Lab 3: Adventure Game

Assignment Objectives:

- Design a finite state machine (FSM) that implements an adventure game using **schematic** input.
- Implement and test your design on an FPGA.

Introduction:

The adventure game that you will be designing has seven different locations and one object (a sword). The game begins in the Cave of Cacophony. To win the game, you must first proceed through the Twisty Tunnel and the Rapid River. From there, you will need to find a Vorpal Sword in the Sword Stash. The sword will allow you to pass through the Dragon Den safely into Victory Vault (at which point you have won the game). If you enter the Dragon Den without the Vorpal Sword, you will be devoured by a dangerous dragon and pass into the Grievous Graveyard (where the game ends). The map for the game is shown in Figure 1.

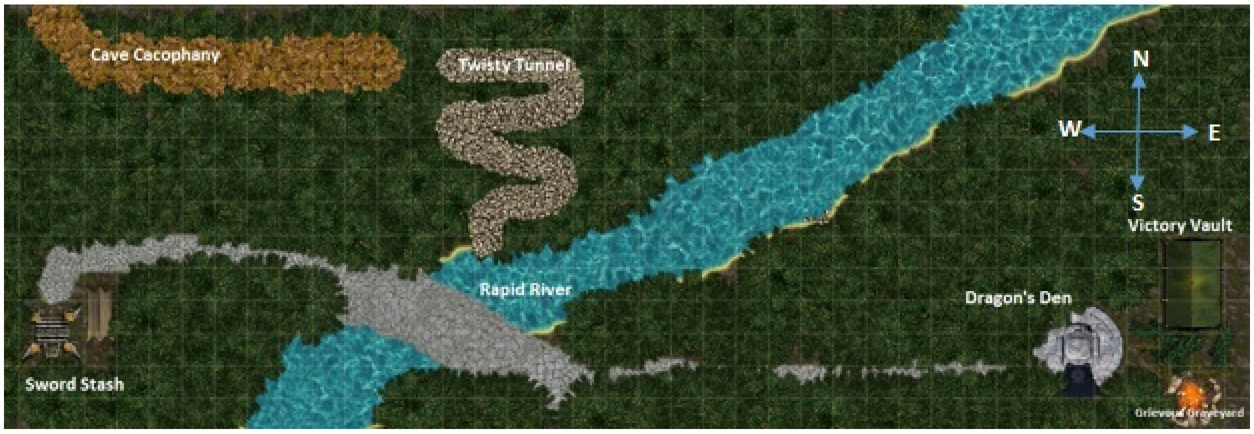

Figure 1. Game map.

This game can be factored into two FSMs communicating with each other. One state machine keeps track of your location, while the other keeps track of whether you currently have the sword.

The state transition diagram of the Location FSM is shown in Figure 2. In the location FSM, each state corresponds to a different location. Upon reset, the initial state is set to the Cave of Cacophony (CC). A player can move among the different locations using the directional inputs N(north), S(south), E(east), and W(west). When in the Sword Stash (SS), the output SW(sword) from the location FSM indicates to the Sword FSM whether the player has found the sword. When in the Dragon Den (DD), the output V (victory) of the Sword FSM determines whether the next state of location FSM will be Victory Vault (VV) if V=1 or Grievous Graveyard (GG) if V=0. Note that **no directional input is needed for the last two states.** When in "GG" and "VV", the FSM asserts the D(dead) and the WIN output, respectively. **Whenever R(reset) is pressed, the FSM gets reset, and the game starts again from "CC" at the next clock edge (synchronous reset).**

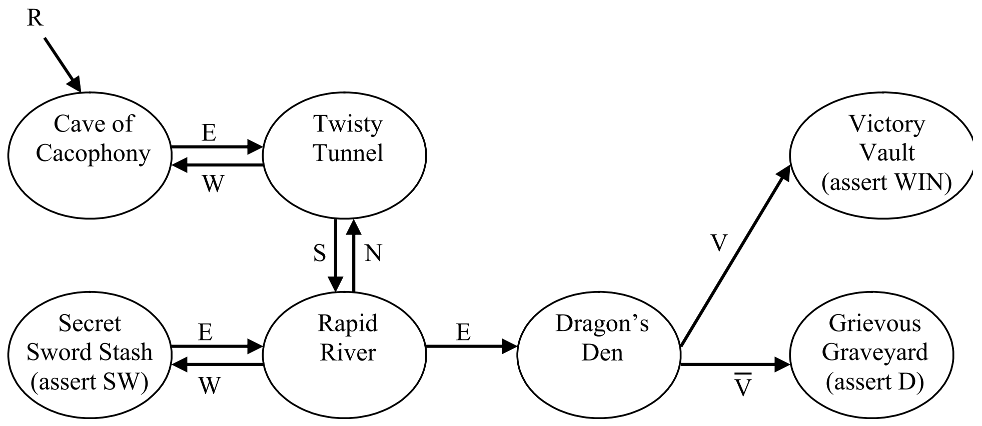
R
Cave of Cacophony
E
W
Twisty Tunnel
S
N
Victory Vault (assert WIN)
V
Secret Sword Stash (assert SW)
E
W
Rapid River
E
Dragon's Den
$\overline{V}$
Grievous Graveyard (assert D)

Figure 2. State transition diagram of the location FSM.

The state transition diagram of the Sword FSM is shown in Figure 3. There are only two states: No Sword (NS) and Has Sword (HS). Upon reset, the initial state is set to "NS". Entering the "SS" location causes the player to pick up a sword, so the transition to the "HS" state is made when the SW input (the SW output of the location FSM that indicates the player is in the "SS") is asserted. Once the "HS" state is reached, the V (vorpal sword) output is asserted, and the sword FSM stays in that state until reset(R).

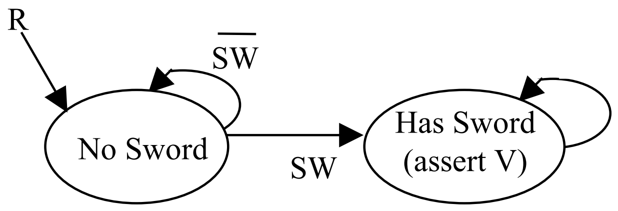

Figure 3. State transition diagram for sword FSM.

The states of these FSMs are stored using registers. Since registers are synchronized by a clock, there must be a clock input to each FSM, which determines when the state transitions may occur.

The block diagrams showing the input/output ports of each state machine are shown in Figure 4.

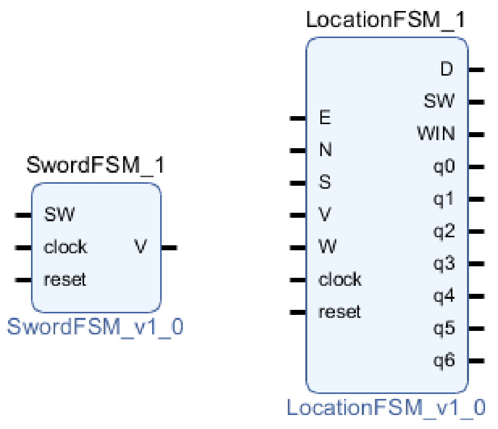

Figure 4. Block diagram of the sword and location FSMs.

Setup:

1) Create a new project.
2) Download the XUP library folder attached with Lab3 and place it into your newly created project folder. The provided XUP library should include an edge detector IP (Edge_Detector), a settable flip-flop (xup_dff_set), and a resettable flip-flop (xup_dff_reset) that you will need for the design.
3) Three testbenches are provided for simulation and functional verification purposes: “Location_fsm_tb.v”, “Sword_fsm_tb.v” and “final_tb.v”. Download and add them to your project.

Things to do:

1. Complete the state transition table for the two FSMs.
   The state transition logic takes both the directional inputs and current state as the inputs. Note that you should assume only one-directional input can be asserted at a time, and the possible values of inputs {N,S,E,W} are limited to: {0 0 0 0}, {0 0 0 1}, {0 0 1 0}, {0 1 0 0}, {1 0 0 0}. This indicates that if any of the directional input is asserted, then the rest of the directional inputs must be 0. In other words, the rest of the directional inputs no long carry information. This is why when one of the directional input is asserted, we can take the rest of the directional inputs as “don’t care”. **In the truth table of the transition logic, it is also important to make sure the current state will remain unchanged if the directional inputs are {N,S,E,W} = 0 0 0 0. Especially when the state is in “GG” or “VV”, it should remain unchanged regardless of the inputs.** Following these principles, the state transition table of the location FSM can be derived as follows (the first two states are provided). Following this clue, you can derive the overall state transition table of the location FSM in **15** lines. Then, derive the state transition table of the **sword FSM** following the same principles.

Input						Output
Current State	N	S	E	W	V	Next State
CC	X	X	1	X	X	TT
CC	X	X	0	X	X	CC
TT	X	X	X	1	X	CC
TT	X	1	X	X	X	RR
TT	X	0	X	0	X	TT

2. Choose your state encoding scheme
   As we discussed in the class, the one-hot state encoding scheme can often simplify your state transition and output logic at the cost of using more state registers. **If you feel that you have a hard time reducing the logic based on a binary state encoding, we recommend you try out the following one-hot encoding instead and see how it makes the difference.** Don't forget to **do** it for the **sword FSM** as well. For the **sword FSM,** there are only two states, so using the binary encoding is fine.

State	Encoding Q[6:0]
CC	0000001
TT	0000010
RR	0000100
SS	0001000
DD	0010000
GG	0100000
VV	1000000

3. Based on the encoding schemes, you can now convert the state transition tables of the location and sword FSMs into **truth tables**.
4. Based on the truth tables, you can derive the corresponding Boolean expressions for the state transition logic. D[i] and Q[i] is the input and output of the i^{th} bit of the state register. With respect to the state transition logic, the inputs include N, S, E, W, V, Q (current state), and the outputs are D (next state). Fill in the logical expressions of the state transition logic in the table below. Don't forget to **do** it for the **Sword FSM** as well.

D[0]=
D[1]=
D[2]=
D[3]=
D[4]=
D[5]=
D[6]=

Boolean expressions of the state transition logic for the location FSM

Note: in the one-hot encoding scheme, we assume only a single bit gets asserted at a time, same as the directional inputs. In other words, when one bit of the state registers gets asserted, the rest of the bits will be deterministic and no longer carry information. For this particular reason, all of the "0"s show up in the current state (as inputs of a truth table) can be considered as "don't care" as well. Take advantage of the don't-cares to simplify your Boolean expressions.

5. Complete the output table of the location and sword FSM, respectively.
   Since the outputs are associated with the states in the two FSMs, the output logic only takes the current state as input. The output table of the location FSM can be derived as follows (the SW output is provided as an example). Following this, you can derive the rest of the output table for both the location and sword FSMs.

Input	**Output**		
Current State	**SW**	**D**	**WIN**
CC	0		
TT	0		
RR	0		
SS	1		
DD	0		
GG	0		
VV	0		

   Based on the state encoding schemes, convert the output table into the truth table of the output logic for each of the FSMs. Then, derive the Boolean expressions of the output logic for each of the FSMs.
6. Design each FSM based on your state encoding scheme and the Boolean expressions using schematic input, respectively.
   Note: To enter the initial state of "0000001" upon a reset, you should use the settable flip-flop "xup_dff_set" (sets output to 1) from the XUP library for Q[0] and the resettable flip-flop "xup_dff_reset" (resets output to 0) for the other register bits.
7. Test each of the FSMs with the provided testbenches (Location_fsm_tb.v and Sword_fsm_tb.v).

Note: To observe the correct behavior of memory-related IPs, avoid any operations in the first 200ns of your simulation by adding a 200ns delay to the beginning of the simulation in all your testbenches.

8. After validating the functionality, create a custom IP block for each of the FSMs. Then, build the entire game system, as shown in Figure 6.

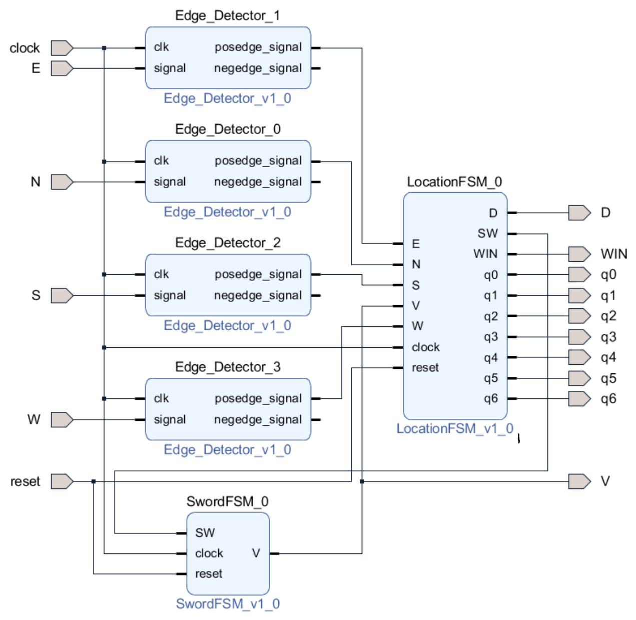

Figure 6. **Final Block Diagram**

9. The provided edge detector IP block detects a positive or negative edge of the input signal. The corresponding output remains high for one clock cycle once an edge is detected at the input. The waveform of the edge detector is shown in Figure 7. Add an edge detector module at each input N, S, E, and W to pass the input signals from the push buttons to your Location_fsm. Any idea about why we are using edge detectors?

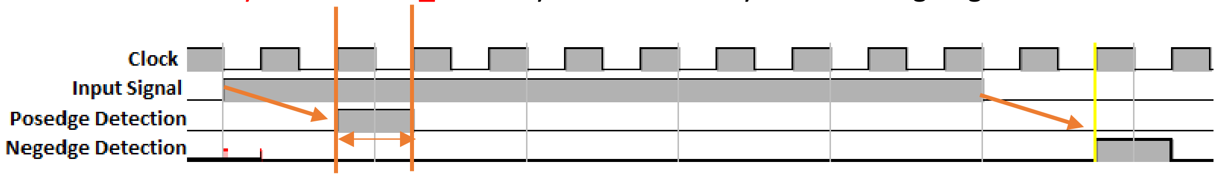

Figure 7. **Edge detector waveform**

10. Verify the functionality of your final system design using the provided testbench (final_tb.v).
Note: To observe the correct behavior of memory-related IPs, avoid any operations in the first 200ns of your simulation by adding a 200ns delay to the beginning of the simulation in all your testbenches.

11. Create a constraint file to define:
    1) The pin assignments shown in Figure 8.

I/O Port	Package Pin	Resource on Board
clock	E3	System Clock
reset	V10	Switch 15
N	M18	Push Button Up
S	P18	Push Button Down
E	M17	Push Button Right
W	P17	Push Button Left
V	R12	LED 16 Blue
D	N16	LED 17 Red
WIN	R11	LED 17 Green
q0	H17	LED0
q1	K15	LED1
q2	J13	LED2
q3	N14	LED3
q4	R18	LED4
q5	V17	LED5
q6	U17	LED6

Figure 8. Pin assignments.

2) A 100 MHz clock signal for timing analysis. The syntax to create a clock in constraint files is:
   create_clock -period 10 [get_ports {*your_clock_port_name*}]
   For further clock options, you may read the Xilinx manual vivado-using-constraints.pdf, around page 80/192.

12. Generate the bit file and test your design on a "Nexys A7-100T board".
13. Read the timing report of your design after implementation and answer the following questions:
    1) Given that the system clock frequency of the Nexys A7-100T board is 100MHz, what is the worst-case setup and hold time slack of your design, respectively?
    2) What is the maximum clock frequency your design can run at? Explain how you derive the answer.
    3) Simulate your design with a 5GHz clock signal to see what will happen if you clock your design at a frequency higher than the maximum frequency? Is it still functional? Why? To observe the system behavior at 5GHz, change the period of the clock in your testbench "final_tb.v" to 0.2ns and re-run "Post-Implementation Timing Simulation". Notice the difference in state transitions at 5GHz and think about why.
       Note: To change clock frequency for "Post Implementation Timing Simulation", you need to modify the value of "Time_Period" at line 52 in final_tb.v. To change clock frequency used for timing analysis, you need to modify the value of -period "10" in the constraint file.

Due Date:

Refer to Canvas.

What to Turn in for Grading

- The submission should be made online via Canvas.
- Compress the submission folder into a zip archive file named cse320-yourlastname-lab03.zip.
- The submission folder should contain a report that includes
  1) The state transition and output tables for each FSM.
  2) The Boolean expressions of the state transition and the output logics for each FSM.
  3) A screenshot of the location FSM schematic (please expand the schematic).
  4) A screenshot of the sword FSM schematic (please expand the schematic).
  5) A screenshot of the top-level schematic of the entire system.
  6) A screenshot of the post-implementation timing report showing the critical path of your design and the worst-case setup time slack.
  7) A screenshot of the post-implementation timing report showing the shortest path of your design and the worst-case hold time slack.
  8) Answers to the above questions about timing.
  9) A screenshot of the simulation waveform of your location FSM (Location_fsm_tb.v) from behavior simulation.
  10) A screenshot of the simulation waveform of your sword FSM (Sword_fsm_tb.v) from behavior simulation.
  11) A screenshot of the simulation waveform of your entire system (final_tb.v) from the "Post Implementation Timing Simulation" at the clock frequency of 100MHz.
  12) A screenshot of the simulation waveform of your entire system (final_tb.v) from the "Post Implementation Timing Simulation" at the clock frequency of 5GHz.
- The submission folder should also contain the following:
  1) A Xilinx Vivado project folder that contains your design, testbenches, and constraint files.
  2) The generated bit file for FPGA programming.
- Failure to follow these instructions may cause a deduction of points while grading your assignment.

Grading Rubrics

Inspection Item	Point
Correct state transition and output tables for the location FSM	8
Correct state transition and output tables for the sword FSM	2
Correct Boolean expressions for the location FSM	8

Correct Boolean expressions for the sword FSM	2
Location FSM schematic	8
Sword FSM schematic	2
Correct simulation waveform under the test of Location_fsm_tb.v	12
Correct simulation waveform under the test of Sword_fsm_tb.v	3
Correct simulation waveform under the test of final_tb.v (Post-Implementation Timing Simulation)	5
Top level schematic	5
A proper constraint file: pin assignments and clock constraints	10
A screenshot of the timing report showing a positive setup time slack	5
A screenshot of the timing report showing a positive hold time slack	5
Working bit file at the clock frequency of 100MHz	10
Correct answers to the questions about timing	10
Simulation waveform of the post-implementation timing simulation at 5GHz	5
Total	100