CapSense

Welcome back to Cypress Academy, PSoC 6 101. In this video, I will show you how to add a CapSense capacitive-sensing interface to the project we have been working on to provide a local interface on the PSoC 6 BLE Pioneer Kit to control the robotic arm.

CapSense is Cypress’ capacitive sensing technology. It’s the same type of technology that nearly everyone uses every day, it’s in your smart phone, wearable device, stove, refrigerator, smart thermostat, it’s everywhere. It enables designers to remove clunky mechanical buttons and switches delivering a sleek and sexy interface. What’s great about Cypress’ CapSense solution? It’s awesome because of how easy it is. Now everyone asks me why it’s so easy and it’s really not that difficult to answer…it only took us a couple dozen guys…years…millions of dollars…

The bottom line is, CapSense is built on over 16-years of research and development, experts optimizing and perfecting the algorithms, sensing technologies and IP, all in an effort to provide you a solution that just works out of the box. Don’t just take my word for it, you’ll see for yourself in the following lesson.

Let’s start with a new project that I’ll call “BasicCapSense”.

In this project I want to use the capacitive slider and buttons on the PSoC 6 BLE board to act as a dimmer and on/off switch for the RED LED. I’ll also use the Blue LED to indicate the status of the dimming function. If the blue LED is on, then dimming is turned off.

So, we’re controlling an LED, so we know we’re going to need our trusty PWM and LED setup that we’ve been using for the last several lessons—so let’s copy and paste that from the BasicI2C lesson’s schematic.

We also need an additional digital output pin for the blue LED. So, drag and drop that in or copy/paste the Red LED digital output pin. Double click and change it’s name to Blue and deselect the hardware control option.

Now we need a CapSense component. Drag and drop that into the design. Let’s configure that component. Starting by renaming it to CapSense, dropping the underscore 1. Now let’s add the linear slider and buttons. Click the plus sign, pick linear slider. Click the plus sign again and add button0, then again for button1.

Now, we’ll use the CSX Mutual-Cap scheme for buttons 0 and 1.

Now, let’s pause here for a moment…CSD and CSX are two different types of patented capacitive sensing schemes that Cypress has developed and perfected over the years. CSD is our self-capacitance mode and CSX is mutual-cap. We have lots more documentation and getting started guides which you can find right here <point>.

Next, click on the advanced tab, CSX settings sub-tab, and set the modulator clock frequency to 12,500. Then click on the widget details sub-tab. The two buttons share the same TX pin. So, click on Button1\_TX and change the selected pins setting to Button0\_TX.

Okay.

Now, let’s assign the pins for the two LEDs and CapSense widgets. The red LED is P0[3], blue LED is P11[1]. Because we have both a CSD and CSX set of CapSense widgets, the schemes require three capacitors that are connected to P7[1], 7[2], 7[7]. The two button RX pins are on P8[1] and 8[2]. The linear slider is a 5 element slider, so 5 pins connected to P8[3] to P8[7]. And finally, the button TX pin is connected P1[0].

Let’s generate the application.

And now the firmware. In the CM4 main application, let’s start the CapSense component, start the scanning, turn on the PWM.

Now, if our hardware block is not busy, then we’re allowed to ask it what the state is. So, check if it’s not busy; if it’s not busy, then process the widgets. Find the middle of the slider, the centroid position. If it’s being touched as indicated by a value less than 0xFFFF, then we’ll set the compare value of the PWM. Since the possible value of the slider is 0 to 100, it matches up nicely with the possible compare values for the PWM.

We’ll now check to see if someone is touching button 0. If it is being touched, then we’ll turn the PWM off and turn on the blue LED.

If someone is touching button 1, then we’ll turn the PWM on and turn off the blue LED.

Finally, we need to update the baselines, which represents the environment that the board is sitting in. This is where some of the magic happens. You want your board to be robust regardless of temperature, humidity, location, etc. This is key in wearables and many other IoT applications.

Then, we need to start the scanning again of the widgets.

That’s it. Build, program and test.

Look, when I move my finger on the slider you can see how the intensity varies. When I push button 0 the blue LED turns on and the slider is disabled. And when I push button 1 the red LED is back on and I can again use the slider to vary the LED intensity.

Sweet!

[Alternate 2-video version]

Now we have a basic CapSense implementation working. In the next video we’ll add this functionality to the BLE-controlled robotic arm project.

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.

Welcome back to Cypress Academy, PSoC 6 101. In this video, I will show you how to take the CapSense implementation we learned about in the previous lesson and add it to our BLE-controlled robotic arm project.

[1-video version]

Now, let’s put this functionality into the robotic arm controller.

[Merge]

Let’s start by copying the CapSense component from the Basic project we just did and pasting it into the MainController schematic.

Next, verify the pin settings by clicking on the pins file in the design wide resources. All the pins should be the same as the Basic project.

First, create capsensetask.h with pragma once and the function prototype for the capsense task.

next create CapSenseTask.c. You will need includes for project.h, pwmTask.h and global .h.

It will only have the capsensetask function which is defined the same was as all of our other tasks.

The slider is going to control the motor position of a given motor. I’ll use the buttons to select which motor the slider is changing.

So, create a variable called currentMotor to keep track of which motor we’re changing. Create a PWM message. This is one of the cool things about an RTOS, you can have multiple, independent tasks sending messages to other tasks—in this case capsense and uart sending PWM task messages.

Now, same as before, we’ll start the capsense component, scan the widgets. In the infinite loop, when the capsense hardware is not busy, we’ll process the widgets.

Next we’ll find the position of the linear slider. If the user is touching the slider, we’ll build a message that contains the slider position and which motor we’re changing; and send it to the PWMQueue.

Then depending on which button is being touched we’ll change the currentMotor variable to change which motor we’re tracking.

Now, update the baselines and start the scanning again.

Lastly, you need to start the capsense task back in main\_cm4.

Build, Program and test.

Now as I press button0, I can run my finger on the slider and change the position of motor 1; if I press button1, I can change the position of motor 2. Excellent!

Now we have CapSense working to control the robotic arm on the PSoC 6 BLE Pioneer Kit. In the next video, I will walk you through an extremely useful tool to debug RTOS applications using a tool called Tracealyzer by Percepio, a partner of ours.

[Alternate ending]

Now we have CapSense working to control the robotic arm on the PSoC 6 BLE Pioneer Kit. In the next set of videos, I will walk you through how to enable the BLE connectivity to begin controlling the robotic arm remotely.

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.