Build a finite state machine as a sequence detector

Goal: Detect sequence 10010 and turn on LED light.

Implementation: Use Mealy Machine. When “10010” is detected, the LED0 in Basys 3 will be on. Will use Pseudo Random Binary Sequence (prbs) to generate the pattern. A task also check the parity of the pattern. If it is odd, the LED1 will be on

**Step 1: Draw the Finite State Diagram**

There are 5 states – S0, S1, S2, S3 and S4 (from initial to end). IN means input to each state and OUT means output of each state. Keep all outputs to be 0 until we reach the final state, S4. The initial state, S0, receives the input from the prbs and final state, S4 decide whether LED0 is on or off. We want to keep buffer in S3. If the IN is 0, we will stay in S3 (not going back to S2). Once we turn on LED0, the initial state will be back to S1 but not S0. In this case, we count IN =1 for the next cycle (i.e. start IN = 1 at S0)

Flow

IN 🡪 S0

S0 🡪 S1 (if IN = 1) otherwise, stay S0

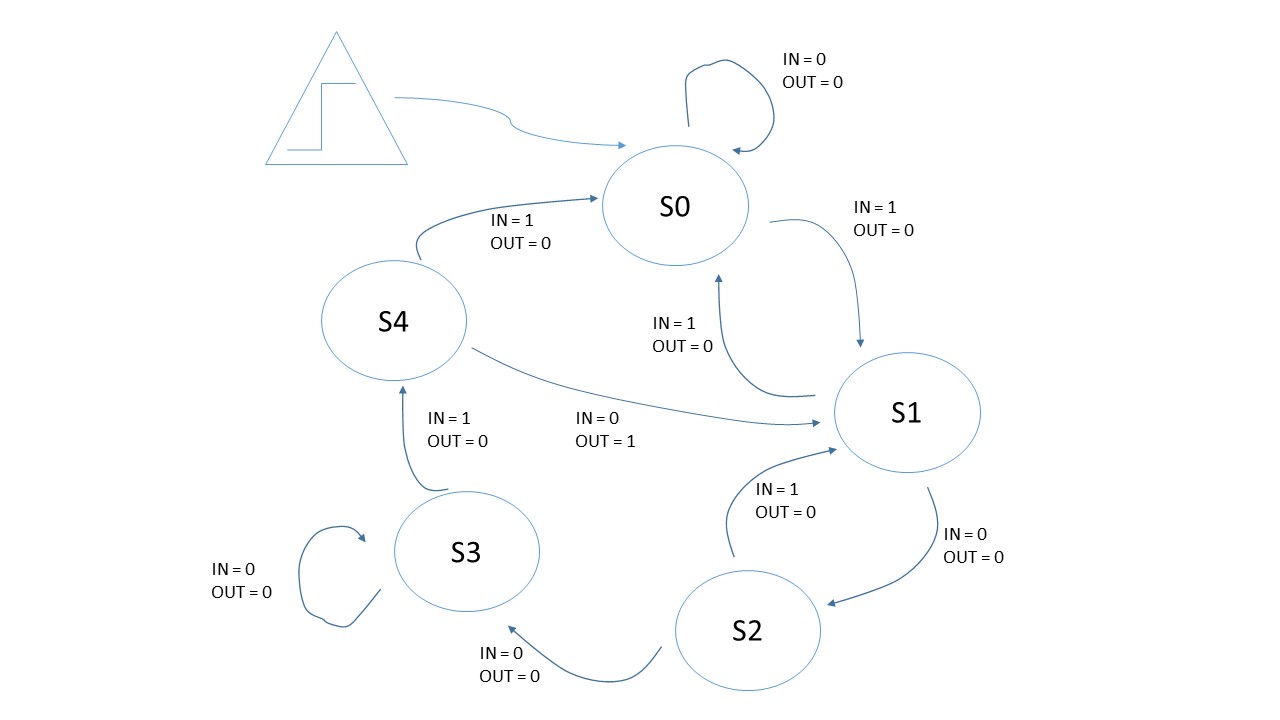
S1 🡪 S2 (if IN = 0) otherwise, S1 🡪 S0

S2 🡪 S3 (if IN = 0) otherwise, S2 🡪 S1

S3 🡪 S4 (if IN = 1) otherwise, stay S3

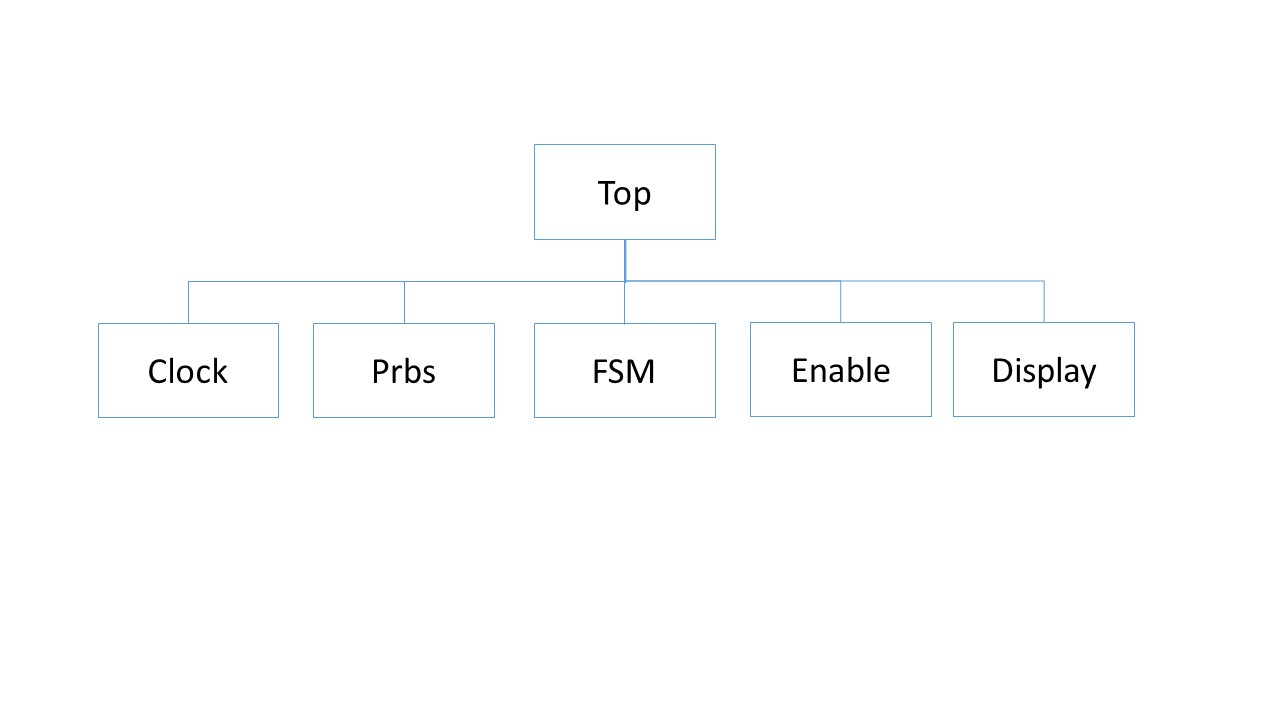
S4 🡪 Turn on LED0 (if IN =0) and back to S1, otherwise S4 🡪 S0

State Diagram



**Step 2: Create Block Diagram**

The modules include finite state machine for sequence detector. Prbs creates the pattern and checks the parity of the pattern. Enable turn on the digit of seven segment displays. Display model how to display the pattern on four seven segment display. Clock create the clock divider and supply the clock to the whole design.



**Step 2: Create clock divider**

Create two clocks to drive prbs, fsm, enable, display. Use BUFG (Global buffer clock, Xilinx primitive) to create a low skew clock as this clock has high fanout (output drives a lot of inputs)

Refer to the stop watch clock divider for details.

**Step 3: Model FSM by Verilog**

Implement the FSM as synchronous FSM, i.e. driven by clock signal or positive edge of the clock. Use parameter to create constant for each state. Assign each state by hot encoding. Initialize the state as S0. This is required to do to prevent error. Create next state logic in the always block through case statement and conditional statement. Assign the output for each state in the always block.

**Step 4: Create enable digit logic**

Refer to the stop watch enable logic for details

**Step 5: Create display logic**

Display “1” and “0” from the prbs pattern in the four digit SSD. Use case statement to model which segment is displayed in “1” or “0”. Add outer case statement (even triggered by concatenation of four enable digits). Inputs are the prbs pattern, digit enable and outputs are segments in each digit.

**Step 6: Create prbs and check the parity**

Psuedo Random Binary Stream (Prbs) is used to generate the random pattern. First, create 18 bits linear feedback shift register (Lfsr) by Exclusive OR (XOR) five bits of lfsr to make right most bit. Shifrt both pbrs and lfsr patterns. Remember to initialize lfsr as all 1’s. Create function PARITY to check the parity of the 18 bits lfsr. Use for loop to XOR all bits in the pattern. Return PARITY by assigning it to the final result of the XOR. If the parity is odd, LED is on.

**Step 7: Create constraint file**

Instead of using Digilent’s master xdc. Create own constraint file.

Refer to the Xilinx reference manual to create timing constraint. Follow the following instructions create I/O constraints.

Timing Constraint: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug945-vivado-using-constraints-tutorial.pdf>

IO Constraint: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug935-vivado-io-clock-planning-tutorial.pdf>

Check the FPGA pin from Basys 3 schematics

**Step 8: Synthesis, Implement and Generate bitstream**

**Step 9: Program bitstream file to Basys 3**