Laboratory 3

Rock Paper Scissors

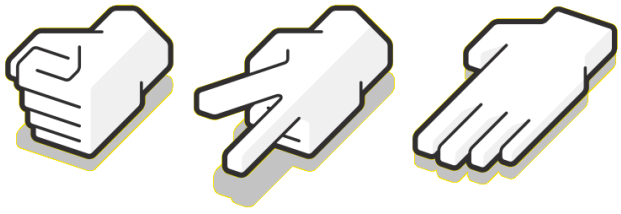
3.1 Outcomes and Objectives

The outcome of this lab is to instantiate a rock, paper scissors games on the FPGA development board. Through this process you will achieve the following learning objectives.

- Analyzing a word statement for a logic function
- Creating a truth table description for a logic function
- Writing concurrent signal assignment statements for a logic function
- Definition of Verilog modules
- Instantiation of Verilog Modules
- Creating a pin assignment for a module
- Synthesizing a module on the FPGA development board

3.2 The Rock Paper Scissors Game

The game of rock, paper, scissors is a two-player game whose goal is to beat the throw of the opposing player. Traditionally, each player throws one of three plays, rock, paper, or scissors by extending their hand in the shape of the object.



The rules of the game state that:

Rock beats scissors Scissors beats paper Paper beats rock

Since prior knowledge of your opponent's throw would provide an unfair advantage, the two players make their throws at the same time. Your goal in this lab is to create a digital

version of rock, paper, scissors on the Altera Cyclone V Board using the inputs and outputs shown in Figure 3.1. Each player will have access to three slide switches and one 7-segment display. Each of the three switches represents one of the three plays and the 7-segment will display the throw when the Play button is pressed. The Win/Lose 7-segment display will show "1" when player 1 wins, "2" when player 2 wins, and "d" when the game is a draw (a tie).

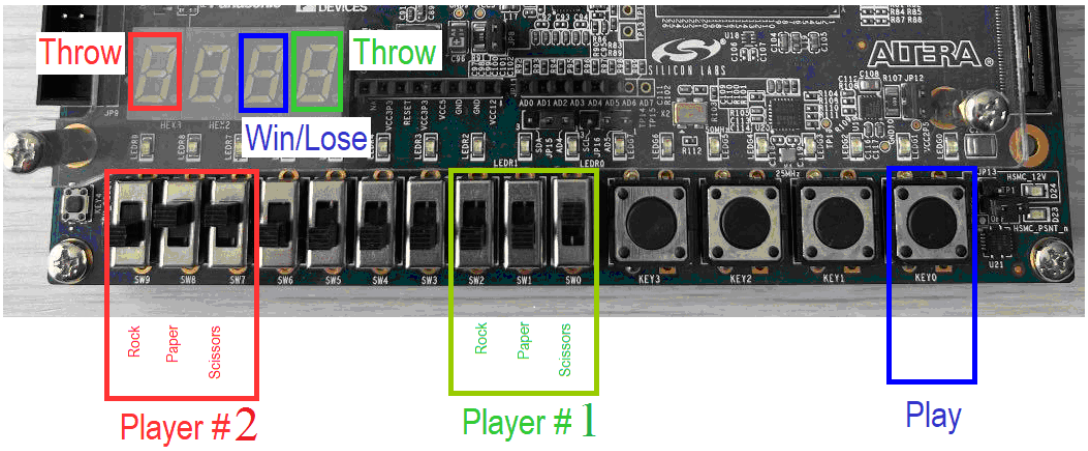


Figure 3.1: The input and output you should use to realize your digital system.

A player will move one of the three slides-switches into the up position to indicate their play. Moving more than one slide-switch, or no slide switch into the up position is in an invalid play. An invalid play always loses to a valid play. If each player throws an invalid play, the game is a draw.

While each player is making their choice of play, their Throw 7-segment display will reflect their choice as shown in Figure 3.2. These patterns are supposed to vaguely resemble the objects.

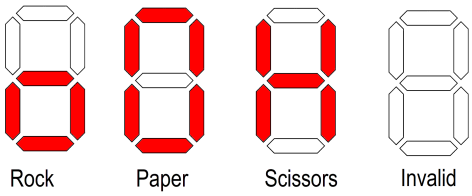


Figure 3.2: Illuminated patters for the different plays.

When the Throw button is pressed, the Win/Lose 7-segment will display "1", "2", or "d" depending on the outcome of the game as shown in Table 3.1. When the Throw button is un-pressed, the Win/Lose 7-segment display is blank.

Table 3.1: The output for every combination of player 1 (P1) and player 2 (P2) throws.

P1 \ P2	Rock	Paper	Scissors	Invalid
Rock	d	2	1	1
Paper	1	d	2	1
Scissors	2	1	d	1
Invalid	2	2	2	d

3.3 System Architecture

The system architecture shown in Figure 3.3 is your guide to building a functioning circuit. As such, we will take a moment to cover some important details of this diagram that will help you write your Verilog code later./

The names outside the FPGA square correspond to the labels in Figure 3.1. Each soft-square (a square with rounded corners) is a Verilog module. Names inside soft-squares, that are adjacent to lines outside the soft-square, are the port names for that module. The instance name and module name of a module are separated by a ":" and usually located along the top edge of the soft-square. Red soft-squares are associated with player 1 and green soft-squares associated with player 2. The names on lines inside the rpsGame soft-square are the signals names you should use in the rpsGame module to connect the 5 modules together. Lines that are slashed with a number denote bit vectors.

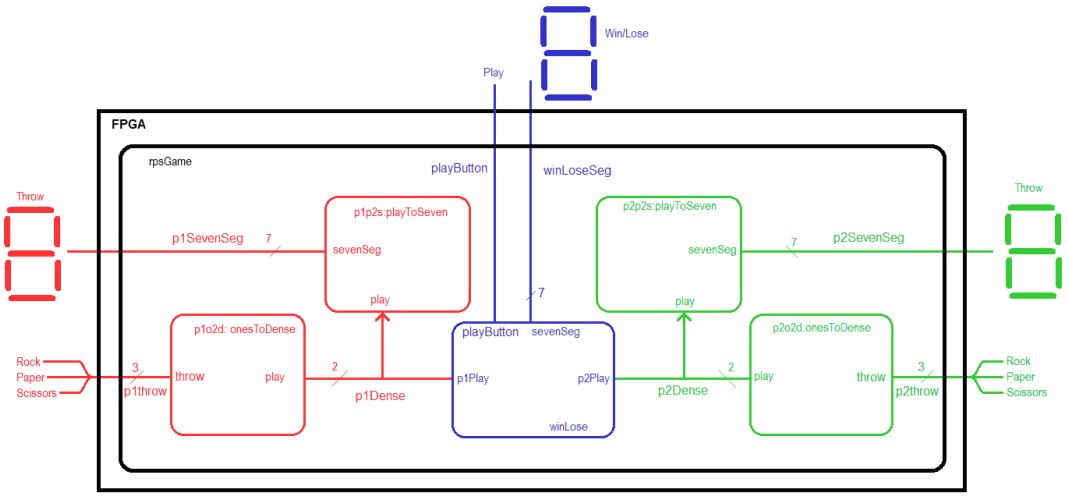


Figure 3.3: System architecture for the rock, paper, scissors system.

3.4 Module: onesToDense

Each player makes their throw selection by placing one of the three slide switches into the up position. As a result of this game mechanic, there are only three valid input combinations for the rock, paper, scissors trio. These are:

- (1,0,0) for when the player moves only the rock slide switch up,
- (0,1,0) for when the player moves only the paper slide switch up,
- (0,0,1) for when the player moves only the scissor slide switch up.

Having a code where only one of the bits is equal logic 1 is called a "ones-hot" code. The "hot" bit being logic 1. A code where every possible combination of bits is assigned a meaning is called a dense code.

This module will convert the input ones-hot code into a dense code. In order to correctly determine the outcome of the game, we need to know when the user has entered an invalid play; the output of this module must be able to represent {rock, paper, scissors, invalid}. You will encode these four combinations in 2-bits as {2'b00, 2b'01, 2'b10, 2'b11} respectively.

In order to write the Verilog code for this module, complete the truth table in Table 3.2 for the onesToDense module.

- r is the state of the rock slide-switch. r=0 slide switch is down. r=1 slide-switch up.
- p is the state of the paper slide-switch. p=0 slide switch is down. p=1 slide-switch up
- s is the state of the scissor slide-switch. s=0 slide switch is down. s=1 slide-switch up
- play = {00} means rock was selected
- play = {01} means paper was selected
- play = {10} means scissor was selected
- play = {11} means invalid selection was made

Table 3.2: The truth table for the onesToDense module.

r	p	s	play	Note
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0	00	Rock
1	0	1		
1	1	0		
1	1	1		

From this truth table, determine the canonical SOP expressions for play[1] and play[0] functions. Do this by writing the canonical SOP expression for the most significant bit of the play output, play[1], in the Table 3.2 truth table while ignoring the LSB. Then proceed to write the canonical SOP expression for the LSB of the play output, play[0], in the Table 3.2 truth table while ignoring the MSB.

play[1] =

play[0] =

Now it's time to write the Verilog code. Incorporate the following into your onesToDense Verilog module:

- Use the module declaration: module onesToDense (throw, play);
- Use vectors for the throw input. The MSB should come from the rock slide-switch and the LSB from the scissors slide-switch.
- Use a vector for the play output.
- Make the input and output port types "wire".
- You may want to break the input vector into its component pieces to correspond to the variable names used in Table 3.2. This will require a wire declaration and two assign statements to give each variable 1-bit from the input vector.

- Use assign statements to realize the AND, OR, NOT logic derived for your canonical SOP expressions.
- Use function04 from lab 1 as a starting point for this module.

3.5 Module: playToSeven

The playToSeven module converts the player throw, represented in the dense coding, to a "graphical" form displayed on the 7-segment display. The symbols for each possible throw are shown in Figure 3.4.

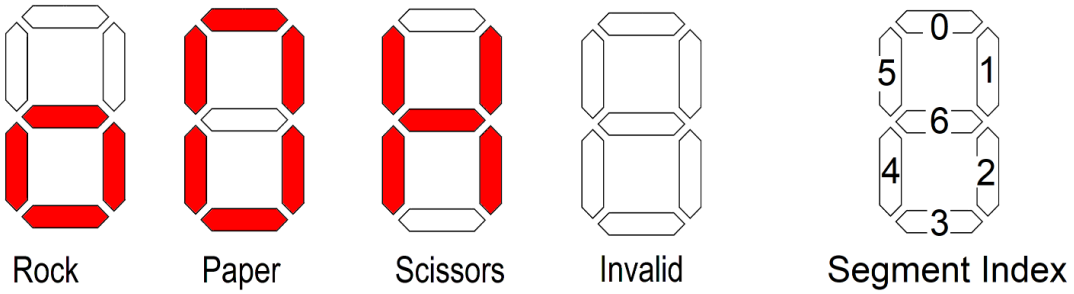


Figure 3.4: 7-segment display patterns for the different throws along with the 7-segment display segment indices.

Use the information in Figure 3.4 to determine the bit patterns needed to generate the four different symbols in Table 3.3. Remember that the LEDs in the segments are active low, meaning that a logic 0 output illuminates an LED segment. In Table 3.3 put a 0 or 1 in each of the numbered column so that each row produces the patterns for its throw. Remember that pPlay codes rock as 00, paper as 01, scissors as 10 and an invalid throw is coded as 11. In the sevenSeg column put the 7-bit code formed by concatenating the bits together. Use proper Verilog syntax to write this 7-bit vector.

Table 3.3: The 7-segment display LEDs to produce the throw patterns.

pPlay	6	5	4	3	2	1	0	sevenSeg	Note
00									Rock
01									Paper
10									Scissors
11								7'b1111111	Invalid

Incorporate the following into your playToSeven Verilog module:

- Use the module declaration: `module playToSeven (pPlay, sevenSeg);`
- Use a vector for the input pPlay and a vector for the sevenSeg output.
- Make the input port type "wire". Make the output port type "reg".
- Use a case statement, embedded in an always statement to realize this module. Enumerate all combinations of the input; do not use a default case
- Use the information in Table 3.3 to assign values to the output.

- Put comments at the end of each case row describing, in words, what the output should look like on the 7-segment display.
- Use the hex2Seven module from lab 2 as the starting point for this module.

3.6 Module: winLose

The winLose module takes the throw from each player (in the dense coding), the push button, and determines what to display on the win/lose 7-segment display. The win/lose 7-segment display is blank when the button is not being pressed, otherwise it will show "1" when player 1 has a winning throw, "2" when player two has the winning throw, "d" when the players have the same throw. The patterns are the same as those you have already created for the hexToSeven module and are shown in Figure 3.5 as a reminder.

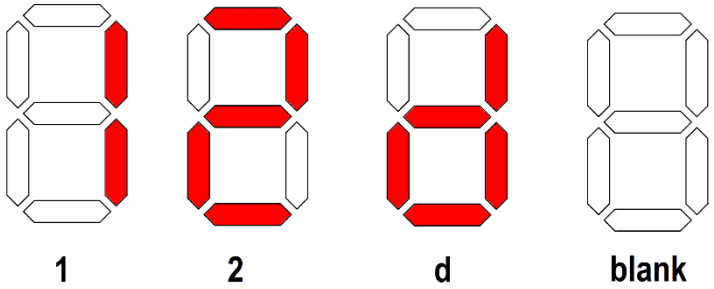


Figure 3.5: The illuminated patterns displayed on the win/lose 7-segment display.

You will use a case statement, based on the information in Table 3.4, to realize this function. In the sevenSeg column put the 7-bit code needed to illuminate the winLose 7-segment display to indicate the outcome of the game. In the Note column put either "1", "2", "Draw", or "Blank" depending on the outcome of the game and button press. Use the bit order given in Figure 3.4.

Table 3.4: Abbreviated truth table for the winLose module.

button	p1Play	p2Play	sevenSeg	Note
0				
0				
0				
0				
0				
0				
0				
0				
0	10 (scissors)	00 (rock)	7'b0100100	2
0				
0				
0				
0				

button	p1Play	p2Play	sevenSeg	Note
0				
0				
0				
1	xx (don't care)	xx (don't care)	7'b1111111	Blank

Incorporate the following into your winLose Verilog module: Use the module and port names given in Figure 3.3.

- Use the module declaration: `module winLose(p1Play, p2Play, playButton, sevenSeg);`
- Use vectors for the p1Play and p2Play inputs.
- Use a vector for the sevenSeg output.
- Make the input port types "wire". Make the output port types "reg".
- You will use a case statement, embedded in an always statement to realize this module.
  - Use brackets to make the vector for the case statement. For example, if button was a 1-bit signal and play was a 2-bit signal, then

```
case ({button, play})
    3'b000:  seg = 7'b1000000;     // display ''0''
    3'b001:  seg = 7'b1111001;     // display ''1''
    3'b010:   seg = 7'b0100100;     // display ''2''
    3'b011:   seg = 7'b0110000;     // display ''3''
    default:  seg = 7'b1111111;     // blank 7-seg
endcase
```

  This code snippet will use the 3-bit value (button as MSB and play as least significant 2-bits) to select one of the rows. Note that the default case handles all the combinations where button is 1.
- Use a default case to handle all the situations where the button is not pressed. The default case catches any unspecified input combinations for the case statement. List the default as the last row in the case list.
- At the end of each "case" row, provide a comment that lists player 1's throw, player 2's throw and the output that is displayed on the 7-segment display. For example, in my program the first case row has a comment that looks like:
  `// P1: Rock P2:Rock Draw`
- Use the hex2Seven module from lab 2 as the starting point for this module.

3.7 Module: rpsGame

The rpsGame module, "glues" together the modules shown in Figure 3.3 and serves as the top-level entity. The Verilog code for this module consists of 5 instantiation statements; one of them is given as the last bullet point item in the list below. For this module, I want you to:

- Use the module declaration:
  `module rpsGame(p1Throw, p1SevenSeg, p2Throw, p2SevenSeg, playButton, winLoseSeg);`
- Make the p1Throw and p2Throw inputs vectors with the MSB coming from the rock slide-switch input and scissors slide-switch as the LSB. You will need to keep this consistent with the pin assignment that you will complete next.
- The `playButton` input is not a vector.
- Use a vector for the `winLoseSeg` output.

- Make the input and output port types "wire".
- You need to create 2 internal vectors. Look carefully at Figure 3.3 and find wires that begin and end inside the rpsGame module. These are the vectors.
- Name the module instances using the names provided in Figure 3.3.
- When you instantiate a module
  - The first term is the name of the module you are instantiating
  - The second term is the instance name of the module
  - The remaining term is the parenthesis list of signal in and out of the module. The order of the signals in the instantiation must be the same as those in the module declaration. Pay special attention to this!
  - For example, in my program I had an instantiation that looked like:
    `onesToDense p1o2d(p1Throw, p1Dense);`

3.8 Testbench

We are forgoing a simulation of today's lab. This requires that you pay especially close attention to the warnings created by the Quartus software. If you are having issues with your design, go through the compilation report and look for unconnected inputs or vector size mismatches.

3.9 Pin-Assignment and Synthesis

Use the image of the FPGA Development Board in Figure 3.1 and the information in the C5G User Guide to determine the FPGA pins associated with the input and output devices used by the rpsGame module.

Table 3.5: Pin assignment tables for the Rock Paper Scissor game.

Segment	Player 1 Throw	Player 2 Throw	Win/Lose
seg[6]	PIN_Y18		
seg[5]		PIN_AC23	
seg[4]			
seg[3]			
seg[2]			
seg[1]			
seg[0]			PIN_AA18

	Player 1 Slide Switch	Player 2 Slide Switch
slide[2]		
slide[1]		
slide[0]	PIN_AC9	

Play Button	Key[0]	

Note, each push-button provides a high logic level when it is not pressed, and provides a low logic level when pressed.

Complete the pin-assignment in Quartus, compile your design and download to the FGPA development boards. Once you get your design working, demonstrate it to a member of the lab team.

3.10 Turn in

You may work in team of at most two. Make a record of your response to the items below and turn them in a single copy as your team's solution on Canvas using the instructions posted there. Include the names of both team members at the top of your solutions. Use complete English sentences to introduce what each of the following listed items (below) is and how it was derived. In addition to this submission, you will be expected to demonstrate your circuit at the beginning of your lab section next week.

onesToDense Module

- Complete Table 3.2 truth table for oneToDense module.
- link Canonical SOP expressions for the play[1] and play[0] functions.
- link Verilog code for the entire. module (courier 8-point font single spaced), leave out header comments.

playToSeven Module

- Complete Table 3.3.
- link Verilog code for the module (courier 8-point font single spaced), leave out header comments.

winLose Module

- Complete Table 3.4 truth table for winLose module.
- link Verilog code for the module (courier 8-point font single spaced), leave out header comments.

rpsGame Module

- link Verilog code for the module (courier 8-point font single spaced), leave out header comments.

Pin Assignment

- Completed pin assignment for 7-segment, slide switches and button in Table 3.5.
- Demonstrate your working design to a lab team member.