BLE - MainController

Welcome back to Cypress Academy, PSoC 6 101. In the last lesson I showed you how to build a BLE Peripheral with a custom service for an LED Dimmer. In this lesson I think that I'll add BLE to the MainController project and modify it from LED dimming to motor control.

First, let's add the BLE component to the schematic from our MainController project. Then configure it for peripheral, 1 connection, and dual core. Once that's done I want to create a motor service… well actually I already did that for you. So, from the GATT Settings tab, right click on the Server and select Add Service From File. Pick out the Motor.service file and hit go… PSoC Creator will load the configuration I created. When you look at it, you will see a custom service called Motor. The motor service will have four characteristics… 2 of which we will use to set the position of the motor… and 2 of which that you will use to make relative position changes. The M1 and M2 characteristics are uint8, and they're configured for read, write and notify. The M1\_REL and M2\_REL are write only. Notice the M1 and M2 characteristics each have a characteristic user description as well as a client characteristic configuration, also known as a CCCD. These are related to notifications, which is something new that I'll tell you more about in a later video.

Now let's setup the GAP settings. Give this beast a name… how about P6ROBOT. On the advertising settings set the Discovery mode to General and no timeout on the fast interval.

Then in the advertising packet let's advertise the local name and the motor service UUID… this will let our remote control find us… and I'll tell you more on that in a future video.

I told you in the previous videos I like to use LED9 to show that there is an active connection… so we'll add a digital pin called LED9, no hardware connection and initialized high… then in the DWR I need to assign it to P13[7].

Next… run generate application to pull in all of the BLE Middleware and let PSoC Creator get everything all connected up for us.

First, we need to make the MAX\_SYSCALL\_INTERRUPT\_PRIORITY change in FreeRTOSConfig.h just like in the previous project.

Remember from the previous video we want to run the controller portion of the BLE in the CM0+, so edit main\_cm0p.c, start the BLE then process events in the main loop.

Now I will create bleTask.h. It will have a pragma once and a definition of the bleTask.

Onto the main\_cm4.c. I'll add the include for the bleTask.h and then startup the BLE Task.

And last but not least I need to actually do the BLE work by creating bleTask.c. This task is going to be a lot like the SimpleBLEPeripheral BLE Task. So I think I'll just start by copying from that project. I will copy from the top all the way to just before the main function.

At the top I'll include global.h.

Now we need to make some changes to the BLE event handler. First instead of just blinking the LED we are just using LED9. So there's no connection and then it'll be off, and when there is a connection it will be on. So let's see here, Cy\_GPIO\_Write(LED9\_PORT,LED9\_NUM,1); …OK now let's delete the TCPWM stuff, and when there is a connection turn on LED9.

Now the write request stuff is going to have to be redone but for now I am going to put and #if 0 and an #endif and see if it will start advertising and we can connect. Obviously we won’t be able to write to the characteristics, but that is OK, it is a good place to start. So, I hit program and let it rip.

We will use the Android version of CySmart this time, so you can see what that looks like. When I startup CySmart, I see that there's a peripheral called P6ROBOT. It's advertising... that’s good. And when I connect… LED9 turns on. I can then open the GATT browser, and I'll see and select the custom service. I can see that there are 4 available which is what it should be based on our service configuration.

All right, now back to the bleTask.c.

First let's add an include for the pwmTask so that I can send it messages.

Next, I want to make a function called updateMotorGatt that will be responsible for actually writing the values into the GATT database… meaning changing the characteristics for the motors.

There are two possibilities for changing the position of the motors. It could happen locally because the user touched the CapSense … or typed on the UART … or it could be getting a command from the BLE Central that's attached. Either way the GATT database needs to be kept up to date.

The reason that the GATT database needs to be kept up to date is because the remote control end wants to be able to read the current position of the motors. So, let's build this function that it can make updates to the GATT database that are initiated locally … for instance by the PWMTask … or remotely... from the BLE Central.

The function takes a motor… remember M1 or M2 from before… a value, and a mysterious flag… actually the flag will tell the system if it is a local write… meaning the motors changed positions locally… or a write from the BLE central side.

When you call the Cy\_BLE\_GATTS\_WriteAttribute function you need to give a pointer to the “cy\_stc\_ble\_gatt\_handle\_value\_pair\_t”… so I'll declare one of those beasts.

Then I'll error check the percent to make sure it’s in range. Then figure out what the handle is of the characteristic we're talking about… either M1 or M2.

Once that is done I'll figure out if it is a local or a remote write. If it's from the peer… or BLE Central side… I'll first write it into the GATT database… then I want to send a message to the PWM that the value of the motor has changed… so I'll build up that message… and I'll send it.

If it is a local write… then I just need to write into the database.

Now… remember earlier I told you about the CCCD… well here is where it comes into play. It's possible for the central side to ask to be notified if the value of a characteristic changes... and remember that we setup M1 and M2 with this notify capability. The CCCD characteristic is where the GATT database keeps track of whether or not the central wants to be notified for a particular characteristic. So, after we write the attribute to the database we need to call the CyBLEGATTSNotification function which will figure out if they have asked for notifications and then send the notification if they have.

Remember I used #if to exclude a section of the BLE event handler. Let's go put it back in.

When you get a CY\_BLE\_EVT\_GATTS\_WRITE\_REQ event there are 6 possibilities. The central wrote an M1 or an M2 … the central wrote an M1 relative or an M2 relative … or lastly the central changed whether or not it wants to be notified of changes to the M1 or the M2 characteristics – in other words the CCCD.

So, let's deal with these six possibilities.

If it is a write to M1 then we will use the handy dandy helper function updateMotorsGatt. Then we'll do the same thing for M2.

If the Central asked for a relative change in M1 then let's make a PWM message that requests a relative change and send it.

And what do you know… it's the same thing for M2 relative.

And finally, if the BLE central changes the CCCD of M1 or M2 then let's make one of these crazy cy\_stc\_ble\_gatts\_db\_attr\_value\_info\_t pairs and fill it out with the connection and value and then write it into the database using the function Cy\_BLE\_GATTS\_WriteAttributeValueCCCD.

That’s it… oh hang on… for all of these cases we need to send a write response so call Cy\_BLE\_GATTS\_WriteRsp.

Now that we've fixed up the BLE event handler you need to make one little change the event loop. Specifically, after you have been woken up you should check to see if the PWM value has changed locally… which you can tell by using the event bits… and if the values have changed then you should update those values of M1 and M2.

All right, hit program and let's try this thing out.

I'll start up CySmart again, connect to the robot, open the GATT database browser, and select the custom service. We still have our 4 attributes available. The first two - M1 and M2 - are read/write/notify and the last two - M1\_REL and M2\_REL - are write only. This makes sense because that's how we defined it. The phone wants to be able to read the current position of the motors from the first two characteristics but reading a "relative position" of the motors is mostly meaningless.

Let's select the first characteristic which is M1 and read its value. Next we'll click on the write button to enter a new desired position. Let's say 0x50 and click OK. Look, the arm moves – that's excellent! Now go back and try the same thing with the second characteristic which will move motor 2.

Alright, that's cool! Now we can move the robot remotely from a BLE connected phone. In the next videos I'm going to show you how to program a second PSoC 6 BLE Pioneer Kit to be a central device. It will then be able to act as a remote control for the robot instead of using a phone.

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