BLE Intro – CySmart to Robotic arm

Welcome back to Cypress Academy, PSoC 6 101. In the chapter 3 videos, I will show you how to add BLE connectivity to the robotic arm project we've been working on. That will include both adding BLE to the MainController as well as building a remote control. Don't worry… we'll start simple and then build up from there, so let's get started.

For this lesson, you will need either a BLE-enabled smart phone and an app from Cypress called CySmart; or the PC version of the CySmart application and the CySmart BLE USB Dongle that's included in the PSoC 6 BLE Pioneer Kit. Whichever version of the CySmart application you choose will be used to read and write BLE control signals to the PSoC 6 BLE robotic arm.

In this video I'll build the equivalent of hello world project… actually, the blinking LED project… for Bluetooth Low Energy. That is also known as the Find Me project.

Let's start by creating a project called 3-1-BLEFindMe. This project will have one task for BLE called "bleTask" which will manage … get this … the BLE. And it will run all of the required stuff to be an immediate alert service or IAS. The immediate alert service is specified by the Bluetooth special interest group and is a simple peripheral that you can tell to be in High Alert, Mild Alert or No Alert. Think of a tag that you could attach to your car keys. When you lose them you could get on your phone and set the tag into one of the three states, for example it could beep and flash in High Alert… just flash in Mild Alert, and be off in the No Alert state.

I am going to use printf and FreeRTOS. So, change the build settings to enable both of those libraries … just like almost every chapter so far.

In FreeRTOS.h, turn off the warning, increase the heap size, enable semaphores and modify the syscall priority.

Once you have that done, let's edit the schematic. First add the BLE component, then a UART component, then four digital output pins … one for LED9 (to indicate a connection), then one for the red, green and blue LEDs to represent the three alert levels – none, mild and high.

Go ahead and change the name of three of the digital output pins to led9, red, blue and set their initial state to high … also known as off. Then set the last pin to be named green and set its output state to low … which will cause the green LED to be on by default indicating the initial state of no alert.

Once you have that done let's go ahead and assign the pins in the DWR… first the UART to P5[0] and P5[1] then blue to P11[1], green to P1[1], led9 to P13[7] and finally the red to P0[3].

Now we need to configure the BLE. This device is going to be a “GAP Peripheral” that accepts one connection. As I have talked about earlier, one of the cool things about PSoC 6 is that it has two cores … remember a CM4 and a CM0+. Our BLE implementation will let you run the controller part of the BLE stack on the CM0+ and the rest on the CM4. To do this you just need to select “dual core”.

Once you have that done… let's edit the GATT Settings. The immediate alert service is specified by the Bluetooth SIG. We made it easy for you to add to your project. Just right click on the “server” and select add service… then immediate alert.

Now that our GATT database is setup, we can configure the GAP settings. First, let's name this device “FindMe” … then setup the advertising settings… let's pick general discovery mode, and no timeout on the advertising….

Now let's configure the advertising packet to have the name of the device as well as the service UUID of the alert service.

That's it for the schematic. Now run build application to bring in all of the middleware and setup all of the connections.

In order to make this work you are going to do something new… that is edit the cm0p.c file. In this file, I need to start the BLE by calling Cy_BLE_Start… then in the main loop I need to call Cy_BLE_ProcessEvents.

With all of that done we are now ready to edit the main_cm4.c to have the main firmware.

First let's setup the includes … project.h … standard io… Freertos.h … task.h … semphr.h. and limits.h.

The LEDs on the board are active low… so I'll setup macros for them. LED ON is a 0 … and LED Off is a 1.

Then we need a semaphore for the BLE interrupt called bleSemaphore and the taskHandle for the BLE Task.

The way that our Bluetooth stack works is that you provide functions which can process events. We call these functions “callbacks”. When something happens that we think is interesting we will call your function with a parameter that will tell you what event occurred, then you need to do the right thing.

For this project we will need to create two callbacks. One for handling the generic Bluetooth events and one for handling the immediate alert specific events. First let's tackle the generic event handler.

Let's declare the callback function …. It will return a void … meaning nothing … I'll call it the genericEventHandler … that makes sense … the Bluetooth stack will pass it an event code which is just a uint32_t and it will give it a void pointer to an event parameter.

All of these event handler functions look about the same. They are just a big switch statement that look at the different kinds of events that can occur in the system. For this example, there are only three events that we care about. When the Stack turns on, when a device is connected (like your phone) and finally when a device is disconnected.

First, when the stack turns on, also known as CY_BLE_EVENT_STACK_ON. For this case all I want to do it printout a message that the stack has turned on… and then I want to start advertising. When I call the start advertisement function, our device will start sending out the advertising packets over the radio. This will let other devices see that we are there, it will tell them our name and will tell them that we have an immediate alert service. Remember earlier we specified all of this information in the BLE customizer on the GAP Settings page.

The next event is the device connected event. When that happens I'll printout a message that we are connected and then turn on led9 to indicate that we have a connection

The last of the generic events is the disconnect event. When that happens I'll printout a disconnect message, turn off led9 and then tell the BLE to start advertising again.

That it for the generic events.

The next function is the event handler for the immediate alert service. Just like the last handler it returns a void … and I'll call it iasEventHandler… the BLE stack passes it an event code and an event parameter.

First, I will declare a uint8_t called alertLevel that I will use to temporarily hold the alert level.

Then I will see what event has been sent to me. It turns out that there is only one possible event, specifically CY_BLE_EVT_IASS_WRITE_CHAR_CMD. In other words, the only event that can occur is that the other side… the phone side … sent me a new value for the alert. I built this function so that if one day in the future there is a new event created I can easily add the ability to handle it to this function.

So, I will call the GetCharacteristicValue function to find out what value the phone sent me. I tell this function that I want the alert level and I tell it to store it in the alert level variable which is one byte long.

Once I know the alert level I can decide what to do using a switch statement. There are three possible alert levels: none, mild and high… all I do I printout the message … then turn on the correct LED.

Simple.

The next step is to build an interrupt service routine. This function will be called every time you need to call the process events function. The problem is that you don't want to call the process events function inside an interrupt service routine. So, what are you going to do? Simple. You are going to give a semaphore which will indicate to your main loop that it needs to call the process events fuction. Now let's look at the ISR. All it does is give the semaphore, and possibly yield to a higher priority task.

You could have skipped all of this ISR business and just have an infinite loop that repeatedly calls the process events function – like we did in the CM0p file. The problem with that is most of the time it wouldn't do anything and as such would be a complete waste of CPU time. Really, we would rather only call it when there is something to do.

The next part of this system is the main bleTask. This task is responsible for initializing the BLE, registering the callbacks, and then running at the right time and calling the process events function.

Declare the task, then printout a message that the BLE task has started and initialize the semaphore that will be used in the ISR. Then start the BLE stack and register the generic event handler callback function.

Now you need to get the stack started. This requires a number of events to be processed… so we will have a while loop that runs process events until the stack is on.

Once the stack is on we can register that we want callbacks when something needs to be done by giving it the ISR, and we want to be called back when the immediate alert service is written to so we register that callback function too.

Finally, we will build the infinite loop. This loop will wait until the semaphore is set in the ISR, then it will call process events. This is cool, because the task will go to sleep until there is something to be done.

That's it for the BLE task.

The last part is the main function, which just initializes the UART, creates the BLE task and then starts the RTOS scheduler.

Now we can build program and test.

When you program the kit, the first thing you see is the green light turn on. That's good. Now you can start CySmart on your phone… pull down to refresh the list of devices…

where I am standing right now I can see a bunch of different Bluetooth devices… but the one that I am interested in right now is called “FindMe”.

When I click on FindMe, you see led9 turn on indicating that we have a BLE connection. Now I can swipe over to the FindMe profile in CySmart. And when I click on that it gives me the option of setting mild alert… hey look the LED is blue… that’s good. Then high alert… and yes the red LED turns on… and if I put it back to no alert I go back to green.

All that seems to work…

Finally, I click back to disconnect and as soon as I do, led9 turns off indicating that it no longer has a Bluetooth connection.

That’s it.

In the next video I am going to show you how to build a custom BLE service for CapSense that we will then integrate into our MainController for the robot.

As always, 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.