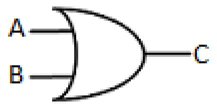
CS61C Fall 2012 – 8 – Combinational Logic and FSM

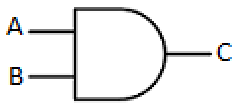
Combinational Logic

OR



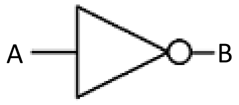
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

AND



A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

NOT



A	B
0	1
1	0

Using these above gates, create a NOR gate, a NAND gate, an XOR gate, and an XNOR gate.

NOR	NAND	XOR	XNOR
Append NOT	Append NOT	Logic Gate of	Append NOT
After OR	After AND	A(~B) + (~A)B	After XOR

Simplify the following Boolean expressions

(~A)B + AB **= (~A + A)B = (1)B = B**

A(A+AB)+A(B+~B)+AA **= A(A) + A(1) + A = A + A + A = A**

(~A)BC + A(~B)C + AB(~C) + ABC **= (~A)BC + A(~B)C + AB(~C+C) = (~A)BC + AB + A(~B)C =(~A)BC + A(B + C)**
= (~A)BC + AB + AC = B((~A)C + A) + AC = B(A + C) + AC = AB + AC + BC

One common Boolean operator is the implication operator (A->B, A implies B, or "if A, then B"). One common function that high-level languages use is the add function. Both can be represented with gates. For these two operations, create truth tables for the input/output values, create a minimalized Boolean expression, and lastly create the logic gates to represent them.

Implication

A	B	C
0	0	1
0	1	1
1	0	0
1	1	1

(~A)(~B) + (~A)B + AB
(~A)(~B) + B
~A + B

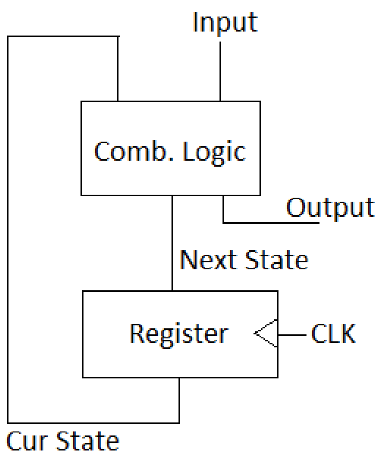
Add

X0	X1	X2	Y0	Y1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(~X0)(~X1)X2+(~X0)X1(~X2)+(X0)(~X1)(~X2)+X0X1X2=Y0
X0(~X1)(~X2) + X0(~X1)X2 + X0X1(~X2) + X0X1X2=Y1
This is one of the operations you need to implement for Project 4. Please minimize the expression and implement the gates yourself.

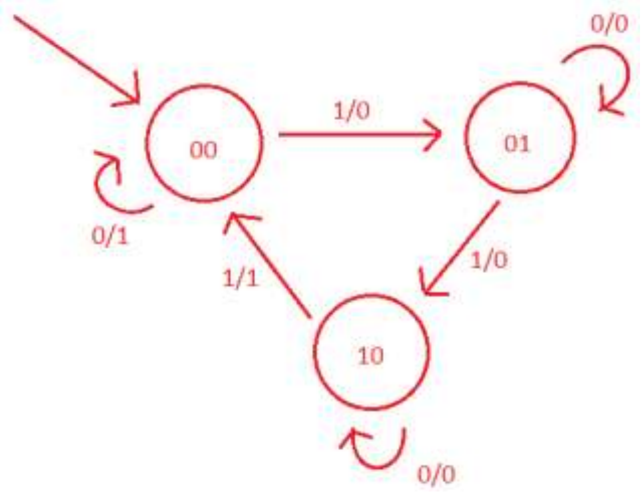
Finite State Machines

FSMs can be an incredibly useful computational tool. They have a straightforward implementation in hardware:



The register holds the current state (encoded as a particular combination of bits), and the combinational logic block maps from {current state, input} to {next state, output}.

Draw a transition diagram for an FSM that can take in an input one bit at a time, and after each input received, output is whether the number of 1s is divisible by 3. Then, assign states binary encodings and complete the truth table for the FSM. Finally, write a simplified Boolean algebra expression that implements the FSM's truth table.



cur. state	input	next state	output
00	0	00	1
00	1	01	0
01	0	01	0
01	1	10	0
10	0	10	0
10	1	00	1

The states each correspond to the number of 1s seen so far, mod 3. When this quantity is 0, 1s seen so far is divisible by 3, and we output 1. Behavior for current state 11 is undefined, since we don't expect our machine to ever reach that state. (Read indices as little endian.)

NS[1]: (~CS[1])CS[0]In + CS[1](~CS[0])(~In)

NS[0]: (~CS[1])(~CS[0])In + (~CS[1])CS[0](~In)=(~CS[1])((~CS[0])In + CS[0](~In)) ~= (~CS[1])(CS[0] XOR In)

Out: (~CS[1])(~CS[0])(~In) + CS[1]*(~CS[0])In=(~CS[0])((~CS[1])(~In) + CS[1]In) ~=CS[0]*(CS[1] XNOR In)