UARTPrintf (2-1 reshoot)

Welcome back to Cypress Academy, PSoC 6 101. In next few videos I will show you the basic building blocks to create our BLE-Controlled robotic arm; this will include a UART terminal interface, PWMs to control the servo motors, EZ-I2C dashboard interface, digital logic-based kill switch, capacitive sensing controls and an advanced technique for debugging RTOS applications. As I go through these videos, I will first create a bare metal implementation so you understand the basics for each peripheral function and then we’ll integrate those functions with an RTOS into the BLE-controlled robotic arm project.

Let’s start with the UART interface to a PC terminal client. The UART interface is great as a rudimentary debug interface when developing an application. It can also be used as a basic communications peripheral for other system ICs.

So, let’s start by adding a new project to our previously used workspace. Let’s call this “BasicUART”.

As with our other projects, let’s drag and drop the UART component on to the schematic. Double click again to edit. Let’s call it “UART”. All the other settings look good, so click okay.

Double click on the pins file under the design wide resources and assign the RX and TX to P5[0] and P5[1].

On the back of the PSoC 6 BLE Pioneer board, you can actually see the silkscreen notes regarding the I/Os. This is a nice quick reference as you build out these projects.

[CUT FROM PREVIOUSLY EDITED VIDEO AND RESHOOT – ~2:04]

So, everybody likes to printf. How do I make that work? To do that I need to retarget stdout to the UART. We’ve built a library that will allow you to retarget which we call Retarget I/O. To include this library into your project, open build settings, click on PDL, scroll to down and click the check box next to Retarget I/O.

Now let’s generate the application. Once that’s done, you’ll see a file called “StandardIO\_user.h” that’s been generated. This is where you configure which UART you want stdout and stdin to point to. Scroll down to the first few lines of code. We need to insert a #include project.h so we can then reference the appropriate UART below. Next, go to the #define IO\_STDOUT\_UART and put in the name of the UART component, in this case, UART\_HW. Then do the same for the IO\_STDIN\_UART #define. And now our retarget I/O library is setup.

We’re going to control the UART with the Cortex-M4, so let’s open the M4 main application file. Start the UART using the API call. Standard IN is typically buffered, which means characters go into a buffer but you don’t know they’re there until you read. I want to turn that off for this program so we can handle each character as it comes in. To do this, we’ll write a line of text, setvbuf( stdin, NULL, \_IONBF, 0).

For this basic project I just want to echo the characters the PSoC 6 receives back to the terminal client. So, let’s create a character variable called c. Let’s show that printf works by printing out “Started UART example”. Now, in our main loop, let’s get a character, see if anything was returned, then print that character.

And that’s it, now time to build, program and test it.

First I’m going to do is open Windows’ device manager to see which COM port the KitProg is attached to. You can see which COM port it’s attached to under Ports and labeled KitProg2 USB-UART.

Now open up your favorite terminal client and attach it to the correct COM port at 115200 baud 8-n-1. I know the baud rate and 8-n-1 setting because they are in the component configuration dialog we saw earlier.

Now in the terminal client whatever I type is now echoed back to me on the screen…in this case, PSoC 6 is awesome! Which I completely agree with!

Ok, so now I’m going to create the main project of our BLE-controlled arm application. File->new project just like before and let’s name this “MainController”. Let’s setup the FreeRTOS and Retarget I/O build settings like we did before in a previous lesson by going to build settings and selecting the checkbox for FreeRTOS and Retarget I/O. Okay, and back to the blank schematic.

I will drag and drop a UART component from component library just like before; rename it UART. And, setup the pins in the design wide resources folder. P5[0] and P5[1] just like last time.

Now I will open the CM4 main application file.

First I will create the UART task like we did with the blinking LED task before in the FreeRTOS intro lesson.

I don’t need any arguments so I’ll use void arg like before. Now, let’s start the UART. Let’s turn off the standard in buffer, just like before. Then, I like to build a command processor that gets a key and then processes that with a big switch statement. First define a character c. Then get C from the UART. Now setup the switch to process the different commands. For now the only command will be the question mark, which will just printout the commands. As we add new commands to the command processor switch I will add more print-f’s to this case as well as additional cases to handle the other commands.

To print, we’ll just use printf command like before and the string you want to printout. When the user presses question-mark for now we will just print “? Prints help”

The last step is to start the task in main… and startup the scheduler.

Build, program.

To test it go to the Terminal again and press ?. OK… good, the help function works and we have a framework to add more commands.

Now we have our first UART interface working. In the next video, I will walk you through adding and configuring the PWM peripherals to control the servos in the robotic arm.

[RE-RECORD OR REUSE FROM PREVIOUS RECORDING]

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.