Motion Sensor Remote

Welcome back to Cypress Academy, PSoC 6 101. In the last video I showed you how to use the PSoC 6 I2C master to interface with the Bosch BMI160 accelerometer. Now we are going to add the motion sensor capability into our robotic arm remote control project.

We'll start out by adding the Bosch library to the MainController project. Make a folder called Bosch and add the existing items to it.

We'll change the build settings for the cm4… compiler…general… add additional include directories… then click add… then new…then add the bmi\_driver directory that we previously cloned from GitHub.

Now we'll add the I2C component to the schematic. Change it to a master.

The we'll add a digital output pin called LED8. This LED will be ON when you are in CapSense mode and OFF when you are in motion mode.

So, the next thing you need to do is assign the pins … P6[0] & P6[1] are the I2C pins. Next run the generate application.

To start the firmware, I am going to create motionTask.h. So, right click… add new item... then header file… then motionTask.h. First, I'll add #pragma once.

When I originally built this application, I had the CapSense task and the motion task running at the same time and independently of each other… I thought that this would be great… but they fought with each other… and the robot arm went crazy and I had to use that kill switch that we put in one of the earlier chapters. So, I decided that what I would do is make the system be in one of two modes, either the CapSense mode or motion mode. FreeRTOS has the concept of event groups which is essentially a thread safe global variable, so we'll use one of these.

The way this is going to work is that when the remote control is sitting still and flat for “a while” – let's call it 3 seconds – I'll put them system into CapSense mode. If you pick up the kit and move it around a little bit, it'll go into motion mode.

To implement this, you need to include the event\_groups.h, then make a definition for the event group called systemInputMode. This will actually be instantiated in the main\_cm4.c. Now I make a bitmask for the two modes. The double left arrow is the shift operator. Finally, let's define the motion task itself.

Now I want to update the main\_cm4.c to include the new stuff. First include the motionTask and the event\_groups.h.

Next, I'll make the variable for the event group called systemInputMode.

Then in main I'll initialize the event group, set the current mode to CapSense and turn off LED8. The last change in main is to start the motionTask.

Now, in capsenseTask.c I need to make two small changes. First, include motionTask.h and second, only call the writePositionFunction when you are in CapSense mode.

Finally, the main event of this whole video… I need to create the motionTask.c. I will copy from the basic motion sensor project's main\_cm4. Specifically, I'll copy all of the includes all the way through the top of main and then I'll paste it into motionTask.c.

At the top I need to add includes for bleTask.h and motionTask.h.

Now scroll all the way down to the motion task. I told you earlier that I will control the mode of the system based on the motion. If the board hasn’t been moved in about a second then I'll switch it into CapSense mode. So, I need to declare a variable which I will use to keep track of the last time the board moved… let's call that variable lastMovement.

In order to calculate the desired motor position, I want to consider the X and the Y acceleration, meaning how the board is being held. In order to make the math easier I will cap the acceleration at plus or minus 1G.

Next, I'll build a little routine that will convert -1G to +1G into 0% to 100% based on the angle the board is sitting at. Right over my shoulder you should see the maths that I did to figure this out.

The next bit of code is used to keep track of the last movement, meaning if the motor change is more than 3 percent from where it was previously, then update the lastMovement variable. It didn't feel like it was worth sending 1% increments over so I didn't send anything for a 0% change, a 1% change, or a 2% change.

Finally, if it has been more than 1000ms since the last movement, I'll set the system mode to CapSense and turn on LED8. Otherwise I'll set the mode to motion and turn off LED8.

If we are in motion mode then we'll send the motion information.

That's it. Let's test this thing. Let's see here - hit the program button and after a bit you can see the remote control turns on and then quicker than anything it connects. See the red light turn on and LED8 turns on indicating that the remote control is flat and not moving and we're in CapSense mode.

Now I'll slide my finger back and forth… and look the robot arm moves.

Now I'll pick up the remote control and look, it turns off LED8 indicating that it has moved to motion mode and you can see the arm move in both axes.

Now we have our fully implemented BLE remote controlled robot. In some of the later videos I'll show you how to add WiFi and cloud connectivity into the mix.

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