CSE 370 Homework 7 Solutions

1. Circuit Timing

The timing diagram for this negative edge detection circuit is shown in Figure 1 below. The slack for this circuit is calculated as follows:

$$T_{cycle} \geq t_{pd\_register} + t_{pd\_logic} + t_{setup}$$

$$t_{slack} = T_{cycle} - t_{pd\_register} - t_{pd\_logic} - t_{setup}$$

where T_{cycle} is the clock period, $t_{pd\_register}$ is the propagation delay through the register, t_{pd_logic} is the propagation delay through the logic, and t_{setup} is the setup time for the flip-flop.

There are 2 long paths in this circuit:
a) register to register: slack = 8 – 2 – 1 – 2 = 3ns
b) input to register: slack = 8 – 3 – 2 – 2 = 1ns
Here, b is the critical path and the slack of the circuit is 1ns.

Thus the setup time of the second flip-flop is violated if more than 1ns of delay is added to the path from input to the second register. In the figure below, the setup/hold window is shown in gray.

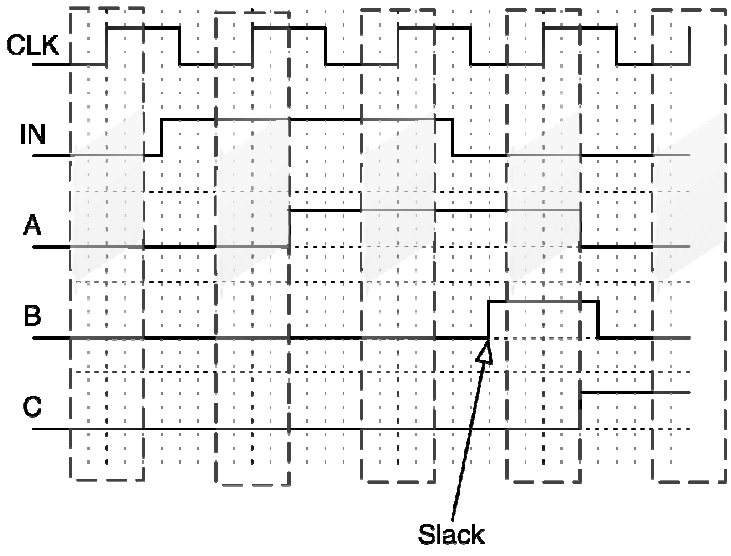

Figure 1: Timing diagram for original negative edge detection circuit.

2. Circuit Timing with Clock Skew = 4ns

The figure below shows the timing diagram with the clock to the right register delayed in time by 4ns. Since B is the input to the 2nd register, it must meet the timing constraints for the clock of that register. The gray boxes show the setup/hold window around clock 2. This shows that both the setup and hold times are violated. It can be seen that this register samples 0 instead of 1 so the output does not change here.

The hold time violation is caused by the change in input 3ns after clk1. Since the clk2 edge for the hold violation is the same edge just delayed by 4ns, there is nothing we can do to avoid this hold violation. For example, slowing down the clock would not help since it all depends on one edge, not two adjacent edges. Therefore, you cannot get this circuit to work.

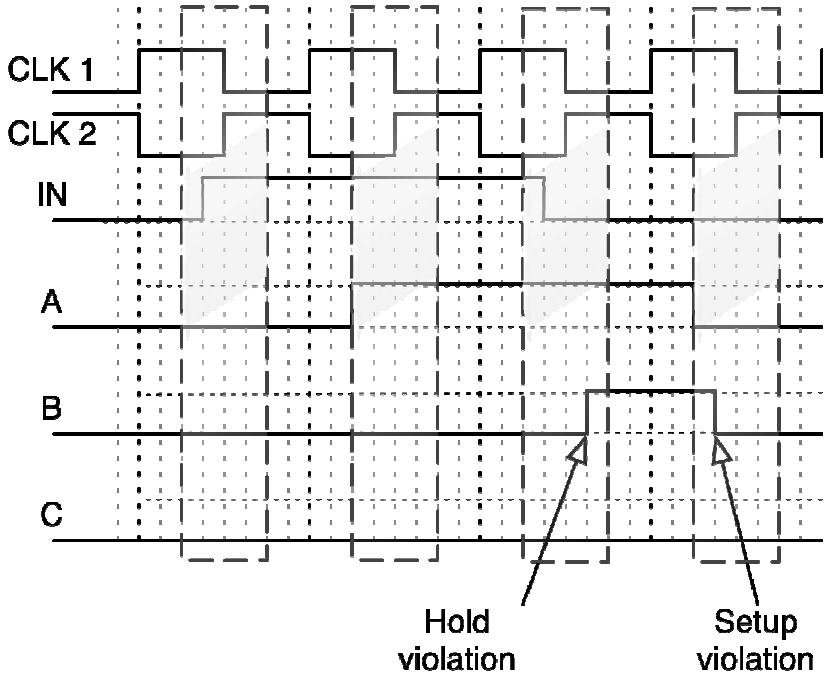

Figure 2: Timing diagram of circuit with 4ns clock skew. Output C is always 0 due to skew.

3. Circuit Timing with Clock Skew = 2ns

Now with a clock skew of 2ns, the output flip-flop attempts to capture the data 2ns earlier than in part2, but the data changes 1ns after the clock edge of the second flip-flop. Therefore the hold time constraint is violated. If we ignore the timing constraint, however, the output will go to 1 as shown.

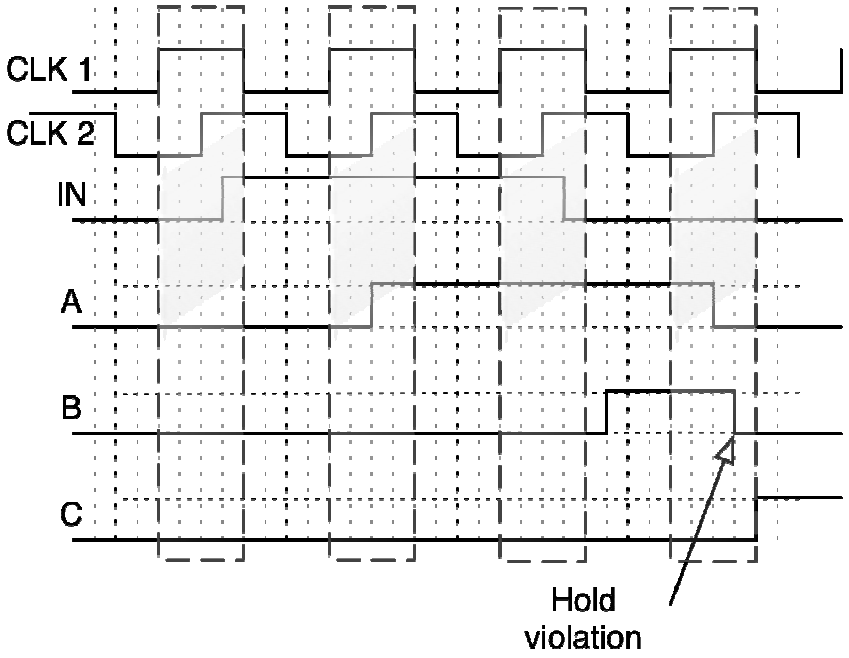

Figure 3: Circuit with 2ns clock skew. Output should go unknown due to hold time violation, but this shows the output if we ignore the hold time constraint.

4. Circuit Timing with Clock Skew = 4ns Reversed

A positive clock skew of 4ns is the same as a negative clock skew of 4ns. If you assume that the input doesn't change, then this is the same as question 2. If you assume the input is fixed relative to CLK, then you get the following timing diagram. The timing analysis of this circuit depends on the timing of the input. If we assume the input is fixed relative to CLK, then it is not possible to fix this circuit. If we can delay the input, and increase the clock period, then we can get this circuit to work.

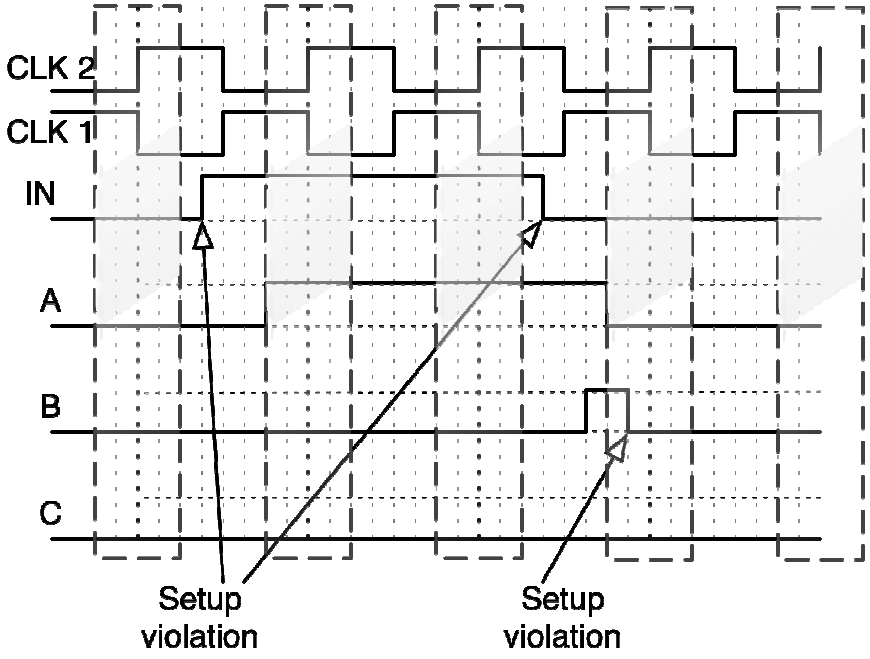

Figure 4: Circuit with 4ns clock skew in the reverse direction. Output goes unknown due to setup time violation on output flip-flop, but this shows the output if we ignore the setup time constraint of the second flip-flop.

5. Arbiter FSM

With the specified timing diagram, you have to use a Moore FSM. You can only use a Mealy if you pass the outputs through a register.

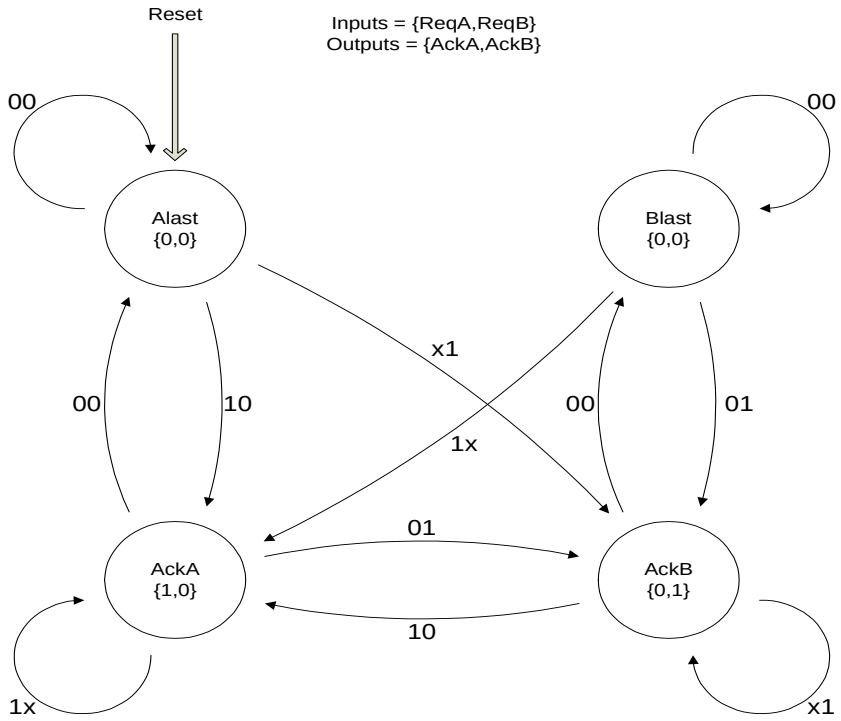

Figure 5: An FSM diagram showing the Moore type implementation of a 2 user arbiter.

State Transition Table

		State Encoding	Alast	Blast	AckA	AckB	
			00	01	10	11	
S[1]	S[0]	ReqA	ReqB	NS[1]	NS[0]	AckA	AckB
0	0	0	0	0	0	0	0
0	0	x	1	1	1	0	0
0	0	1	0	1	0	0	0
0	1	0	0	0	1	0	0
0	1	0	1	1	1	0	0
0	1	1	x	1	0	0	0
1	0	0	0	0	0	1	0
1	0	0	1	1	1	1	0
1	0	1	x	1	0	1	0
1	1	0	0	0	1	0	1
1	1	x	1	1	1	0	1
1	1	1	0	1	0	0	1

Table 1: State Transition Table for the 2 user arbiter.

Next State and Output Equations

$$NS[1] = \mathrm{Re}\,qA \mid \mathrm{Re}\,qB$$
$$NS[0] = (\overline{\mathrm{Re}\,qA} \cdot \mathrm{Re}\,qB) \mid (S[0] \cdot \overline{\mathrm{Re}\,qA}) \mid (S[1] \cdot S[0] \cdot \mathrm{Re}\,qB) \mid (\overline{S[1]} \cdot \overline{S[0]} \cdot \mathrm{Re}\,qB)$$
$$AckA = S[1] \cdot \overline{S[0]}$$
$$AckB = S[1] \cdot S[0]$$

Circuit Implementation

Note: The boxes in the following circuit produce the functions for next state and the outputs that were specified in the previous section.

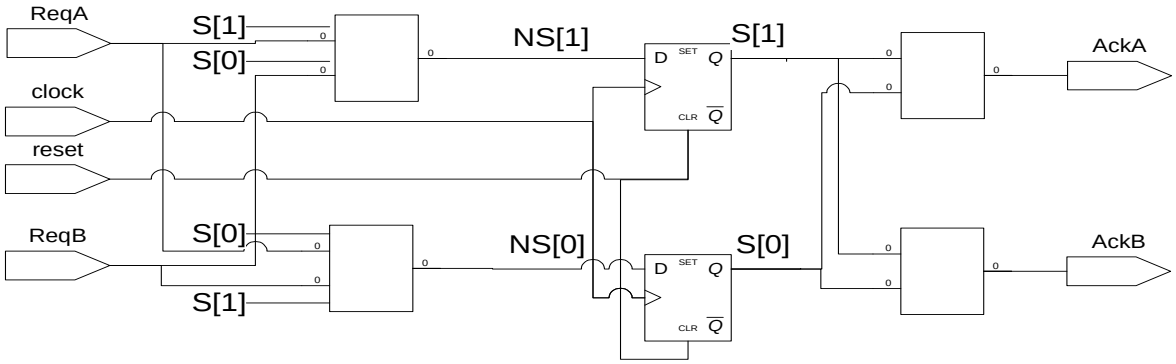

Figure 6: Circuit implementation of the 2 user arbiter.

6. Program Counter and Squares Calculation

Efficient Program to Compute the Squares:

```
// Simple program that generates squares
// R0=result, R1=increment, R2=dx (d/dx(x^2) = 2)
10110_000_00_000_000   // R0 = 0
10010_001_00_000_000   // R1 = R0 + 1      ...R1 = 1
10010_010_00_001_000   // R2 = R1 + 1      ...R2 = 2
10000_000_00_000_001   // R0 = R0 + R1     ...R0 = 1
10000_001_00_001_010   // R1 = R1 + R2
10000_000_00_000_001   // R0 = R0 + R1     ...R0 = 4
10000_001_00_001_010   // R1 = R1 + R2
10000_000_00_000_001   // R0 = R0 + R1     ...R0 = 9
10000_001_00_001_010   // R1 = R1 + R2
10000_000_00_000_001   // R0 = R0 + R1     ...R0 = 16
10000_001_00_001_010   // R1 = R1 + R2
10000_000_00_000_001   // R0 = R0 + R1     ...R0 = 25
10000_001_00_001_010   // R1 = R1 + R2
10000_000_00_000_001   // R0 = R0 + R1     ...R0 = 36
```

Simulation Output: