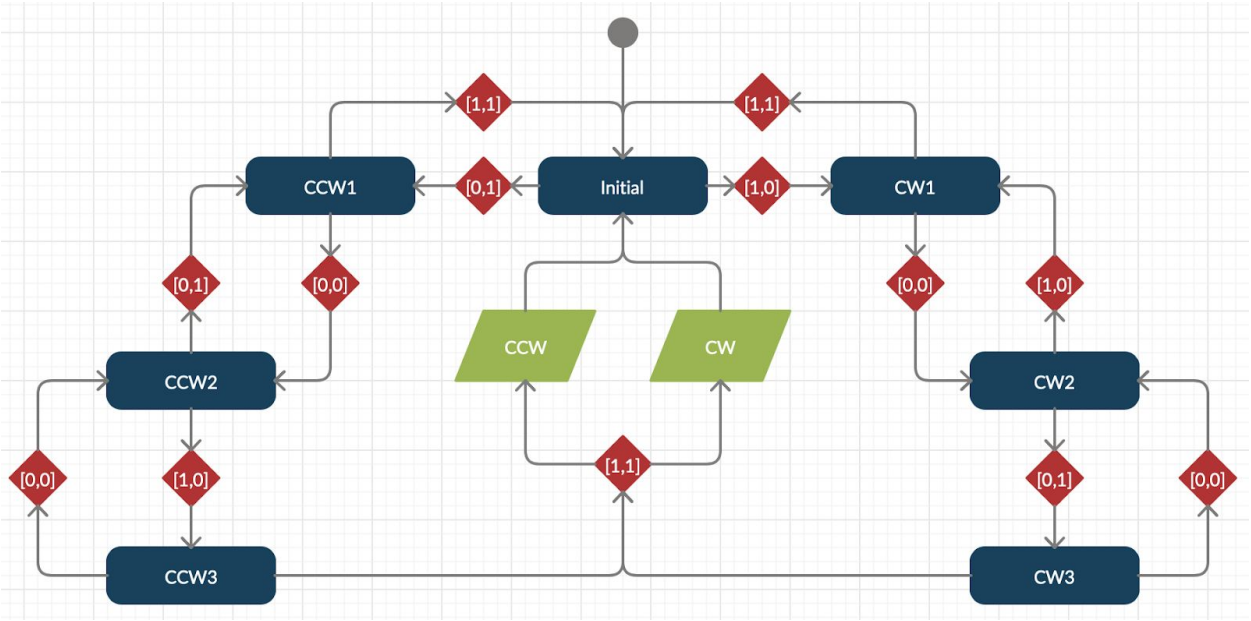
ECE 153A Lab 2A: Rotary Encoder

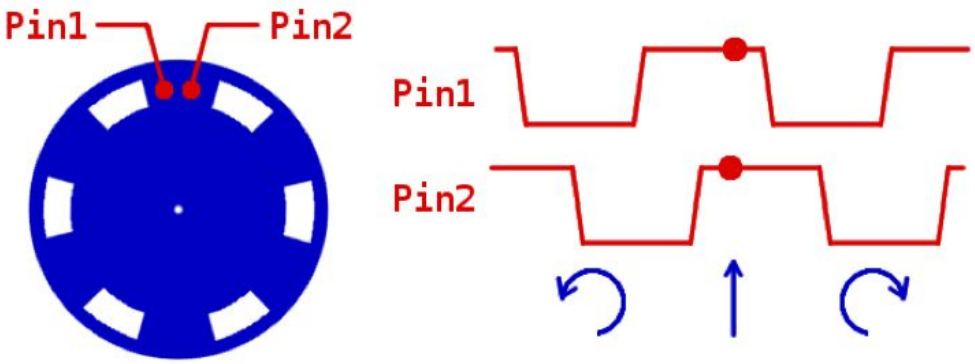
Luke Lewis | Eduardo Olmos

Purpose

In this lab we will be using Finite State Machines to map the internal sequence of a rotary encoder's clockwise and counterclockwise clicks to move a led index on a circular linked array of 16 leds situated on our board. A click counterclockwise shifts the led to the left, clockwise shifts the led to the right wrapping around at the bounds. Additionally we will utilize the button on the rotary encoder to cycle power to the led. We are also going to flash a status led to indicate the machine is working as intended. Our machine is to be very responsive and robust against rapidly and similar fast/simultaneous inputs . This means that we need to make sure debouncing is very well controlled. We will also be diving into the Vivado Hardware Manager to debug our code.

Flow





Results

A rotary encoder is a quadrature encoder. There are two pins that rest at high. The sequence for a counter-clock wise click is Pin1 goes low, then Pin2 goes low, then Pin1 goes high, and finally Pin2 goes high. Whereas for a clockwise click, the sequence is reversed: Pin2 goes low first, then Pin1 goes low, then Pin2 goes high, and finally Pin1 goes high. Notably, the two pins never transition at the exact same time. Therefore to know which direction our rotary encoder is twisting in, we can break the sequence into states that reflect a change of only one bit at a time and the machine will output the direction when they reach back to the resting state.

Debouncing of the twists of the encoder is accomplished by looping between steps of the sequence. It is important to note that only one of the pins changes at a time until it settles and then the other changes. It is also important to note that we are guaranteed by the rotary that the action of a click that no matter if it is bouncing between steps, it will eventually progress and complete the sequence. The states are labeled according to the order of the steps for each direction. If you're at the CW1 and Pin 1 changes from 1 to 0, it progresses to CW2. However Pin1 might change back from a 0 to 1 so it must be that we move back to CW1 as a result. Eventually Pin1 will settle at 0 and we would have progressed in our sequence. This logic applies to all steps except the final step of each respective direction. If you're at CW3 and Pin 1 changes to a 1, it progresses to the final resting state, but if it bounces and changes back to a 0, we do not step back into CW3. We bounce to CCW1 and then back to the resting state until it settles there and it outputs for the respective direction.

Debouncing of the button press of the decoder was accomplished with the use of a flag and a timer. As soon as we encounter the first time the button is pressed, we set a flag rEncoderButtonPressed to 1. This flag is set only the very first time it sees it was pushed, no matter if it bounces. The time is recorded every time the encoder interrupts even if it is bouncing, it should capture the last time it saw any input change from the button. It is important to note that we are running our interrupt handler atomically so this would help a little with the bouncing value. In our loop, we first check to see if the flag was set and if the current input of the button is not pressed. We then check to see if it has been 2000000 cycles since the time we recorded when the encoder was interrupted. We then either turn the led on or off and reset the timer. We found 2000000 to be a good value that allows us to press the button really fast and have it respond quickly. We defined our system so that you can hold the button down and turn it and have the leds respond only if the led is on. To trigger the power cycle of the led, you must release after pressing the button.

	Delay in Cycles
Input change to GPIO interrupt	2048 - 2039 = 9
GPIO interrupt to Processor interrupt	2054 - 2048 = 6 cycles
Processor interrupt to address 0x00000010	2058 - 2054 = 4 cycles
0x00000010 to interrupt Handler	2310 - 2058 = 252

The MSR register is the machine status register, which serves as a register containing control and status bits for the processor. All of the bits (except the reserved bits 1:16) serve a purpose in dictating the processor's behaviour. Bit 30 is the interrupt enable bit, where a 1 means they are enabled, and oppositely a 0 means interrupts are disabled.