CS 61C Combinational Logic, SDS and FSMs

Fall 2023 Discussion 6

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	$A(\overline{A}) = 0$	$A + \overline{A} = 1$
De Morgan's	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}(\overline{B})$

Simplify the following Boolean expressions:

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

$$\begin{aligned}(A+B)(A+\overline{B})C &= (AA+A\overline{B}+BA+B\overline{B})C\\ &= (A+A(\overline{B}+B)+0)C\\ &= (A+A)C\\ &= AC\end{aligned}$$

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

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

Alternatively, you can prove to yourself that X + $\overline{X}Y = X + Y. Then,$

$\overline{A}\overline{C} + A\overline{C} + AC$

$= (\overline{A} + A)\overline{C} + AC$

$= \overline{C} + AC$

$= \overline{C} + A$

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

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

2 Combinational Logic Design

Logic gates can be connected together to create a variety of useful functions. In this question, we will implement a simplified version of the memory write mask for the RISC-V CPU. The memory write mask looks at the store instruction given and decides which of the four bytes (in one word of memory) to write to. It is four bits long - each bit is one if we should write to the corresponding byte, but zero if we shouldn't. For simplicity, assume that all memory addresses used in store instruction are word-aligned. Here's a truth table for the simpified memory mask:

Instruction	**funct3**	**Out**
sb	000	0001
sh	001	0011
sw	010	1111
(undefined)	011-111	xxxx

The x's for the final entry of the table indicates that any output is fine in that case.

2.1 Write out and simplify boolean expressions for each of the output bits in terms of the funct3 (input) bits f_2, f_1, f_0.

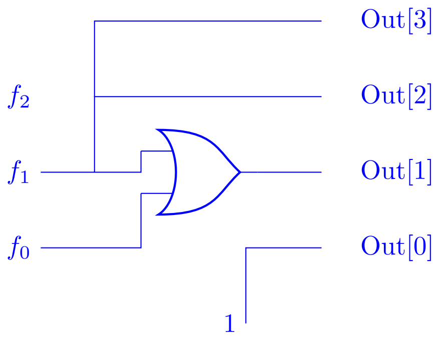
We'll work on the bits from right to left. For Out[0], its value is one in all input cases that are defined, so we can just set Out[0] $= 1$.

For the other bits, we can write out the base canonical form first, but then we can bring in any amount of terms from the undefined cases (equivalent to setting this bit to one in that undefined case) to simplify the expression.

Out[1] $= \overline{f_2}\,\overline{f_1}f_0 + \overline{f_2}f_1\overline{f_0}$ from the cases given. Since all of the cases where f_2=1 are undefined, we can bring in some of those terms to cancel out the f_2's: Out[1] $= \overline{f_2}\,\overline{f_1}f_0 + f_2\overline{f_1}f_0 + \overline{f_2}f_1\overline{f_0} + f_2f_1\overline{f_0} = \overline{f_1}f_0 + f_1\overline{f_0}$. We can go one step further: the input 011 is also undefined, so if we bring in that term (and the corresponding f_2=1 term 111), we end up with Out[1] $= \overline{f_1}f_0 + f_1\overline{f_0} + f_1f_0 = f_1 + f_0$.

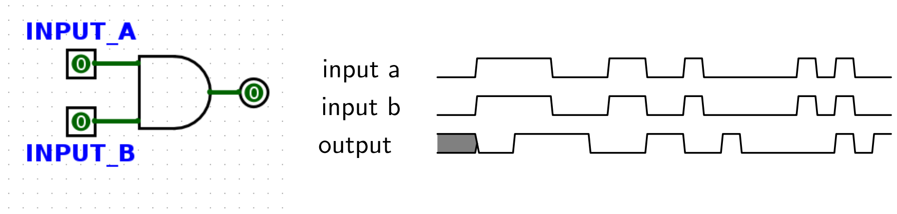
Following a similar process, the final two bits simplify to Out[3] $=$ Out[2] $= f_1$.

$\boxed{2.2}$ Draw out the boolean circuit for this memory write mask based on your simplified expressions above. You may use constants 0 and 1, and the logic gates AND, OR, NOT.



3 State Intro

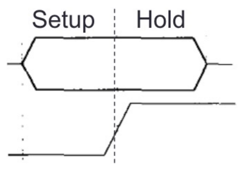
There are two basic types of circuits: combinational logic circuits and state elements. **Combinational logic** circuits simply change based on their inputs after whatever propagation delay is associated with them. For example, if an AND gate (pictured below) has an associated propagation delay of 2ps, its output will change based on its input as follows:



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

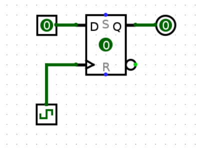
State elements, on the other hand, can *remember* their inputs even after the inputs change. State elements change value based on a clock signal. A rising edge-triggered register, for example, samples its input at the rising edge of the clock (when the clock signal goes from 0 to 1).

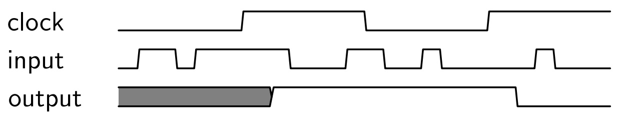
Like logic gates, registers also have a delay associated with them before their output will reflect the input that was sampled. This is called the **clk-to-q** delay. ("Q" often indicates output). This is the time between the rising edge of the clock signal and the time the register's output reflects the input change.



The input to 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 generally included in clk-to-q delay, so clk-to-q time will usually be greater than or equal to hold time. Logically, the fact that clk-to-q $\geq$ hold time makes sense since it only takes clk-to-q seconds to copy the value over, so there's no need to have the value fed into the register for any longer.

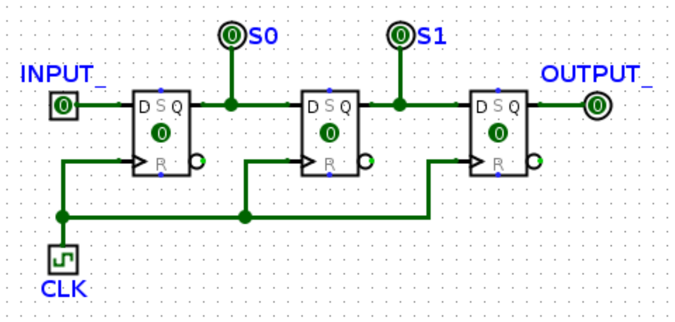
For the following register circuit, assume **setup** of 2.5ps, **hold** time of 1.5ps, and a **clk-to-q** time of 1.5ps. The clock signal has a period of 13ps.

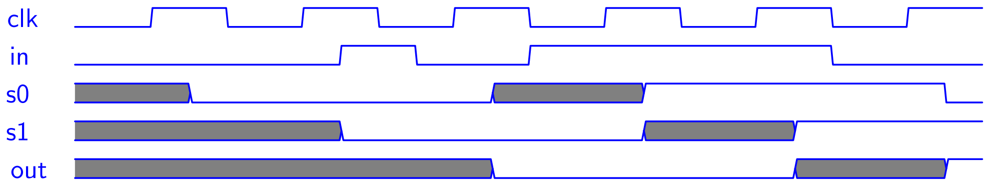




You'll notice that the value of the output in the diagram above doesn't change immediately after the rising edge of the clock. Until enough time has passed for the output to reflect the input, the value held by the output is garbage; this is represented by the shaded gray part of the output graph. Clock 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.

3.1 For the following circuit, 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.



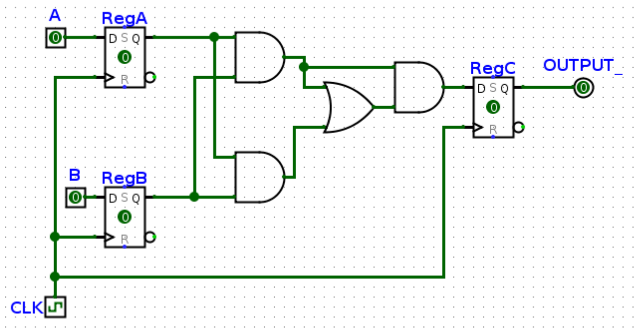


At a rising clock edge, the value that the output of a register will update to is the value of the input of the register sampled at this rising clock edge. However, notice that the output will not update until clk to q after the rising clock edge.

Gray signals:

- the output of a register at the very beginning, before a rising clock edge.
- the input into a register did not satisfy setup or hold times, so the output of the register will be garbage. For example, at the 3rd rising clock edge, "in" changed 2 ps before, violating the setup time of 4ps. Clk-to-q after this 3rd rising edge, the s0 is garbage.
- the input into the register was garbage

3.2 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?



Hold time of RegC is the amount of time that the input into RegC must be stable for after a rising clock edge. We need to first consider the input to RegC and how it might change after a rising clock edge. Then, we need to consider the shortest amount of time that it could take for this input to change in order to determine max hold time. Think about why this is the case. What could happen if the hold time of RegC was greater? Again, the time we are considering is after a rising clock edge.

We look at the path of the input coming from a previous register. The input to RegC comes from the output of Reg A and B fed into some combinational logic (the AND and OR gates). How is the input related to a rising clock edge? Because this input signal path is the output of a register (A and B), it can be updated clk-to-q after the rising clock edge (taking on the values of the inputs to Reg A and B sampled at this rising clock edge). After this clk-to-q after the rising edge, the signal flows through the CL, and it takes CL delay time to reach RegC. To determine the minimum amount of time after the rising clock edge for the input to RegC to change, we consider the shortest CL path/delay.

Max hold time RegC = (clk-to-q of A or B) + shortest CL time = 4 + (5 + 5) = 14 ns.

To determine minimum clock cycle time (time between two rising edges), consider every path between two registers. The clock cycle must be greater than the time for the output of the first register to update after the rising clock edge (clk-to-q delay) plus the longest CL delay of the CL path that this output feeds into. Recall, the output of a CL block updates delay time after its input changes. Then, the signal flowing out of the CL path and into the second register must be stable (or finalized with the correct value) setup time before the next rising clock edge. The minimum clock cycle ensures a register will sample the correct value at a rising clock edge.

Therefore, the minimum acceptable clock cycle time is clk-to-q + longest CL time + setup time. In the circuit above, there is a path between registers A/B and C. Min clock cycle = 4 + (5 + 5 + 5) + 6 = 25 ns.
25 ns corresponds to a clock frequency of $(1/(25 * 10^{-9}))s^{-1} = 40MHz$

Conceptual questions:

3.3 Simplifying boolean logic expressions will not affect the performance of the hardware implementation.

False. Different gate arrangements that implement the same logic can have different propagation delays, which can affect the allowable clock speed.

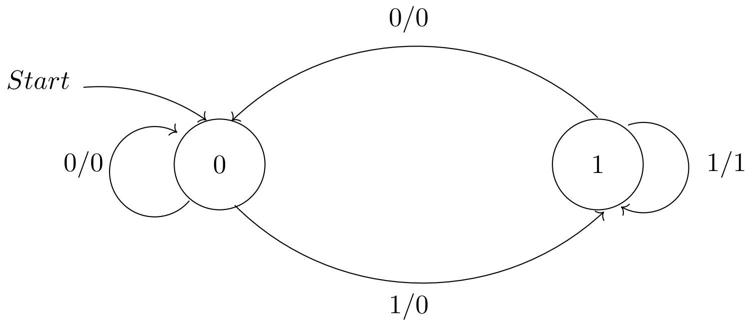
3.4 The fewer logic gates, the faster the circuit (assuming each gate has the same propagation delays).

False. Propagation delays add to the allowable clock speed with the depth of the circuit, so a wide circuit with more gates in parallel can have less delay than just a few gates arranged in sequence.

4 Finite State Machines

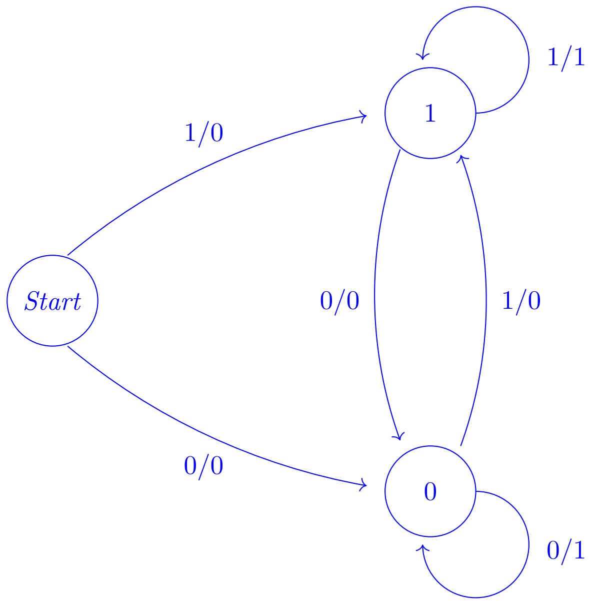
Automatons are machines that receive input and use various states to produce output. A finite state machine is a type of simple automaton where the next state and output depend only on the current state and input. Each state is represented by a circle, and every proper finite state machine has a starting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

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

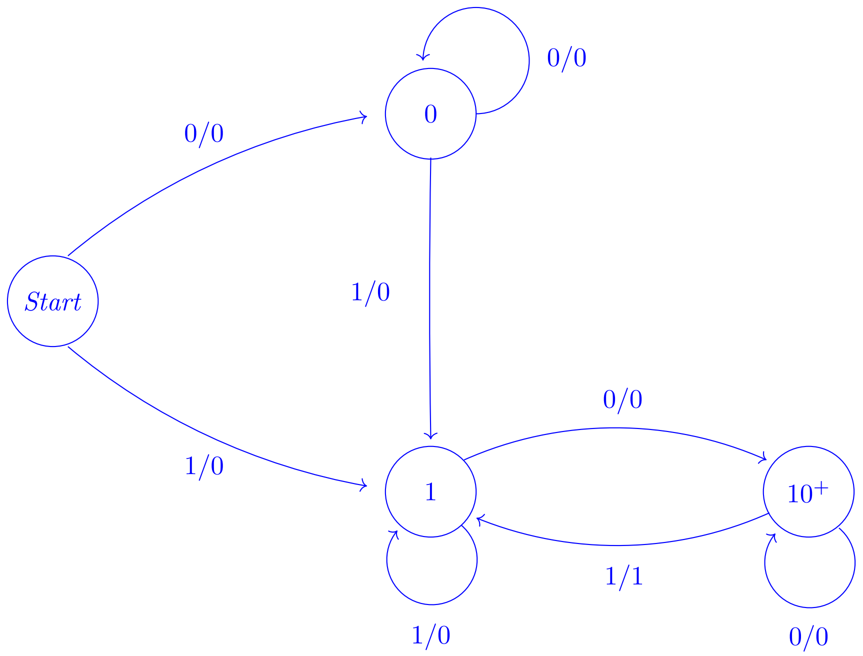


The FSM outputs a 1 if it detects the pattern "11".
The FSM would output "001000000110"

4.2 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.



$\boxed{4.3}$ Draw an FSM that will output a 1 if it recognizes the regex pattern {10+1}. (That is, if the input forms a pattern of a 1, followed by one or more 0s, followed by a 1.)


Start
0
1
10^+
0/0
0/0
1/0
1/0
0/0
1/1
1/0
0/0