CS 61C Summer 2019

Logic,SDS

Discussion 6: June 15, 2019

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

$$\begin{aligned}(A + B)(A + \bar{B})C &= (A + B\bar{B})C \\ &= AC\end{aligned}$$

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

$$\begin{aligned}\bar{A}\bar{C}(\bar{B} + B) + A\bar{C}(B + \bar{B}) + AC(B + \bar{B}) &= \bar{A}\bar{C} + A\bar{C} + AC \\ &= \bar{A}\bar{C} + A\bar{C} + A\bar{C} + AC \\ &= (\bar{A} + A)\bar{C} + A(\bar{C} + C) \\ &= A + \bar{C}\end{aligned}$$

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

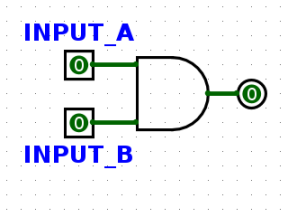
$$\begin{aligned}\overline{A(\bar{B}\bar{C}+BC)} &= \bar{A}+\overline{\bar{B}\bar{C}+BC} \\ &= \bar{A}+\overline{\bar{B}\bar{C}}\,\overline{BC} \\ &= \bar{A}+(B+C)(\bar{B}+\bar{C}) \\ &= \bar{A}+B\bar{C}+\bar{B}C\end{aligned}$$

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

$$\begin{aligned}\overline{A}(A+B)+(B+AA)(A+\overline{B} &= (\overline{A}A+\overline{A}B)+(B+AA)(A+\overline{B}) \\ &= \overline{A}B+(B+AA)(A+\overline{B}) \\ &= \overline{A}B+(B+A)(A+\overline{B}) \\ &= \overline{A}B+(BA+AA+B\overline{B}+A\overline{B}) \\ &= \overline{A}B+(BA+A+A\overline{B}) \\ &= \overline{A}B+A \\ &= A+B\end{aligned}$$

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 2ps, its output will change based on its input as follows:



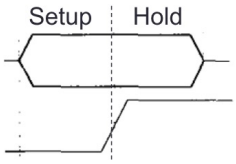
input a	2L 4H 3L 2H 2L 1H 5L 1H 1L 1H 2L
input b	2L 4H 3L 2H 2L 1H 5L 1H 1L 1H 2L
output	2U 2L 4H 3L 2H 2L 1H 5L 1H 1L 1H

Where U, L, and H refer to an undefined, low (0), or high (1) signal respectively, for some number of nanoseconds.

You should notice that the output of this AND gate always changes 2ps after its inputs change.

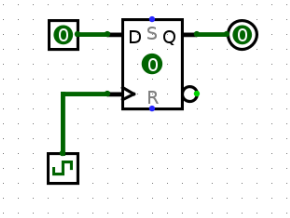
State elements, on the other hand, can *remember* their inputs even after the inputs change. State elements change value based on a clock signal. A rising edge-triggered register, for example, samples its input at the rising edge of the clock (when the clock signal goes from 0 to 1).

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). This is the time between the rising edge of the clock signal and the time the register's output reflects the input change.



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 the input must be stable is called the `setup` time, and the time after the rising edge the input 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.

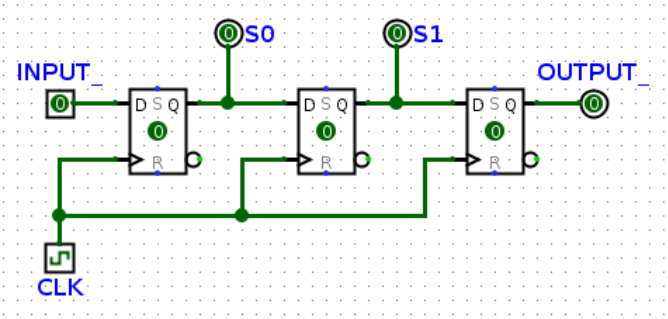
For the following register circuit, assume **setup** of 2.5ps, **hold** time of 1.5ps, and a **clk-to-q** time of 1.5ps. The clock signal has a period of 13ps.



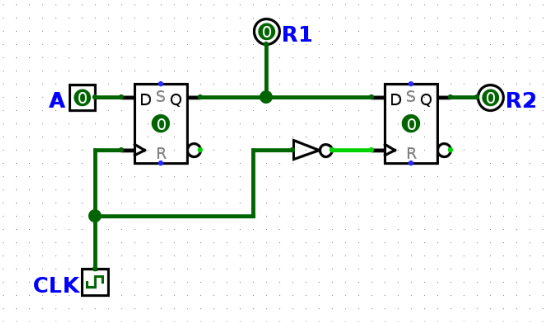
clock 6.5L 6.5H 6.5L 6.5H
input 1L 2H 1L 5H 3L 2H 2L 1H 5L 1H 3L
output 8U 13H 5L

You'll notice that the value of the output in the diagram above doesn't change immediately after the rising edge of the clock. 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 8ps. For every register, clk-to-q delay is 2ps, setup time is 4ps, and hold time is 2ps. NOT gates have a 2ps propagation delay

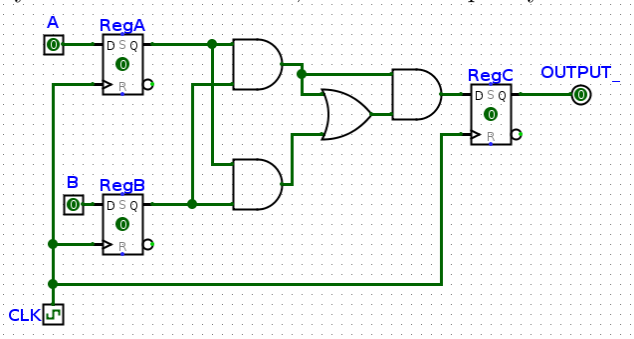


clk 4L 4H 4L 4H 4L 4H 4L 4H 4L 4H 4L 4H
in 14L 4H 6L 16H 8L
s0 6U 16L 8U 16H 2L
s1 14U 16L 8U 10H
out 22U 16L 8U 2H



clk 4L 4H 4L 4H 4L 4H 4L 4H 4L 4H 4L 4H
!clk 4H 4L 4H 4L 4H 4L 4H 4L 4H 4L 4H 4L
A 6H 8L 8H 2L 6H 8L 2H 6L 2H
R1 6U 8H 8L 16H 10L
R2 12U 8H 8L 16H 4L

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?



The maximum allowable hold time for RegC is how long it takes for RegC's input to change, so (clk-to-q of A or B) + shortest CL time = 4 + (5 + 5) = 14 ns.

The minimum acceptable clock cycle time is clk-to-q + longest CL time + setup time = 4 + (5 + 5 + 5) + 6 = 25 ns.

25 ns corresponds to a clock frequency of $(1/(25 \quad 10^{-9}))s^{-1} = 40MHz$

3 Finite State Machines

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"?