An 8-bit Pseudorandom Number Generator

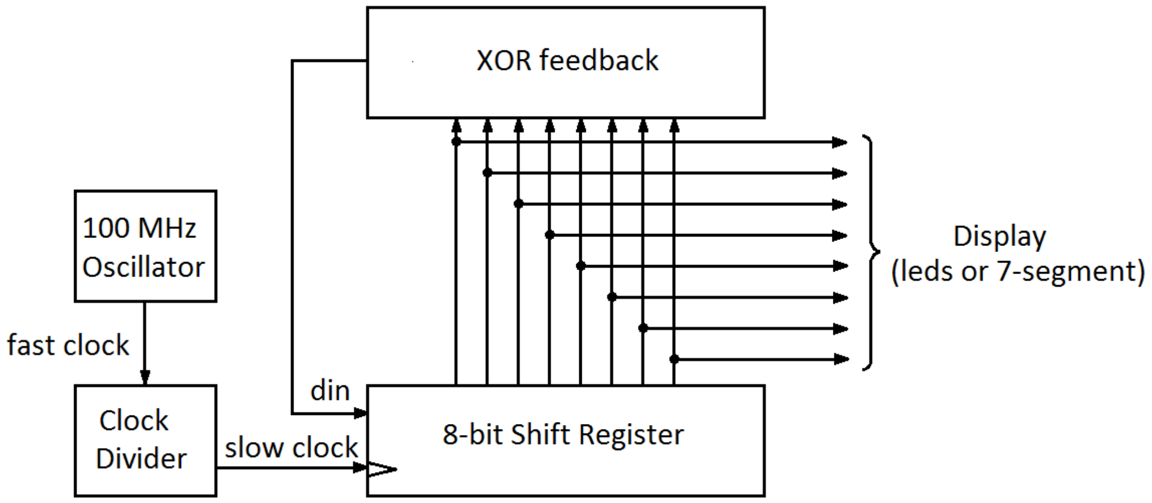
PURPOSE - This lab will require you to design and implement an 8-bit pseudo random number generator on the Basys 3 board.

1. **Introduction**

A Linear Feedback Shift Register (LFSR) is a shift register with two or more of the flip-flop outputs XORed together and fed back to the input of the shift register. The inputs to the XOR gate are referred to as taps. For an *n*-bit shift register there are 2^n-1 possible binary patterns which can be output. The value of all 0s is omitted as it will always repeat itself. Depending on the choice of taps there may be multiple sequences with varying lengths. For some set of taps we will generate maximal length sequence of all 2^n-1 possible patterns.

This arrangement is also called a pseudo-random number generator since the shift register outputs will appear to be a sequence of binary random numbers. The set of taps which generates the maximal sequence for shift registers of n= 4 to 32 bits can be found in Roth et al (2016).

For this lab we will construct an 8-bit linear feedback shift register. The block diagram of this LFSRis shown in the figure below.



The Basys 3 board has a 100 MHz connected to pin W5 of the Artix-7 FPGA. However, this frequency is far too fast for us to see the pseudo-random sequence output from the shift register, so we use a clock divider to reduce the frequency to about 1 Hz. Thus, the counter will count once per second.

To implement the 8-bit LFSR we will use three modules. The first module is the clock divider, the second block is an 8-bit LFSR that models the shift register with XOR feedback, and the third block is a top- level module that instantiates and interconnects the clock divider and the LFSR. This top level block can be synthesized and downloaded into the FPGA. A constraints file will connect the outputs to 8 leds and the reset to a push button. With this initial arrangement you will be able to see the 8-leds display a new, seemingly random, 8-bit value appear every second.

***2.* Start Vivado as was done in the previous labs.**

.

3. Create a new project called *lab4*. The hardware is the same as in the previous labs.

4. **Type the following Verilog code. Then save it as *clock_divider.v* and add it to the project.**

```
module clock_divider (input cin, output cout);

parameter timeconst = 60;//constant
integer count0;
integer count1;
integer count2;
integer count3;
reg d;
reg cout;

initial begin
  count0=0;
  count1=0;
  count2=0;
  count3=0;
  d = 0;
 end
 always @ (posedge cin )begin
     count0 <= (count0 + 1);
     if ((count0 == timeconst))begin
          count0 <= 0;
          count1 <= (count1 + 1);
     end
     else if ((count1 == timeconst)) begin
          count1 <= 0;
          count2 <= (count2 + 1);
     end
     else if ((count2 == timeconst)) begin
          count2 <= 0;
          count3 <= (count3 + 1);
     end
     else if ((count3 == timeconst)) begin
          count3 <= 0;
          d <= ~ (d);
     end
     cout <= d;

  end

  endmodule
```

Try to figure out the operation of the above code. You will feed the 100 MHz external clock into `cin`, and then a 1 Hz signal will come out at `cout`. This frequency is adjustable, however, according to the following formula:

Output Frequency = 100000000 / (2 * (TIMECONST ^ 4))

5. Create a Verilog module to model the LFSR, save it as lfsr.v, add it to your project.

This module should have a first line

```
module lfsr (output reg [7:0] q,
             input [7:0] seed,
              input clock,
              input rst)
```

For an 8-bit shift register with flip-flop outputs q[0], q[1],…,q[7], and the feedback input serially to flop-flop 0, we have taps at q[1], q[2], q[3], and q[7] to produce the maximal length sequence. The XOR feedback can therefore be modelled with a single continuous assignment statement as

```
                assign din = q[1]^q[2]^q[3]^q[7];
```

For the shift register you should use a single always construction. Examples of modeling a shift register can be found in the supplementary notes or Roth et al (2016). The shift register should be reset to the value of seed when the rst input is asserted. You must write and design your own shift register.

6. **Type the following Verilog code. Then save it as *lab4_top.v* and add it to your project.**

```
module lab4_top(output [7:0] q,
                input clk,
                input rst
                );

     reg [7:0] seed = 8'b00000001;
     wire cout;

     clock_divider CDIV (clk,cout);
     lfsr LFSR (q,seed,cout,rst);

endmodule
```

Draw a new block diagram (as the one at the beginning of this lab manual) showing all defined signals in the Verilog code above.

7. **Write a custom testbench code.** The test bench should drive the required **clk** and **rst** pins and observe the pseudo random number sequence in the waveform viewer. Compile a table of the expected number sequence versus the one observed.
   Note: The test bench may be extremely slow due to the clock divider. Instead of instantiating the lab4_top you can directly instantiate the lfsr module and test that.

8. **Synthesis**

8.1. Analyze all your Verilog sources as learned during the first lab.

8.2. Locking in pins

Now that all of the Verilog design files are added and correctly analyzed, you need to specify the FPGA pin numbers that you wish to use. Create a new design constraint file *lab4_top.xdc*, and add it to your project. This file should be

```
 # contraints file (.xdc file) for lab 4

 # Clock signal
 set_property PACKAGE_PIN W5 [get_ports clk]
 set_property IOSTANDARD LVCMOS33 [get_ports clk]
 # UNCOMMENT the below lines to generate a virtual clock required for timing
 # analysis
#create_clock -add -name sys_clk_pin -period 12.00 -waveform {0 6} [get_ports
clk]
#set_input_delay  -clock [get_clocks sys_clk_pin]  -add_delay 0.000 [get_ports
-filter { NAME =~ "*" && DIRECTION == "IN" } ]
#set_output_delay -clock [get_clocks sys_clk_pin]  -add_delay 0.000 [get_ports
-filter { NAME =~ "*" && DIRECTION == "OUT" } ]

 # leds
 set_property PACKAGE_PIN U16 [get_ports {q[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {q[0]}]
 set_property PACKAGE_PIN E19 [get_ports {q[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {q[1]}]
 set_property PACKAGE_PIN U19 [get_ports {q[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {q[2]}]
 set_property PACKAGE_PIN V19 [get_ports {q[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {q[3]}]
 set_property PACKAGE_PIN W18 [get_ports {q[4]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {q[4]}]
 set_property PACKAGE_PIN U15 [get_ports {q[5]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {q[5]}]
 set_property PACKAGE_PIN U14 [get_ports {q[6]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {q[6]}]
 set_property PACKAGE_PIN V14 [get_ports {q[7]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {q[7]}]

 #Buttons
 set_property PACKAGE_PIN U18 [get_ports rst]
      set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

Verify that the pin assignments are correct by referring to the manual. Now save the file and exit the editing program.

8.3. Synthesize your top level entity.

8.4. Implement your design as in previous labs.

8.5. After implementation is accomplished verify the pad assignment reading the corresponding report file. Among others, you should verify that the correct pins have been assigned to all inputs and outputs by viewing the corresponding reports under in the *Design Summary (Implemented) -> Detailed Reports*.

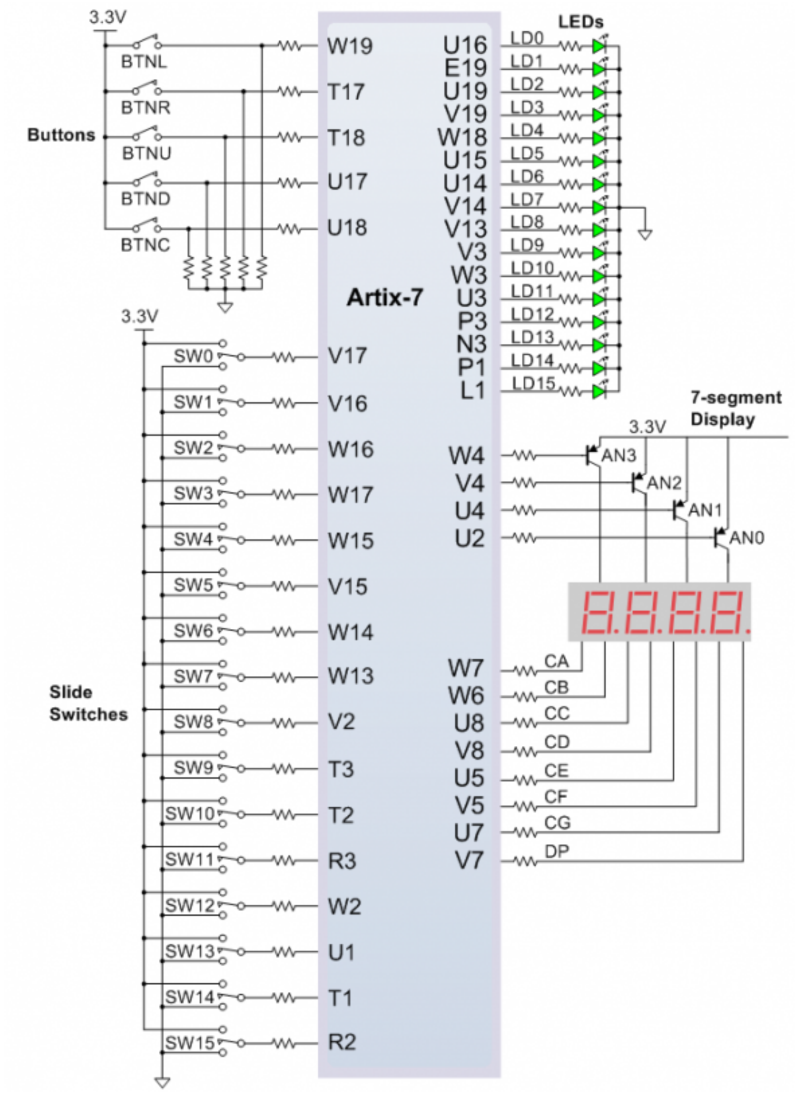
8.6. Generate a bitstream File

9. **Downloading and Testing**

Use the hardware manager in Vivado to download the bitstream to the FPGA. If everything is working properly, after you download the file the leds will immediately start displaying a sequence of pseudo random numbers, once per second. If it does not, make sure to retrace your steps. Also, make sure that your Basys 3 board is working properly.

10. **Modifying the design**

Now, modify your design such that the output drives a 7-segment LED to **display the hex characters** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F, corresponding to your 8-bit pseudo random number.



Do this by coding a Verilog module to implement a decoder from binary to decimal. which will drive the 7 segments of the 7-segment LED properly. The **easiest** way to do this is to implement the binary to 7-segment conversion with a 16-element array in your module. **In addition, add an input to the top_level module which will read 8-switches with the value of seed.** You will also need to update the .xdc file to connect the elements of seed to the switches.

You will have to implement and **demonstrate to your TA the correct functionality** (by downloading the bitstream to the board) of your modified design (with 7-Segment Display) during the lab session in order for you to obtain the full grade for this lab.

You will have to include into your lab note-book all the Verilog files of the modified counter.

(Use the reference figure on the left to define the new constraints)

Lab Notebook deliverables:

- Brief description of what you did in the lab, what steps were followed
- Discussion of results:
  a. What were the test cases you used and discuss why you chose the set of test cases, what changes did you do to test the basic LFSR(without 7-seg)?
  b. If you had to modify your design to meet your design discuss what methodology you followed, etc. (i.e. to add the 7-seg display)
  c. Attach snippet of the reports of the tool.
     i. Table and Simulation waveforms (behavioral and post timing) showing all the testcases used above. (w/o 7-seg)
     ii. Schematic screenshot of synthesized design. (w/o 7-seg)
     iii. Utilization report: synthesis and implementation (w/ 7-seg and w/o 7-seg)
     iv. Timing report (w/o 7-seg)
     v. Power report (w/o 7-seg)

  NOTE: Only attach relevant sections, not the entire report.
- Summary or Conclusion:
  a. Were all design goals met?
  b. What were the difficulties encountered?
  c. Any improvement you would like to make time permitting? Etc.…
- Verilog Source code:
  (Font: Courier, size 8pt, single line spacing, no paragraph spacing)
- All files must be labeled (ex. Figure 1: Simulation of eight-bit adder…). You should also highlight more interesting parts of your attachments.

SUMMARY – In this lab you used Vivado to design a pseudo random number generator which was implemented in the Basys 3 board.