Engs 31 / CoSc 56

Homework 4: RTL Design

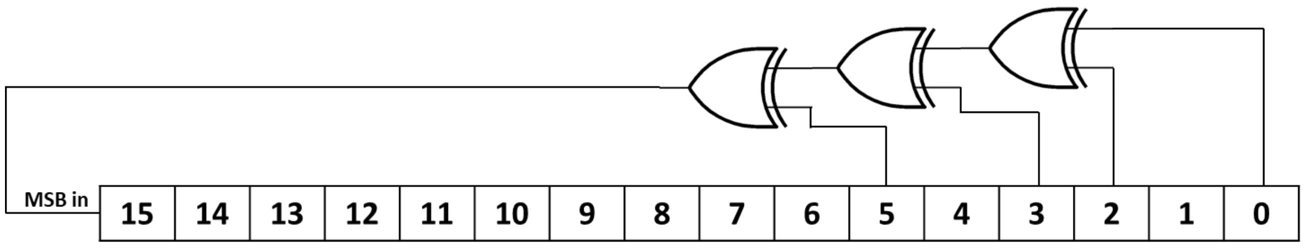
Due Wednesday, May 1

*Complete the following design and VHDL programming assignments. The VHDL assignments should be completed using EDA Playground or Vivado. (**50 points total**)*

1. **White Noise Generator (15 Points)**

You are going to design a white noise generator. You will generate the white noise by producing a sequence of pseudorandom numbers that will be sent to a digital-to-analog converter (DAC) connected to a speaker. In this problem, you will not worry about the DAC interface, you will just focus on generating the stream of pseudorandom numbers.

In order to generate the pseudorandom numbers, you will need to design a Fibonacci linear-feedback shift register (LFSR). An LFSR is a simple circuit that can generate pseudorandom numbers while cycling through all possibilities. A diagram for a 16-bit Fibonacci LFSR is shown below and more details can be found on the Wikipedia page.



16-bit shift register

Your design will make use of the 16-bit Fibonacci LFSR to generate a stream of random numbers. You want to produce a random number every 25 μs (i.e., the sampling rate of the DAC is 40 kHz). Your system clock is 1 MHz. The component has the following inputs and outputs:

Inputs: **clk** – 1-MHz clock input

Reset – A 1-bit control signal that tells the shift register to perform a parallel load operation to set the register equal to the input, **Seed**.

Seed - A 16-bit std_logic_vector that contains the starting number for the LFSR. This number gets loaded into the register when **Reset** is 1.

Outputs

Random_Number – 16-bit std_logic_vector that gets updated at a rate of 40 kHz. The value should remain constant for 25 µs until the next pseudorandom number is ready.

Number_Ready – 1-bit control signal that goes high for 1 clock cycle when each new pseudorandom number is available.

Part a) Draw an RTL-level block diagram of the system. This can be done without a controller. Use only datapath components. Clearly label all components, signals, inputs, and outputs.

Part b) Write well-commented VHDL code implementing the design using EDA Playground. Design a testbench to thoroughly test the solution. Provide a copy of your code and a link to your EDA Playground solution in your homework submission.

2. **Code Guessing Game** (**35 Points**)

You must design a simple logic game for two players. The overall rules of the game are as follows:

1) Player A thinks of a 4-digit code. Each digit can take on a hexadecimal value ranging from 0 to F. Examples of valid codes are "1 2 3 4" and "A 0 9 B." Player A enters the code one digit at a time. After 4 digits have been entered, the game automatically moves onto the guessing stage.

2) Player B tries to guess the code by entering their own 4-digit sequence. After the 4 digits have been entered, the game compares the guess to the code. For each digit, one of three outputs is asserted TRUE: **GT**, **EQ**, or **LT**. **GT** is TRUE if the particular digit of the code is greater than the guess. **EQ** is true if the particular digit of the guess is correct. Otherwise, **LT** is TRUE. This gives Player B feedback on whether to make each digit larger or smaller.

3) If all the digits of the guess are correct, then Player B wins. If not all digits are correct, then repeat step 2 a total of 4 times. If the guess is not correct after 4 guesses, then Player B loses and the game resets.

Here is an example of how a game might play out:

Player A enters the code of "A 4 1 C."

Player B guesses "8 8 8 8"

The game gives the following feedback to Player B:
GT = "1001" (since A > 8 and C > 8)
EQ = "0000" (since none of the guesses are correct)
LT = "0110" (Since 4 < 8 and 1 < 8)

Player B guesses "A 4 3 B"

The game gives the following feedback to Player B:
GT = "0001" (Since C > B)
EQ = "1100" (Since the first two guesses are correct)
LT = "0010" (Since 1 < 3)

Finally, Player B guesses "A 4 1 C" The game ends and Player B wins.

This game logic will make use of the following inputs and outputs:

Inputs: **clk** – Clock input

Number – A 4-bit std_logic_vector that contains a single digit that the user has selected. Depending on the state of the game, this will either be a digit that Player A inserts for the code or a digit that Player B entered as part of their guess.

KeyPress – This is a 1-bit control signal that goes high for 1 clock cycle each time the user enters a new number. So when the user enters a new digit, the **Number** input will update to the selected number, and **KeyPress** will go high for one clock period. See the supplied Testbench for how the inputs are applied.

Outputs: **GT** – A 4-bit std_logic_vector where each bit of **GT** corresponds to a single digit of the guess. If the code digit is greater than the guess digit, the bit is 1. Otherwise it is zero.

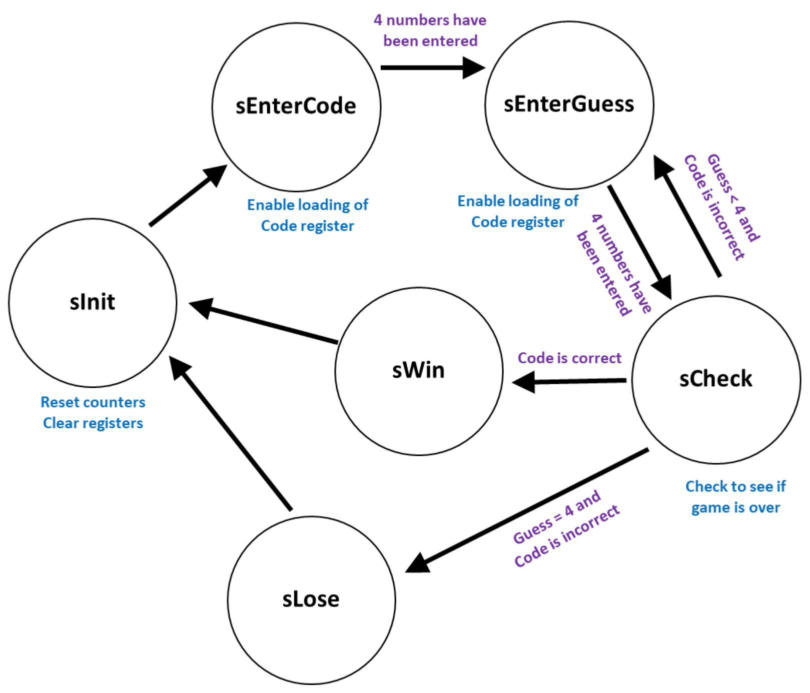
EQ – A 4-bit std_logic_vector where each bit of **EQ** corresponds to a single digit of the guess. If the code digit is equal to the guess digit, the bit is 1. Otherwise it is zero.

LT – A 4-bit std_logic_vector where each bit of **LT** corresponds to a single digit of the guess. If the code digit is less than the guess digit, the bit is 1. Otherwise it is zero.

Win – A 1-bit control signal that goes high for a single clock cycle after Player B correctly guesses the code

Lose – A 1-bit control signal that goes high for a single clock cycle after Player B makes 4 incorrect guesses.

Part a) A high-level state machine is provided below. Make sure you understand the overall operation of the game. Then, identify and list the datapath components that you will need to use in the design (e.g., registers, counters, etc.).


4 numbers have been entered
sEnterCode
sEnterGuess
Guess < 4 and Code is incorrect
Enable loading of Code register
Enable loading of Code register
4 numbers have been entered
sInit
sWin
Code is correct
sCheck
Reset counters Clear registers
Check to see if game is over
Guess = 4 and Code is incorrect
sLose

Part b) Construct a datapath for the game. Name all control signals and component ports using descriptive names. You should use higher level components (such as counters and comparators) whenever possible, instead of using gate-level logic. Use of a few gates as "glue logic" is acceptable.

Part c) Modify the high-level state machine from Part a) to be a finite state machine (FSM). Remember that all inputs and outputs to the FSM must be Boolean. All higher level operations are performed in the datapath and control signals are used to communicate between the two. Make sure all control signal naming is consistent with the datapath.

Part d) Write well-commented VHDL code that implements your design. Starter code that includes a reasonable testbench is available here: https://www.edaplayground.com/x/eZax Make sure to include an annotated waveform with your submission that verifies correct operation.