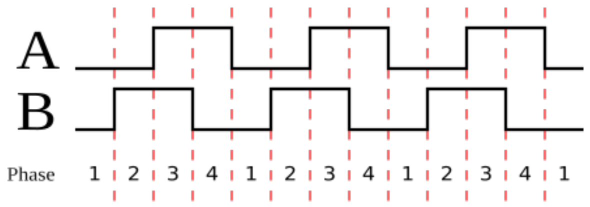
ECEN 2350: Digital Logic

Assignment #13

Make a quadrature encoder counter with your Boolean Board.

Quadrature (or Incremental) encoders are widely uses for a variety of uses, including motion control, robotics, speed sensors (e.g. in anti-lock brakes), and as an input feedback device. It is characterized by two digital signals which arrive at their decoder separated in phase by 90°:

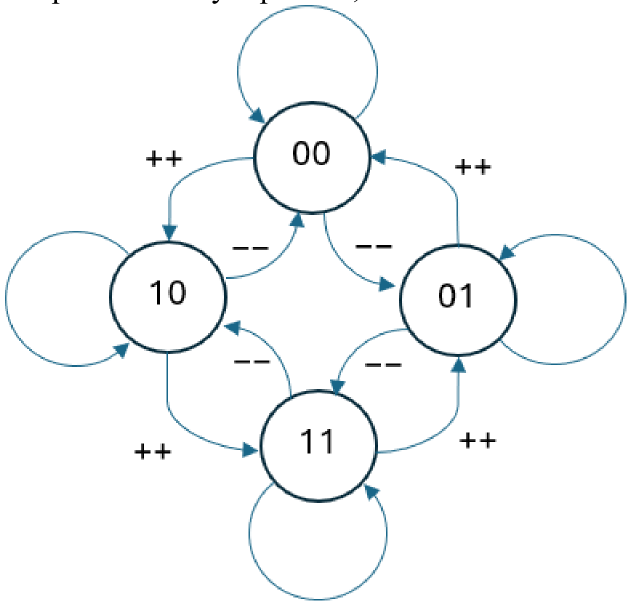


Review the Wikipedia page https://en.wikipedia.org/wiki/Incremental_encoder, a decent summary of the technology and its uses.

1. [15 points] Make a reliable Finite State Machine (FSM) in a Verilog module named `QuadratureEncoder`, to input the quadrature encoder outputs {A, B} and a reset signal, and output a 16-bit count.

   Separately, create a Verilog module named `proto_QuadratureEncoder`, to test your `QuadratureEncoder` module, simulating the {A, B} signals using switches `sw[0]` and `sw[15]`, and using `btn[0]` as the reset signal.

   Your Finite State Machine should number its states according to the gray-code sequence which the encoder outputs naturally represent, i.e. in a forward direction:



   Your count should use the `HexEncoder` and related modules you made in prior labs. The count may be presented in either decimal (hard) or hexadecimal (easy).

If you decide to use decimal encoding, and since the Lab 11 decimal counter only counts up (here it must also count down), you may get some implementation ideas by reviewing the design of standard (TTL) parts which do the same function like the 74LS192 BCD up/down counter. Note how this part forms one 4-bit BCD digit logic unit with separate up/down clock inputs and separate "carry-out" (CO) and "borrow-out" (BO) signals, which can feed into digits to the left and right.

IF you use the encoder signals (`sw[0]` and `sw[15]`) as clocks (i.e. `always @(posedge sw[0])`), you will have to adjust your constraints file to permit it to be used as a clock. Consult the instructions in Lab 9, e.g.

```
   set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets sw[0]];
```

If you use the 100MHz system clock (`mclk, clk` or a derivative), you may want to construct a synchronizer to assure the stability of the input signals, as described and demonstrated in class (see below).

Tips to make your state machine more stable & reliable:

- Use a single clock for the whole state machine, to avoid setup and hold violations.
- Use a Moore machine. Make your outputs a combinational function of your state.
- Use a Verilog `case` statement to determine the next-state from the current state.
- Use a synchronizer, as shown in lecture, when you have unpredictable (e.g. Human) inputs. This avoids instabilities caused by setup/hold time violations. Here is example code for a single bit synchronizer:

```
module sync(input clock, input in, output reg out);
    reg  data;

    always @(negedge clock)
        data <= in;

    always @(posedge clock)
        out  <= data;
endmodule
```

Submit your solution as the file `lab13-q1.txt`

If you would like to test your circuit with an actual hardware encoder, One will be available made available in lab. You may feed your quadrature encoder via the Boolean Board Pmod connector GPIO signals. Please request needed constraints file information for this.

2. [10 points] Unless you have already done so, reorganize the Verilog module interfaces for the `HexEncoder` and related modules to be standalone modules, with inputs and outputs that do not directly connect to the Boolean Board's devices. If you have not already, write a `HexDisplay4` module which uses the `HexEncoder` module and also generates the digit enable (`D0_AN[3:0]`) signals. It will need a clock input to sequence/multiplex the digits to the 7-segment display. Module prototypes should be thus:

```
module hexEncode(input [3:0] bin, output reg [7:0] hex);

module HexDisplay4(input clock, input [15:0] in,
                   output reg [3:0] digit_ena, output [7:0] seg);
```

Note that the `HexDisplay4` module can be used and re-used in any future project. Modules should be modular! :-)