PWM

Welcome back to Cypress Academy, PSoC 6 101. In this video, I will show you how to setup the PWMs to control the servos in the robotic arm for our BLE-controlled robotic arm project. Let’s get started.

Let’s make a new project to understand how the TCPWM component works.

Make a new project, let’s call it “BasicTCPWM”. This project will vary the brightness of a LED with a very simple delay scheme.

I will add the TCPWM component by dragging and dropping it in to the schematic. Let’s rename it to PWM and set the period to 100 and compare to 0.

I will now drag in the pin component. Again, we’ll rename it to RED.

Now for the clock, drag and drop that component and we can keep the default configuration of the clock alone.

Make sure you wire the clock to the PWM and the pin to the PWM as I did when I dragged the components into the design.

Now to assign the pin component to the physical pin on the PSoC 6 device. Again P0[3].

Time to generate the application.

Now let’s go the CM4 main application file.

I will add a local variable to main function called compareValue and set the default value to 50.

Start the PWM using the simple start API command.

In the infinite for loop, I will change the compare value of the PWM using the “CY\_TCPWM\_PWM\_SetCompare0” API call and the PWM\_HW macro. There are multiple counters in the PWM, so the setcompare function needs to know which one I want to change. In this case, I’ll use the macro PWM\_CNT\_NUM. Now the last argument is used to set the compare value, I’ll set it to the local variable we created, compareValue.

Now I want to increment the local variable, compareValue, by 1 and mod it by 100 so it counts up to 100, the maximum period we set before, and resets to 0.

Finally we need a delay so we can actually see the change, so let’s use the CyDelay API and 20 milliseconds. This will make the LED slowly dim and back to maximum brightness.

Now, build, program and test.

Awesome! I love blinky LEDs!

Now let’s add the PWM functions to our BLE-controlled robotic arm project, “MainController”.

The servo motor in my robotic arm requires a 20hz pulse and the pulse width needs to be between one and two milliseconds. At one millisecond it’s turned all the way to the left and at two, it’s all the way to the right and half a millisecond, right in the middle.

So, to make all the math work, we’ll choose a 12MHz clock, pre divide it by 4 and then set the period to 60,000. Now the compare value we’ll go between 3,000 and 6,000 to give us full range control on the motors. So, 1% will be 3-clock ticks, in other words 3,003 is 1%, 3,006 is 2%, etc.

So, I’ll start by dragging and dropping one PWM from the catalog on to the schematic. Let’s setup the period and compare to 60,000 and 3,000 respectively; turn on the divide by 4 pre-divider. And click okay.

We want to control two motors to start, so let’s copy that component and paste a second one into our design. It will be named PWM\_2 and that’s convenient since PWM\_1 I’ll use to control motor 1 and PWM\_2 for motor 2. The copy paste function is also nice in that all of the settings from PWM\_1 were also copied over to PWM\_2 saving us some time.

Now let’s add two digital output pins to our design…I like the copy-paste function, so I’ll pull in one and rename it “M\_1” for motor 1. A quick copy/paste and I now have “M\_2” as well.

Let’s connect those pins to the PWMs’ PWM output line on the component.

Next, we need a clock. I’ll drag and drop a clock in and set it to 12MHz. Then wire it to both of the PWMs.

One of the most forgotten steps in a design, is to assign the pins; so let’s do that now before we forget. Go to the pins settings and we’ll set M\_1 to P0[2] and M\_2 to P5[5] which matches up to the H-bridge pins on my Arduino shield I’m using.

Let’s generate the application and start working on some firmware.

Back to the CM4 main application file. I’ll start at the top of the file and add a structure to send messages to the PWM. This structure will allow me to tell the PWM task, that we’ll create next, what motor we want to change, an absolute percent change we want to make or a relative percent change. I’ll call this structure “PWM\_Message\_t”.

I’ll use an RTOS queue to handle the messages so they are handled in the order they’re received. I’ll call it PWM\_Queue.

Next thing I’m going to do is to create two helper functions—first, turn compare values into percent values; and second, turn percent values back into compare values.

Now I’ll create the PWMTask. When the task starts I’ll start the two PWMs using the start API command; then in an infinite loop I’ll wait to receive an RTOS command from the queue. When I get a command from the queue, I’ll figure out what the hardware and counter numbers are so I can use the appropriate macros; and then error check to ensure I didn’t get an incorrect value.

Then what I’ll do is, if the message coming in wants to make a relative change in percent, I’ll get the current compare value, convert it to percent, make the change. If the message calls for an absolute value change, I’ll make the change. Then I’ll update compare value of the appropriate PWM.

Now, I need a way to test this, so I’m going to add it to the UART command set we defined earlier. In the UARTTask, I’m going to use the commands “o” and “p” to change the relative percent value of motor one by negative 10% and positive 10%, respectively. And I’ll use commands “k” and “l” to do the same for motor 2. So, in the case statement for the command ‘p’, I’ll set the message to one, to control motor one, absolute percent to minus one since we’re doing a relative change, and relative percent change to 10. Then queue the message and break. Now I’ll do the same for the other commands.

Are we done? No! We can’t forget to add the messages to the help command, “?”. So, I’ll do that real quick.

And that’s it…that’s the beauty of an RTOS. Simplifying complex designs.

Let’s build, program and test…

Now we have a functioning UART-controlled robotic arm…but we’re not done yet, this is supposed to be a BLE-controlled robotic arm. So, we have a little more work still to go. In the next video, I will walk you through setting up an I2C control interface.

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.