I2C in the Main Controller

Welcome back to Cypress Academy, PSoC 6 101. In this video, I will show you how to add the I2C interface to our BLE controlled robotic arm project. I’m going to use the bridge control panel as a dashboard to display the position of the motors. So, as we change the position of the motors you’ll see a graph in the bridge control panel showing where they are. Does that sound cool? Yeah, I think so!

Open the BLE controlled robotic arm project schematic. Add the EZ-I2C component from the catalog. Change its name to EZI2C and leave the rest of the settings as their default.

Assign the pins to P6[1] and P6[0] for SDA and SCL.

Generate the application.

Then we create and edit the ezi2cTask.h. This file will just have the #pragma once and a definition of the ezi2cTask.

Once that's done we can edit the CM4 main application. When the whole main controller is done there will be two ways to display updates to the position of the motors - specifically the EZI2C and the BLE. In order for everyone to know that there's been a change to the PWMs I am going to use an event group. An event group is essentially an RTOS safe global uint32 variable. All of the tasks can read the status of an event group without fear of a race condition. They can also wait for a change in the event group. This is how we are going to communicate from the PWM task to the ezi2cTask.

Let's get to editing. In main\_cm4.c, first include the ezi2ctask.h and the FreeRTOS event\_groups.h. Next, define the event group called pwmEventGroup. And finally in the main function I'll initialize the PWM event group by calling xEventGroupCreate and start the EZI2C task.

Before anyone can use the PWM event group I need to add it to global.h. In this file I'll add the include for the event\_groups.h, I'll add an extern for the pwmEventGroup and finally create a definition of the event group. Basically, a bit mask. The first bit will be for the I2C and the second bit will be for the BLE.

In pwmTask.c all I need to do is set all of the bits when the PWM has changed using the xEventGroupSetBits RTOS command.

Now let’s create and edit the file ezi2cTask.c. This is a fairly straight forward file. First include the project dot h, the FreeRTOS dot h, and the pwmTask dot h. Next make the function ezi2cTask. I’ll setup a variable to store the percent value of the motors so that the I2C master - the bridge control panel in this case - can use it to read and display the data. I’ll call it motorPercent, an array of two unsigned 8-bit integers representing the percentages of the two motors.

I’ll start the EZI2C component. Setup the buffer and mark it as read only. Then, start the infinite loop and then initialize the motorPercent array with the current value of each motor’s PWM compare value. Then I’ll wait for a change event in the group, and then do it again.

What happens is the task will go to sleep until the pwmTask changes the event group. When that happens, the task will wake up, then update the values in the buffer, and then it'll wait for another change.

That’s it! Slick isn’t it?

Alright, now build, program and test this beast. Remember, if you still have the bridge control panel open and connected to the kit, you need to disconnect the kit from the software by clicking on the disconnect icon in the bridge control panel. Then go back to the PSoC Creator and hit the program button again. Okay.

Now that we've got it programed, go back to the bridge control panel and re-connect to the kit.

Let’s setup the chart so we can graph the motor percent values. Go to the chart menu, select variable settings. For the first row, click "active" and type "M1" as the variable name. In the second row, click "active" and "M2". Okay.

Next in the editor I’ll type w 8 0 r 8 @M1 @M2 . What this does is it sets the offset pointer to zero then reads in two bytes which it then stores in M1 and M2. Now press enter a few times to see if it works. Alright, that's good.

Now just go to the charts tab and hit repeat.

You can change the values using your UART interface that we built before. Remember, press the keys o, p, and k and l in the terminal client. Alright, now as we change those values you can see the waveforms going up and down. Hey, that's what I call success.

Now we have an I2C dashboard interface for our robotic arm project, in the next video, I will walk you through how to add a custom digital logic component to implement the terminator-kill, or the safety-switch to the design that will safely stop the motors.

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 @askioexpert with your comments, suggestions, criticisms and questions.