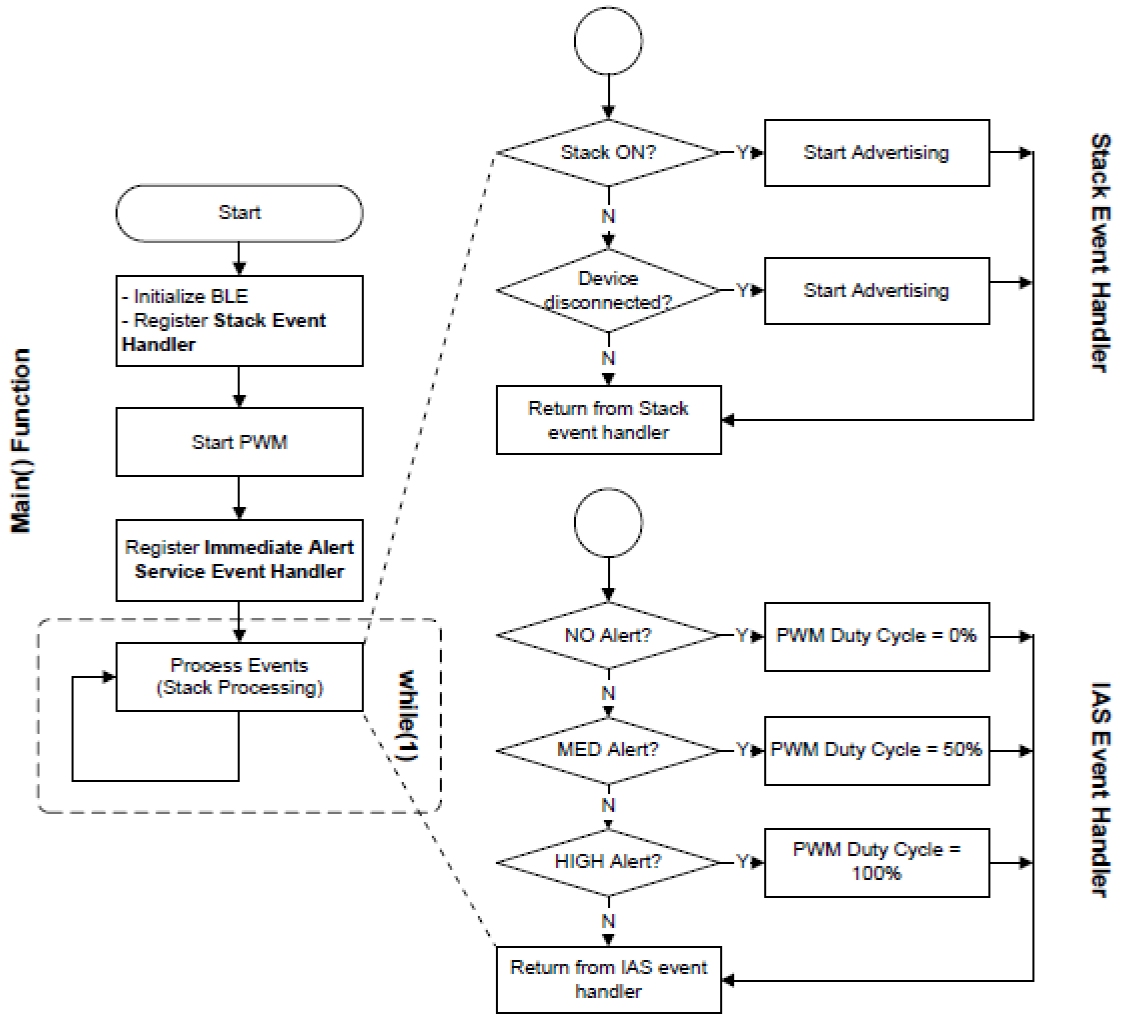
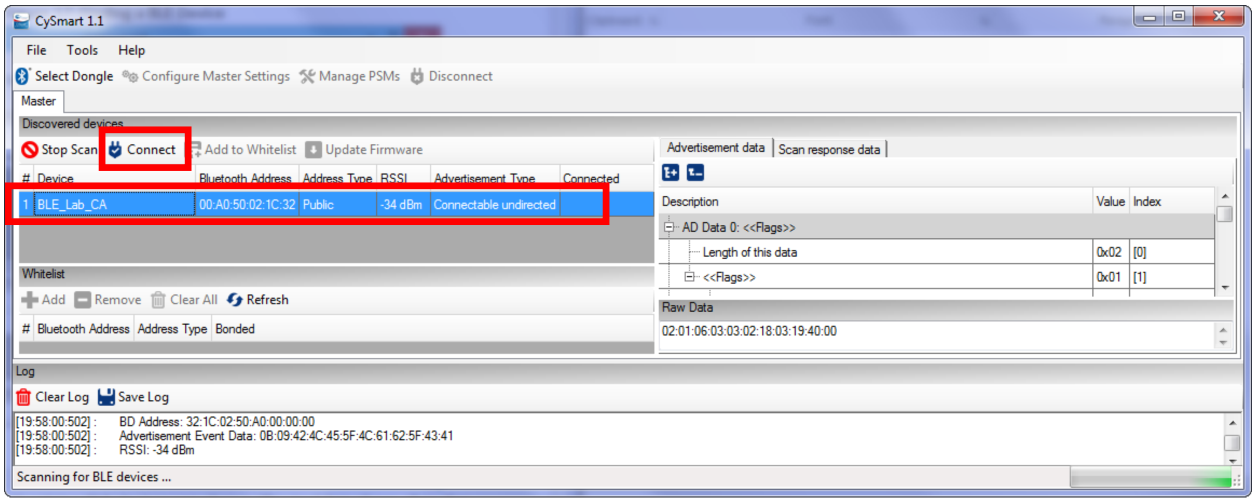


Figure 16: Firmware Flow

17. CySmart Scanning

We plugged the BLE-USB Bridge (included with the BLE Pioneer Kit) in my computer's USB port, and then launched CySmart 1.1. By connecting the Cypress BLE Dongle and using it to scan BLE device, we can find the BLE Pioneer Kit, as shown in Figure 17.



#	Device	Bluetooth Address	Address Type	RSSI	Advertisement Type	Connected
1	BLE_Lab_CA	00:A0:50:02:1C:32	Public	-34 dBm	Connectable undirected	

#	Bluetooth Address	Address Type	Bonded

Description	Value	Index
AD Data 0: <<Flags>>		
Length of this data	0x02	[0]
<<Flags>>	0x01	[1]

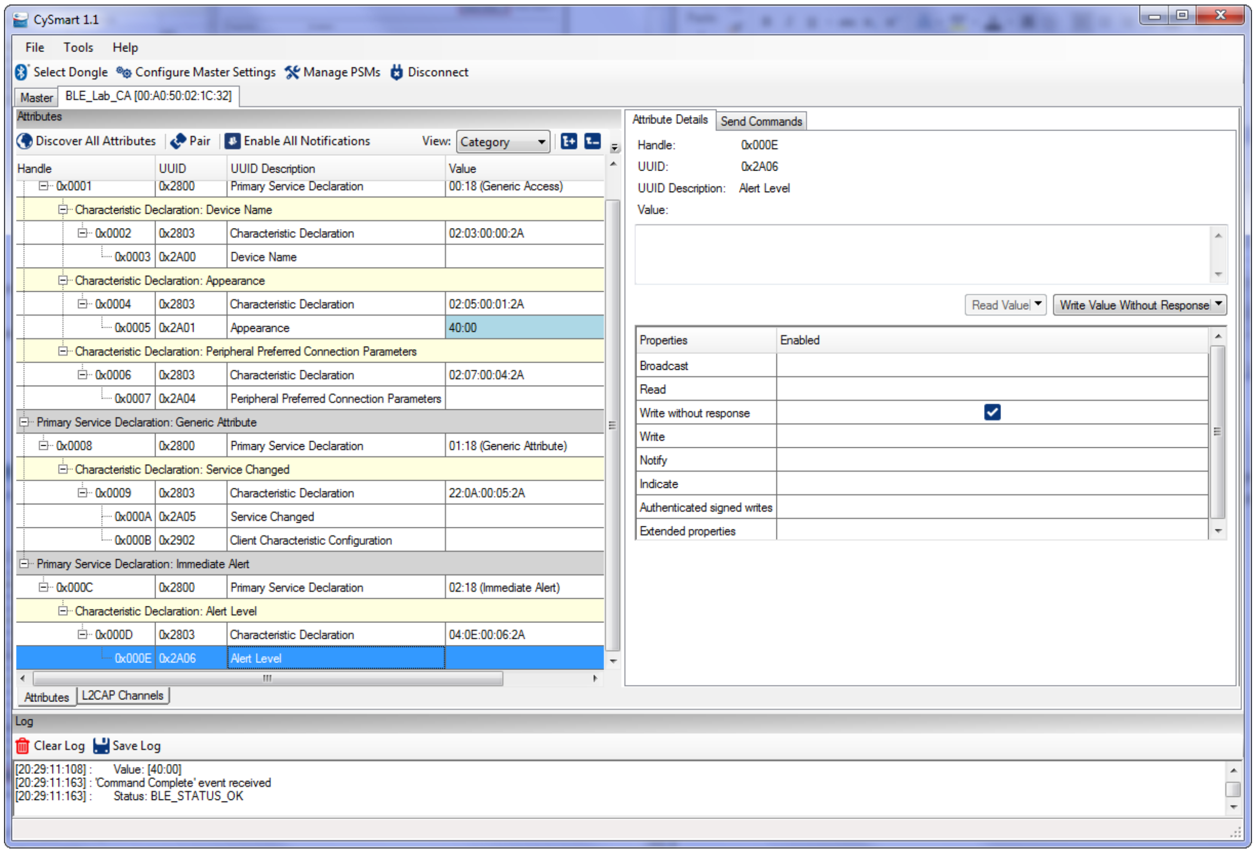
```
02:01:06:03:03:02:18:03:19:40:00
```

```
[19:58:00:502] :    BD Address: 32:1C:02:50:A0:00:00:00
[19:58:00:502] :    Advertisement Event Data: 0B:09:42:4C:45:5F:4C:61:62:5F:43:41
[19:58:00:502] :    RSSI: -34 dBm
Scanning for BLE devices ...
```

Figure 17: CySmart Scanning Results

18. Discover all attributes

I clicked Discover All Attributes to list all the Attributes in the device, with their respective UUIDs (Universally Unique Identifier) and descriptions. Then, I located the Alert Level Attribute for the Immediate Alert Service. See Figure 18.



Handle	UUID	UUID Description	Value
0x0001	0x2800	Primary Service Declaration	00:18 (Generic Access)
Characteristic Declaration: Device Name			
0x0002	0x2803	Characteristic Declaration	02:03:00:00:2A
0x0003	0x2A00	Device Name	
Characteristic Declaration: Appearance			
0x0004	0x2803	Characteristic Declaration	02:05:00:01:2A
0x0005	0x2A01	Appearance	40:00
Characteristic Declaration: Peripheral Preferred Connection Parameters			
0x0006	0x2803	Characteristic Declaration	02:07:00:04:2A
0x0007	0x2A04	Peripheral Preferred Connection Parameters	
Primary Service Declaration: Generic Attribute			
0x0008	0x2800	Primary Service Declaration	01:18 (Generic Attribute)
Characteristic Declaration: Service Changed			
0x0009	0x2803	Characteristic Declaration	22:0A:00:05:2A
0x000A	0x2A05	Service Changed	
0x000B	0x2902	Client Characteristic Configuration	
Primary Service Declaration: Immediate Alert			
0x000C	0x2800	Primary Service Declaration	02:18 (Immediate Alert)
Characteristic Declaration: Alert Level			
0x000D	0x2803	Characteristic Declaration	04:0E:00:06:2A
0x000E	0x2A06	Alert Level	

Handle: 0x000E
UUID: 0x2A06
UUID Description: Alert Level
Value:

Properties	Enabled
Broadcast	
Read	
Write without response	✓
Write	
Notify	
Indicate	
Authenticated signed writes	
Extended properties	

```
[20:29:11:108] :    Value: [40:00]
[20:29:11:163] : 'Command Complete' event received
[20:29:11:163] :    Status: BLE_STATUS_OK
```

Figure 18: Alert Level Attribute

RESULTS

As shown in Figure 19, value of 0 can turn off the LED while 1 starts blinking the LED, and 2 can be set to keep the LED always on.

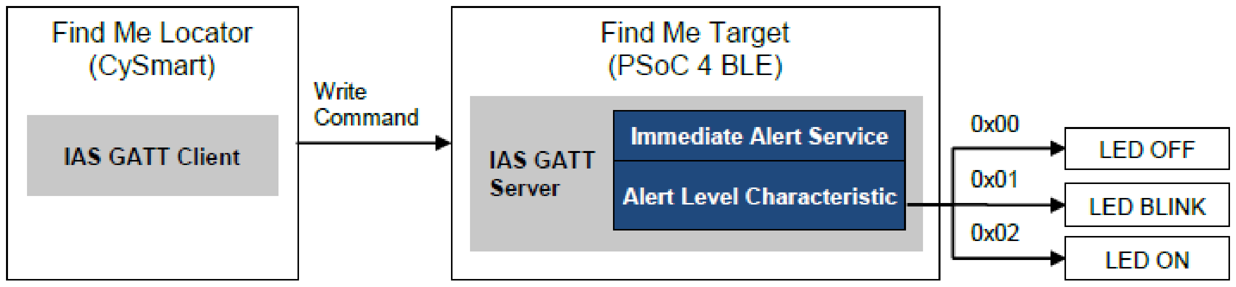
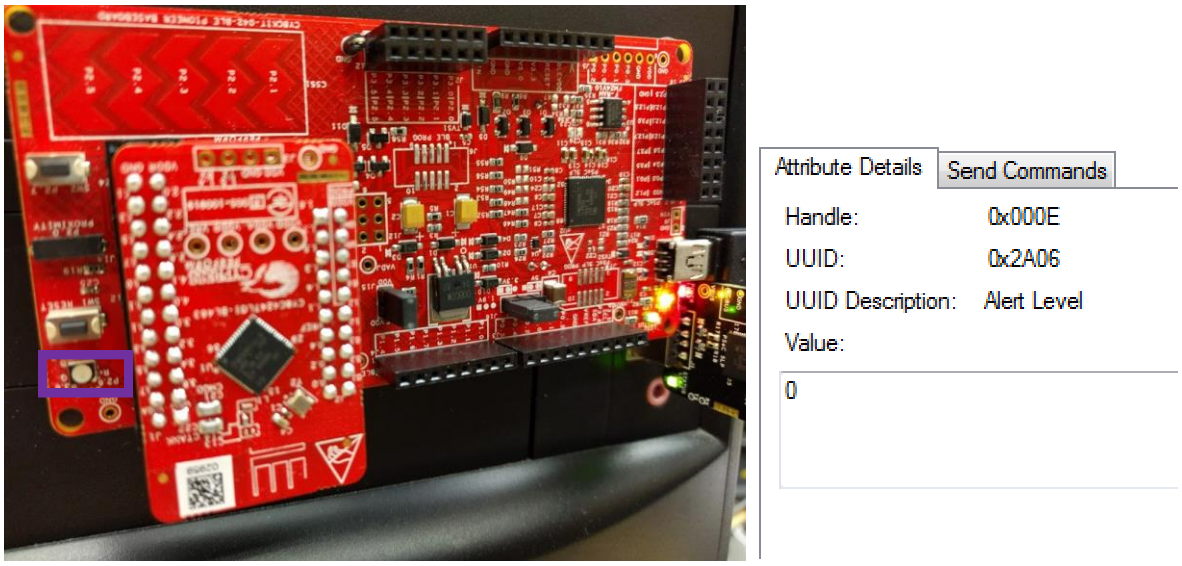


Figure 19: Find Me Target Description

1. Turn off

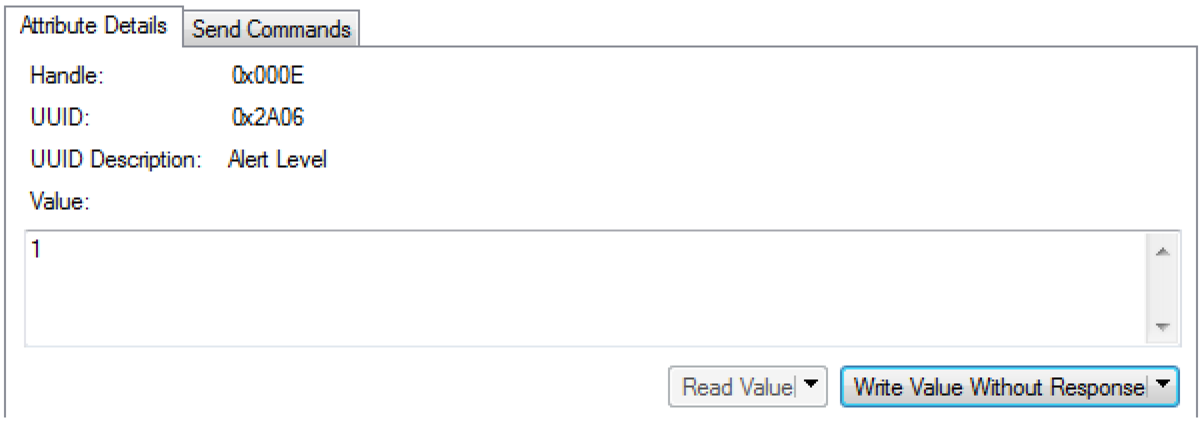


Handle: 0x000E
UUID: 0x2A06
UUID Description: Alert Level
Value:
0

Figure 20: Turn OFF

2. Blinking

It is hard to capture the state of blinking. Therefore, the photo in this state is not included.

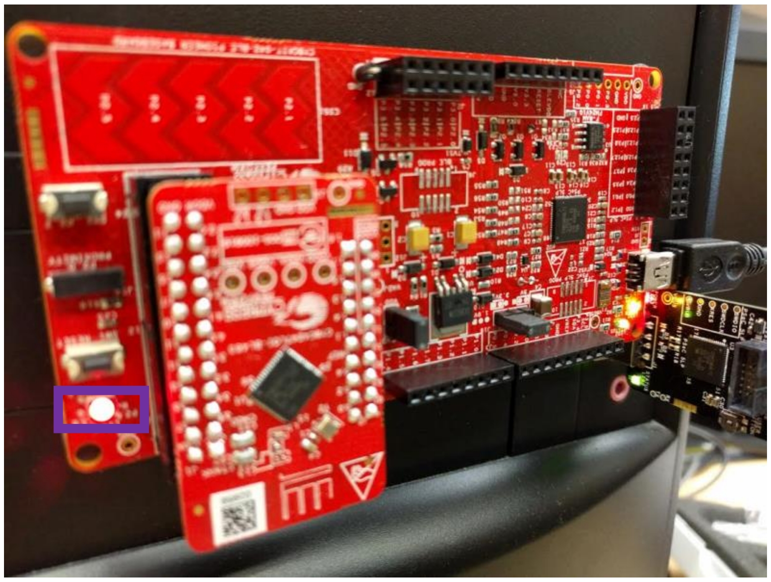

Attribute Details | Send Commands

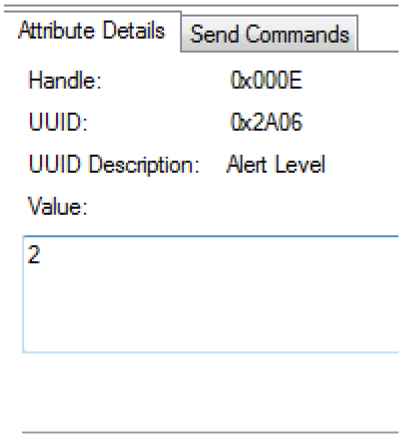
Handle: 0x000E
UUID: 0x2A06
UUID Description: Alert Level
Value:

```
1
```

Figure 20: Blinking State

3. Turn on




Attribute Details | Send Commands

Handle: 0x000E
UUID: 0x2A06
UUID Description: Alert Level
Value:

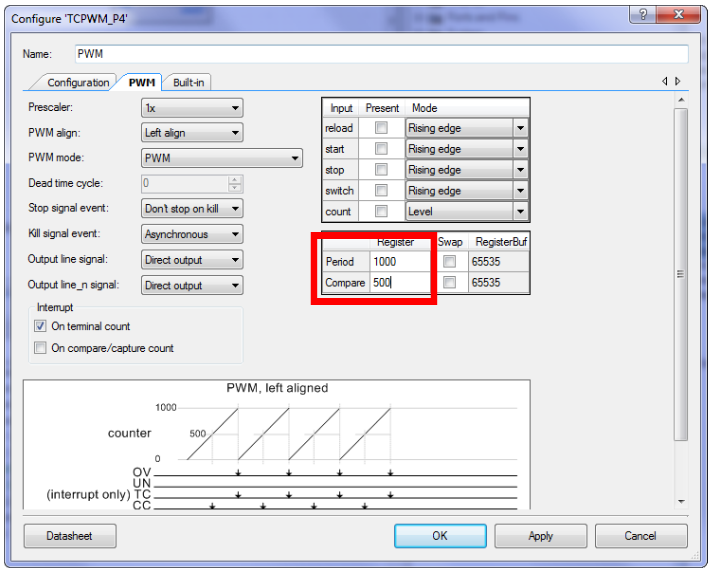
```
2
```

Figure 20: Turn ON

ADDITIONAL EXERCISES

1. Change the LED blink rate to 1-Hz.

At present, the LED blink rate is 2 Hz. In order to reduce the frequency to 1 Hz, the period of PWM should be set as 1000 due to the fact the clock frequency is 1000 Hz. I also changed to compare value to 500.


Configure 'TCPWM_P4'

Name: PWM

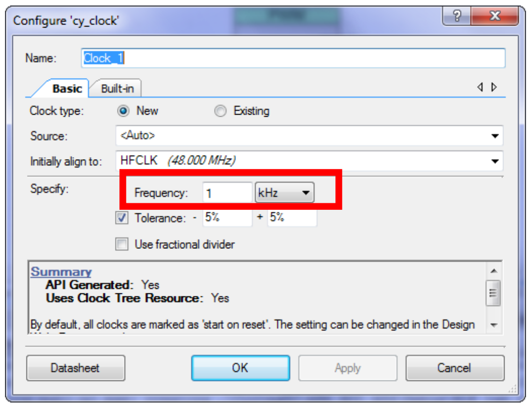
Configuration | **PWM** | Built-in

Prescaler:	1x
PWM align:	Left align
PWM mode:	PWM
Dead time cycle:	0
Stop signal event:	Don't stop on kill
Kill signal event:	Asynchronous
Output line signal:	Direct output
Output line_n signal:	Direct output

Interrupt
- [x] On terminal count
- [ ] On compare/capture count

Input	Present	Mode
reload	☐	Rising edge
start	☐	Rising edge
stop	☐	Rising edge
switch	☐	Rising edge
count	☐	Level

	Register	Swap	RegisterBuf
Period	1000	☐	65535
Compare	500	☐	65535


Configure 'cy_clock'

Name: Clock_1

Basic | Built-in

Clock type: [✓ New] Existing

Source: <Auto>

Initially align to: HFCLK *(48.000 MHz)*

Specify: Frequency: 1 kHz

- [x] Tolerance: - 5% + 5%
- [ ] Use fractional divider

Summary
API Generated: Yes
Uses Clock Tree Resource: Yes

By default, all clocks are marked as 'start on reset'. The setting can be changed in the Design

Figure 21: PWM & Clock Configuration

At the same time, we made modifications in the Main.c file in correspond to the change of PWM period.

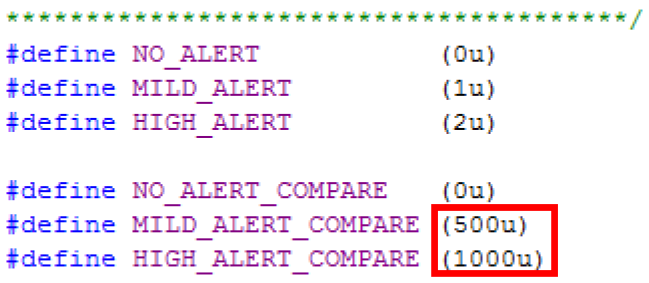
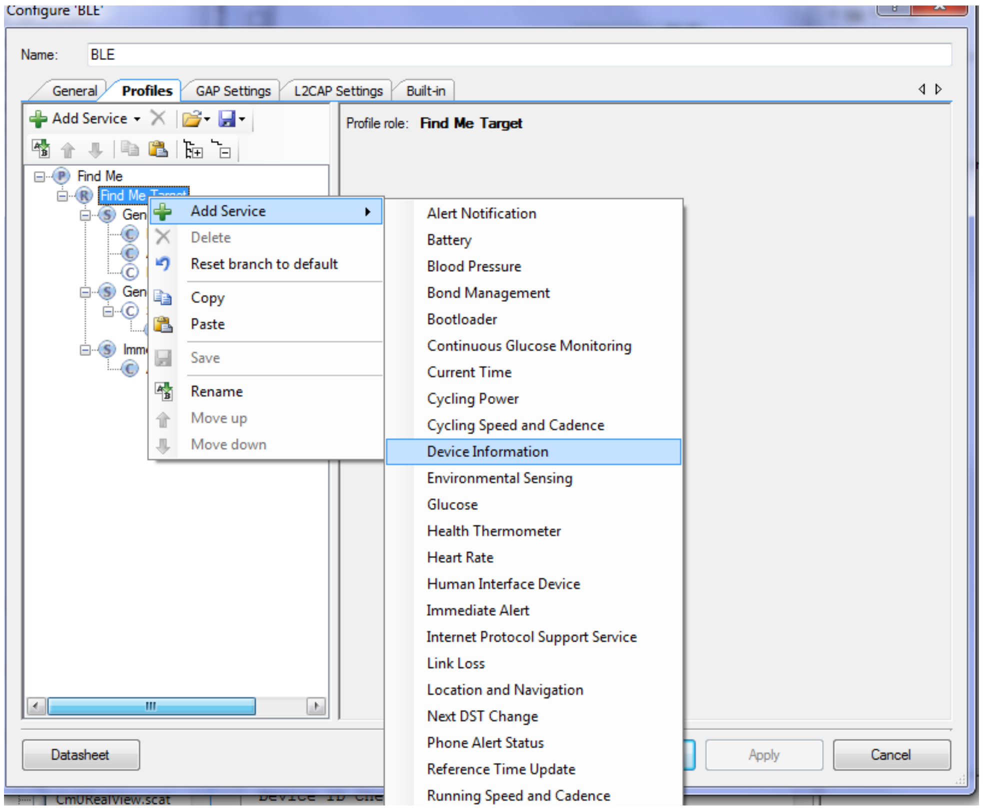

```
*****************************************/
#define NO_ALERT                (0u)
#define MILD_ALERT              (1u)
#define HIGH_ALERT              (2u)

#define NO_ALERT_COMPARE        (0u)
#define MILD_ALERT_COMPARE      (500u)
#define HIGH_ALERT_COMPARE      (1000u)
```

Figure 21: Modification in Main.c

The LED blinks at 1 Hz. Since it is hard to capture the state of blinking, the photo is not included.

2. Add the Device Information Service (DIS) to the Find Me Profile.

The Additional Services can be added by right-clicking the Find Me Target in the BLE Component Configuration Tool, and then selecting Add Service.


Configure 'BLE'

Name: BLE

Profile role: Find Me Target

Figure 21: Add the DIS

CONCLUSIONS

From Lab 2, I learned how to create a schematic and configure components. The PSoC creator provided a wonderful platform for users to develop their own circuits and realize intended functions. The platform saves a lot of work for users. And my effort is focused on how to configure components and understand the meaning of each parameter. Although I came across some incompatibility problems between CySmart 1.2 and the Cypress BLE Dongle, I eventually realized the BLE Find Me function.

The additional exercises 1 gave me a deeper understanding of the blinking frequency and how to change it. The second additional exercise showed more configuration options which are especially useful to develop a more complex embedded system. The last additional exercise helped to understand the similarity and difference between PSoC and PRoC.