BLE Peripheral - Custom Service

Welcome back to Cypress Academy, PSoC 6 101. In the last lesson I showed you how to build your first PSoC 6 BLE project. In that project we used a Bluetooth SIG specified service … specifically the Immediate Alert Service. While I was looking through the Bluetooth SIG specifications for custom services I couldn't find a terminator robot service definition anywhere. That's a problem because we know that we want to control that thing with Bluetooth. So, what do we do? Well the answer to that question is simple. We need a custom service. Is that going to be hard? Nope… not at all.

What is this project going to do?

1. When the device turns on it is going to start advertising
2. When there is no connection the red led is going to blink
3. When there is a connection the red led is going to turn off
4. And when there is a connection the other side… also called the GAP Central … it will be able to change the brightness of an LED.

Simple enough eh? All right let's build a project. Start by creating a new PSoC 6 projected called 3-2-SimpleBLEPeripheral. Next, lets change the build settings to include FreeRTOS with a heap setting of 4 and the Standard IO redirection.

Now I'll go to the schematic and add the BLE Component, A UART, two PWMs, Two digital output pins and two clocks.

Let's wire the two output pins to the two PWMs. And wire the two clocks to the two PWMs.

Now for some configuration. First lets setup the blinking PWM. Double click it… then change its name to PWM_BLINK. Set the compare value to 500 and the period to 999. Now change the input clock to 1kHz. And finally rename the LED to be RED. So, I'm sure that you guys remember from before that this will result in a 1Hz blinking LED.

Let's follow almost the same process for the other PWM circuit. First change the name of the pin to GREEN. Then change the name of the PWM to be PWM_DIM. Change the period to 100 and the compare value to 0. We will leave this clock at the default 1MHz value so that the LED will appear dim instead of blinking. This circuit will give us a dimmer… and by changing the value of the compare from 0 to 100 it will change the brightness from off to all the way on. Cool.

Now let's configure the BLE. On the general tab I am going to run this BLE in dual processor mode… just like I did in the last example. The device is going to be a peripheral and only allow 1 connection.

Click on the GATT settings. Then right click on the server and select add service… notice that there is no killer robot service in the list of predefined services… maybe I'll send a note to Misha to get him to add it… just joking… In this case we are going to create a custom service… so select “Custom service”.

Right click on the custom service and select rename .. then change the name to LED. Now let's change the name of the custom characteristic to be green. On the other side… remember the GAP central or phone side… we want them to be able to write a uint8 so leave this set as uint8 and click “write” so that the permission will be set as writable. Next let's add some information so that the other side knows what “green" means. Right click and add descriptor … characteristic user description… then let's type a description like “Green Brightness 0-100”. Finally, delete the custom descriptor as it is not needed.

Now let's configure the GAP settings. First give this bad boy a name… how about P6LED. Then change the advertising settings to General Discovery mode and set it so that it never times out (which wastes power… but makes it easier to find).

The last step in the BLE configuration is to setup the advertising packet to have the device name and the fact that it has a custom service.

Now we will go to the Design Wide Resources and configure the pins. Set the UART to P5[0] and p5[1] then GREEN to p1[1] and RED to p0[3]

OK… run generate application to get PSoC Creator to do its magic.

Let's modify the FreeRTOS.h to get rid of the warning, include semaphores, have more heap and set the MAX_SYSCALL interrupt priority just like in the last example.

Then I'll modify the stdio_user.h to know about our project and which UART we are using.

In order for the system to run in dual core mode I need to add the BLE processing commands to the main_cm0p.c … first launch the controller part of the stack… then infinitely loop and process events.

And now we are ready for the main event… main_cm4.c.

At the top I need to have all of the required includes… project.h … FreeRTOS.h, etc.

Then I need to declare the two variables that will be used to signal from the BLE interrupt service routine… a task handle for the BLE task… and a semaphore called bleSemaphore.

Remember from the last video I told you that you need to build an event handler… well let's do it. In this case, instead of a separate generic event handler and a service specific event handler, the generic event handler will take care of everything. As I told you before, the BLE stack will call this function with an event code and an event parameter to tell you what is happening in BLE land.

I am going to make the code a little bit cleaner by making a variable called writereqparameter. I'll tell you about it in a second.

For this project there are four events that I am interested in.

Stack on … which happens when the stack turns on.

Gap disconnected … which happens when the remote device disconnects.

Connection indication … which happens when there is a connection.

And write request …which happens with the other side sends you a write request. This event will be generated when a write happens to the GREEN characteristic in our custom LED service.

As before … the event handler function is just a big switch.

So… when the stack turns on… OR when there has been a disconnection we are going to do exactly the same thing. Specifically, I start the blinking red LED PWM. Then I'll start advertising… and finally I'll reset the green LED PWM and then disable it.

The next case handles a new connection. In this case I'll turn off the blinking red LED PWM and startup the green LED dimmer PWM.

The last case occurs when the other side writes to my device. Up until now we have only looked at the event codes. Now we need to use the other parameter to the function which we declared as a void pointer… as you guys remember a void pointer is a generic pointer… it can point to anything…. Well in the write request case that pointer is going to point to a structure called the write request parameter. So, I will cast the void pointer to a pointer of type cy_stc_ble_gatts_write_cmd_req_paramter_t … which is quite a mouth full.

The write request parameter has a bunch of useful information in it… including which GATT characteristic was written by the central. So, the next thing that I need to do is make sure that the GAP Central was actually writing into the Green Brightness characteristic.

If it is… then I extract the value that it wrote…. Make sure that it 100 or less (remember the value from the PWM)…. Then it will update the compare value in the PWM which will change the brightness.

Finally, I have to call the WriteRsp function to…

Now that I have built the BLE handler, all we are left to do is make the BLE interrupt service routine, the BLE task and the startup code.

I'll start by copying and pasting that code from the last project as it is almost exactly the same. This time we don’t have an IAS callback… so I'll delete that from the BLE task. Then in main I'll start the two PWMs.

Now… the moment of truth. Hit the little chip button to build and program this dog.

All right… the light is blinking red… that’s good.

Last time I showed you CySmart on the iPhone. This time I'm going to run the PC version of CySmart. The application communicates using a BLE dongle that is included in the kit, so I'll connect that first and then run the application.

I'll connect to the dongle, and then start a scan. P6LED shows up so that's good.

When I select it and click Connect, the red LED turns off. Then, I'll click on Discover All Attributes so that I can see my GATT database. I can read the value for the Characteristic User Description at the bottom to see that it is the Green Brightness characteristic.

Then if I click on the characteristic value itself, I can write a new value into the characteristic. I'll start by writing hex 64 which is also known as 100 %... and look the green LED is full bright.

Now I'll try hex 10… yup dimmer…

And when I try 0 … the green goes off…

I'll put 64 back in for full brightness…

Now disconnect… and look the green goes off and the red starts blinking again. Cool.

Now we know how to build a custom characteristic… in the next video I'll add BLE to our main controller… specifically I'll add characteristics for the two motors so that we can change the position of our robot arm with a remote control.

You can post your comments and questions in our PSoC 6 community or as always you are welcome to email me at alan_hawse@cypress.com or tweet me at @askioexpert with your comments, suggestions, criticisms and questions.