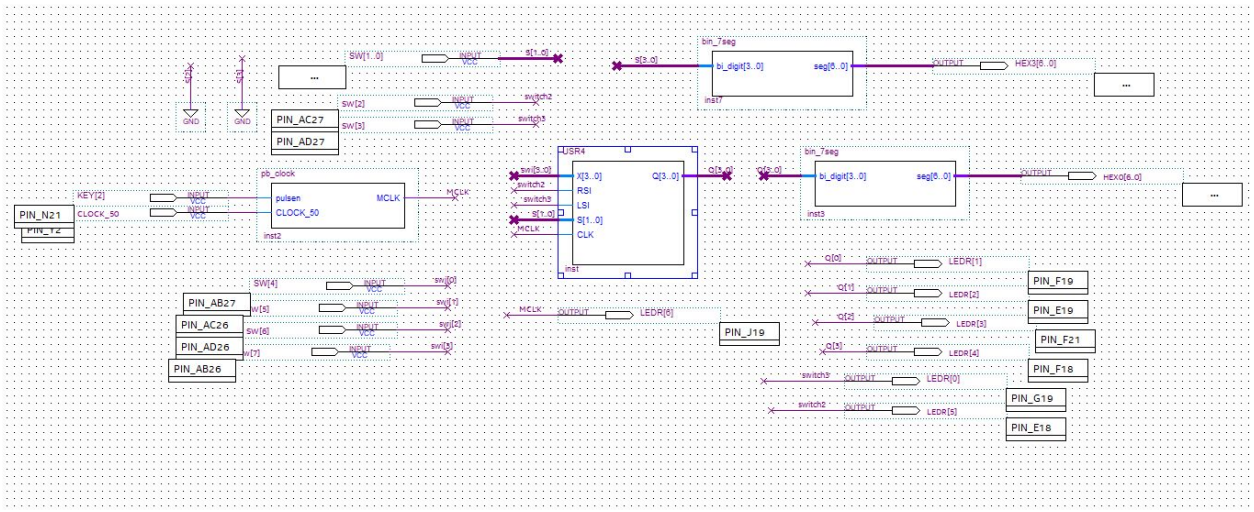
Liam Connole
Dillon Kanai
COEN 21 Lab
November 21, 2019

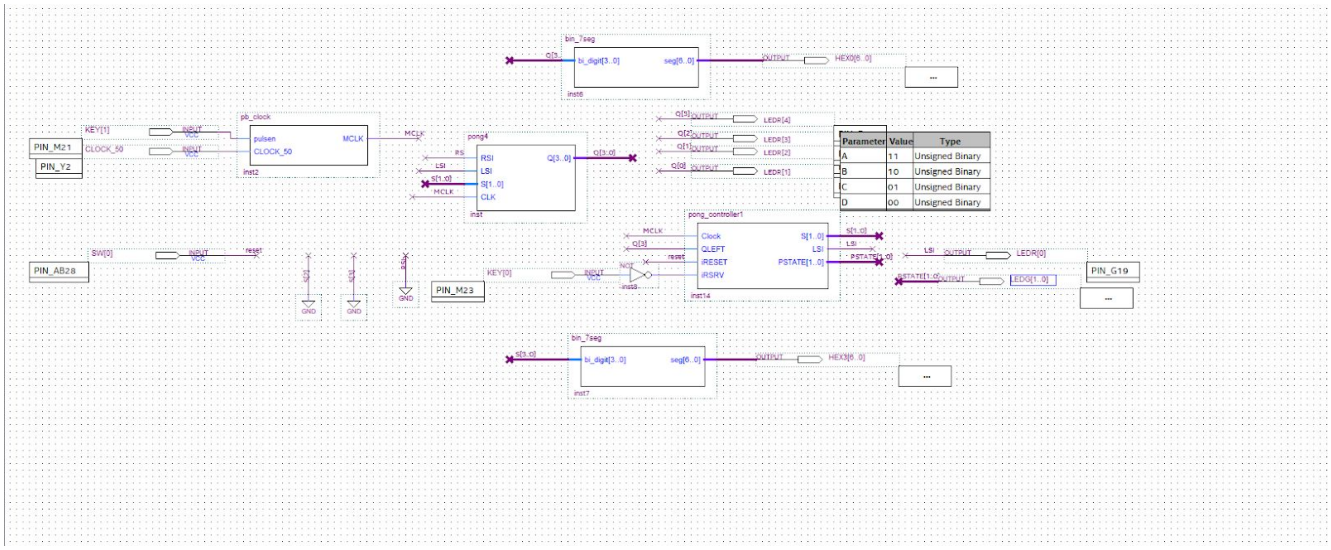
Lab Report 8

Schematics, Test Plan, Verilog, and Results

Schematic for Part 1:



Schematic for Part 2:



Verilog for module USR4:

```
module USR4(X, RSI, LSI, S, CLK, Q);
input RSI, LSI, CLK;
input [3:0] X;
input [1:0] S;
output reg [3:0] Q;
reg [3:0] FlopInput;

always @(posedge CLK)
Q = FlopInput;

always @(*)
begin
case(S)
2'b00: FlopInput = Q;
2'b01: FlopInput= {RSI, Q[3:1]};
2'b10: FlopInput = {Q[2:0], LSI};
2'b11: FlopInput = X;
endcase
end

endmodule
```

Verilog for module pong4:

```
module pong4 (RSI, LSI, S, CLK, Q);
input RSI, LSI, CLK;
input [1:0] S;
output reg [3:0] Q;
reg [3:0] FlopInput;

always @(posedge CLK)
Q = FlopInput;

always @(*)
begin
case(S)
2'b00: FlopInput = Q;
2'b01: FlopInput= {RSI, Q[3:1]};
2'b10: FlopInput = {Q[2:0], LSI};
2'b11: FlopInput = 4'b0000;
endcase
end

endmodule

```

Verilog for module pong_controller1:

```
module pong_controller1(Clock, QLEFT, iRESET, iRSRV, S, LSI, PSTATE);
input Clock, QLEFT, iRESET, iRSRV;
output reg [1:0] S;
output reg LSI;
parameter [1:0] A = 2'b11, B = 2'b10, C = 2'b01, D = 2'b00;
reg [1:0] NSTATE;
output reg[1:0] PSTATE;

always @ |(*)
case(PSTATE)
A:
begin
LSI = 1'b0;
S = 2'b11;
if(iRSRV)
NSTATE = B;
else
NSTATE = A;
end
B:
begin
LSI = 1'b1;
S = 2'b10;
if(iRESET)
NSTATE = A;
else
NSTATE = C;
end
C:
begin
LSI = 1'b0;
S = 2'b10;
 if(iRESET)
NSTATE = A;
else if(QLEFT)
NSTATE = D;
else
NSTATE = C;
end
D:
begin
LSI = 1'b0;
S = 2'b00;
NSTATE = A;
end
endcase


always @(posedge Clock)
PSTATE <= NSTATE;

```

Test Plan:

1. First, we will test the hold operation for the register, As the clock oscillates between 0 and 1, S will be kept at 00 in order to hold the values. In this instance, LSI and RSI do not matter since we do not need values to shift into the register. The hold operation will prove functional if the initial value in the registers does not change throughout the waveform simulation.
2. Second, we will test the shift right operation. To do so, we will keep S at 01 while RSI will change between 0 and 1 at every rising edge of the clock. The right shift operation will work correctly if it successfully shifts the values present in the register to the right continually, while also shifting the correct serial input (RSI) at each rising edge of the clock.
3. Third, we will test the left shift operation. This process is very similar to the testing of the right shift operation, though we will now keep S at 10 and have LSI change between 0 and 1 at every rising edge of the clock. Proving the functionality of this operation will be the same, we simply need to see if the LSI is correct at every shift, and that the values are correctly being shifted at each rising edge of the clock.
4. Fourth, we will test external load operation. To test this operation, we will simply need to keep S at 11, and look at each index of Q. At the rising edge of the clock, if each index of Q matches the values at each index of X, the load input, then the operation works as it should.

Observed results:

During the creation of our schematic and implementation of the lab using the FPGA, all of our results were as expected. For part 1, we had some difficulty with assigning the pins to the proper inputs and outputs in our schematic as well as some syntax errors, but our execution of part 1 was successful otherwise. The outputs of the shift register, represented by the red LEDs, move according to the combination in which the S switches were flipped on or off. Also, the parallel data in the register could be specified by switches and replaced the outputs of the register when both S switches were flipped on.

For part 2, we had slightly more difficulty than part 1. We had several verilog errors that hindered our progress, but were easily fixed and applied to our schematic. Additionally, we had an issue with push button pin assignments, since there was a gap between the two push buttons that were being used, resulting in the FPGA not working correctly. When our FPGA finally functioned as intended, we used one of our push buttons to indicate a serve, and another button could hold the value while depressed. We had 2 green LEDs to specify the current state, and these lights changed as the asserted input was shifted left in order to indicate the events that were occurring.

Questions:

2. Describe the most challenging part of this lab.

The most difficult part of the lab was the pin assignments. There were many variables to take into account, especially in part 1. It took significantly longer than previous labs to put all the assignments in. Additionally, writing the verilog file for the state machine proved to be quite challenging in terms of making sure everything was syntactically flawless.

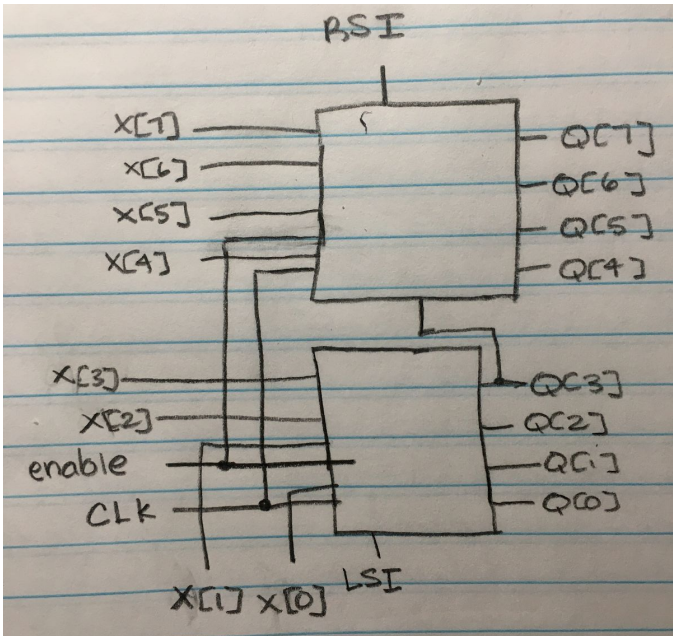
3. **For lab part 1, if you loaded the register with 1101, what value would that represent if you interpreted it as an unsigned integer? With RSI=0, if you shifted to the right what binary pattern would you see and how would it be interpreted as an unsigned number? With RSI = 0, if you shifted to the right a second time what binary pattern would you see and how would it be interpreted as an unsigned number? Explain how right shifting is related to dividing by two.**

If we loaded the register with 1101, it would represent the unsigned number 13. If RSI equals 0, it would replace Q[3] with 0. Q[0] will lose its current value and inherit the value from Q[1]. The final value is 0110, which is 6. If we shifted a second time where RSI = 0, we will get 0011, which is 3. This is related to dividing by two. 13 divided by 2 is 6.5. The actual value received from the register is 6. This implies that the register rounds down if the number received isn't a whole number. The same is demonstrated by the next shift. 6 divided by 2 is 3. Since three is a whole number, there is no need to round down.

4. In the previous question, for unsigned interpretation of the values, RSI was set to zero. To divide by 2 using right shift for signed integer interpretation using 2's complement representation, what should RSI be? Why?
It depends, if we are dividing positive numbers, RSI needs to be 0. If we are dividing negative numbers, RSI needs to be 1. Since negative numbers are the inverse of their positive counterparts, a 1 is a 0 in positive terms. So, by adding in a 1, we are dividing by two just like we did with positive numbers.

5. How would two 4 bit universal shift registers be connected to form an 8-bit universal shift register? Show a schematic in which each 4-bit register is shown as a block component with inputs and outputs. Do not show the internal components and connection of the 4-bit universal shift register.

All we need to do to connect two 4 bit USRs is to make sure that the value Q3 value is transferred over to the next block, or the Q4 value is transferred over the previous block. If there is an enable switch, we also need to make sure that value also transfers over to the next block. Furthermore, we need to make sure a clock signal is connected to both blocks, and left shift value is placed at the beginning of the first shift register block, and the right shift value input is placed at the end of the second shift register block.



6. For lab part 2, if the controller is in the idle state and the right server input is high during an active clock edge, what is the next state? For the next 6 active clock edges, specify the state that the controller is in after each clock edge.

The next state will be the sRSRV state. Next, sMOVL for the rest of the 6 active clock edges because sRSRV has no effect on which direction we go, assuming all the other inputs are 0.

7. If by mistake, QLEFT were connected to Q[0] instead of Q[3], how would your answer to the previous question change?

QLEFT asserts when the ball is in the leftmost position, just before it shifts out of the register, meaning that the assertion of this signal would indicate that the game has ended. If QLEFT were connected to Q[3], the state would transition to sENDL immediately and the game would end directly after the controller enters the sMOVL state. The answer to the previous question would change in the sense that the controller would still move to the sRSRV state while the right server input is high and then to sMOVL, but would transition to sENDL on the clock edge after that. Then, the state would go back to sIDLE and would repeat the previous cycle for the remaining clock edges because the right server input is high.

8. Show how you would modify your state transition diagram to allow both players to serve. How many additional states would you need? What additional inputs and outputs would be needed?

We would need twice as many states as we do now. Since each player has the same states as each other, there would be no reason to have any more or any less. Likewise, we need twice as many inputs/outputs. One set for the left player reacting to the state of the ball and one set for the right player. We would have to edit how sENDL behaves however. The pong ball simply doesn't stop just because it reaches the leftmost position. So, we need to add a couple variables there. Perhaps an extra variable to see whether the right player returns the ball or misses.