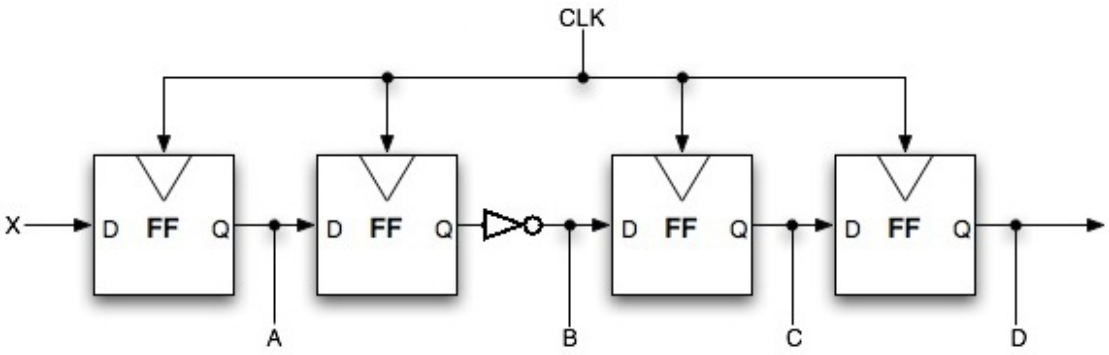
Q1 Waveform Diagrams

4 Points

REMINDER: Gradescope does not have submission history for in-browser assignments such as lecture questions and homeworks. If you press "save answer" or "submit" after the deadline, the autograder *will* mark it as late and you'll get 0 points in the next TCP rerun. When you review old homework and lecture assignments, do not click any buttons otherwise you risk getting a zero on that assignment.



Consider the chain of flip-flops (FF) shown here. Assume that input X alternates between 0 and 1, 10ns after every rising edge. Initially, X is **0** (so 10ns after the first rising edge it should be 1), and all Flip-Flops are stable and set to **1**. Assume one clock cycle is **40 ns**. Given the clock signal, draw the waveform for input X, and the signals at points A, B, C, and D in the circuit for the first 6 clock cycles. Assume that the clk-to-Q delay is **5 ns**, the setup time is **0 ns**, and the hold time is **5ns**. Assume that flip-flops are *positive-edge-triggered* : they take their new value on the rising edge of the clock. Assume the first rising edge occurs at time = 0. Assume all other unstated propagation delays (i.e., through wires and logic) are zero.

You should fill out the waveform diagram below to help you answer the following questions. However, you only have to submit the answers to the questions, not the whole diagram. Consider six clock cycles (so six rising edges) as shown in the diagram. Assume the diagram is cut off 10ns after the last rising edge. You only need to consider from t = [0, 210] ns for this problem.

Q1.1

1 Point

How how many times does the value at B change?

5

Q1.2

1 Point

How many times does the value at D change?

4

Q1.3

1 Point

At which time(s) does the value at A becomes stable at 0?

Format your answer as a sorted comma-separated list of times with **no** extra whitespace. (e.g., "3,4,5").

5,85,165

Q1.4
1 Point

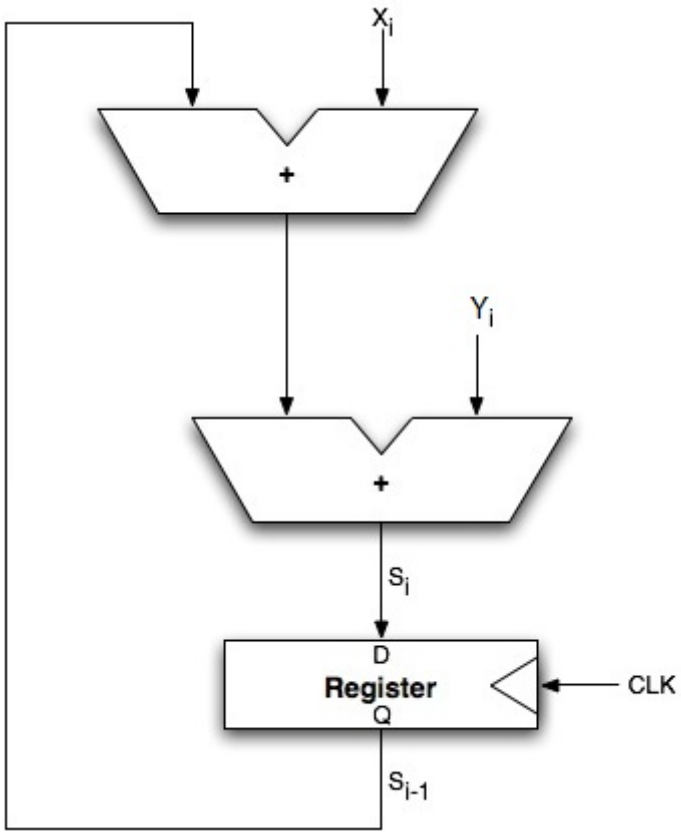
At which time(s) does the value at D becomes stable at 0?

Format your answer as a sorted comma-separated list of times with **no** extra whitespace. (e.g., "3,4,5").

45,165

Q2 Clock Frequency
8 Points

Consider this circuit. It accumulates two arguments at a time, arriving at each rising edge of the clock. You are given the following: the adder propagation delay is 2 ns, the register setup time is 2 ns, the register hold time is 4 ns, the register clk-to-q delay is 3 ns, and the clock frequency is 100 MHz.



As a reminder from lecture:

- **hold time** is the amount of time ***after*** the rising edge of the clock that the input into a register needs to remain stable (i.e. the same) in order for the input to be successfully saved in the register.
- **setup time** is the amount of time ***before*** the next rising edge of the clock that the input into a register needs to remain stable in order for the input to be successfully copied to the output of the register.
- **clk-q-delay** is the amount of time ***after*** the rising edge of the clock that it takes for the new/updated output of the register to show up on the Q terminal (i.e. the output of the register). You can think of this as the logic delay for a register to output its result (just like the propagation delay for an AND gate to output its result).
  For any circuit, the following must be met:

1. $t_{hold} \leq t_{shortest-path}$
2. $t_{clk-2-q} + t_{longest-combinational-path} + t_{setup} \leq t_{clk\_cycle}$

Conceptually, **(1)** means that once a clock cycle starts and a register begins saving its current input into its contents, the input into a register cannot change again (or else the register won't know what it should be updating its contents to!), and the fastest way the register input could change is through the shortest path. **(2)** means that your clock period needs to be long enough for your new

register input to be calculated correctly, which means that you need enough time for the longest path to finish and then to setup the register for updating.

Q2.1

4 Points

For how many nanoseconds must S_i stay stable after the clock trigger?

4

After the clock trigger, after how many nanoseconds will S_i change due to Y_i (ignore X_i)?

2

Is the hold time requirement of the register met?

- ○ Yes
- ◉ No

Would this accumulator work properly?

- ○ Yes
- ◉ No

Q2.2

2 Points

Now assume that the arguments arrive **2 ns *after*** each clock trigger.

Give the max delay, also known as critical path delay.

6

Give the maximum clock frequency at which the circuit will work (in MHz).

Hint: $\text{Frequency} = \frac{1}{\text{Period}}$

Round your decimal to the nearest tenth.

Q2.3
2 Points

Now we want to rearrange the components of this accumulator.

For each of the following suggestions, state how it will affect the max clock frequency.

Swap X_i with Y_i

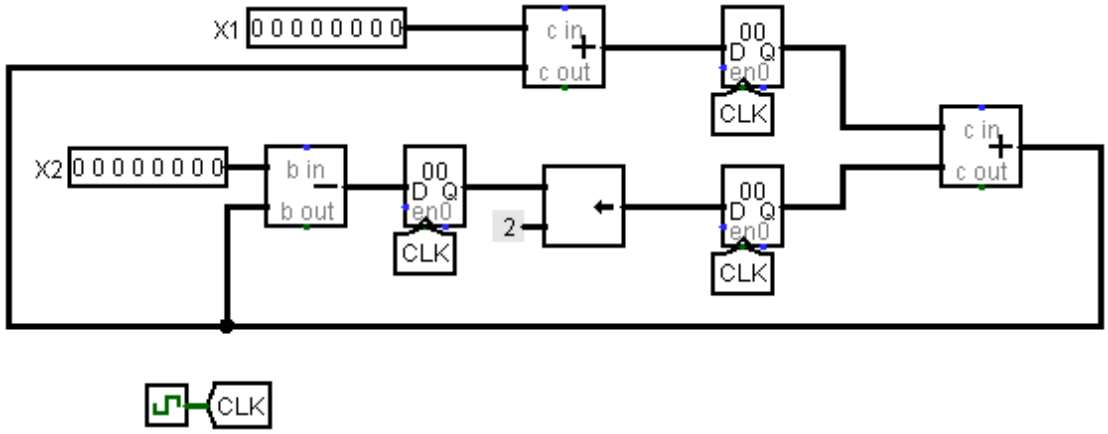
- ○ Increases max clock frequency
- ◉ Doesn't change max clock frequency
- ○ Decreases max clock frequency

Add X_i and Y_i first, then add the result of that with the output of the register.

- ◉ Increases max clock frequency
- ○ Doesn't change max clock frequency
- ○ Decreases max clock frequency

Q3 Circuits and Delays
5 Points

Consider the circuit shown below. All registers have a clock-to-Q delay of 2ns and a setup time of 3ns. The delay for the addition block is 2ns, the delay for the subtract block is 4ns, and the delay for the shift block is 1ns. The inputs X1 and X2 come from registers that have the same specifications as previously described. Make sure you know why the formulas work and what happens if you don't obey the formulas (given in hints).



Q3.1

1 Point

What is the shortest clock period that this circuit can have? Your answer should be in nanoseconds. (Hint: Remember, shortest CLK = CLK-to-Q + Setup + Longest CL.)

11

Q3.2

2 Points

What is the maximum hold time that the registers can have for the circuit to still function properly? Your answer should be in nanoseconds. (Hint: Remember, max Hold = CLK-to-Q + shortest CL.)

3

Would the answer change if the inputs (X1 and X2) arrived right at the rising edge of the clock and are stable for the remainder of each clock cycle? (i.e. X1 and X2 are no longer in registers)

- (●) Yes
- ( ) No

Q3.3
2 Points

Consider the setup presented in the previous problem, where X1 and X2 are no longer in registers. What is the maximum hold time the registers can have now? Your answer should be in nanoseconds.

2

Assuming we operating at the shortest clock period described in problem 3.1, which of the following changes could result in incorrect outputs due to **timing alone**? i.e. would cause a setup or hold time violation

- [ ] Increasing the clock period
- [x] Decreasing the clock period
- [ ] Adding an addition block directly after the shift block
- [x] Adding an shift block directly after the rightmost adder

Q4 Boolean Algebra
5 Points

Simplify each expression by algebraic manipulation:

(Use "`~`" for `NOT`, "`*`" for `AND`, and "`+`" for `OR`)

Q4.1
1 Point

`~(x * ~x)`

1

Q4.2

1 Point

`~(x*~x + ~x*~x) + x`

x

Q4.3

1 Point

a*(a + b + c + ...)

a

Q4.4

1 Point

`(~a + ~b)(a + ~b)`

~b

Q4.5

1 Point

Which of the following simplification is wrong?

- [ ] `a + b + ab = a + b`
- [x] `(x + z)(~x + y)(z + ~y) = (x + z)(~x + y)`
- [ ] `~x + ~y + ~x * y * ~z = ~x + ~y`
- [ ] `~w * ~(wxyz) = ~w`
- [ ] None of the above

Q5 Truth Table

5 Points

We'd like to design an FSM that would take an infinite stream of bits and output 1 **twice** if it sees two consecutive 0's. In other words, given some input X_i, if $X_{i-2} = X_{i-1} = 0$ or $X_{i-1} = X_i = 0$, then it will output 1. Otherwise, it will output 0.

To design FSMs, start by drawing a state diagram (example). Then convert it into a truth table mapping each state and input to a next state and an output. It's important to name the states meaningfully so that it is easily understandable (for example, Seen0 and Seen00).

We've started a truth table for a three-state FSM that works as described above.

CurState	Input	NextState	Output
Start	0	Seen0	**(a)**
Start	1	**(b)**	0
Seen0	0	Seen00	**(c)**
Seen0	1	Start	0
Seen00	0	**(d)**	1
Seen00	1	**(e)**	1

Answer the questions below to fill in the remaining blanks.

(a)

0

(b)

Start

(c)

1

(d)

Seen00

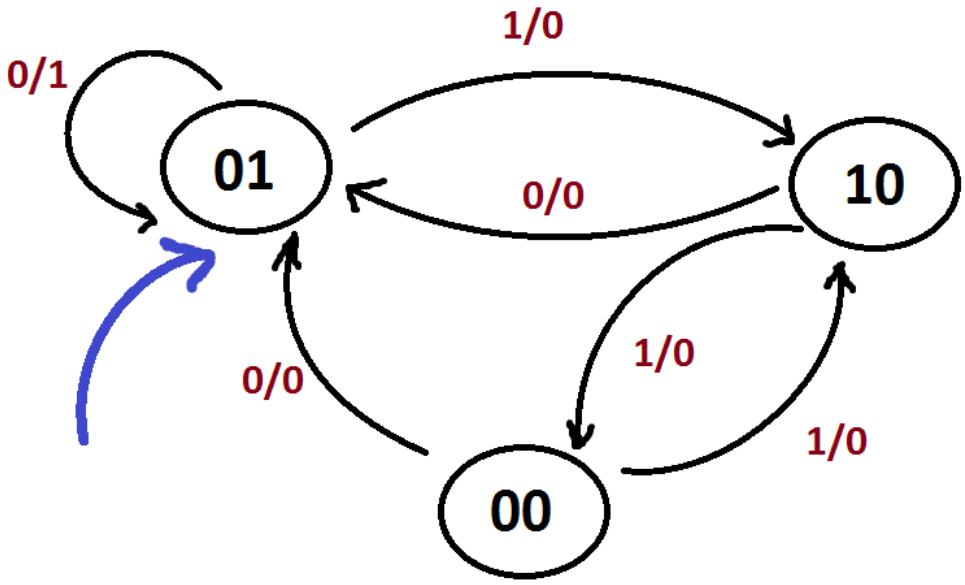
(e)

Start

Q6 Finite State Machine

6 Points

Consider the following finite state machine.



Q6.1

3 Points

Come up with the MOST simplified boolean expressions for determining bits for the next state and the output bit given the current state and the input bit.

We'll label the input bit as `In`, and the left bits as `Cur_1` and `Next_1` for start state and next state left bits, respectively, and do the same for right bits `Cur_0` and `Next_0`.

Format your answer using C syntax for bit operations.

```
Out =_________
```

- [ ] `0`
- [ ] `1`
- [ ] `In`
- [ ] `~In`
- [ ] `Cur_0`
- [ ] `~Cur_0`
- [ ] `Cur_1`
- [ ] `~Cur_1`
- [ ] `In&Cur_0`
- [ ] `In&Cur_1`
- [x] `~In&Cur_0`
- [ ] `~In&Cur_1`
- [ ] `In&~Cur_0`
- [ ] `In&~Cur_1`
- [ ] `~In&~Cur_0`
- [ ] `~In&~Cur_1`

`Next_1 =_____`

- [ ] `0`
- [ ] `1`
- [ ] `In`
- [ ] `~In`
- [ ] `Cur_0`
- [ ] `~Cur_0`
- [ ] `Cur_1`
- [ ] `~Cur_1`
- [ ] `In&Cur_0`
- [ ] `In&Cur_1`
- [ ] `~In&Cur_0`
- [ ] `~In&Cur_1`
- [ ] `In&~Cur_0`
- [x] `In&~Cur_1`
- [ ] `~In&~Cur_0`
- [ ] `~In&~Cur_1`

`Next_0 =_____`

- [ ] `0`
- [ ] `1`
- [ ] `In`
- [x] `~In`
- [ ] `Cur_0`
- [ ] `~Cur_0`
- [ ] `Cur_1`
- [ ] `~Cur_1`
- [ ] `In&Cur_0`
- [ ] `In&Cur_1`
- [ ] `~In&Cur_0`
- [ ] `~In&Cur_1`
- [ ] `In&~Cur_0`
- [ ] `In&~Cur_1`
- [ ] `~In&~Cur_0`
- [ ] `~In&~Cur_1`

Q6.2

3 Points

`fsmCompute` takes one bit at a time as input. Fill in the blanks below so that it behaves as according to the FSM above. Hint: your expressions from the

previous should come in handy, along with some bitwise operators. Also, note how the state is a static variable, so it is maintained across function calls.

Hint: `x` is a signed int, so be careful, `~0 == -1 != 1`. Also, you cannot use the logical operators, only bitwise operators.

```
/* Called once per "clock cycle."
Assume input x is 0 or 1.
Updates state and outputs FSM output (0 or 1). */

int fsmCompute(int x) {
    int output;
    static unsigned int curr_state = 0x1;
    static unsigned int next_state = 0x1;
    curr_state = ____________________________;
    output = ____________________________;
    next_state = ____________________________;
    return output;
}
```

`curr_state =_____`

- [ ] `0`
- [ ] `1`
- [x] `next_state`
- [ ] `~next_state`
- [ ] `curr_state`
- [ ] `~curr_state`
- [ ] `x`
- [ ] `~x`
- [ ] `((x)&curr_state)`
- [ ] `((x)&curr_state&1)`
- [ ] `((~x)&curr_state)`
- [ ] `((~x)&curr_state&1)`
- [ ] (((x&~(curr_state >> 1)) << 1) | ((~x)))
- [ ] ((((~(curr_state >> 1)) << 1) | ((~x)&1))
- [ ] (((x&~(curr_state >> 1)) << 1) | ((~x)&1))
- [ ] (((x&~(curr_state << 1)) >> 1) | ((~x)&1))

`output =_____`

- [ ] `0`
- [ ] `1`
- [ ] `next_state`
- [ ] `~next_state`
- [ ] `curr_state`
- [ ] `~curr_state`
- [ ] `x`
- [ ] `~x`
- [ ] `((x)&curr_state)`
- [ ] `((x)&curr_state&1)`
- [ ] `((~x)&curr_state)`
- [x] `((~x)&curr_state&1)`
- [ ] `(((x&~(curr_state >> 1)) << 1) | ((~x)))`
- [ ] `(((~(curr_state >> 1)) << 1) | ((~x)&1))`
- [ ] `(((x&~(curr_state >> 1)) << 1) | ((~x)&1))`
- [ ] `(((x&~(curr_state << 1)) >> 1) | ((~x)&1))`

`next_state =_____`

- [ ] `0`
- [ ] `1`
- [ ] `next_state`
- [ ] `~next_state`
- [ ] `curr_state`
- [ ] `~curr_state`
- [ ] `x`
- [ ] `~x`
- [ ] `((x)&curr_state)`
- [ ] `((x)&curr_state&1)`
- [ ] `((~x)&curr_state)`
- [ ] `((~x)&curr_state&1)`
- [ ] `(((x&~(curr_state >> 1)) << 1) | ((~x)))`
- [ ] `(((~(curr_state >> 1)) << 1) | ((~x)&1))`
- [x] `(((x&~(curr_state >> 1)) << 1) | ((~x)&1))`
- [ ] `(((x&~(curr_state << 1)) >> 1) | ((~x)&1))`

HW05 - Logic, Timing

GRADED

STUDENT
Somya Mohindra

TOTAL POINTS
31 / 33 pts

QUESTION 1

Waveform Diagrams — **4** / 4 pts

1.1 (no title) — **1** / 1 pt

1.2 (no title) — **1** / 1 pt

1.3 (no title) — **1** / 1 pt

1.4 (no title) — **1** / 1 pt

QUESTION 2

Clock Frequency — **6** / 8 pts

2.1 (no title) — **4** / 4 pts

2.2 (no title) — **0** / 2 pts

2.3 (no title) — **2** / 2 pts

QUESTION 3

Circuits and Delays — **5** / 5 pts

3.1 (no title) — **1** / 1 pt

3.2 (no title) — **2** / 2 pts

3.3 (no title) — **2** / 2 pts

QUESTION 4

Boolean Algebra — **5** / 5 pts

4.1 (no title) — **1** / 1 pt

4.2 (no title) — **1** / 1 pt

4.3 (no title) — **1** / 1 pt

4.4 (no title) — **1** / 1 pt

4.5 (no title) — **1** / 1 pt

QUESTION 5

Truth Table — **5** / 5 pts

QUESTION 6

Finite State Machine **6** / 6 pts

6.1 (no title) **3** / 3 pts

6.2 (no title) **3** / 3 pts