Kill Switch

Welcome back to Cypress Academy, PSoC 6 101. In this video, I will show you how to add custom digital logic to our BLE controlled robotic arm to safely stop the servo motors and bypass any UART or I2C commands. We will also use the RGB LED on the PSoC 6 Pioneer Board to show the status of the kill-switch.

Why a kill switch in a BLE controlled robotic arm you might ask? Well because all robots should always have a kill switch – I mean, have you guys seen the Terminator movies? - and #2 because my son, bless his heart, he's an avid engineer and is a tinkerer just like me and while we're working on the robotic arm project, he 100% guaranteed will be swinging the thing every which way and we definitely need to have a kill switch to turn if off.

Alright, all joking aside, the real reason for a kill switch is to remove the CPU from a safety critical decision. Our digital logic responds in nanoseconds, and CPUs are much slower, think microseconds, depending on what’s going on in your application.

So, we’re going to make a simple t-flip-flop controlled by an external switch on the PSoC 6 BLE Pioneer board. The t-flip-flop will be implemented in the digital logic block inside of the PSoC 6. This block is also called the Universal Digital Block, or UDB for short. There’s a lot more documentation on this and entire video series on this that you can find here:

Let’s get started.

I’ll create a new project called “BasicKillSwitch”. For this project I want to use a button on the PSoC 6 BLE Pioneer board to act as the kill switch. When I enable the kill switch, I expect the red LED to immediately turn on and blink. When I disable the kill switch, I expect to see a solid green LED indicating all is well.

To do this, I’ll drag and drop a t-flip-flop from the digital logic folder in the component catalog. Add a digital input pin component. Let’s rename that pin to “SW” and set the drive mode to resistive pull-up on the general tab of the pin dialog. I’m doing that because the push button on the PSoC 6 BLE kit is active low.

Now, I’ll connect the switch to the t-flip-flop through an inverter to make the switch act on the falling edge, like when the button is pushed there is a falling edge. I also need a logic high for the t-input of the flip-flop. With this configuration, the output of the t-flip-flop will toggle each time the button is pressed. To enable a blinking LED, we’re going to need a PWM. So, I’ll drag and drop a PWM into my design and then configure it. I’ll rename the PWM to be LEDBlink. I’ll set the period to 1,000 and compare value to 500. I need a clock to input to the PWM, so another drag, drop, and configure and look there we have a 1KHz clock.

Now I have two LEDs in this project, a RED LED and a Green LED. So, let’s grab two digital output pins and rename them to be RED and GREEN. One thing to note with the LEDs is that they are active low.

Now for a little logic fun. When the t-flip-flop output is high, disabled, I want to turn the Green LED on. So, a simple inverter logic block will do the trick. Drag and drop that and then wire it up. Alright, now that looks good. Now if the t-flip-flop is low, enabled, I want to blink the red LED, so we’ll use a digital multiplexer and the output of the t-flip-flop to select the channel to send to the LED. We’ll send the LEDBlink PWM output to the multiplexer input 0 and a logic high to the multiplexer input 1. So, when the t-flip-flop output is low the PWM output is feeding the RED LED, making it blink.

Let’s not forget to assign the pins for the switch, the red LED and the green LED to P0[4], P0[3], and P1[1] respectively.

In the main\_cm4 you just need to start the PWM by calling the LEDBlink\_Start().

And that’s it. Build, program and test once again. Alright, I love those blinking LEDs!

In the next video I will integrate the kill switch into our main controller robot arm.

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, and suggestions, and criticisms (and yes, I do get them), and your questions. And once again, thank you for your time.