CS 61C Fall 2018

Logic and State

Discussion 10: October 29, 2018

1 Boolean Logic

In digital electronics, it is often important to get certain outputs based on your inputs, as laid out by a truth table. Truth tables map directly to Boolean expressions, and Boolean expressions map directly to logic gates. However, in order to minimize the number of logic gates needed to implement a circuit, it is often useful to simplify long Boolean expressions.

We can simplify expressions using the nine key laws of Boolean algebra:

Name	AND Form	OR form
Commutative	$AB = BA$	$A + B = B + A$
Associative	$AB(C) = A(BC)$	$A + (B + C) = (A + B) + C$
Identity	$1A = A$	$0 + A = A$
Null	$0A = 0$	$1 + A = 1$
Absorption	$A(A + B) = A$	$A + AB = A$
Distributive	$(A + B)(A + C) = A + BC$	$A(B + C) = AB + AC$
Idempotent	$A(A) = A$	$A + A = A$
Inverse	$A(\overline{A}) = 0$	$A + \overline{A} = 1$
Demorgan's	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}(\overline{B})$

1.1 Simplify the following Boolean expressions:

(a) $(A + B)(A + \overline{B})C$

$(A+B)(A+\bar{B}) = A$ ← by analysis or can write out

$A + AB + A\bar{B} + B\bar{B}$

$A(B+\bar{B}) = A$; $B\bar{B} = 0$

$\Rightarrow AC$

(b) $\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + A\bar{B}\bar{C} + ABC + A\bar{B}C$

$\bar{A}\bar{C}(\bar{B}+B) + A\bar{C}(B+\bar{B}) + AC(B+\bar{B})$; $(B+\bar{B}) = 1$

$\bar{A}\bar{C} + A\bar{C} + AC = \bar{C}(\bar{A}+A) + AC = \bar{C} + AC$ ← simpler

or we could try $= \bar{A}\bar{C} + A\bar{C} + A\bar{C} + AC$

$= \bar{C}(\bar{A}+A) + A(\bar{C}+C) = \bar{C} + A$

(c) $\overline{A(\bar{B}\bar{C} + BC)}$

Demorgans: $\bar{A} + \overline{\bar{B}\bar{C} + BC}$

again: $\bar{A} + \overline{(\bar{B}\bar{C})}\overline{(BC)} = \bar{A} + (B+C)(\bar{B}+\bar{C})$

$= \bar{A} + B\bar{C} + \bar{B}C$

(d) $\overline{A}(A+B)+(B+AA)(A+\overline{B})$ = $\bar{A}B + (B+A)(A+\bar{B})$

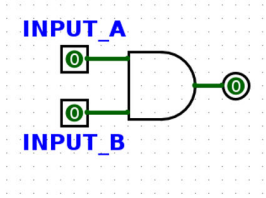
A from (a)

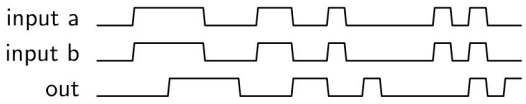
$= \bar{A}B + A$

$= B + A$

2 State Intro

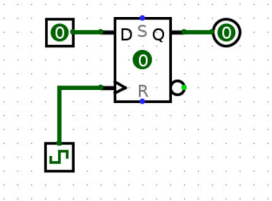
There are two basic types of circuits: combinational logic circuits and state elements. **Combinational logic** circuits simply change based on their inputs after whatever propagation delay is associated with them. For example, if an AND gate (pictured below) has an associated propagation delay of 5ns, its output will change based on its input as follows:

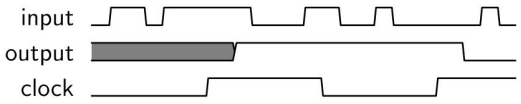




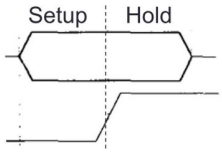
You should notice that the output of this AND gate always changes 5ns after its inputs change.

State elements, on the other hand, can *remember* their inputs even after the inputs change. State elements change value based on a clock signal. A rising edge-triggered register, for example, samples its input at the rising edge of the clock (when the clock signal goes from 0 to 1).





You'll notice that the value of the output in the diagram above doesn't change immediately after the rising edge of the clock. Like logic gates, registers also have a delay associated with them before their output will reflect the input that was sampled. This is called the **clk-to-q** delay. ("Q" often indicates output).

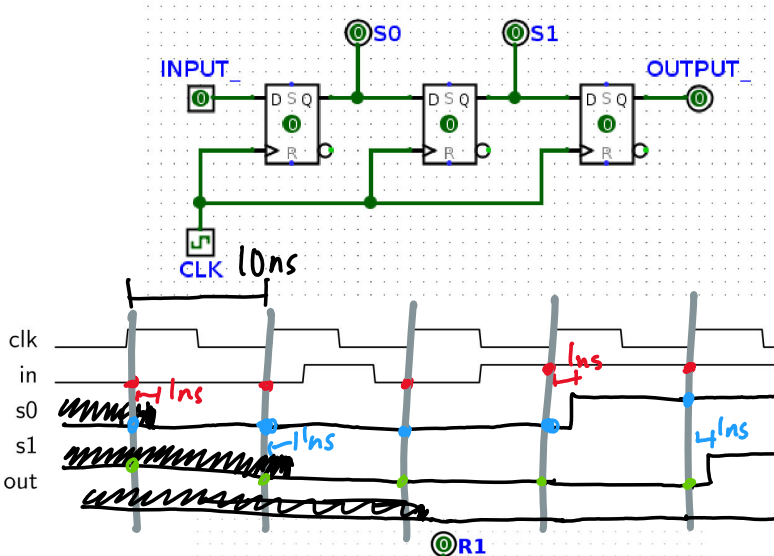


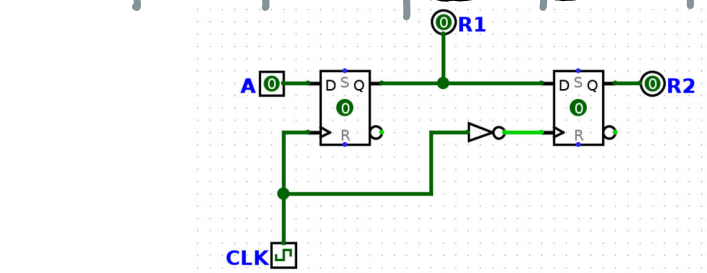
The input the register samples has to be stable for a certain amount of time around the rising edge of the clock for the input to be sampled accurately. The amount of time before the rising edge it must be stable is called the **setup** time, while the time after the rising edge it must be stable is called the **hold** time. Hold time is included in clk-to-q delay, so clk-to-q time will always be greater than or equal to hold time.

Clock cycle time must be small enough that inputs to registers don't change within the hold time and large enough to account for clk-to-q times, setup times, and combinational logic delays.

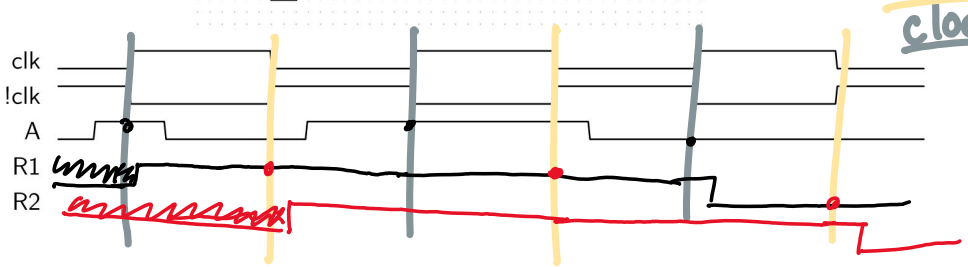
2.1 For the following 2 circuits, fill out the timing diagram. The clock period (rising edge to rising edge) is 10ns. For every register, clk-to-q delay is 1ns and setup/hold time are 0ns. There is no delay associated with a NOT gate.

1) Draw Clock Ticks
2) find value @ rising edge
3) propagate signal



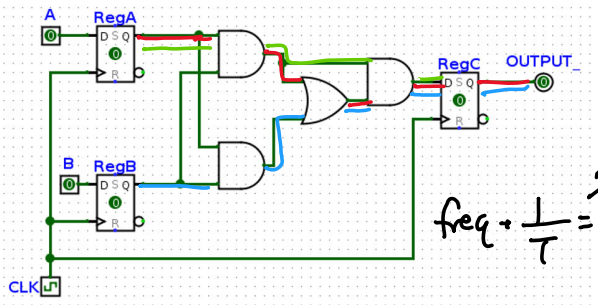


!clock
clock



2.2 In the circuit below, RegA and RegB have setup, hold, and clk-to-q times of 4ns, all logic gates have a delay of 5ns, and RegC has a setup time of 6ns. What is the maximum allowable hold time for RegC? What is the minimum acceptable clock cycle time for this circuit, and clock frequency does it correspond to?

min cycle: changes have to propagate ⇒ longest path



freq = $\frac{1}{T}$ = 40MHz

(1) = 4 + 5 + 5 + 5 + 6 = 25ns

(2) = 4 + 5 + 5 + 5 + 6 = 25 ns

clk-to-q | logic delay | setup

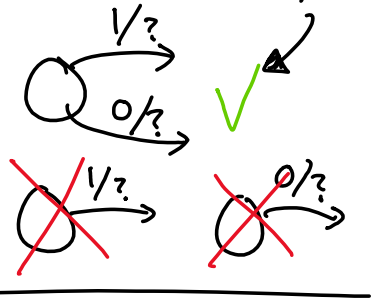
max hold time: first time it changes (breaks hold)

⇒ fastest path ⇒ 4 + 5 + 5 = 14 ns

Transition: (s1) —in/out→ (s2)

3 Finite State Machines

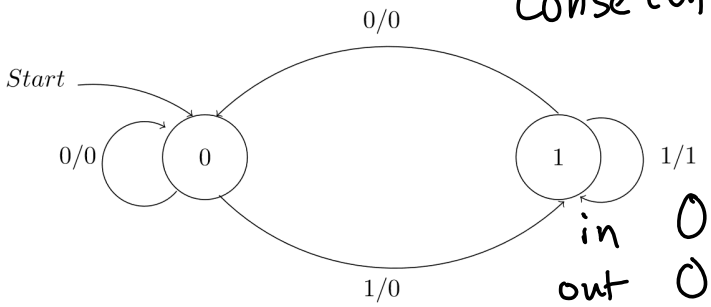
valid FSM reminder: handle 0 or 1 input



Automatons are machines that receive input and use various states to produce output. A finite state machine is a type of simple automaton where the next state and output depend only on the current state and input. Each state is represented by a circle, and every proper finite state machine has a starting state, signified either with the label "Start" or a single arrow leading into it. Each transition between states is labeled [input]/[output].

3.1 What pattern in a bitstring does the FSM below detect? What would it output for the input bitstring "011001001110"?

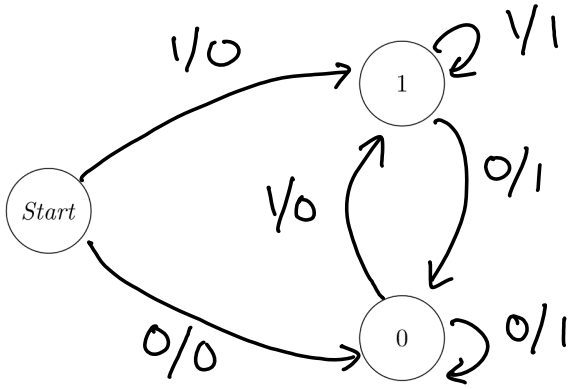
consecutive 1's (see's "11")



in 0 1 1 0 0 1 0 0 1 1 1 0

out 0 0 1 0 0 0 0 0 0 1 1 0

3.2 Fill in the following FSM for outputting a 1 whenever we have two repeating bits as the most recent bits, and a 0 otherwise. You may not need all states.



test 0 0 1 1 0 1 0 0

out 0 1 0 1 0 0 0 1 ✓

3.3 Write an FSM that will output a 1 if it recognizes the regex pattern {10+1}.