PSoC Academy: How to Create a PSoC BLE iOS App
Lesson 4: Write the Firmware

00:00:09 Hello. I'm Alan Hawse. Let's get right into it. At this point we've created the schematic and configured all the components. We've set the pins and now we're ready to work on the firmware. I'm going to just go through the firmware slowly. I'll explain what all the pieces and parts do, and then you'll be able to copy it for yourself, because we're going to post it on Cypress.com. So, first of all, I create a uint16 that will represent the finger position on the slider.

00:00:40 This will be a global variable, so it's accessible inside of the BLE firmware. The second variable I create is the capsenseNotify. This will be set to 0 when notifications are off, and it will be set to 1 when notifications are on. The next thing I'm going to do is I'll create a function which I'll call updateLed. This function, when it's called, will update the GATT database with the current state of the LED.

00:01:10 First thing it does is, if the BLE is connected – and you can find that out with the CyBle_GetState function – if it's not connected, then you don't want to update the GATT database, so you'll just immediately return from this function. If it is connected, then you want to update the CYBLE_LEDCAPSENSE_LED_CHAR_HANDLE. Now, remember, when you set up the profile and the services and the characteristics in the GATT database, inside of the BLE

component, we named the LED characteristic led.

00:01:52 When you do that, Creator will automatically build a pound define that will reference you into that table – CYBLE_LEDCAPSENSE_LED_CHAR_HANDLE. Once you do that, you read the value of the pin with the read function and you flip it. The reason you flip it is the LED is active low, which means it's 0 when the light's on and it's 1 when the light's off.

00:02:22 But, as far as the users on the other side, it makes more sense to send a 1 to turn on and a 0 to turn off. So, I'll do the handy dandy bang to flip the state. The next thing you do is you tell it that it's one byte, and then you actually write it into the GATT database with the Cyble_GattsWriteAttributeValue function.

00:02:49 So now we have a helper function, updateLed. This function can be called anywhere in your source code, and all it does is, if it's connected it reads the stage of the LED and writes it into the GATT database. The next function that we need is the exact same thing for CapSense. So, we'll start by checking to find out if we're connected. If we're not connected, then just return out of this thing, because there's no need to update the database when you're not connected.

00:03:20 If you're connected, it will fall past that if statement and you'll just want to update the database. So, in the above function, we did

LEDCAPSENSE_LED_CHAR_HANDLE. This time we'll want to talk to the CapSense handle. And Creator created a pound define for you, just like it did in the above case. This is based on the name you give it when you configure the characteristic when you're setting up the component.

00:03:52 The finger position was the global variable that we defined at the top of the program. This is a 16-bit unsigned integer, and so first thing you need to do is you need to cast it into an unsigned 8-bit integer pointer, then you need to tell it it's two bytes, and then you copy it into the database with the GattsWriteAttributeValue function, just like you did above.

00:04:17 The difference in this function and the previous function is, do you remember when we set up the component, specifically when we set up the CapSense, we clicked on the notify button. Notify is a feature of BLE where the other side, the phone side, can say, every time you get a change in the value, I'd like to know about the value. In other words, it can register for notifications of changes in the value.

00:04:49 I have a global variable called CapSenseNotify which gets set when the Client Characteristic Configuration Descriptor gets set, and if it's set, and the CapSense has changed, then I'll send out a notification. So at this point I've created two helper functions, one

to update the database with the LED value, one to update the database with the CapSense value. Now on to the next section.

00:05:20 When you start up BLE inside of Creator, which I'll show you in the main loop, the next thing you have to do is you have to tell it, what is the event handler. And so we're going to create a BLE event handler. I call this event handler the BleCallBack. So, the BleCallBack is essentially a giant case statement which allows me to deal with the different kinds of events that get called from the BLE firmware.

00:05:50 The first two events that I'll handle are the stack turning on, or, the stack being disconnected. In other words, when you first start going, that gives you the event CYBLE_EVT_STACK_ON. And then after you've been connected and you get a disconnect, that event is called CYBLE_EVT_GAP_DEVICE_DISCONNECTED. I'll handle these two events exactly the same way.

00:06:19 And what I'll do is, I'll turn off notifications, I'll start the advertising function again, and I'll start the PWM that drives the blinking blue LED. So there's two situations where you want the blue light to blink. You want it to start blinking when the chip first turns on to indicate that it's disconnected. The other time you want it to start blinking is after you've been disconnected by the remote side, the iPhone side.

00:06:49 So when you've gotten disconnected, the BLE stack will send you the CYBLE_EVT_GAP_DEVICE_DISCONNECTED message. And when you get that message, you start the PWM going so the blue light blinks. The next event that we'll need to handle is, what happens when you've gotten a connection established from the remote side? Well, the first thing that you need to do is you need to update the GATT database with the current state of the LED.

00:07:20 So, it's convenient, we wrote a little function, grabs the state of the LED and puts it in the GATT database. So we'll call that function. The other thing that we need to do is we need to update the GATT database with the CapSense value. This is so the remote end can read the state of the LED and it can read the state of the CapSense. That's handled automatically for you by our BLE stack firmware which is completely free and comes with our BLE component.

00:07:50 The other thing we want to do is, after you've gotten a connection established, you want to turn off the blinking blue light. So, the light will blink blue when you're disconnected, and it will be turned off when you're connected. The next event that we can get is a Write event. This event is called CYBLE_EVT_GATTS_WRITE_REQ. This event gets called when the remote side wants to write into your GATT database.

00:08:19 So, when it asks to write into your GATT database, our stack will tell you which of the characteristics it's trying to write. And what you need to do is you need to look at those characteristics and do something slightly different based on which characteristic is training to be written. So, in the first case you say: Is the LEDCAPSENSE_LED_CHAR trying to be written? In other words, did the remote side try to change the state of the LED?

00:08:49 And if the answer to that is yes, then, go ahead and update your database with whatever the remote side did by calling the GattsWriteAttributeValue, just like you did in the UpdateLED function. The other thing you want to do is, when they write into your attribute for the LED, into your characteristics specifically for the LED, you want to turn on the LED or turn off the LED, based on what they wrote. Remember I told you the red LED is active low.

00:09:21 So when they write a 1, you want it to turn on, and when they write a 0, you want it to turn off – except for the fact that when you write a 0 it'll turn on. So you need to use the bang to flip the state of what they wrote. Okay. So we're talking about the different Write requests. The first Write request is if they try to write the LED characteristic. The second Write request is if they try to change the notification flag for the CapSense characteristic.

00:09:53 So that is called

CYBLE_LEDCAPSENSE_CAPSENSE_CAPSENSECCCD_DESC_HANDLE. Remember in the previous case we changed the name of the characteristic CapSense descriptor to be capsensecccd so we wouldn't have to type as much. When that happens, you want to go ahead and write into the GATT database with the WriteAttributeValue function, and then you want to turn on the flag, or turn off the flag, based on whether they turned it on or turned it off.

00:10:27 This way, when you call the updateCapsense function other places in your firmware, if they've said that they're interested in notifications, the notifications will be sent out. So we'll modify the global variable that we're using to keep track of whether the CapSense Notify is turned on or off. The last thing you need to do, the BLE spec says that when you get written to, you need to give a response.

00:10:52 So, in either one of the cases, whether they do the LED or whether they do the CapSenseCCCD, you need to send a response. And you do that with the CyBle_GattsWriteRsp function. So at this point we've got the update the LED function, we've got the update the CapSense function and we've got the BLE stack event handler function. The last function that you need is the main function, and it has two basic parts.

00:11:21 It has initialization and it has the main loop. In the initialization, you'll need to turn on the global interrupts. This is done with the CyGlobalIntEnable macro. All of these things that we're doing, both BLE and CapSense, are interrupt-based, so you need to have the interrupts on. The next thing you need to do is you need to start the CapSense component, and then you need to get the CapSense component to initialize its baselines.

00:11:51 Initializing the baselines is a function that gets rid of the baseline noise that's in your system as well as deals with the parasitic capacitance attached to the sensors, the slider that is attached to your pins. So that's the first step of getting the CapSense going. Then, the main loop is very trivial. If the CapSense is not busy – in other words, if the CapSense has completed its scanning process - then grab the finger position using the GetCentroidPos function, update your baselines, start the scan again, and then call your helper function that we talked about earlier.

00:12:30 That function will take the finger position and it will write it into the characteristic for CapSense if you're connected. If it's not connected, it doesn't do anything. So, we've dealt with the CapSense, and you need to do that every time through your infinite loop. The next thing you need to do is you need to give the BLE stack the opportunity to process its events. That's done with the

CyBle_ProcessEvents function – very easy.

00:12:58 And then the last thing you want to do is, you want to tell the BLE that it's welcome to go to sleep if it can, in which case you'll save power. You do that with the CyBle_EnterLPM. That's it. That's your whole loop. So, from the top, we have a function called updateLed which will update the GATT database with the state of the LED. We have a function called updateCapsense which will update the GATT database with the state of the CapSense.

00:13:28 We have a BLE event handler that will handle the stack turning on or a disconnection, and it will handle writes of the characteristics from the iPhone side. And then the last thing is we have the main loop, where we initialize the CapSense, run the CapSense in the infinite loop and process the BLE events. That's pretty simple.

00:13:53 In the next video, I'll show you how to debug your firmware and make sure that it's working with CySmart.