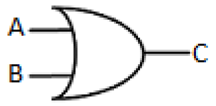
CS61C Fall 2012 – 8 – Combinational Logic and FSM

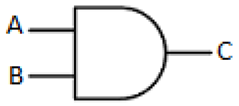
Combinational Logic

OR



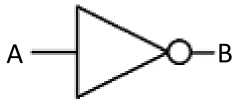
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

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.

Simplify the following Boolean expressions
(~A)B + AB

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

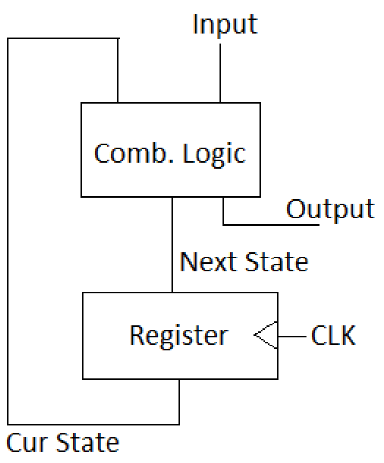
(~A)BC + A(~B)C + AB(~C) + ABC

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 **Add**

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.