| **VIDEO P6-2-3a-I2C Version 2** | | |
| --- | --- | --- |
| Cell | **Visuals** | **Audio** |
| 1 |  | Welcome back to Cypress Academy, PSoC 6 101. In this video, I will show you how to setup an I2C dash board for our BLE-controlled robotic arm project. An I2C control and debug interface is handy to understand because you will more than likely have multiple sensors or ICs in a typical system that will require this type of interface. Let’s start first by learning about the I2C peripheral and the EZ-I2C component. |
| 2 | SCREEN CAPTURE:  P6-2-3a-I2C\_capture1.trec | Let’s start with a new project. I’ll call it “BasicI2C”.  Again, I love copying and pasting as it saves so much time, so let’s go back to the schematic view of the BasicTCPWM project we did in the prior lesson. Let’s copy the entire schematic and paste that into our BasicI2C project.  I want to show you an additional feature of the TCPWM component, so let’s open up the configuration dialog. Click on the advanced tab. Here you can see more configuration options like the kill and start input, which we’ll talk about in the next lesson, as well as how you can change the polarity of the PWM output. In this case, because the LED on the kit is active low, and I want to use larger duty cycles for a brighter LED, because that just makes sense, I’m going to invert the output of the PWM. Now hit okay.  Now, let’s add the EZ-I2C component from the catalog. Double click it and let’s rename it to EZI2C without the underscore 1. You can also see the other configuration settings like the address and data rate. Click okay and now on to configuring the pins.  Let’s make sure we assign the pin for the LED to P0[3]. Then SDA, the data signal, assign that to P6[1] and SCL, the clock, to P6[0]. |
| 3 | SCREEN CAPTURE:  P6-2-3a-I2C\_capture2.trec | Now on to the Cortex-M4 main application.  Let’s start the PWM and EZ-I2C components with the simple start API call. |
| 4 |  | The EZ-I2C component implements an EEPROM I2C scheme. In this scheme, the first byte written on an I2C write command is an offset location followed by the data to be written starting at that location. Reads use the same offset location as the previous write. This allows a single I2C slave to have 256 different register locations that can be used to store data. You can also use a 2-byte offset which allows 64K locations. This scheme is interrupt driven and designed to easily setup an I2C slave in your firmware. |
| 5 | SCREEN CAPTURE:  P6-2-3a-I2C\_capture3.trec | We’ll initialize a read/write buffer with the set buffer API call and a local variable that we’ll declare called myBuffer. Because this variable can be changed by the interrupt service routine I’ll use the volatile flag to let the compiler know not to optimize the variable out.  In the main loop, we’ll simply use the contents of the variable myBuffer to update the compare value of the PWM using the standard API call that we used in the previous BasicTCPWM project.  Then, we’ll put the CPU to sleep and tell it to wait for the next interrupt before waking up. Any I2C command from the master will generate an interrupt which will wake up the CPU. |
| 6 |  | Now, build, program and test… |
| 7 | SCREEN CAPTURE:  P6-2-3a-I2C\_capture4.trec  VIDEO:  Show LED bright/dim with when I2C commands are sent. May want to do a split-screen here with bridge control panel on the left and a close-up of the kit on the right.  TEXT ON SCREEN (show when the 3 lines are added):  To add a new line, press **Control-Enter**.  To execute a line, select it and press **Enter** or click the **Send** button. | To test this, I’ll use Cypress’ Bridge Control Panel tool that comes with PSoC Creator. I’ll open the tool and click on the KitProg entry at the bottom of the screen. At the top I’m going to write different compare values to send to the PSoC 6 to change the LED intensity. I’ll do this by typing: w, for write, 8, for the address, 0, for the register offset, and the hex value for the intensity, 0 to 100; and then p, for an I2C stop. For example, w 8 0 32 p, sets the PWM at a 50% duty cycle – because hex 32 is 50 - for half intensity.  Awesome.  One note about the bridge control panel – if you want to re-program the PSoC, you must disconnect from the bridge control panel first. Otherwise PSoC Creator will not be able to connect to the KitProg. Once you finish reprogramming, just select the KitProg again from the bridge control panel to reconnect. |
| 8 |  | Now we that we understand how to implement EZ-I2C with PSoC 6, next we’ll add it to our BLE-controlled robotic arm project. |
| 9 | TEXT ON SCREEN:  Cypress Developer Community  community.cypress.com  VIDEO:  Show video of ARH email and twitter windows. | 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. |