Lab 2 – Rotary Encoder Input

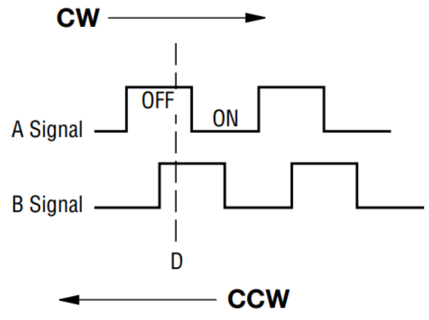
1 Objectives

1. Implement a simple finite state machine.
2. Read input from a rotary encoder.

2 Introduction

In this lab you will design and implement a circuit to interpret inputs from a rotary encoder. The rotary encoder will be used to increment and decrement counts that will be displayed on the four digit 7-segment LED display.

The rotary encoder generates two outputs A and B which are 90 degrees out of phase as shown in the diagram to the right.



You will be provided the partial code for the top level module, lab2.

You must design and implement the **`encoder`** module as follows:

- The module has two input signals, **`a`** and **`b`**, from the rotary encoder, and a clock input **`clk`**.
- The module will use the information from the current **`a`** and **`b`** inputs and the previous **`a`** and **`b`** inputs to determine if the position of the encoder has changed.
- The outputs **`cw`** and **`ccw`** should pulse for one clock cycle if the encoder inputs change in such a way indicating that the encoder has been turned clockwise or counterclockwise respectively.
- All synchronous logic should use the **`clk`** input (ie no flip-flops should be triggered on the **`a`** or **`b`** inputs).

3 Prelab Exercises

Write a Verilog module (**`encoder.sv`**) that meets the requirements given in the introduction.

Download lab2 files from the Learning Hub. Follow the simulation instructions in Appendix A of Lab 1 using the supplied **`encoder_tb.sv`** testbench to check that your module operates as expected. The simulation results will be displayed as PASS/FAIL messages on the Modelsim Transcript window. To debug your design you can examine the signal waveforms. The Modelsim IDE also has debugging features that allow you to set breakpoints, single-step and examine signal values during simulation. Once all tests pass, save a screenshot of the simulation waveform as well as a screenshot of the Transcript window to include in your lab hand in. The Transcript window screenshot should include the lines from “run -all” to “Break in Module...”.

4 Lab Activities

Complete the **`lab2`** module provided by adding code where you see “***ADD CODE HERE***”. Follow the instructions in Appedix B of Lab 1 to synthesise the completed **`lab2.sv`** file along with your other modules. Be sure to download and import the pin file before synthesis. Plug in the FPGA board and program it. Note that the TI Educational BoosterPack MKII must be in place to complete the ground connection for encoder 2. If you forget your BoosterPack, a jumper wire can be used.

5 Lab Improvements

You may have noticed that you can count by ones by carefully turning the knobs, however, the knobs only stop in positions every four counts. In addition, it would be much more natural to have a count in decimal than in hex. Write a module called **`enc2bcd`** that generates a binary coded decimal count (0 to 99) and only counts up/down every fourth pulse of **`cw`**/**`ccw`**.

The module declaration should look like:

```
module enc2bcd (input logic clk, cw, ccw, output logic [7:0] bcd_count) ;
```

Add two instances of your your **`enc2bcd`** module to replace the following code:

```
// encoder counts: enc1_count & enc2_count (increment when cw=1, decrement when ccw=1)
always_ff @(posedge CLOCK_50)  begin

  // encoder 1 count
  if (enc1_cw) enc1_count <= enc1_count + 1'b1;
  else if (enc1_ccw) enc1_count <= enc1_count - 1'b1;

  //encoder 2 count
  if (enc2_cw) enc2_count <= enc2_count + 1'b1;
  else if (enc2_ccw) enc2_count <= enc2_count - 1'b1;

end
```