3-2b 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. Then configure it for peripheral, 1 connection, dual core. Once that is 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 in 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 that you will use to make relative changes to the position. The M1 and M2 characteristics are uint8 that is read, write and notify while 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 later in this 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 advertising.

Then in the advertising packet let's advertise the local name and the motor service UUID… this will let our remote control find us… 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 add a digital pin called LED9, no hardware connection and initialized high… then onto the DWR where I need to assign it to P13[7].

Next… generate application to pull in all of the BLE Middleware and let PSoC creator get everything connected.

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 BLE in the CM0+ … so edit main_cm0p.c … start the BLE and then process events in the main loop.

Now I will create bleTask.h … it will have 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 that I'll 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 a blinking LED…we are using just led9…. So when there is no connection it will 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); … now 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, 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 a peripheral called P6ROBOT is advertising... that's good. And when I connect… led9 turns on. I can then open the GATT database browser, and then I'll 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, back to 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 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 UART commands … or it could be getting a command from the BLE Central that is attached to it. Either way the GATT database needs to be kept up to date.

The reason that the GATT database needs to be 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 so 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 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 a “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 the handle of the characteristic we are talking about… either M1 or M2.

Once that is done I'll figure out if it is a local write or a remote write. If it is 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 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 is 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 that I used #if to exclude a section of the BLE event handler… now let's go put it back in.

When you get a CY_BLE_EVT_GATTS_WRITE_REQ there are 6 possibilities. The central wrote M1 or M2 … the central wrote M1 relative or M2 relative … or lastly the central changed whether or not it wants to be notified of changes to the M1 or M2 characteristics.

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 do the same 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 then send it.

And what do you know… the same for M2 relative.

And finally, if the BLE central changed the CCCD of M1 or M2… then let's make one of these crazy cy_stc_ble_gatts_db_attr_value_info_t … 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 have 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 it has… then you should update the 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 the central – that is, 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 meaningless.

Let's select the first characteristic which is M1 and read its value. Next click the write button, enter a new desired position such as 0x50 and click OK. The arm moved – excellent! Now go back and try the same thing with the second characteristic which will move motor 2.

Cool! Now we can move the robot remotely from a BLE connected phone. In the next few videos I'll 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 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.