FreeRTOS!

Welcome back to Cypress Academy, PSoC 6 101. In this video I will show you how to run a FreeRTOS on PSoC 6.

So, you ask why an RTOS? Well, when you have multiple cores, more memory and higher frequencies, you can do a lot more. When you build devices that have multiple sensors, interfaces, communications, etc., you need help managing all of those resources and access to the cores. An RTOS is used just for that purpose, to manage the complexity of your design.

FreeRTOS is one of the most commonly used open source RTOS’ in the embedded market and a great place to start. So, let’s get right into it.

We’re going to create a new project again. This time, we’ll call it “HelloWorldFreeRTOS”.

Now we’re going to modify the build settings to automatically add FreeRTOS into our project. Right click on the project in the workspace explorer and select build settings. Pick the Peripheral Driver Library. Click FreeRTOS and choose your memory management scheme, in this case we’ll just keep the default.

Hit okay.

Then we’re going to add the digital output pin component, call it red again, turn off the hardware connection.

Let’s assign the pin, P0[3] again.

Let’s generate the application…this is going to bring in all of the drivers we need for the pin and the cores but, this time, it will also bring in the FreeRTOS source code for us to use in our application including adding a template file, called FreeRTOS config.h.

Let’s open up the FreeRTOSconfig.h file. This is the standard template from FreeRTOS. PSoC Creator gives you a warning that you should modify this file in case you forget. All of the defaults here are reasonable so, I’ll just comment out the warning.

So, now, let’s open the main CM4.c file. Let’s create a function called ledtask using the standard FreeRTOS task function prototype that requires a void pointer. We’re not using the argument so we’re going to tell the compiler that we’re not using the argument so, a simple void arg will do. Now all this task is going to do is to infinitely read and write the GPIOs for the red pin. Instead of using the CyDelay API like last time, though, we’re going to use the FreeRTOS task delay, vTaskDelay for 500 milliseconds.

Now in the main function, let’s create the task. We’ll use the FreeRTOS API command xTaskCreate, call the ledtask function, give it a name “LED Task”, a minimum stack size, we’re not passing any parameters, task priority of one and we don’t need a task handle.

Next we need to start the FreeRTOS scheduler by calling the API vTaskStartScheduler which starts and never returns.

Now build, program and test.

Notice we didn’t have to include all of the FreeRTOS headers. Those are already included by default in the project.h once we changed the build settings. Congrats you’re now up and running with PSoC 6. In the next set of videos, I will walk you through the PSoC 6 peripherals we’re going to use in the BLE-controlled robotic arm, one step at a time.

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.