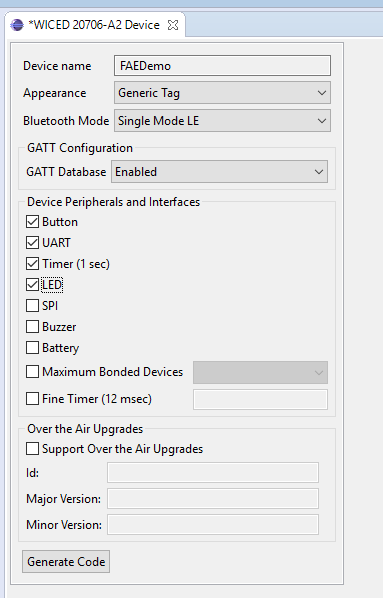
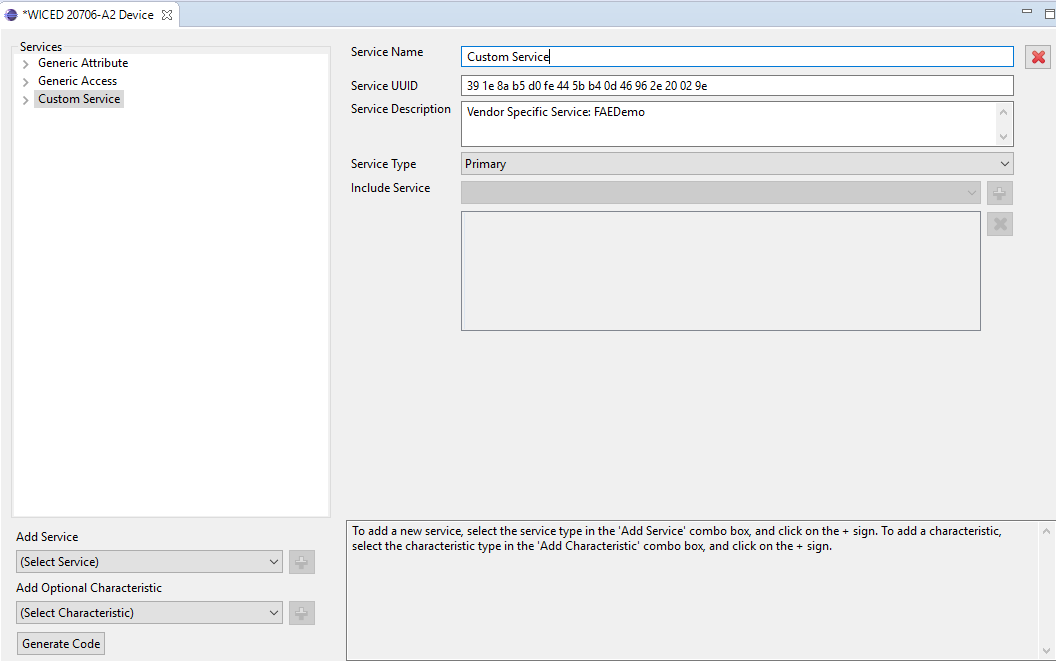
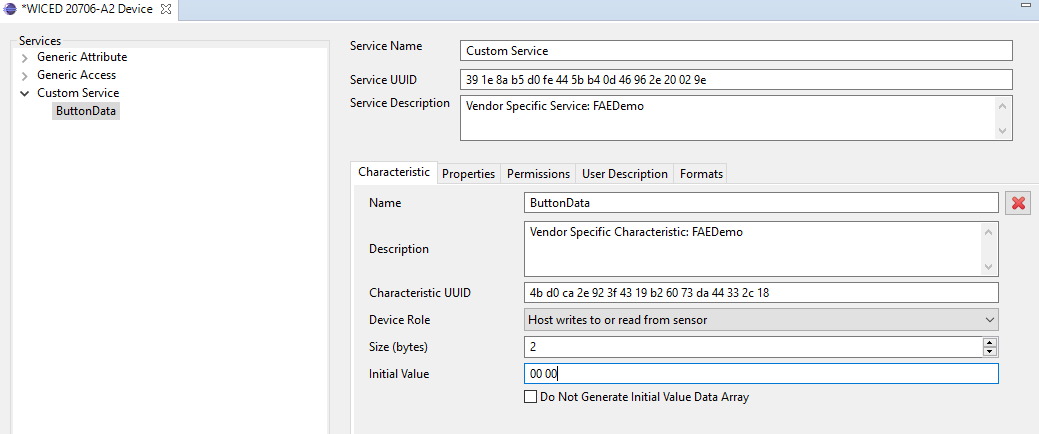
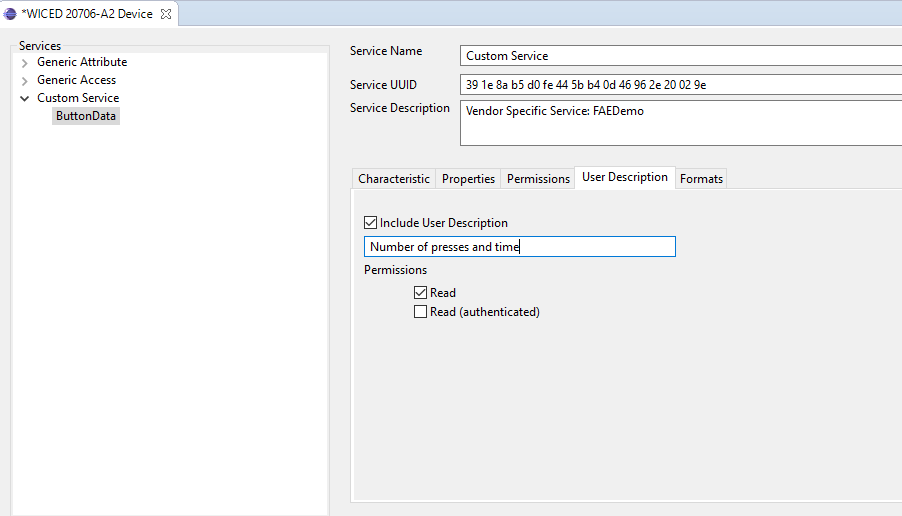
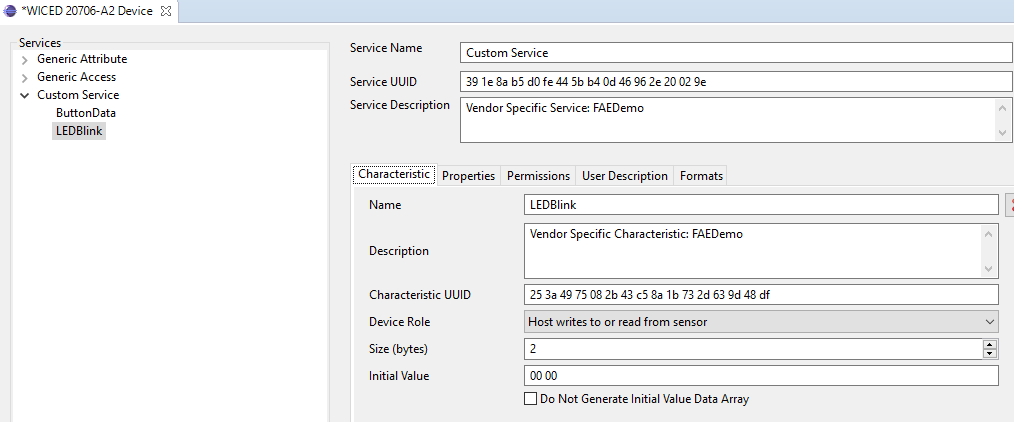
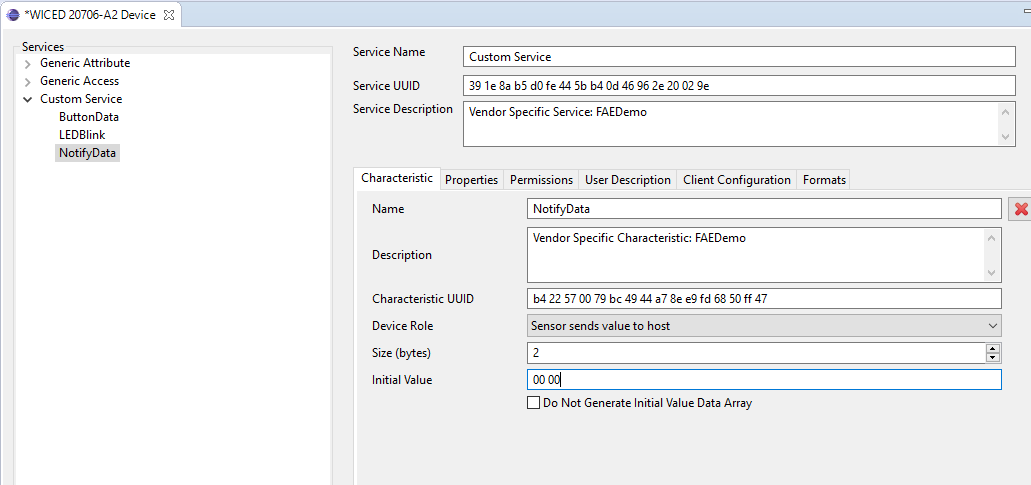
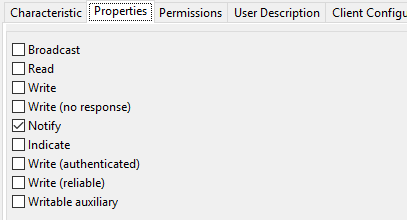
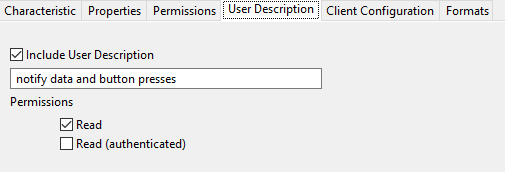
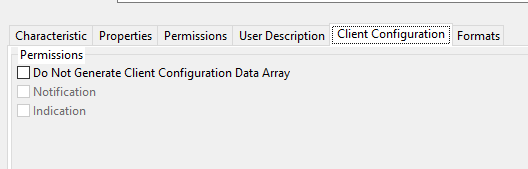
**Create Project**

This project is created in WICED Studio 4.1 with the CYW20719 Kit, and tested with CYSmart 1.2 and CY5677.

1. File, New, WICED Bluetooth Designer
2. Give your application a name – FAEDemo . Click Finish
3. This generates a GUI to configure some system peripherals as well as what services and characteristics are to be setup.
4. On the Device Settings page, change appearance to Generic Tag, set Bluetooth mode to Single Mode LE, make sure the GATT Database is Enabled, and then configure your other peripherals and interfaces. For most applications use the Button, UART, Timer (1sec) and LED:
   1. 
5. Click on the characteristics tab on the bottom. This is where we configure the services, characteristics and permissions.
6. For this application we want to show a custom service with 3 characteristics:
   * 1. Read Service to read the number of button presses and the time between presses
     2. Write Service to blink an LED by x times using the first byte we write
     3. Notify service to send data continuously to the central device, once notifications have been enabled
7. To do this we first add a service, Vendor Specific Service, and give it a name – CustomService in this example:
   1. 
8. We then add optional characteristics.
9. For the read service we select Vendor Specific Characteristic, rename it as ButtonData, change the role to Host reads or Writes from sensor, set the size to 2 bytes, and the initial values to 00 00
   1. 
10. We can then configure the properties and permissions, as well as adding user descriptions if required. For this service the default properties and permissions are ok. In the user description, tick the box to include user description and add some text – Number of presses and time
    1. 
11. We then add another characteristic for the Write data. This time call it LEDBlink, and the user description should have something like number of blinks.
    1. 
12. The 3rd characteristic is for the notify service. This will send a counter updated every second, as well as the current total number of button presses. This has slightly different properties and setup as shown:
    1. 
    2. 
    3. 
    4. 
13. The client configuration will be the handle that is written to enable or disable notifications in this example.
14. Once completed, press generate code. This will create a database c and h file, an application c file, a make file and Bluetooth config file. We will now modify these to make our application work.
15. We will start off in FAEDemo.c – this file contains all the application functions and callbacks for the device application.
16. Firstly we are going to add some defines and declarations of functions and flags that will be used in the project. On line 28 add the following:

uint8\_t NotificationsEnabled=0; //local flag for notifications being turned on or off

uint16\_t ConnectionId=0; //local variable to store the connection id in

uint8\_t ConnectedFlag=0;//flag to show if device is connected to a central or not

**static** **void** **Send\_Message**(**void**); //declaration of function to send a notification message

1. We declare some variables/flags that will be used for connected/not connected and notifications enabled/not enabled, and then declare the function we will use to send messages.
2. Wiced automatically creates function prototypes and call back functions, so most of the other changes are to add specific code functionality to the application. We want to declare that we are going to use the LED, which we will do inside the app\_init function. Go to line 240, and before the } add the following line:

wiced\_bt\_app\_hal\_led\_init();

1. The next change is in the interrupt\_handler function which is called whenever a button press is detected. In here we are going to change the code a little. We want to increment the counter when the button is pressed, then get the time since last button press, and store the time in the read value array. We are also going to copy the number of presses to the notify value array, before clearing the timers on release. The code should look like this:

/\* This is the interrupt handler \*/

**void** **faedemo\_interrupt\_handler**( **void**\* user\_data, uint8\_t value )

{

/\* Check if button has been pressed or released \*/

**if**( wiced\_hal\_gpio\_get\_pin\_input\_status(WICED\_GPIO\_PIN\_BUTTON) == WICED\_BUTTON\_PRESSED\_VALUE )

{

WICED\_BT\_TRACE("button pressed - keep pressed for 10 seconds to reset\n");

/\* Store timer count at time of button press \*/

app\_timer\_count\_at\_button\_pressed = app\_timer\_count;

faedemo\_custom\_service\_buttondata[0]++;//increment first byte as a counter for button press

faedemo\_custom\_service\_buttondata[1] = app\_timer\_count\_at\_button\_pressed; //set time pressed as second byte

faedemo\_custom\_service\_notifydata[1] = faedemo\_custom\_service\_buttondata[0]; //also copy number of presses to 2nd byte of notification package

}

**else**

{

app\_timer\_count=0;//reset timer so that we get a count between presses

app\_timer\_count\_at\_button\_pressed = 0;

}

}

1. The next change is to what happens when we get a start/stop advertising event. The tool creates an advertisement\_stopped function automatically. We want to configure our code to restart advertising if the device is not connected, or to exit if it is. We can replace the code inside the function with the following:

**void** **faedemo\_advertisement\_stopped**( **void** )

{

WICED\_BT\_TRACE("Advertisement stopped\n");

**if**(ConnectedFlag==0) //check if not connected then restart advertising

{

wiced\_bt\_start\_advertisements(*BTM\_BLE\_ADVERT\_UNDIRECTED\_HIGH*, 0, NULL); //RESTART ADVERTISING

}

**else** **if**(ConnectedFlag==1)//if we are connected just exit this fucntion

{

**return**;

}

}

1. The next change is to the timeout function. This is the 1 sec timer function, that typically just updates a counter that is used in the button press function. We are going to add a call here to send a notification every second as well. To do this add the following code on line 327:

Send\_Message();//send a notification every second if enabled

1. The next change is to the set\_value function. This is called every time a write occurs on one of the enabled characteristics. In our application we could have writes to the LEDBlink Characteristic or to the notification client configuration characteristic. For the led blink we want to copy the first byte of data and pass this to a function to blink the led X number of times. For the notify configuration we want to copy the first value to the notifications enabled flag. The code to do this should ultimately look like this:

wiced\_bt\_gatt\_status\_t **faedemo\_set\_value**( uint16\_t attr\_handle, uint16\_t conn\_id, uint8\_t \*p\_val, uint16\_t len )

{

**int** i = 0;

wiced\_bool\_t isHandleInTable = WICED\_FALSE;

wiced\_bool\_t validLen = WICED\_FALSE;

wiced\_bt\_gatt\_status\_t res = *WICED\_BT\_GATT\_INVALID\_HANDLE*;

// Check for a matching handle entry

**for** (i = 0; i < faedemo\_gatt\_db\_ext\_attr\_tbl\_size; i++)

{

**if** (faedemo\_gatt\_db\_ext\_attr\_tbl[i].handle == attr\_handle)

{

// Detected a matching handle in external lookup table

isHandleInTable = WICED\_TRUE;

// Verify that size constraints have been met

validLen = (faedemo\_gatt\_db\_ext\_attr\_tbl[i].max\_len >= len);

**if** (validLen)

{

// Value fits within the supplied buffer; copy over the value

faedemo\_gatt\_db\_ext\_attr\_tbl[i].cur\_len = len;

memcpy(faedemo\_gatt\_db\_ext\_attr\_tbl[i].p\_data, p\_val, len);

res = *WICED\_BT\_GATT\_SUCCESS*;

// **TODO**: Add code for any action required when this attribute is written

// For example you may need to write the value into NVRAM if it needs to be persistent

**switch** ( attr\_handle )

{

**case** HDLC\_CUSTOM\_SERVICE\_BUTTONDATA\_VALUE:

**break**;

**case** HDLC\_CUSTOM\_SERVICE\_LEDBLINK\_VALUE:

faedemo\_custom\_service\_ledblink[0]=p\_val[0]; //copy the first byte of data to led blink value

faedemo\_custom\_service\_ledblink[1]=p\_val[1]; //copy the second byte of data to led blink value

wiced\_bt\_app\_hal\_led\_blink( 250, 250, faedemo\_custom\_service\_ledblink[0] ); //blink the led by the amount in byte 1 of the array

**break**;

**case** HDLD\_CUSTOM\_SERVICE\_NOTIFYDATA\_CLIENT\_CONFIGURATION:

NotificationsEnabled = p\_val[0]; //set local flag to the first byte of the packet. to enable or disable notifications

**break**;

}

}

**else**

{

// Value to write does not meet size constraints

res = *WICED\_BT\_GATT\_INVALID\_ATTR\_LEN*;

}

**break**;

}

}

**if** (!isHandleInTable)

{

// **TODO**: Add code to write value using handles not contained within external lookup table

// This can apply when the option is enabled to not generate initial value arrays.

// If the value for the current handle is successfully written then set the result using:

// res = WICED\_BT\_GATT\_SUCCESS;

**switch** ( attr\_handle )

{

**default**:

// The write operation was not performed for the indicated handle

WICED\_BT\_TRACE("Write Request to Invalid Handle: 0x%x\n", attr\_handle);

res = *WICED\_BT\_GATT\_WRITE\_NOT\_PERMIT*;

**break**;

}

}

**return** res;

}

1. The next change is in the connect\_callback function. Here we want to copy the connection id to our local variable, and set a flag when connected, as well as clearing flags on disconnect and restart advertisements. The code should look like this:

/\* GATT Connection Status Callback \*/

wiced\_bt\_gatt\_status\_t **faedemo\_connect\_callback**( wiced\_bt\_gatt\_connection\_status\_t \*p\_conn\_status )

{

wiced\_bt\_gatt\_status\_t status = *WICED\_BT\_GATT\_ERROR*;

**if** ( NULL != p\_conn\_status )

{

**if** ( p\_conn\_status->connected )

{

// Device has connected

WICED\_BT\_TRACE("Connected : BDA '%B', Connection ID '%d'\n", p\_conn\_status->bd\_addr, p\_conn\_status->conn\_id );

ConnectionId = p\_conn\_status->conn\_id; //copy the connection ID locally

ConnectedFlag=1;//set the connected flag

}

**else**

{

// Device has disconnected

WICED\_BT\_TRACE("Disconnected : BDA '%B', Connection ID '%d', Reason '%d'\n", p\_conn\_status->bd\_addr, p\_conn\_status->conn\_id, p\_conn\_status->reason );

ConnectionId=0; //clear the connection id

NotificationsEnabled=0; //disable notifications

ConnectedFlag=0;//clear the connected flag

wiced\_bt\_start\_advertisements(*BTM\_BLE\_ADVERT\_UNDIRECTED\_HIGH*, 0, NULL); //restart advertising

}

status = *WICED\_BT\_GATT\_SUCCESS*;

}

**return** status;

1. The final change in this file is to add our function to send a notification message. We only want to send a message if connected device has enable notifications by writing to the client configuration handle. If enabled we will send the an incrementing value on every notification as well as the current number of button presses. The code to do that is below:

//function to send a notification to a connected device when notifications have been enabled

**void** **Send\_Message**(**void**)

{

**if**(NotificationsEnabled==0) //if notifications are disabled

{

faedemo\_custom\_service\_notifydata[0]=0; //reset the data to send to 0

**return**;

}

**else** **if**(NotificationsEnabled==1) //if notifications equal 1

{

faedemo\_custom\_service\_notifydata[0]++; //increment the first byte counter

uint8\_t \*p\_attr = (uint8\_t\*)faedemo\_custom\_service\_notifydata; //create pointer to the notification array

wiced\_bt\_gatt\_send\_notification( ConnectionId, HDLC\_CUSTOM\_SERVICE\_NOTIFYDATA\_VALUE, **sizeof**(faedemo\_custom\_service\_notifydata), p\_attr );

//send a notification with the notifydata handle

}

**else** //if we get corrupt data

{

faedemo\_custom\_service\_notifydata[0]=0;//reset data to 0 and exit

**return**;

}

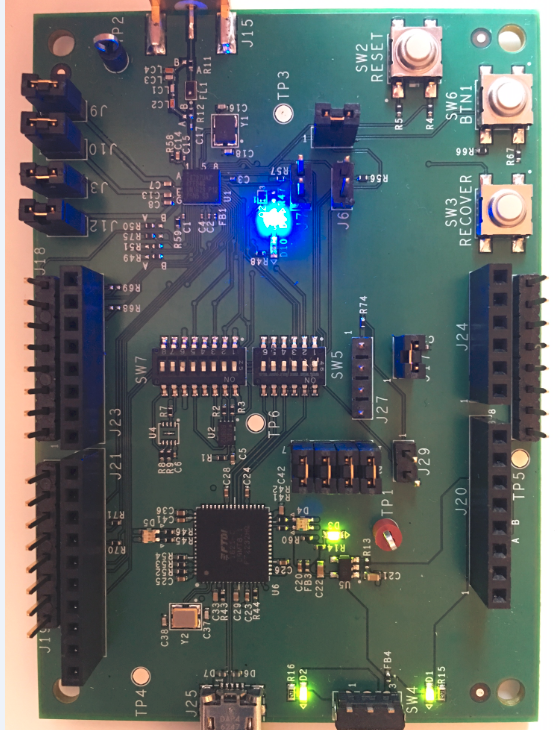
}

1. We now need to save our changes (Control and S), and then open up the db.c file. In here we need to make one change. The tool by default expects a secure connection. It therefore restricts writes to the client configuration handle unless authentication has taken place. As we don’t intend to show this feature we need to remove this restriction. On line 67 we want to delete the highlighted section of code:

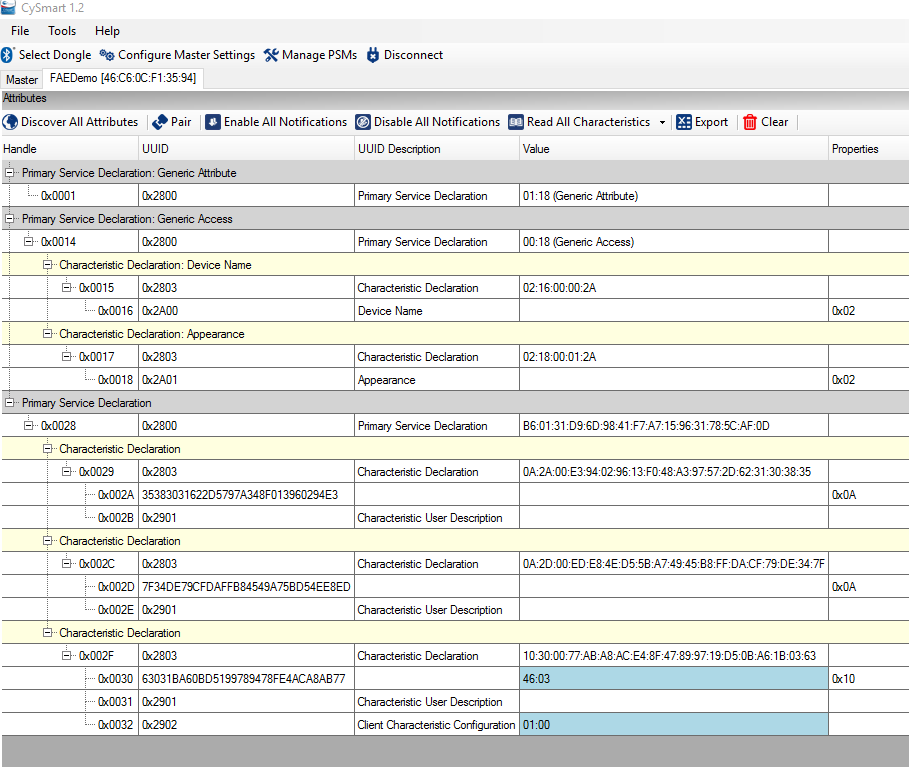
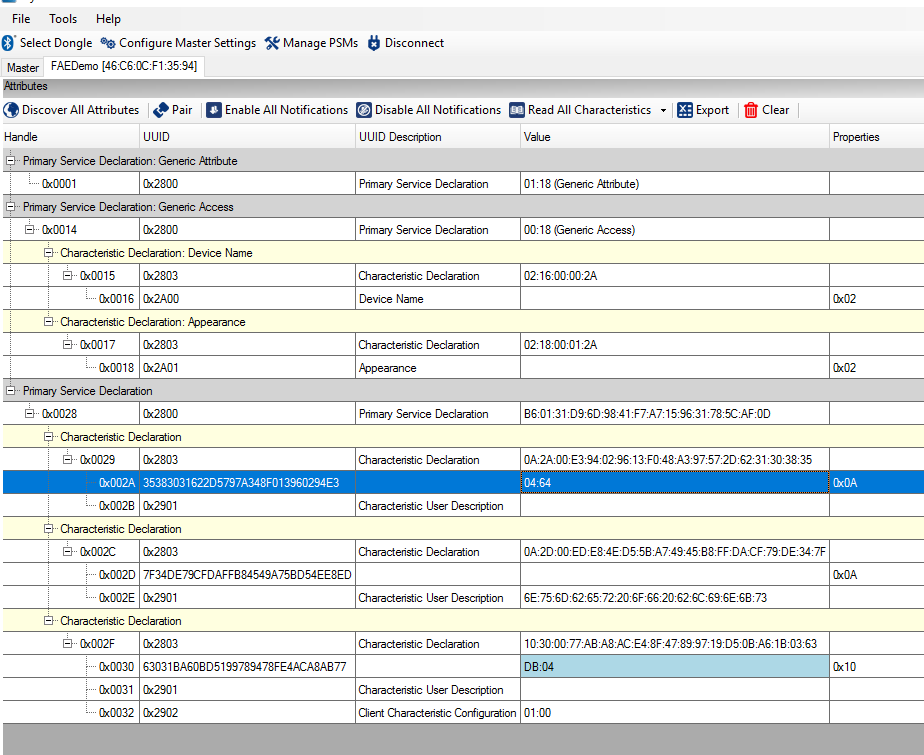
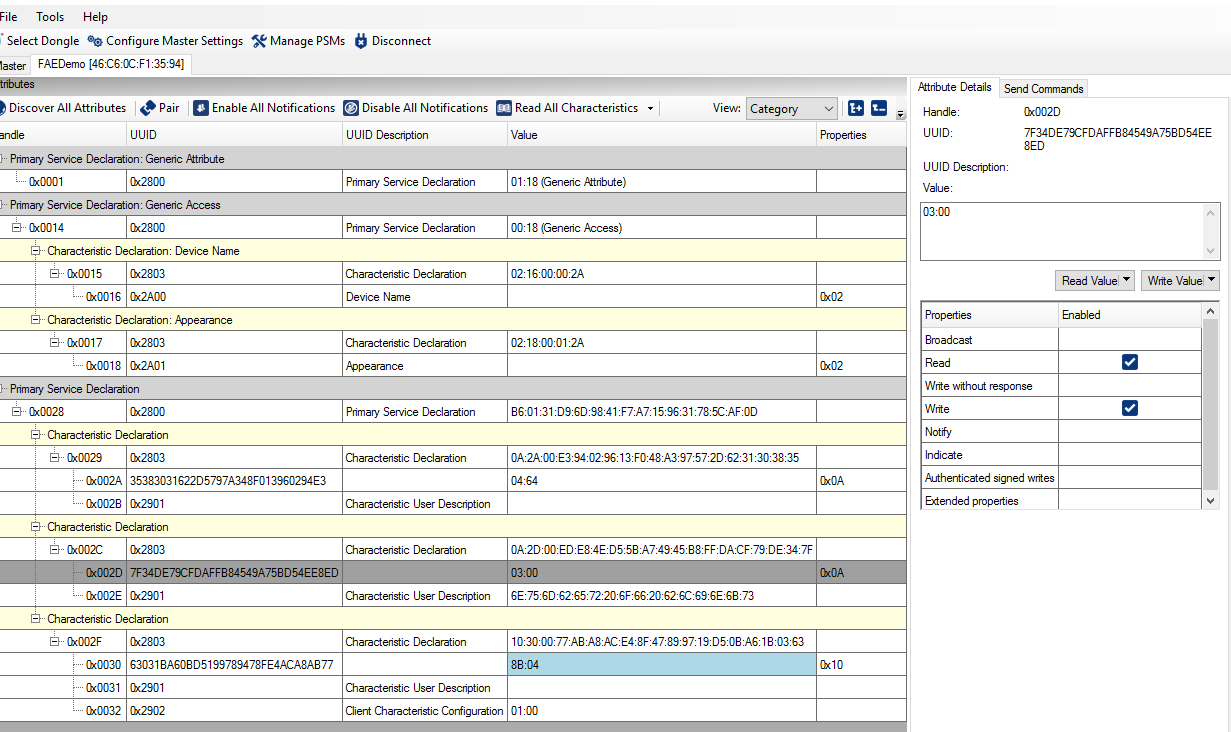
/\* Descriptor 'Client Characteristic Configuration' \*/

CHAR\_DESCRIPTOR\_UUID16\_WRITABLE (HDLD\_CUSTOM\_SERVICE\_NOTIFYDATA\_CLIENT\_CONFIGURATION,

*UUID\_DESCRIPTOR\_CLIENT\_CHARACTERISTIC\_CONFIGURATION*, LEGATTDB\_PERM\_READABLE | LEGATTDB\_PERM\_WRITE\_REQ | LEGATTDB\_PERM\_AUTH\_WRITABLE),

1. With this done, save our changes. We next need to create a new make instruction. We need to know which com port the board is plugged in to. This can be found in Device Manager.
2. The make file should match this – FAEDemo-BCM920706\_P49 UART=COMxx download where the xx is your com number i.e. 16
3. We need to make sure the switches on the 20706 board are configured correctly. As the 20706 does not have onboard flash, we need to configure it send the data to the on kit flash. The settings should match this photo:
   1. 
4. With the board attached, double click on the make target and download the application to your board.

**Testing**

1. To test the design, plug a CY5677 dongle in to your PC and launch CYSmart. On start of scanning it should detect a device called FAEDemo.
2. Connect to the device and discover all attributes.
3. If you enable all notifications you should see data starting to come in every second, with an incrementing first byte. If you press SW6 you should see the second byte reflect the number of presses, in this image, 3
   1. 
4. If we then read the attribute 0x002A we should see the number of presses logged and the time between the last touch and the previous touch. In this instance it shows 4 touches and 100 seconds between touched (64hex)
   1. 
5. if we click on attribute 0x002D we can then write 2 bytes of data to this characteristic. The first byte will reflect the number of led blinks to expect, the second byte currently does nothing. In this example we write 03 00 to the characteristic, and we should expect the led on the kit to blink 3 times
   1. 
6. This completes the testing. The application can be modified further to add extra features but it shows being able to read, write, notify and configure the device.