CSE 100/L Logic Design
Fall 2025

Lab Assignment 4

Prelab deadline:

- Tuesday, October 28th by 11:59PM.

Code Submission deadlines:

- 100% Before the end of your section Thursday, October 30th.
- 95% after the end of your section Thursday, October 30th.
- 80% after 11:59PM Thursday, October 30th.
- 60% after 11:59PM Sunday, November 2nd.
- 40% after 11:59PM Monday, November 3rd.
- 20% after 11:59PM Tuesday, November 4th.
- 0% after 11:59PM Wednesday, November 5th.

Demo deadline:

- Code may be demonstrated until the end of the next lab assignment. You must submit before you demo, and the code you submit must be the code you demo. You may be required to download the code from Canvas to demo.

Write-Up deadline:

- Thursday, November 6th by 11:59PM. No late exceptions.

Back To the Future Token Opportunity

There is one token available in this lab:

- Milestone 1: Demo the first Milestone, before the end of the day on Thursday, October 30th. Check the end of the lab doc for a description of the milestone.

Academic Integrity Policy:

The Academic Integrity Policy is stated in the syllabus on Canvas. Students are responsible for reading the syllabus. Not reading the syllabus is not an excuse for academic misconduct.

- Work submitted in CSE 100 must be yours alone. Sharing/sending code, submitting code written by others, and submitting (any) code from generative AI **are all violations of academic integrity**.
- Working together on a whiteboard, on paper, debugging waveforms, explaining concepts and/or examples, is encouraged, **as long as no solution code is shared**.

Verilog Operator and Procedural Block Constraints:

In this lab you can use any of the following combinational operators:

- All combinational logic must be implemented using only `assign` statements.
- Bit-wise Operators: &, |, $\sim$, $\wedge$, and $\sim \wedge$

- Concatenation and Replication: {} and {{}}

Your designs must be **synchronous with the system clock** specified in the lab. This means:

- use only positive edge-triggered flip-flops (FDRE),
- not use asynchronous clears or pre-sets of any sequential elements,
- connect only **the system clock** as input to the clock pins of any sequential components
- not connect **the system clock** as the input to any other logic.
- you may **not** use any procedural blocks in your design (i.e. `always@`, `always_ff@`, `always_comb`).

You may only use **`assign`** statements and FDREs in your design. Use of disallowed operators or procedural blocks will be considered behavioral and will result in a **score of 0**!

Overview

In this lab you will implement a state machine as part of a sequential circuit for a single player game called Quick Decode. Be sure to complete the prelab so that you can simulate your state machine and components. The prelab steps through simulating a hypothetical state machine called MsgChk (which is not related to this lab). When simulating your state machine you will want to add the state bus to your waveform viewer. When you ask for help in the lab, the first question that will be asked is whether you have a simulation showing the error.

Resistance is futile: you will need to use the simulator!

Design Overview

You will use the BASYS3 board to implement the following single player game in which a target number is presented and the switch associated with the number must be flipped within two seconds.

1. A **Go** signal is given (pushbutton `btnC` is pressed) to start each round, however the round will not begin unless all switches are set to 0.
2. In each round, a random 4-bit binary value (the target number) is selected and displayed on the leftmost digit.
3. The round ends as soon as any switch is flipped or when two seconds have elapsed, which ever occurs first.
4. If an incorrect switch is flipped or no switch is flipped, then a point is lost and the led next to the correct switch flashes for four seconds.
5. If the correct switch is flipped then the current score flashes for two seconds, then it is incremented and flashes for two more seconds.
6. The correct switch corresponds to the value of the target number. For example if the target number is C, then switch `sw[12]`, the fourth from the left, should be flipped.
7. After 4 seconds, either way, if the score is less than 4 and greater than negative 4, then a new round can begin with a **Go** signal if the switches are all reset back to 0.
8. But if the score is currently 4 then the game has been won. The score will continue flashing and all 16 leds will also flash.
9. And if the score is currently -4, the correct led will continue flashing and the game is over. This is a loss.
10. The **Go** signal will have no effect when the game is over, win or lose.
11. The score is always displayed on the rightmost digit of the 7-segment display. When it is less than 0, it is represented as with a "-" on `an[1]` and its magnitude on `an[0]`.
12. The target number after the **Go** signal. It will remain on the display until the 4 seconds after a correct or incorrect or no switch is flipped. It will also remain displayed if the game is over.

On the BASYS3 board,

- `btnC` will be used as the "Go" signal.
- `btnR` will be used as global reset (as usual) and should only be connected to the module **qsec_clks.v** described below.
- `qsec` is high for one clock cycle every 1/4 of a second and is provided by **qsec_clks.v**. The signal `qsec` should NOT be used as a clock. It can be connected to clock enable inputs possibly with other logic to control the rate at which a counter advances, and to a counter to make a score flash. This counter may also be useful for making leds and digits flash.

Below is the block diagram of your entire system; it has two pushbuttons, the 16 switches, and the BASYS3 clock (`clkin`) as inputs, and the outputs are the 7-segment displays and 16 leds.

(The BASYS3 clock clkin and global reset btnR are inputs, but will not be part of the rest of your logic. They are connected only to the module **qsec_clks.v**.)

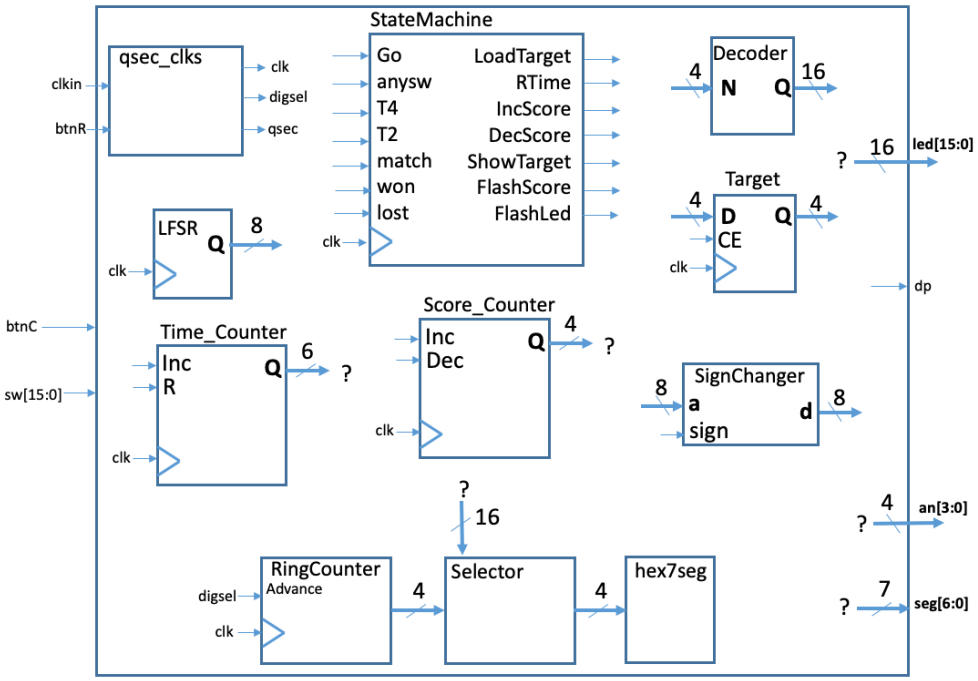

Figure 1: Lab 4 overview

Below is an example of what the top level ports list can look like:

```
module top (
    input clkin,
    input [15:0] sw,
    input btnC,
    input btnR,
    output [3:0] an,
    output [6:0] seg,
    output [15:0] led
);

//logic

endmodule
```

You may reuse code from your previous labs, but it must be your own code.

Random Number Generator

You will use a Linear Feedback Shift Register (LFSR) to generate a random 8-bit binary number. Below is an 8-bit linear feedback shift register.

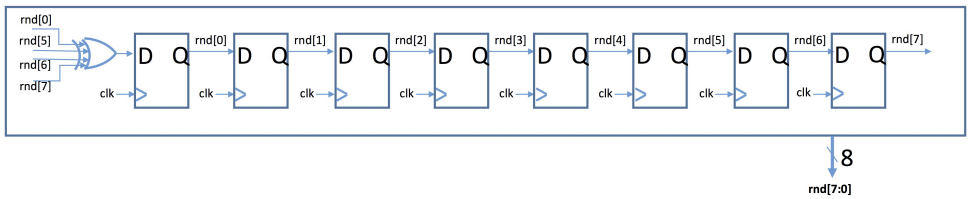

Figure 2: 8-bit Linear Feedback Shift Register

This LFSR is simply an 8-bit shift register where the input to the first register is the XOR of specific bits in the register. If all the bits in the register are 0 then the LFSR output will always be 0's. But otherwise it will go through a sequence of all 255 non-zero states before it repeats.

This sequence is not random, but reading the LFSR at random times (assuming it cycles through enough states fast enough) will give you a random 8-bit number in the same way a roulette wheel provides a random outcome. The choice of inputs into the XOR gate is not arbitrary so be sure to use the correct inputs.

Below is the module port list for the **lfsr** module:

```
module lfsr(
    input clk_i,
    output [7:0] q_o
);

//logic

endmodule
```

Time Counter

You will need a loadable counter that can load a 6-bit binary value and count down to 0 (or vice versa). This will be your method of reseting the counter. Since you have completed Lab 3 you should have one handy. You can actually use your 16-bit counter and leave the top bits unconnected. The tools will remove any logic that is not useful and that will turn your 16-bit counter into a smaller counter. In case it wasn't obvious enough you should directly reuse your UD16L counter from Lab 3. Below is the module port list for the **time_counter** module:

```
module time_counter(
    input clk_i,
    input inc_i,
    input dec_i,
    input reset_i, //input ld_i
    input [5:0] din,
    output [5:0] q_o
);

//logic

endmodule
```

Decoder

An n-bit decoder has n inputs and 2n outputs. Only one of the outputs is high, the one corresponding to the value represented by the n-bit inputs. Specifically, the ith output is high when the input represents the value i in binary. You can read a bit about decoders in Section 2.11.1 of the text. Comparing the decoder output with the switches will provide the match input to the state machine that indicates whether the correct or incorrect switch was flipped.
Below is the module port list for the **decoder** module:

```
module decoder(
    input [3:0] in_i,
    output [15:0] out_o
);

//logic

endmodule
```

State Machine (FSM)

In the overview diagram there is a block for the state machine you must build. You may add additional inputs and/or outputs, particularly for debugging. Start by drawing a state diagram. Make sure you consider all cases (including edge cases), for example if Anow and Bnow are flipped exactly when time expires.

Tip 1: We *strongly* recommend that you have a correct state machine simulating **before** your second section.

Tip 2: Getting a state machine right usually requires several iterations. It is likely that in demonstrating your design the TA will discover some case that is not properly handled. Often changes to the state machine are not simple. A complete redesign may be necessary. Trying to patch it by changing one signal here or there, or inserting a gate or gates **almost always makes things worse.**

Tip 3: The BASYS board has plenty of LEDs. Use them to help debug your state machine, by driving the state bits to the LEDs.

This is the beginning of your digital logic design journey. Please leave yourself enough time to revise!

Required FSM IO

Although you as a designer have a choice of adding additional IO in the state machine, there are a couple of IO signals that are required.

1. **Go**: To initiate a round, the **Go** signal must be set high for a cycle while the FSM waits for a new round to start. This **Go** signal is tied to `btnC`.

   Note that the inputs from btnC is not synchronized with your clock (it is asynchronous). You should pass this input through a syncronizer (D Flip-Flop) before using them in your design.

2. **Anysw**: This is an input that tells the FSM that one of the switches on the board has been flipped.

3. **FourSecs** and **TwoSecs**: These signals are created using the output of the **Time Counter** module with some additional logic. Hint: is there a signal created by a module that you know oscillates at a known frequency?

4. **Match**: This signal is used to determine whether the player correctly matched the switch that corresponds to the value of the target number. Equilavence is easily checked with a single logic gate. Hint: it is one of the lesser known ones.

5. **Won** and **Lost**: These are the signals that tell the FSM whether the game has been won or lost in its entirety.

Gratuitous Advice: Getting a state machine right usually requires several iterations. It is likely that in demonstrating your design the TA will discover some case that is not properly handled. Often changes to the state machine are not simple. A complete redesign may be necessary. Trying to patch it by changing one signal here or there, or inserting a gate/FF, or more, almost always makes things worse. Please leave yourself enough time. Hurrying will introduce more bugs that you will need to hunt down. It is strongly suggested that your entire design be entered and simulating before your second section.
Below is the module port list for the **fsm** module:

```
module fsm(
    input clk_i,
    input Go_i,
    input four_secs_i,
    input two_secs_i,
    input match_i,
    input anysw_i,
    input won_i,
    input lost_i,
    output load_target_o,
    output reset_timer_o,
    output inc_score_o,
    output dec_score_o,
    output show_target_o,
    output flash_score_o,
    output flash_led_o
);

//logic


endmodule
```

Recommended Design Procedure

1. Read all of the instructions below before beginning any design.
2. Read the **entering a state machine with Verilog** tutorial in Canvas and complete the pre-lab.
3. Make the modules for your components and if they have flip flops make sure they initialize to the appropriate values. For example, your LFSR should start with the contents `8'b10000000` when the global reset (`btnR`) is asserted, not the Go signal (`btnC`).
4. Using the state diagram you drew in the pre-lab, obtain the next state and output logic equations. Remember to use one-hot encoding.
5. Enter the logic for your next state and output equations using the **assign** statement and provide flip-flops to hold your present state.
6. Remember that the global reset will reset all flip-flops. Make sure this will put your state machine into its initial state.
7. Simulate your state machine (state logic with flip-flops). It is much easier to catch problems when you control the FourSecs and TwoSecs signals rather than waiting many clock cycles for them in the simulation of the whole design.
8. In your top level, connect your state machine, LFSR, decoder, counters, registers, the inputs and outputs, and anything else you need.
9. Create a net named `clk`, connect it to the clock input of your state machine, LFSR, 8-bit loadable counter and any other sequential components. This is the **system clock** for the design. It is the only signal which can be used as a clock in your design.
10. Download **qsec_clks.v**, found in the Canvas files, into your project directory.
11. In the Vivado Project Manager, add it to your project. Make sure you select the option to copy it into your project.
12. Add an instance of the module **qsec_clks** to your top level as follows:

```
qsec_clks slowit (
    .clkin(clkin),
    .greset(btnR),
    .clk(clk),
    .digsel(digsel),
    .qsec(qsec));
```

    `clkin` is your system clock. The signal `digsel` should be used to advance the Ring Counter for the 7-segment displays; **it should not be used as a clock!!!**

    `qsec` is high for one clock cycle each 1/4 second (4 times per second) and should be used to advance the Time Counter; **it should not be used as a clock!!!**

13. Simulate your entire project. Although each of your parts may have passed your simulation tests, it is possible they may not work well together.
14. You should simulate until the signals from your timer go high to make sure your State Machine and the Time counter are working properly together.

- You should test several different scenarios (see requirements for the Write-Up below). The TAs may ask for others as well. Be sure to display the signals mentioned below.
- The `qsec` signal is not 1/4 second during simulation; it is intentionally much faster so that you don't have to simulate large numbers of cycles before the green light turns off. You can also reduce the time by choosing when the Go signal (`btnC`) goes high. Simulate and look at the output of your LFSR. Find a small value and make `btnC` high at that time so that your Time Counter loads a small value. The LFSR is not random. What is random is the moment you sample it, and this can be controlled in the simulation.

15. ***Implement your design, configure the FPGA and demonstrate your design to the TA. The TA will ask to see your simulation results for the top-level.***

    - Be prepared to explain your waveforms and make sure the test scenarios requested in the write-up (see below) are shown in the simulator wave window ahead of your demonstration.
    - The TA will also ask you to demonstrate these scenarios on the board (you may use the cheat switches to show the results of concurrent flips).

16. Remember to archive your project and back it up the zipfile. Make sure you upload it to Canvas if you are not certain of demoing before the next deadline.

17. Once your working design has been demonstrated, make sure your working project's zipfile is uploaded to the Lab submission assignment. The zipped project file must be submitted by the due date of the write-up. You can continue to improve your project for the write-up if desired, but you should submit the version you demonstrated.

Rubric and Token Opportunities

The late policy is applied on top of the rubric shown below.

- 100% - Fully Working Lab 4, exactly following the specification provided in the lab document.
- 90% - Fully working Lab 4, with some small error at the discretion of the TA.
- 80% - Overall Gameplay works, multiple rounds can be played, 2- and 4-second delays are correctly implemented but the digits do not flash for success/failure.
- 70% - Overall Gameplay works, multiple rounds can be played, 2- and 4-second delays are not implemented in addition to digits not flashing.
- 60% - A single round of gameplay works correctly. Buttons or switches are used to transition between states and update numbers.

BTTF Tokens

Reaching the following milestones on time will allow you to earn BTTF tokens. You must demo the milestone by the deadline to earn the token.

BTTF Tokens

1. Milestone 1 (1 token): Demonstrate your in-progress Lab 4 project:

   Debugging state machines is often an arduous task. To get this token you must show your TAs a detailed state machine diagram AND implement it on the FPGA. This diagram can be on paper or on any drawing software, such as draw.io or OneNote. Use the buttons and switches to probe the FSM and output all outputs onto the LEDs. Among these outputs, you can output the state encoding itself to track the current state of the FSM. This is common practice from previous students in 100 and 125/225.

   To earn the BTTF token, you must showcase that the provided state machine diagram is correctly implemented on the FPGA during a demo. This means YOU must explain your simulation, identify your inputs/outputs, show how your states are transitioning and explain why. You cannot just show the TA a waveform and expect to get a token. This token is not available after the lab deadline.

Write-Up

See the lab write-up guide for general instructions on what to include in the written report and how to submit electronically. In the write-up for Lab 4 be sure to:

- Include your state diagram, next state and output logic equations. The diagram can be hand drawn or drawn using with a program(aka app).
- Describe each state of your machine in words.

  Example (not necessarily from this lab): "The machine is in state WIN when player has won the game. In this state an[1:0] flash in unison and alternate with an[3]. The machine stays in state WIN for four seconds then goes back to IDLE."
- For the appendix you will need to provide screenshots of the Waveform Viewer showing simulation results for each of the scenarios below. The following input and output signals should be included: clk, btnR, btnC, sw[], led[], as well as the LFSR contents, the Time Counter, the state bits of the State Machine. Display buses as buses in hex.
  - Player winning by matching the correct switch with the target.
  - Player loses by not matching the correct switch with the target.
- The following supplementary material for Lab 4 should be submitted electronically as appendices in your report **within the one PDF file (not in separate files)**.
  - All of your schematics for new modules.
  - A screenshots of the Waveform Viewer showing your simulation results.
  - The scanned notebook pages or other drawings for this lab.