PWM

Welcome back to Cypress Academy, PSoC 6 101. In this video, I will show you how to setup the PSoC PWMs - which we call TCPWMs - to control the servos in the robotic arm for our BLE controlled robotic arm project. TCPWM stands for Timer, Counter, PWM, because these blocks can be used for any of those functions. In this video, we'll just use them as PWMs.

Alright, let’s get started.

First, if you've never used a PWM let me tell you a little bit about them. They are actually really simple little devices. PWMs have in input clock and one digital output signal. Inside of them there is a counter and two registers. When you clock the PWM the counter counts up. When the value in the counter reaches the value in the first register – often called the compare register - the output state toggles. Then the counter keeps counting until the value reaches the value in the 2nd register - which is called the period - at which point the output toggles again.

Note, there's a billion little variants… sometimes the counters count up, sometimes they count down… sometimes there's multiple outputs… sometimes they are staring high, sometimes they start low… sometimes there's dead bands… there's a bunch of those options and in fact our TCPWM has all of the ones you might every possibly want but that is the basic idea of a PWM.

Inside of the PSoC 6 there are two banks of PWMs. The first bank has 24 PWMs that have 16-bit counters. The second bank has 8 PWMs with 32-bit counters. By having a 32-bit counter, you can get a much higher resolution output waveform but generally 16-bits works well which is why that's what we have mostly. You can read more about the TCPWM in chapter 27 of the PSoC 6 technical reference manual.

Allrighty… 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 an LED with a very simple delay scheme.

If you blink an LED fast – and I mean really fast - like more than about 30Hz or so - your eye is not going to see the blinking at all. If you vary the amount of time that the LED is on versus the time that the LED is off, it will appear either brighter or dimmer based on that ratio – the ratio of the on time to the off time. If it is on all of the time, that will appear to your eye to be the brightest it will ever be… if it is on only 10% of the time then it will appear very dim. This on/off ratio is called the duty cycle.

PWMs are perfect for controlling duty cycle. The ratio of the compare value to the period is exactly the duty cycle of the PWM. If I set a period to be 100 then the compare value can be between 0 and 100… which is also known as percent. And this is exactly what I am going to do in the next several PWM examples.

Start by adding the TCPWM component by dragging and dropping it in to the schematic. Let’s rename it PWM and set the period to 100 and the compare value to 0.

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

Now for the clock, drag and drop that component and we'll keep it at its default configuration.

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

Now assign the pin component to the physical pin on the PSoC 6 device. In this case a red LED is connected to P0[3].

It's time to generate the application.

Now let’s go the CM4 main application file.

I will add a local variable to the 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. Each PWM hardware block has multiple PWM counters, 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 actual compare value. I’ll set it to the local variable we created earlier called compareValue.

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

Finally, we need a delay, so we can actually see the change. Let’s use the CyDelay API which I hate because it just makes the CPU burn CPU time, but in this case it's OK. So, we'll set the CyDelay API to be 20 milliseconds. This will make the LED go slowly dim and then slowly back up to maximum brightness.

Now, build, program and test.

Awesome! I really love making these blinky LED projects!

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 @askioexpert with your comments, suggestions, criticisms and questions. Thank you.