Mechtron 3TB4: Lab 2

Building a Hardware Interface Using an FPGA

Reports Due:
Prelab: At the start of your lab session
Post-lab: At the start of your next tutorial session

Goals

- To practice building a more complicated circuit using Verilog.
- To practice more Verilog features.
- To learn to build an FSM using Verilog.

System Description

In this lab, you will create a device allowing two users to compete in testing their reaction times. When a light signal is given, the users press their keys on the DE1-SoC board as quickly as possible. The user who presses their key earlier is the winner. To achieve this, you will design a clock divider, a counter, a hexadecimal to binary-coded decimal translator, a seven-segment display handler, a random number generator, and a top-level module interfacing the submodules, as well as other auxiliary modules. You will describe this interface in the Verilog Hardware Description Language (HDL) and implement it on the FPGA on the DE1-SoC board. Some modules from your stopwatch project in the previous tutorial can be used in this project (they may require some modification).

Your system will work in the following way:

- After your project is downloaded to the DE1-SoC board, or upon the reset button (KEY1) being pressed, the HEX LEDs will blink at a frequency of several Hz, and all the red LEDRs will be off.
- The HEX LEDs will turn off after blinking for several seconds.
- The HEX LEDs will be off for a random length of time. They will then turn on, a timer will start to run and the timer's time in milliseconds will be displayed on the HEX LEDs.
- Once the HEX LEDs are on, two users can press their user buttons (KEY0 and KEY3, respectively). The user who presses their button earlier wins the contest.
- The timer reading stops once a user button is pressed. One more red LEDR will be turned on for the winner.
- LEDR0 to LEDR4 indicate the number of times that the KEY0 user has won the contest. LEDR9 to LEDR5 indicate the number of times that the KEY3 user has won the contest. The first time the KEY0 user wins the contest, the LEDR0 is on, the second time, LEDR1 is on, and so on. The first time the KEY3 user wins the contest, the LEDR9 is on, the second time for the KEY3 user, LEDR8 is on, and so on.

- Pressing the KEY2 will continue/resume the contest.
- The system should be able to detect cheating. If KEY0 is pressed and held before the HEX LEDs turn on, no one will win, and "1's will be displayed on the HEX LEDs. If the KEY3 user cheated, no one wins and "2"s will be displayed on the HEX LEDs.

Hint: You may need to use a multiplexer to set input signals to your HEX LEDs to turn them off, blink or display different numbers. You may need to use an FSM to set the selection signals for the multiplexer.

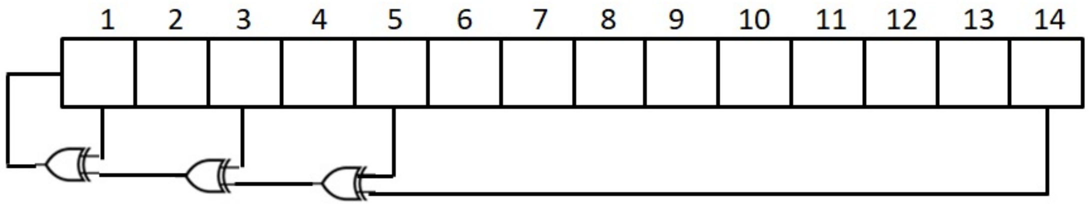
LFSR Counters for Random number generation

To keep the HEX LEDs off for a random length of time, a (pseudo) random number must be generated. Although Verilog provides a `$random` function, it is for simulation usage and cannot be synthesized. Quartus Prime comes with a random number generator IP, but it does not work (it is not clear to us why this is the case). In this lab, we will implement an LFSR (Linear Feedback Shift Register) on the FPGA to generate random numbers for our application.

An LFSR is a shift register whose input bit is the feedback of a linear function of some subset of its contents. The linear function can either be exclusive-or (XOR) or exclusive nor (XNOR). Those bits that are chosen to be XORed or XNORed to be fed back are called taps.

With XOR as the linear function, the LFSR's initial value, the seed, cannot be all zeroes, otherwise the LFSR will be "locked up" in the all zeroes state. For the same reason, if XNOR is used as the linear function, the seed cannot be all ones.

Two types of LFSRs are commonly used to produce pseudo random numbers. They are the Fibonacci LFSR and the Galois LFSR. For a Fibonacci LFSR, the taps are XORed sequentially with the output bit, and the XORed result is fed back as the input bit (see Figure 1). For a Galois LFSR, the output bit is fed back as the input bit, and is XORed individually with the bits before the taps. The XORed results are fed into the taps (see Figure 2).



A 14 bits Fibonacci LFSR, with taps on bits 14, 5, 3 and 1

Figure 1: Schematic Diagram of a Fibonacci LFSR

The LFSRs do not produce random numbers directly, instead, they produce pseudo random sequences of zeroes and ones. These sequences of bits can be used for generating pseudo random numbers. For a given LFSR with a fixed seed (initial value) these sequences can be predicted and the bit sequences will eventually repeat. However, if this repeating cycle is long enough, it is still practical to use an LFSR to generate random numbers (the resulting sequence of numbers can pass statistical tests for randomness). To maximize the length of the repeating cycle for a given LFSR, taps have to be appropriately chosen. Xilinx

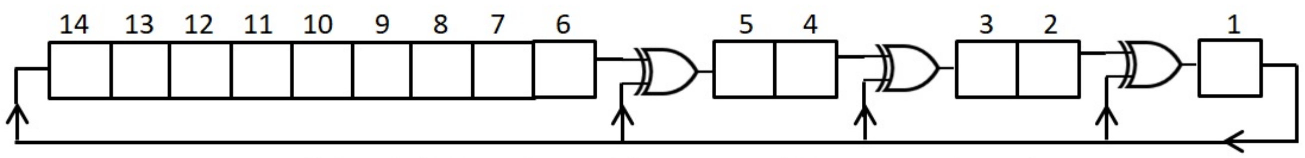


Figure 2: Schematic Diagram of a Galois LFSR

has provided a table of these taps for LFSRs from 3 bits to 168 bits. Please see the Xilinx Application Note for Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators.

Additional details on LFSRs can be found at https://en.wikipedia.org/wiki/Linear-feedback_shift_register and by performing a search on the term – there is lots of material out there on LFSRs.

LFSRs are simple to synthesize and can be efficiently implemented on an FGPA. You can use either a Fibonacci or Galois LFSR in your lab project. To make your random number generator more practical, you should use a longer LFSR. You may also need to do some work to make your random numbers fall within an appropriate range such that the resulting delays are reasonable.

Activities

Pre-lab [20]

The following activities must be completed and submitted as part 1 of the lab report (the prelab) at the start of your lab session.

1. Write your Verilog module to generate random numbers. The random numbers should be between 1000 to 5000 (or a similar range), so that they can be easily translated to a reasonable delay (in milliseconds). When a random number that falls within your range is generated, output a signal to indicate that a random number is ready. You may use the following skeleton to generate 14 bit random numbers:

```
module random (input clk, reset_n, resume_n, output reg [13:0] random,
                                output reg rnd_ready);
//for 14 bits Liner Feedback Shift Register,
//the Taps that need to be XNORed are: 14, 5, 3, 1

wire xnor_taps, and_allbits, feedback;

reg [13:0] reg_values;

reg enable=1;

always @ (posedge clk, negedge reset_n, negedge resume_n)
begin
        if (!reset_n)
                begin
```

```
                        reg_values<=14'b11111111111111;
                                //the LFSR cannot be all 0 at beginning.
                        enable<=1;
                        rnd_ready<=0;
                end

        else if (!resume_n)
                begin
                        enable<=1;
                        rnd_ready<=0;
                        reg_values<=reg_values;
                end
        else
                begin
                        if (enable)
                        begin
                                reg_values[13]=reg_values[0];
                                reg_values[12:5]=reg_values[13:6];

                                reg_values[4]<=reg_values[0] ^ reg_values[5];
                                //  tap 5 of the diagram from the lab manual

                                reg_values[3]=reg_values[4];

                                reg_values[2]<=reg_values[0] ^ reg_values[3];
                                //  tap 3 of the diagram from the lab manual

                                reg_values[1]=reg_values[2];

                                reg_values[0]<=reg_values[0] ^ reg_values[1];
                                //  tap 1 of the diagram from the lab manual

                                /* fill your code here to make sure the random */
                                /*  number is between 1000 and 5000 */

                        end //end of ENABLE.
                end
end

endmodule
```

2. Based on the project requirements, design your FSM and draw your FSM diagram.

3. Write your Verilog code for your FSM.

   Please note: In a combinational block, if a variable or a reg is not assigned a value, in some cases it will try to hold its previous value, and a latch will be inferred. An inferred latch is a danger for your circuit design as it may yield unexpected results. Coding an FSM will inevitably use many `case` or `if` statements, so care must be given to make sure all the cases are covered and all the variables are assigned values in each case. To be more clear:

- Try to include the `default` condition when using a `case` statement.
- Try to include an `else` condition when using an `if` statement.
- For variables that will be affected by the `case` or `if` statement, try to assign them default intial values.

Assigning a variable to itself will still infer a latch. In case you need a variable to remember its previous value, use a clock edge sensitive block to expressively make it a latch.

Latches can also be inferred by missing signals from a sensitivity list, so for the `always` block with level sensitivity list, try to use `always @(*)`.

4. Write a Verilog module or modules to make the HEX LEDs blink at a frequency of several Hz.
5. Write any additional Verilog modules that are necessary to complete your lab project.

In the lab [40]

1. [**10**] Create a new Quartus project to test your random number generator. Each time the reset or resume button is pressed, a new random number should be generated. Display the random numbers that fall within your desired range using the HEX LEDs on the DE1-SoC board and demonstrate your project to one of the TAs.
2. Download the starter kit project for Lab2 from the course website. Compile and download the project to your DE1-SoC board. The starter kit project works in the following way:
   (a) After the starter kit project is downloaded to the board, the HEX LEDs will blink.
   (b) After 6 seconds, the LEDR9 is turned on, indicating that player 2 has won once.
   (c) Each time KEY2 (the key to pause or resume the contest) is pressed, after 6 seconds, one more LEDR is turned on on the left side, indicating that player 2 has won one more time.
   (d) Pressing KEY1 resets the project.
3. [**30**] Complete your project for your reaction time contest device based on the starter kit project, or by creating your own project. Test and debug your project. Demonstrate your working device to one of the TAs.

Lab Reports (Part 2) [10]

This part of the lab concerns your presentation and report writing skills. Describe what you did in this lab, include the code used as well as screenshots to support your report. Answer the following questions.

1. Which type of LFSR have you used in your project? How many bits long was your LFSR? What taps have you used for your LFSR? Explain your choices.
2. What are the advantages or disadvantages if the LFSR is too short, or too long?
3. Open the compilation report in Quartus, and report the following numbers:

- Total number of logic elements used by your circuit.
- Total number of registers.
- Total number of pins.
- The maximum number of logic elements that can fit on the FPGA that you used.

Please submit one report per group.