P6-RC3-5b-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 accelerator. 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…Add the existing items to it.

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 cloned from git previously.

Add the I2C component to the schematic…Change it to a master.

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 sensor mode.

Assign the pins … P6[0] & P6[1]. Then 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 the kill switch. So, I decided that what I would do is make the system be in one of two modes, either 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 those.

The way this is going to work is that when the remote control is sitting still and flat for “a while” it will put them system in CapSense mode. If you pick up the kit it will go into motion controller 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 main cm4.c. Now I make a bitmask for the two modes… the double left arrow is the shift operator.

Finally, I'll define the motion task.

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 a 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.

In the 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… I need to create 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… then 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 moved in about 1 second then I'll switch the system into CapSense mode. So, I need to declare a variable which will keep track of the last time the board moved… called lastMovement.

In order to calculate the desired motor position, I want to consider the x and 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 1 g.

Next, I build a little routine that will convert -1g to +1g into 0 to 100% based on the angle the board is at… right over my shoulder you should see the maths…

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.

If it has been more than 1000ms since the last movement, I'll set the system mode to CapSense and turn on led8. Otherwise set the mode to motion and turn off led8.

Finally, if we are in motion mode… send the motion information.

All right let's test this thing… hit program… and after a bit you can see the remote control turns on… connects… see the red light turn on… and led8 turns on because the control is flat and not moving.

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

Now I'll pick up the remote control… and look it turns off led8 indicating it is in motion mode and you can see the arm move in both axis…

So, 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.

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.