Basic Motion Sensor

Welcome back to Cypress Academy, PSoC 6 101. Now that we have the fundamentals down for interfacing with the thermistor, lets focus now on the motion sensor.

The motion sensor that’s on the E-ink display shield board is a 6-axes motion sensor from Bosch, the IMU160. To communicate with this sensor, a digital interface is required, so for this lesson, I’ll be using the I2C master component to communicate and receive data from this sensor. And Ill use the UART to print out the acceleration data.

When I start looking at an I2C sensor I always like to make sure that I understand how to talk to it. So Ill go get a datasheet from the Bosch website. Hey that is a nice picture. But I need the datasheet so Ill click Documents and Drivers. First Ill look at the datasheet to so what is going on. OK I get it… this is a normal register based device. On page 5 you can see there is a list of registers… and the CHIP ID register looks interesting… so I click it.

What this says that if I read the 8-bit value in I2C Register 0 I should get 11010001 also known as D1… But what is the address of the chip? Scan a little bit further down in the datasheet and lookey there.. on page 90 I find the I2C address of the chip is 0x68.

OK enough documentation… lets see if we can talk to it with the bridge control panel. Startup BCP … then click attach to the kitprog… then press list devices. Lookey there D0/68 shows up. The list devices but sends out all the I2C addresses and see who answers back. So D0/68 makes sense. Now lets see if the chipid register has the right value… lets write 68 0 then read 68 x stop … sure enough the chip respond back with D1. That’s good.

Now all I need to do is develop a driver that knows how to read and write all of those register…. No just joking. If you look back on the Bosch website you will see that they provide a link to github …. Which has a nice C-Driver. Sweet.

All right lets start this thing by creating a new project, I’ll call it Basic Motion Sensor. Let’s drag and drop the I2C component in our schematic. Then drag in the UART component. Next Ill set the pins, P6[0] & P6[1] for the I2C… and P5[0] and P5[1] for the UART. Then Ill go to the build setting and turn on STDIO and FreeRTOS … next Ill run generate application to assemble all of the firmware into a project.

Once that is done I need to modify FreeRTOS.h to get rid of the warning and increase the size of the heap.

Now I need to fix up stdio\_user.h so I can printf… ill include project.h and update the two macros to uart\_1\_hw … all right we are cooking with gas...

In order to use the Bosch driver the first thing to do is download it into my workspace by opening up a terminal, CD-ing to my workspace … then running git clone git@github.com:BoschSensortec/BMI160\_driver.git

I am running this on a mac… so I have git built in… but if you are running on a PC you can use Cygwin to git… or you can download a zip file

Now that I have the Bosch driver, I need to tell the compiler where it can find the include files. To that I:

1. Right click the project and change the build settings
2. Click on CM4 ARM GCC settings
3. Then compiler
4. Then general
5. I need to add the BMI Driver to the include path… so I click on Additional Include Directories.
6. Press the dot dot dot
7. Then click new
8. Then naviage to the include path… which will be dot dot BMI160 driver

Now I can add the actual files to my project.

First, Ill click on the CM4 and select add new folder … Ill call it Bosch

Then I click on my new folder… and right click add existing item… navigate to the right folder on my disk… then select the two dot hs and the dot c … this gets the files to be part of my project.

Now we are ready to write the firmware… so go to the main\_cm4.c … at the top add includes for FreeRTOS.h, task.h stdio and the bmi160.h

Then create a variable of type struct bmi160\_dev which Ill call bmi160Dev. This structure is used as the interface to your specific BMI160

Now that the driver is part of my project I need to create the Bosch HAL. There are two functions that you need to create. Once called BMI160BurstWrite which can write values via the I2C Master into the device…. And one called BMI160BurstRead which can read the values via the I2C master into your firmware.

Obviously, you can type this code from my screen… or if I were you I would go get it out of my PSOC Creator workspace. But its your choice.

First the burst write. It takes 4 arguments. The I2C address, the register you want to write, the data you want to write and finally the number of bytes you want to write. Ok this is pretty easy.

1. Send a start using the PDL function Cy\_SCB\_I2CMasterSendStart
2. Now send the register you want to write
3. Then for loop through all the bytes and write them using Cy\_SCB\_MasterWriteByte
4. Finally send a stop using Cy\_SCB\_I2C\_MasterSendStop

Now I need to create the read function. The way that it works is it sends an I2C start, then writes the address it wants to read… then it sends an I2C restart … then it sends I2C reads with an ACK until it is done reading… then it sends the final read and a NAK. And finally sends a stop.

Now I need to create a function to initialize the chip. Ill call it bmiInit. This function will

Wait for 10ms for the BMI to boot

Then ill setup the BMI structure with a function pointer to the read … then the write … then the delay function… and finally the I2C address of my BMI160.

Once the structure is setup, then I can call the initaizliation function.

Now I need to configure the chip… first Ill setup the GYRO, output data rate… range… and bandwidth.

Ill put it in normal power mode….

Then I setup the accelerometer part of the chip… first the output data rate to 1600hz … then the range … bandwidth … and power mode.

Next I call the function to set my configuration…. Finally wait 50ms for it to take effect.

After all of this junk Im finally ready to get some acceleration numbers. So ill create a task called motionTask.

It will startup the I2C Master

Then Startup the BMI160…

The driver library has a function called “vmi160 get sensor data” you have to pass a pointer to a structure for it to save the data of type struct bmi160 sensor data… so Ill declare it... this will return the acceleration for x,y,z as a counts integer between -32767 and +32768. I have it set at 2g so 32768 counts is plus 2G

Finally the main loop which will infinitely loop… first reading the sensor data, turning “counts” into g and finally printing it out on the uart.

Now that I have a task, I create the main, start the uart, create the motion sensor task… and finally start the scheduler.

Now build program debug…

When I start the terminal program I can see that with the kit sitting on my desk it is 0,0,1… when I turn it over I can see that it is 0,0,-1 … that makes sense as the earth is pulling on the kit with 1g… now turn it on one side… yup 1,0,0 and the other way -1,0,0… good.

In the next video ill add the accelerometer to the remote-control 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.