Lesson 5: Using the EZ-PD SDK

Hello. My name is Alan Hawse. Welcome back to Cypress Academy. In the last video, I changed the configuration of the CCG3 device to remove one of the power profiles. In this video, I will take this one step further and add additional functionality to the device using the EZ-PD software development kit (SDK).

I will use the user switch on the base board to send a command to get the source capabilities of the Type-C power adapter. That is, every time the switch is pressed, the CCG3 will ask the adapter for its source capabilities.

You will be able to see this using the CY4500 USB PD Protocol Analyzer like I showed you in the previous video.

The SDK is a PSoC Creator project. If you're not familiar with PSoC Creator, go watch my series of short videos based on our CY8CKIT-042 PSoC 4 Pioneer kit. You can find the videos and a link to purchase the kit at www.cypress.com/psoc101. In fact, you can find all of the video training series that we offer at www.cypress.com/training.

To begin, I'm going to start by making a copy of the project from the install directory into another writable directory so that I'm not modifying the files in my original installation directory. I will also remove the folders for the CCG2 and CCG4 devices as I'm only interested in the CCG3 firmware for now. I'll make all of the files in the copy writable and then open the Workspace in PSoC Creator by double clicking on the Workspace file. Once PSoC Creator starts, I'll see a window like this.

On the left hand side is the Workspace Explorer window. You will see that there are two projects: notebook and notebook_noboot. They are the same except for the second project cannot be bootloaded using the USB-Serial interface that's provided on the CCG3 devkit. It does, however, allow runtime debugging. To use that project, a Miniprog3 must be used to program the CCG3 device. For today, I will use the default bootloadable project. You can see that it is shown in bold on the workspace explorer which means that it is the active project in the workspace.

Under the project, you will see the schematic, the design wide resources, the header files, and the source files. These files contain all of the default CCG3 Type-C functionality that I have been demonstrating up to this point.

In the center of the screen you can see the schematic for the project. It has several pins, several clocks, an I2C interface that's used for the host

processor interface (HPI), it has an I2C interface that's used to control the USB super speed multiplexor, and the bootoloadable component. I'm going to leave all of that alone and I'll start by adding a new schematic page for our new functionality. I'll call this page "User Input". Next, I'll drag and drop a digital input pin from the Component Catalog, which is found on the right side, onto the schematic. I'll double click this new pin, and I'll change its name to "UserButton". I'll also turn off the HW connection and change the drive mode to "Resistive pull up" because the switch is active low and pulls it low when you press the button. On the "Input" tab, I'll enable a falling edge interrupt and set it as a dedicated interrupt.

Next, I'll add an interrupt component and I'll connect it to the irq terminal of the pin. I'll call this interrupt "UserButtonInt".

My schematic changes are complete. Next, I'll double click on the notebook.cydwr file. This contains the design wide resources. I need to assign the UserButton component to the pin that connects to the switch on the board. In this case it is port 2, pin 0 (P2[0]).

The next thing I'm going to do is generate the application using the "Build -> Generate Application". This will allow PSoC Creator to assist me when writing the code because it will know about all of the component APIs and it will be able to suggest code completions, which will aid me in quickly creating my firmware.

Finally, I will double click on main.c in the Source Files and I'll add the interrupt service routine (ISR). This routine will clear the interrupt request, and then request the source capabilities from the port partner using the CCG3 APIs provided to you in this project.

You can read the complete documentation about the functions used here and the other APIs related to the CCGx firmware family in the API guide which is located in the Documentation folders of the SDK.

In the initialization section of the main function, I just need to associate the ISR that I just created to the interrupt component and start it.

There is one other change that needs to be done. The same pin that the user switch is connected to also connects to an LED that is toggled on and off with a frequency of 1Hz automatically by the firmware. This interferes with the switch operation. In order to disa**b**le it, open the config.h file and change the value for APP_FW_LED_ENABLE. Specifically, change it from a 1 to a 0 to disable it.

Now the schematic, design wide resource assignments, and code are complete. I will build the project using "Build -> Build notebook". If

everything is OK, you will see a Build Succeeded message in the output window.

Finally, I need to load the new firmware into the device. I'll follow exactly the same process that I showed you in the last video. First, I'll move J3 to pins 2 -3 and connect the kit using the USB Mini-B connector just like in the previous video. Next, I'll open up the EZ-PD Configuration Utility and click on "Firmware Update". First click on the "NOTEBOOK" and then enter the path to the firmware. Note that there are two firmware files generated by PSoC Creator – and you should enter both. The path from where you copied the project is CCG3-CCG4, Firmware, projects, CYPD3125-40LQXI, notebook.cydsn, CortexM0, ARM_GCC_493, Debug. Wow, that's a mouthful. Yes, that's quite the path, but that's where it gets stored.

Then click "Program" to update the alternate firmware. Once programming has completed, unplug the USB Mini-B, move J3 back to pins 1 and 2, connect the EZ-PD Protocol Analyzer, start the EZ-PD Protocol Analyzer Utility, and start capturing data. Then connect the Type-C power adapter and wait for a power contract to be established. Clear out the log and then press the user switch. See how the CCG3 sends a get source capabilities request to the port partner, and the adapter responds with its capabilities each time the switch is pressed? It also establishes a new power contract each time.

So now I've shown you how you can add your own special functionality to the Cypress Type-C devices using the SDK. We provide many, many features in the APIs to support Type-C. In addition, the Cypress Type-C controller is a fully featured MCU with Type-C hardware integrated in, this means that the possibilities for your products is almost limitless.

As always, you are welcome to email me at alan_hawse@cypress.com or tweet me @askiotexpert with your comments, suggestions, criticisms, and questions.