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 a custom service I couldn’t find a terminator robot service definition anywhere in the spec, and that's a problem because we know 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. When there is a connection, the other side - also called the GAP Central - 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, let's 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 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 – the DIM means dimmer. Change the period to 100 and the compare value to 0. We'll leave this clock at the default 1MHz value so that the LED will appear to dim instead of blinking. This circuit is a perfect implementation of a dimmer… and by changing the compare from 0 to 100 it will change the brightness from completely off to all the way on. Alright, that's pretty cool.

Now let's configure the BLE. On the general tab I'm 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 and get him to add it as a default… alright, just joking again Misha. In this case we are going to create a custom service, so select – let’s see - “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 the 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 a descriptor … characteristic user description… then let's type a description like, oh say “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 so that it will never time out (which wastes a bunch of power but it certainly 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 – the LED 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 warnings, include semaphores, have more heap and set the MAX\_SYSCALL to a good priority.

Then I'll modify the stdio\_user.h to know about our project – the project.h. - and which UART we are using – UART\_1\_HW.

In order for the system to run in dual core mode I need to add BLE processing commands to the main\_cm0p.c … first launch the controller part of the stack… then infinitely loop and process events.

OK, 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 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 that. 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 just a second.

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

Stack On … which happens when – oh yeah - 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 statement.

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

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

The final 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 just a generic pointer… it can point to anything in memory. Well in the case of a write request that pointer is going to point to a structure called the write request parameter. So, I'll cast the void pointer to a pointer of type cy\_stc\_ble\_gatts\_write\_cmd\_req\_paramter\_t. That's quite the mouthful – it only took me 4 times to say it in this video.

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 to the Green Brightness characteristic.

If it is then I extract the value that it wrote, make sure that it less than 100 (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 send the response. You might remember that in BLE there are two kinds of writes – a write with response and a write without response. In this case we've implemented the write with response.

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 - it really is almost exactly the same. This time we don’t have an IAS callback… so I'll delete that from the task. Then in the main I'll start the two PWMs.

And now, finally, if you are ready for the moment of truth, hit the little chip button and build and program this dog.

All right, look, this is good - the red LED is blinking – no connection.

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

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

When I select and click Connect, the red LED turns off – OK – that makes sense. 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%. Look the green LED – it's full bright – hey that's good.

Now let's see here, I'll try hex 10… yup it's dimmer…

And when I try 0 … the green LED goes completely off…

I'll put 64 back in for full brightness…

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

Now that 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 the robot arm with a remote control.

Alright boys and girls, as always you can post your comments and questions in our PSoC 6 community or you are welcome to email me at alan\_hawse@cypress.com or tweet me @askioexpert with your comments, and your suggestions, and your criticisms or your questions. Hey, thanks again and I hope you are having fun.