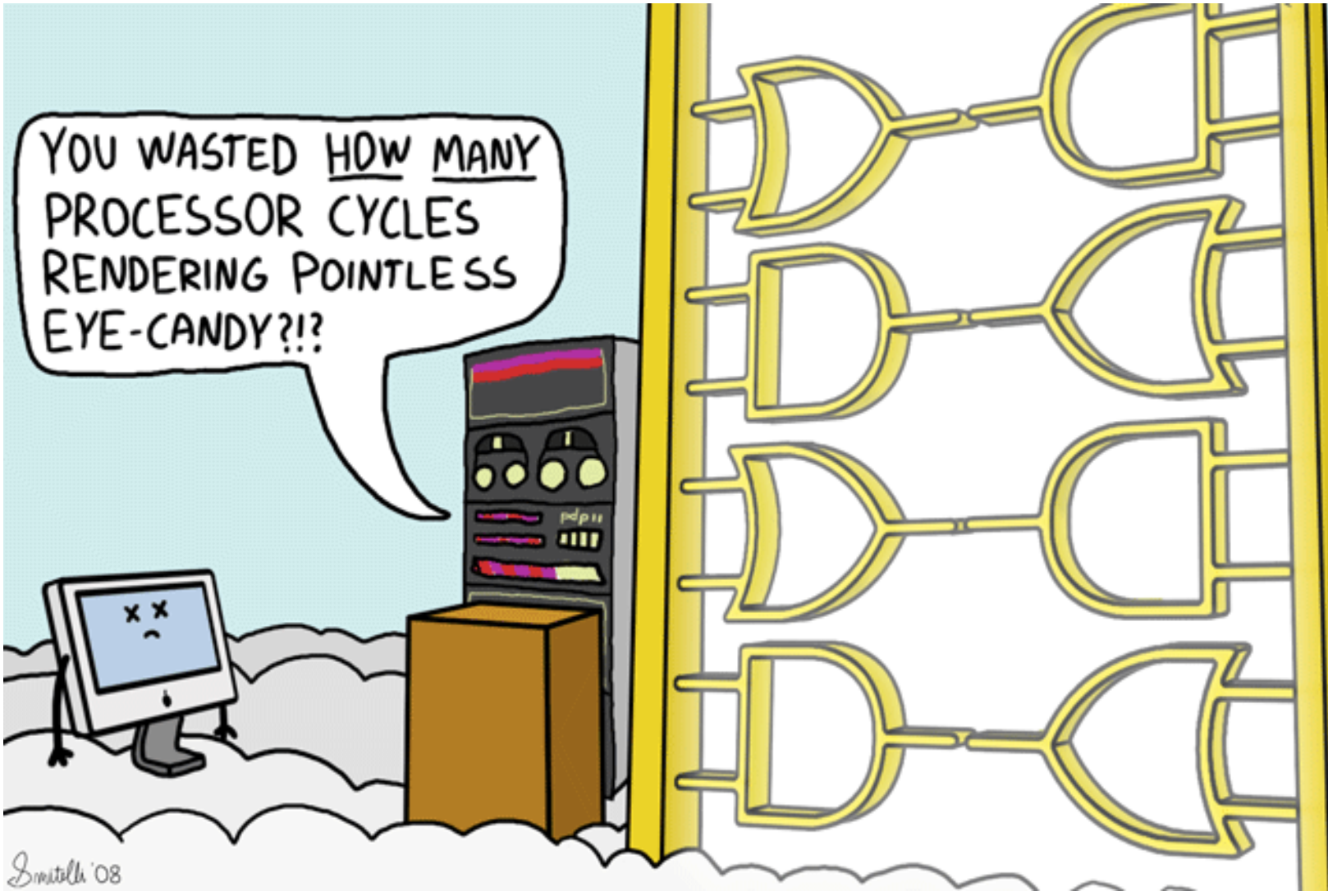
MS1A NOTES


YOU WASTED HOW MANY PROCESSOR CYCLES RENDERING POINTLESS EYE-CANDY?!?
pdp 11
'08

M51A Notes:

BCD ⇒ Binary Coded Decimal. We have each for bit value based on a weight ex. 8421.

Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111

Binary to Hex: treat each 4 bits translate to one hexadecimal
Binary to Octal: treat each 3 bits translate to one octal value.

Fractions: $(39.6875)_{10}$ ⇒ 100111.1001 ⇒ $(2^{-1}, 2^{-2}, 2^{-3}\ldots)$ $(8^{-1}, 8^{-2}, 8^{-3}\ldots)$ $(16^{-1}, 16^{-2}, 16^{-3}\ldots)$

(39 = Int, 6875 = fraction; 100111 = Int, 1001 = Decimal)

Unsigned: (Only positive)

Signed: (Two's complement) ⇒ the most significant bit would determine sign.

Binary to Unsigned: $\sum_{i=0}^{n-1} x_i 2^i$; Binary to Signed: $-(2^n) + \sum_{i=0}^{n-1} x_i \cdot 2^i$

Unsigned Range: 2^{n-1} ; Signed Range: -2^{n-1} to $2^{n-1}-1$

Detecting Overflow: (Only if they have same sign). If two positive numbers add to a negative or if two negative add to positive there is overflow

Understanding Gray Code:

- Two successive values only have a 1 bit difference.

- Alias: Reflected Binary Code, RBC
  Unit Distance Coding

- Binary to Gray Code:
  - Record MSB
  - Add msb to next bit, neglect carries.
  - Repeat until lsb.

Ex:

$13_{10} \rightarrow (1101)_2 \rightarrow g_3 = b_3 \rightarrow (1011)_g$

$b_3 - b_0$

$g_2 = b_2 + b_3$

$g_1 = b_1 + b_2$

$g_0 = b_0 + b_1$

Table 1.6
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

Binary Logic:

each variable has exactly two distinct values 0 or 1 ⇒ where 0 is false and 1 is true

Logic gate: a specific piece of hardware that does a specific logic operation.

And: (x AND Y is equal to z) represented as * or no operator

Or: (x Or Y is equal to z) represented by a +

Not: represented by a bar or '. ex: $\bar{x}$, x'

AND:

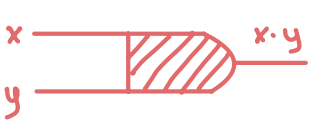
x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

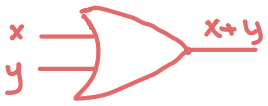
OR:

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

NOT:

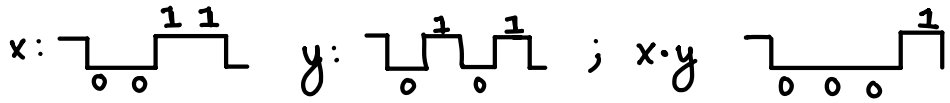
x	x'
0	1
1	0



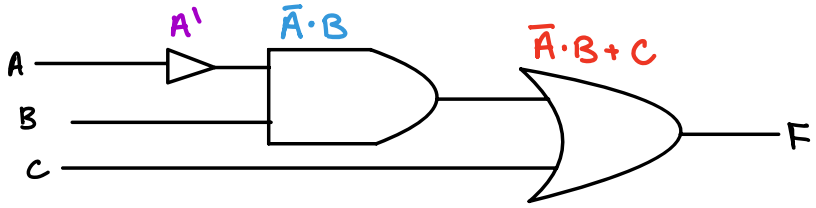


* We can have more than two inputs to and, or logic gates

Timing Diagram: a zero is at a low and a one is considered a high.



Example:



Draw the truth table for this circuit: number of rows = 2^n

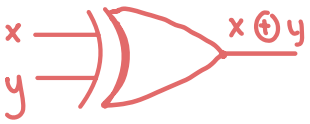
The best stratagey is to deal with each circut element one at a time.

A	B	C	A'	Ā·B	Ā·B + C
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	0	0
1	1	1	0	0	1

More Gates:

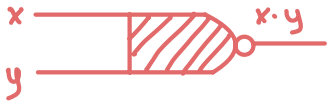
XOR: Exclusive or represented by $\oplus$ $\quad * (\bar{A} \cdot B) + (A \cdot \bar{B}) = A \oplus B$

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0



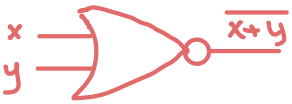
NAND: Not And Gate: $\overline{(x \cdot y)}$

x	y	$\overline{x \cdot y}$
0	0	1
0	1	1
1	0	1
1	1	0



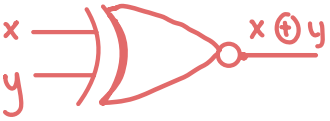
NOR: Not oR Gate: $\overline{x+y}$

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0



XNOR: Not XOR $\overline{x \oplus y}$

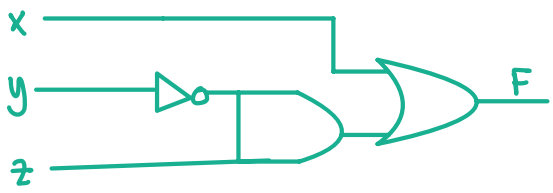
x	y	$\overline{x \oplus y}$
0	0	1
0	1	0
1	0	0
1	1	1



Boolean Algebra: is an algebra dealing with binary variables and logical operations

Boolean Function: is described by boolean equation, truth table, or a logical circuit diagram.

Ex: $F(x,y,z) = x + \bar{y}z$



x	y	z	F
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0

Duality Principle:

IF we exchange every symbol by it's dual formula we get the dual result.

Postulates and Theorems of Boolean Algebra:

$X + 0 = X$	$X \cdot 1 = X$
$X + X' = 1$	$X \cdot X' = 0$
$X + X = X$	$X \cdot X = X$
$X + 1 = 1$	$X \cdot 0 = X$
$(X')' = X$	
$X + Y = Y + X$	$XY = YX$
$X + (Y + Z) = (X + Y) + Z$	$X(YZ) = (XY)Z$
$X(Y + Z) = XY + XZ$	$X + YZ = (X + Y)(X + Z)$
$(X + Y)' = X'Y'$	$(XY)' = X' + Y' \Rightarrow$ DeMorgan
$X + XY = X$	$X(X + Y) = X \Rightarrow$ absorption

* to verify these identities we can draw out their truth tables

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

* $AB + A' = B + A'$

General Form of DeMorgans Theorem:

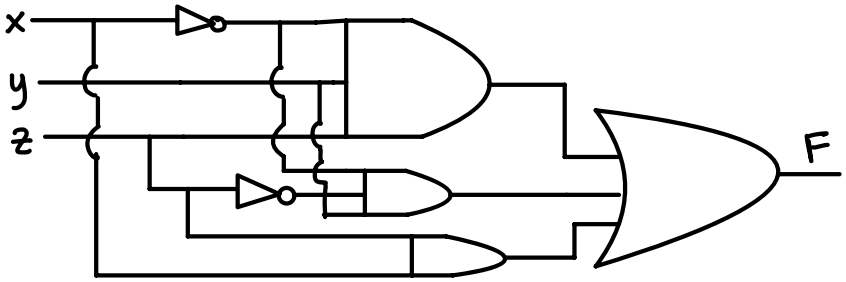
$$\overline{x_1 + x_2 + x_3 + \dots x_n} = \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \dots \bar{x}_n$$

$$\overline{x_1 \cdot x_2 \cdot x_3 \dots x_n} = \bar{x}_1 + \bar{x}_2 + \bar{x}_3 + \dots + \bar{x}_n$$

$F_{(x,y,z)} = \bar{x}yz + \bar{x}y\bar{z} + xz$; 8 literals 3 terms

$= \bar{x}y(z + \bar{z}) + xz \Rightarrow \bar{x}y(1) + xz \Rightarrow \bar{x}y + xz$

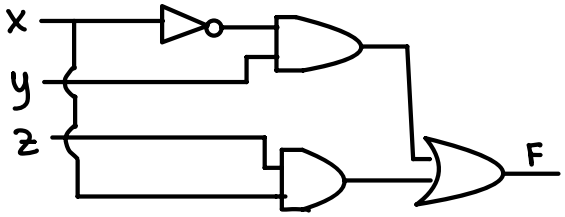
Expanded diagram



Truth Table:

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Simplified Diagram:

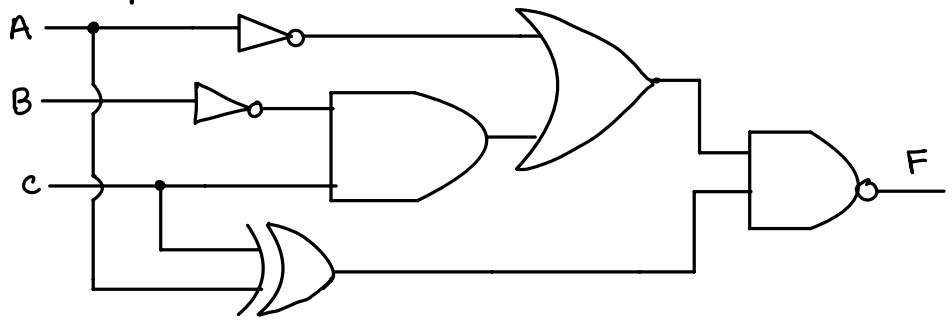


To quickly get truth table values from a function:

Take each term in the function. For each literal in the term if an input is not present it can be either zero or one. If it is present assign it a 1. If it is a not assign it a zero. The corresponding combination in the truth table would evaluate to a 1 the rest of the combinations would evaluate to zero.

$\bar{x}y \Rightarrow$ x is zero; $y = 1$; $z = 1,0 \Rightarrow$ 011 or 010; $xz \Rightarrow$ 101, 111

Example:



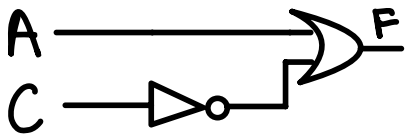
Writing the function: $F_{(A,B,C)} = \overline{(\bar{A} + \bar{B}C)(C \oplus A)}$

$\Rightarrow F_{(A,B,C)} = \overline{(\bar{A}+\bar{B}C)(C+A)(\bar{C}+\bar{A})} \Rightarrow \overline{\bar{A}+\bar{B}C} + \overline{C+A} + \overline{\bar{C}+\bar{A}}$

$\Rightarrow \bar{\bar{A}}\,\overline{(\bar{B}C)} + \overline{C+A} + \overline{\bar{C}+\bar{A}} \Rightarrow \bar{\bar{A}}(\bar{\bar{B}}+\bar{C}) + \bar{C}\bar{A} + \bar{\bar{C}}\bar{\bar{A}}$

$\Rightarrow A(B+\bar{C}) + \bar{C}\bar{A} + CA \Rightarrow AB + A\bar{C} + \bar{C}\bar{A} + CA \Rightarrow A + AB + \bar{C}\bar{A}$

$\Rightarrow A + \bar{C}\bar{A} \Rightarrow A + \bar{C}$



Truth tables:

a,b,c	f
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

a,c	f
0 0	1
0 1	0
1 0	1
1 1	1

Sum of Product: additive between terms → $F = \bar{x}yz + x\bar{y}\bar{z}$

Product of Sum: multiplicative between terms. → $(x+y)(\bar{x}+\bar{z})$

Complement of a function:

- Interchange 1s and 0s in a truth table
- Derived algebrically by using DeMorgan's theorm

Ex: $F_1(x,y,z) = \bar{x}y\bar{z} + \bar{x}\bar{y}z$

$$\overline{F_1} = \overline{(\bar{x}y\bar{z} + \bar{x}\bar{y}z)}$$

$$\Rightarrow \overline{\bar{x}y\bar{z}}\,\overline{(\bar{x}\bar{y}z)}$$

$$\Rightarrow (x+\bar{y}+z)(x+y+\bar{z}) \rightarrow \text{Product of Sums}$$

$$\Rightarrow x + xy + x\bar{z} + x\bar{y} + \bar{y}\bar{z} + xz + zy + z\bar{z}$$

$$\Rightarrow x(1+y) + x(\bar{z}+z) + x\bar{y} + \bar{y}\bar{z} + yz$$

$$\Rightarrow x + x + x\bar{y} + \bar{y}\bar{z} + yz$$

$$\Rightarrow x + \bar{y}\bar{z} + yz \rightarrow \text{Sum of Products}$$

Minterms: m_j

- a product term in which all the variables appear exactly once, either complemented or un-complemented
- represents exactly one combination of the binary variables in a truth table.
- has the value 1 for that combination
- 2^n distinct minterms for n variables

Minterm for 3-variables:

x	y	z	Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\bar{x}\bar{y}\bar{z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\bar{x}\bar{y}z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\bar{x}y\bar{z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\bar{x}yz$	m_3	0	0	0	1	0	0	0	0
1	0	0	$x\bar{y}\bar{z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$x\bar{y}z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$xy\bar{z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	xyz	m_7	0	0	0	0	0	0	0	1

MaxTerms: M_J

- a sum term in which all the variables appear exactly once, either complemented or un-complemented
- represents exactly one combination of the binary variables in a truth table.
- has the value 0 for that combination
- 2^n distinct maxterms for n variables

MaxTerm for 3-variables:

x	y	z	Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$x+y+z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$x+y+\bar{z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$x+\bar{y}+z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$x+\bar{y}+\bar{z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\bar{x}+y+z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\bar{x}+y+\bar{z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\bar{x}+\bar{y}+z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\bar{x}+\bar{y}+\bar{z}$	M_7	1	1	1	1	1	1	1	0

Relation between: Min and Maxterm: $M_J = \overline{m_j}$

Represent as a sum of Minterms:

$F = \bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + x\bar{y}z + xyz$ ⇒ using the table above convert each product term to a minterm

$F = m_0 + m_2 + m_5 + m_7$ ⇒ $F = \sum m(0,2,5,7)$

Represent F in Product of Maxterms: $F_1 = M_1 \cdot M_3 \cdot M_4 \cdot M_6$ ⇒ the same numbers of minterms that weren't included.

$F = \prod M(1,3,4,6)$

Minterm - Maxterm of 4 variables

X	Y	Z	W	SOP	POS	MINTERM	MAXTERM
0	0	0	0	X'Y'Z'W'	X + Y + Z + W	m0	M0
0	0	0	1	X'Y'Z'W	X + Y + Z + W'	m1	M1
0	0	1	0	X'Y'Z'W	X + Y + Z' + W	m2	M2
0	0	1	1	X'Y'ZW	X + Y + Z' + W'	m3	M3
0	1	0	0	X'YZ'W'	X + Y' + Z + W	m4	M4
0	1	0	1	X'Y'ZW'	X + Y' + Z + W'	m5	M5
0	1	1	0	X'Y'Z'W	X + Y' + Z' + W	m6	M6
0	1	1	1	X'YZW	X + Y' + Z' + W'	m7	M7
1	0	0	0	XY'Z'W'	X' + Y + Z + W	m8	M8
1	0	0	1	XY'Z'W	X' + Y + Z + W'	m9	M9
1	0	1	0	XY'ZW'	X' + Y + Z' + W	m10	M10
1	0	1	1	XY'ZW	X' + Y + Z' + W'	m11	M11
1	1	0	0	XYZ'W'	X' + Y' + Z + W	m12	M12
1	1	0	1	XYZ'W	X' + Y' + Z + W'	m13	M13
1	1	1	0	XYZW'	X' + Y' + Z' + W	m14	M14
1	1	1	1	XYZW	X' + Y' + Z' + W'	m15	M15

Minterm - Maxterm of two variables

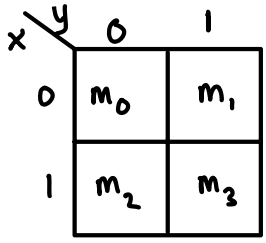
Index	Minterm	Maxterm
0	$\bar{x}\,\bar{y}$	$x + y$
1	$\bar{x}\,y$	$x + \bar{y}$
2	$x\,\bar{y}$	$\bar{x} + y$
3	$x\,y$	$\bar{x} + \bar{y}$

Karnaugh Map: (K- ap)

Adjacent squares in a K-map will only differ by a single term
A K-variable K-map will have K adjacent cells.

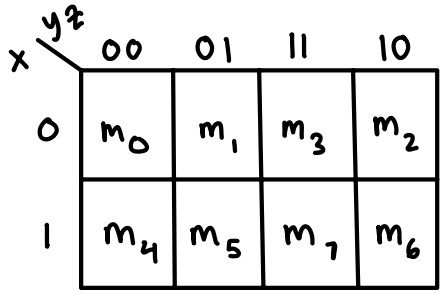
Grouping 2 adj → removes 1 literal
Grouping 4 adj → removes 2 literals
Grouping 2^n adj → removes n literals

2-Variable: x, y



m_0 has a neighbors of m_1, m_2
$m_1 \rightarrow m_0, m_3$
$m_2 \rightarrow m_3, m_0$
$m_3 \rightarrow m_1, m_2$

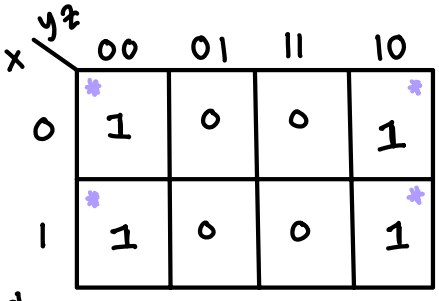
3-Variable: x, y, z



* neighbors wrap around so m_2 is a neighbor of m_0.

Example: * Astericks indicate combined cells

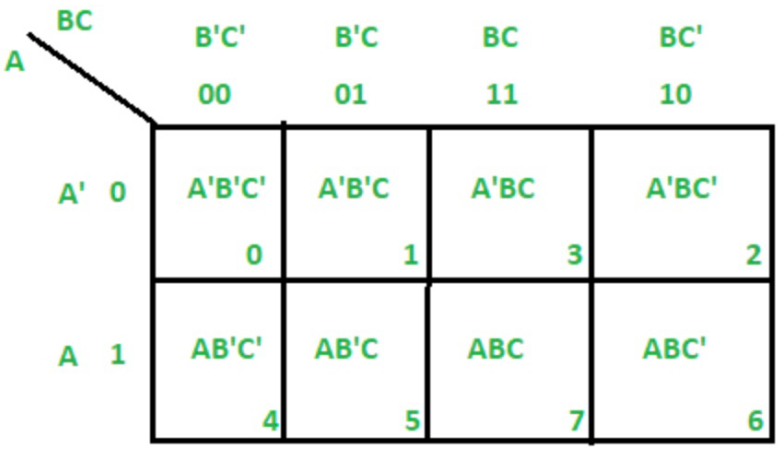
$F(x,y,z) = \sum m(0,2,4,6)$



x is complemented and un complemented (both 0 and 1) along x. And y is also complemented and un-complemented

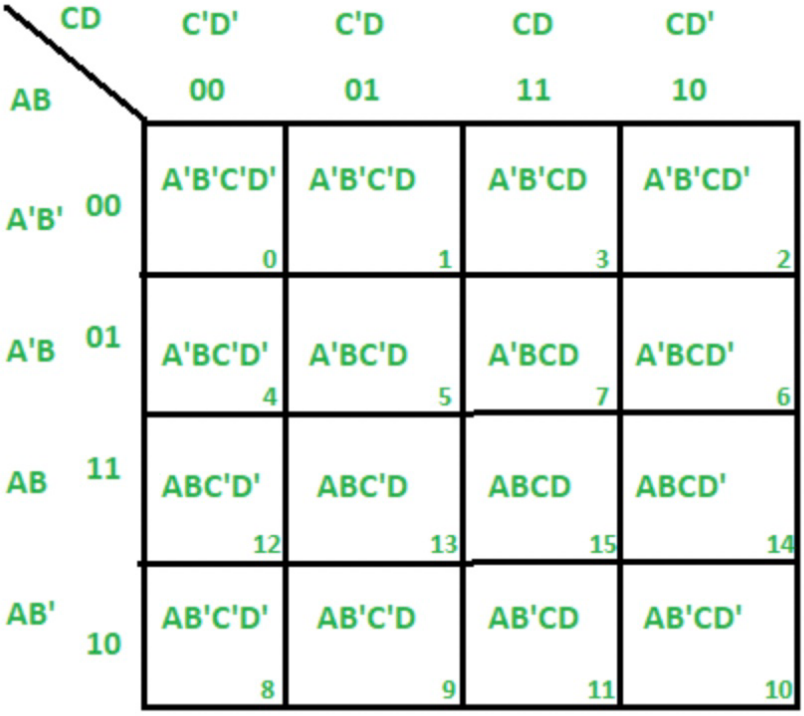
$F(x,y,z) = \overline{z}$

3 variable K-map



SOP(MINTERMS)

4-Variable Map



SOP(MINTERMS)

* If we have a corner of 1's we can combine it into a group of 4.

Implicants: a product term is an implicant if the function has the value 1 for all minterms of the product.

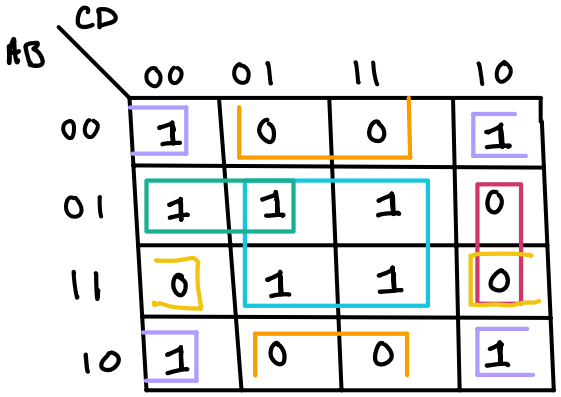
Prime implicants: All the possible large group of combined adj cells in the map.

Essential Prime Implicant: A product term is an essential implicant if there is a minterm that is only covered by the prime implicant.

Conversion between SOP and POS using K-Maps:

Example: Express the following: $F(A,B,C,D) = \Pi(1,3,6,9,11,12,14)$

*convert to minterm: $\sum(0,2,4,5,7,8,10,13,15)$



① Put 1's where the minterms are

* SOP: will deal with the ones divide the k-map into it's essential primes and see what terms cancel out.

Purple: $(B'D')$; Blue: (BD) ; Green: $(A'BC') \Rightarrow B'D' + BD + A'BC'$

② Fill in the rest with zeros:

③ Group the zero's for their essential prime:

④. Orange we are left with the terms B' and $D \Rightarrow$ not both of these
$\Rightarrow$ we get B and $D' \Rightarrow$ and then add these $\Rightarrow (B + D')$
Yellow $\Rightarrow ABD' \Rightarrow (A' + B' + D)$
Red $\Rightarrow BCD' \Rightarrow (B' + C' + D)$
$\Rightarrow (B + D')(A' + B' + D)(B' + C' + D)$

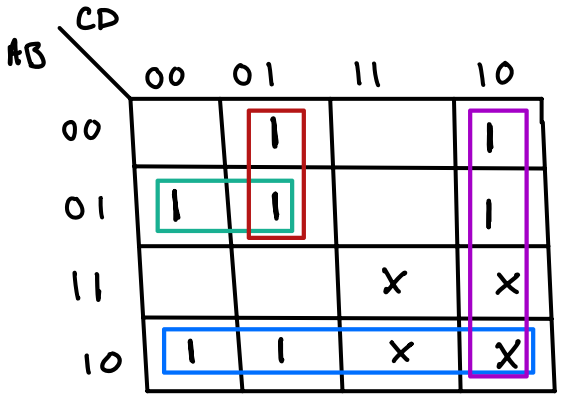
SOP $\Rightarrow B'D' + BD + A'BC'$

POS $\Rightarrow (B + D')(A' + B' + D)(B' + C' + D)$

Don't Care Condition:

Suppose we don't care about the output for certain functions at a certain value. This allows us to simplify a boolean expression further. These are denoted by x on a K-map and can be combined into our grouping of ones.

Ex: $f(A,B,C,D) = \sum m(1,2,4,5,6,8,9)$
$d(A,B,C,D) = \sum m(10,11,14,15) \Rightarrow$ don't cares



Purple: CD'
Blue: AB'
Red: $A'C'D$
Green: $A'BC'$

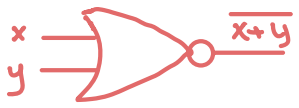
$A\bar{B} + C\bar{D} + \bar{A}\bar{C}D + \bar{A}B\bar{C}$

Universal Gates:

A gate in which we can implement any boolean function using only that single gate: two of these gates are NAND and NOR

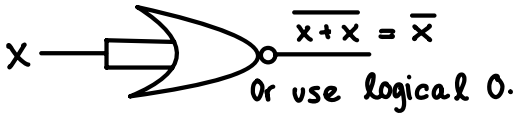
NAND $\Rightarrow \overline{(x \cdot y)}$

NOR $\Rightarrow \overline{(x+y)}$

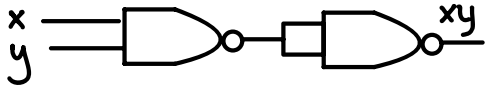


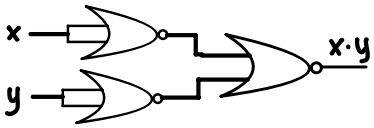
Implement All the Gates Using only NAND and NOR

Not gate:

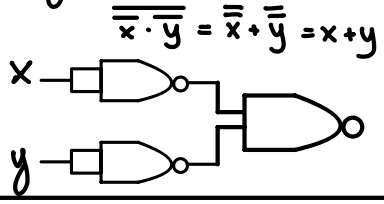


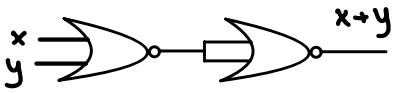
AND gate:





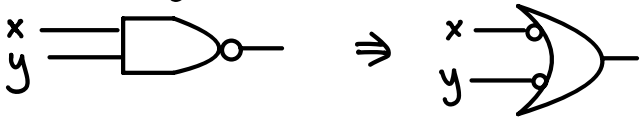
OR gate:



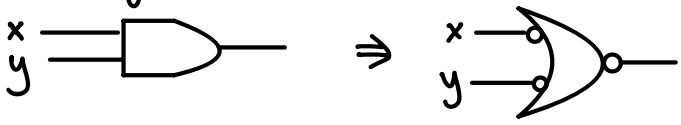


Equivilent Gates:

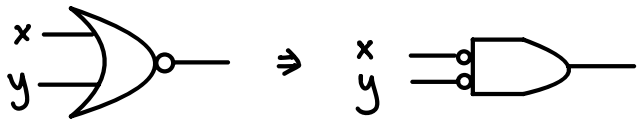
A NAND gate is equivilent to an inverted input or gate



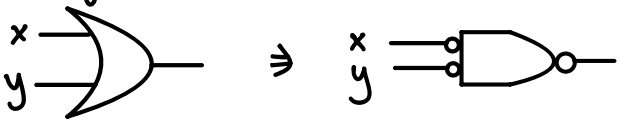
A AND gate is equivilent to an inverted input NOR gate



A NOR gate is equivilent to an inverted input and gate



A OR gate is equivilent to an inverted input NAND gate



Two level implementation:

* SOP: → AND gates will be in the first level and a single or gate will be in the second level.

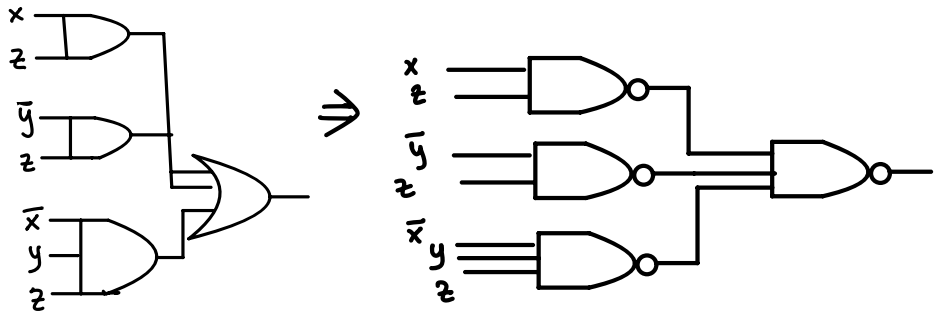
  POS: → OR gates will be in the first level and a single and gate in the second level.

* Using Inverters to Complement is not considered an inverter.

SOP ⇉ implemented using NAND gates
POS ⇉ implemented using only NOR gates.

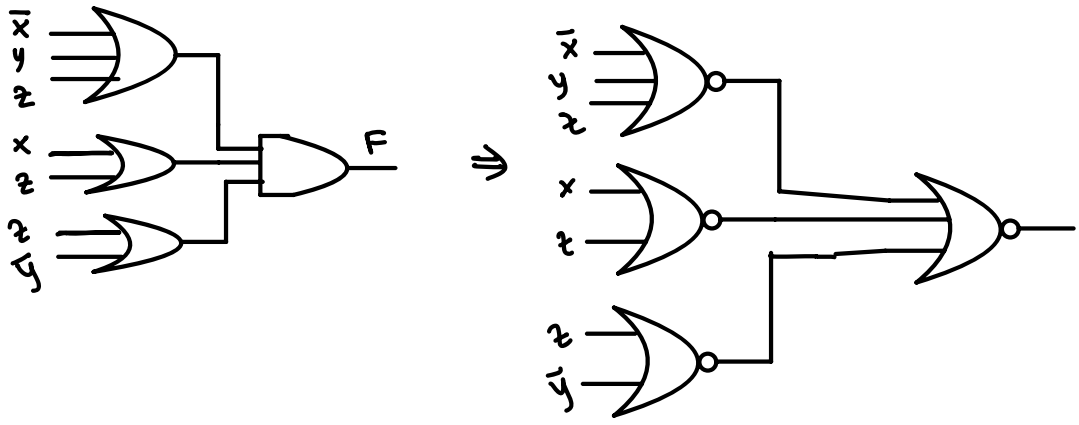
When we have a SOP form we can convert each of the gates to a nand gate. Rember the anology of pushing bubbles

Ex: $F = xz + z\bar{y} + \bar{x}y\bar{z}$



When we have a POS notation we convert each gate to a NOR Gate;

Ex: $F = (x+z)(\bar{y}+z)(\bar{x}+y+z)$



Properties of XOR:

1). $\overline{x} \oplus y = \overline{x \oplus y}$

2). $x \oplus y = \overline{x} \oplus \overline{y}$

3). $x \oplus y = y \oplus x$

4). $x \oplus 0 = x$

$x \oplus 1 = \overline{x}$

$x \oplus x = 0$

5). $\underbrace{x \oplus x \oplus x \oplus x \ldots \oplus x}_{k \text{ times}} = \begin{cases} 0 \text{ if } k \text{ is even} \\ x, \text{ if } k \text{ is odd} \end{cases}$

6). if $x \oplus y = 0$ then $x = y$

7). if $x \oplus y = z \oplus y$ then $x = z$

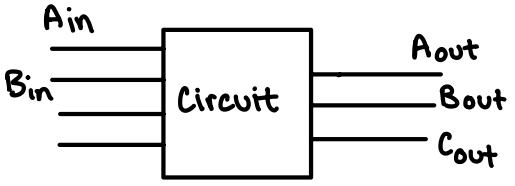
Combinational Circuits:

Combinational circuits don't use any memory. The previous state of input does not have any effect on the present state of the circuit.

The output of a combinational circuit at any instant of time, depends only on the current input
It will have n-number inputs and m number of outputs.

Block diagram ⇒ what is inside the block is hidden, we don't bother about it, think of it as "abstraction" or "obfuscation"

ALU ⇒ arithmetic logic unit, is part of the cpu and performs arithmetic and bitwise operations



⇒ example of a block diagrm

Half-Adder :

Add two single 1 bit values

Ex: $0+0=0$; $0+1=1$; $1+1=10$; $1+0=1$

Two outputs

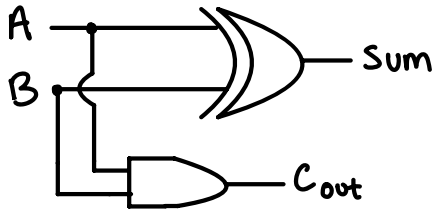
A	B	Sum	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

→ $Sum_{(a,b)} = A'B + AB'$ ⇒ using minterms

$Cout_{(a,b)} = AB$ ⇒ using minterms

The half adder circuit would look like the following:

* $A'B + AB' = A \oplus B$



Full Adder :

add any number bits.

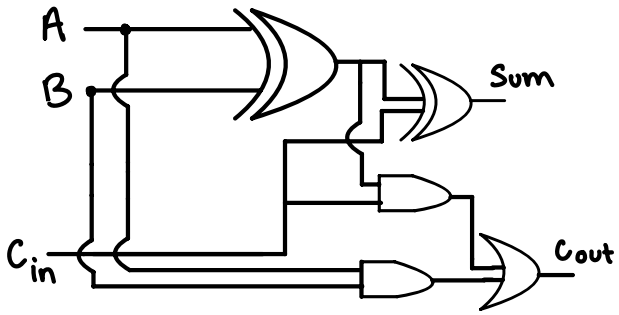
A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$Sum = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$

$Sum = C_{in} \oplus A \oplus B$

$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$

$C_{out} = C_{in}(A \oplus B) + AB$



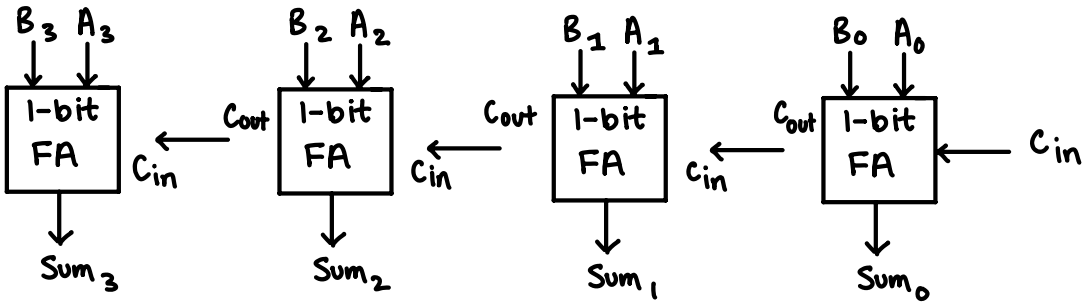
HA + HA = FA

two half adders combined give a full adder

The above circuit example is only a 1 bit full adder. So for example if were to attempt the addition of 4 bits we would need to replicates this 4 times:

We will do this using a block diagrm:

Ex: $A = 1010 \Rightarrow A + B = 1111$
$B = 0101$

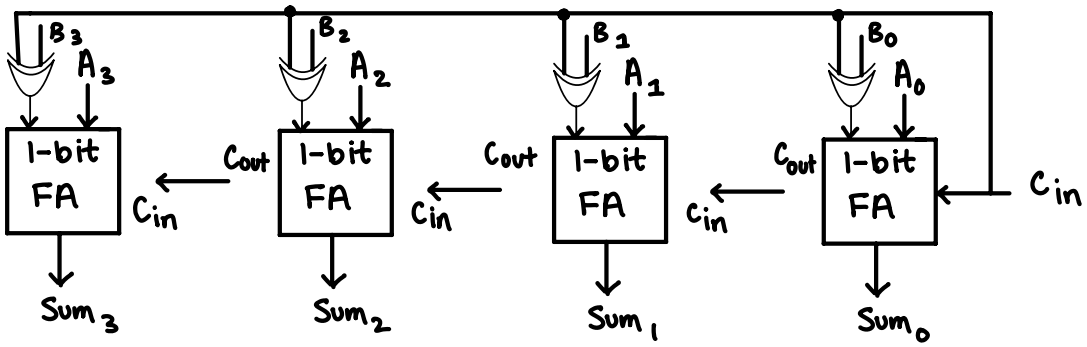


A Full adder can also handle subtraction:

Ex: $A = 1010$ $\quad A - B \Rightarrow A + \overline{B} + 1$
$B = 0101 \Rightarrow \overline{B} = 1010 \Rightarrow \overline{B} = B \oplus 1$

So if we wanted to apply this process to a digital circuit: we can say that c_{in} be a 1 to account for the +1. And we can then xor c_{in} and the b_0 to get the complement

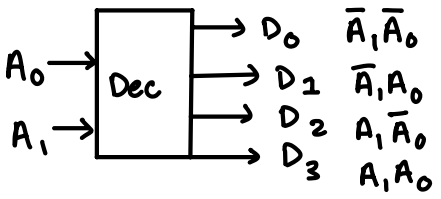
Final Full Adder circuit for both subtraction and addition



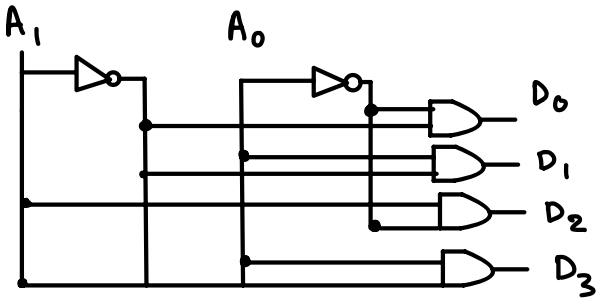
Decoder:

a decoder is a circuit that decodes an input code.
a decoder will have n inputs and 2^{n-1} outputs. ⇒ refered to as a minterm generator

Ex: 2 to 4 decoder:



A_1A_0	D_0	D_1	D_2	D_3
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1



⇒ logic gate construction of a decoder

A decoder can have an enabaling wire that will determine if it on or off. If the enabaling wire is zero then the decoder would be inactive.

Ex: Using a decoder and external gates design:

$F_1 = \bar{x}y\bar{z} + xz$
$F_2 = x\bar{y}\bar{z} + \bar{x}y$
$F_3 = \bar{x}\bar{y}\bar{z} + xy$

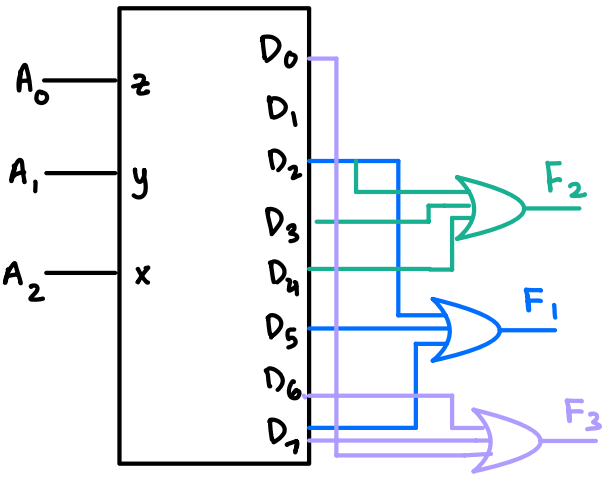
Step 1: Draw the truth table for the functions
Step 2: For wherever there is a 1 label the minterm.
Step 3: Draw the block diagrm for the n input decoder.
Step 4: Combine the individual minterms for each function using an or gate.

* Note how the z relates to A_0 and x relates to A_2

Step 1 & 2:

x y z	f_1	f_2	f_3
0 0 0	0	0	1 m_0
0 0 1	0	0	0
0 1 0	1 m_2	1 m_2	0
0 1 1	0	1 m_3	0
1 0 0	0	1 m_4	0
1 0 1	1 m_5	0	0
1 1 0	0	0	1 m_6
1 1 1	1 m_7	0	1 m_7

Steps 3 & 4:

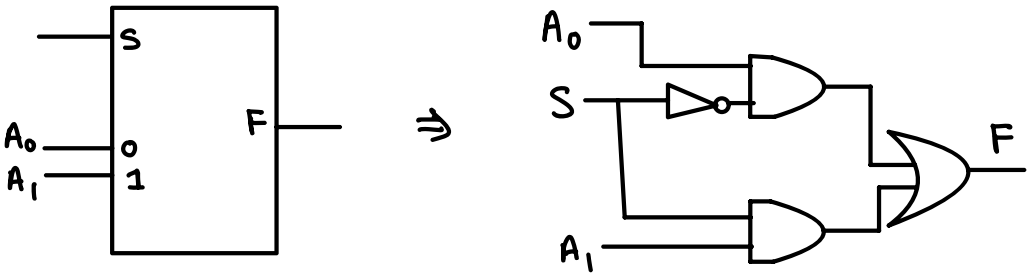


Multiplexer:

A multiplexer is a combinational circuit that selects binary information, from one of many inputs.

The selection of a particular input line is controlled by a set of selection lines:

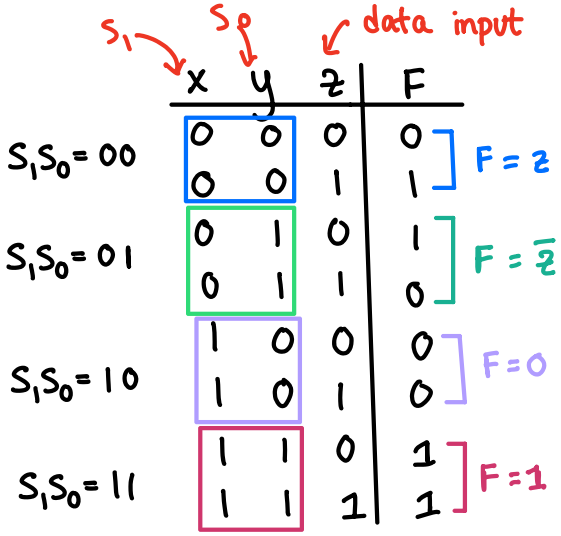
2^n input lines, n selection lines ; where s is the selector line

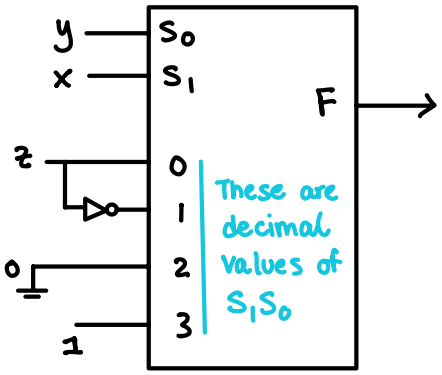


The above example was a 2x1 selector. What would a 4x1 box look like. We would see that there are 4 inputs ⇒ 2^2 with 2 selection lines since $4 = 2^2$.

Ex: use a mux to design the boolean function f:
$F(x,y,z) = \sum(1,2,6,7)$

we have three variables so $n=3$ and we will have $n-1$ selection lines so: $n-1=2$. Using this we see that 2^{n-1} is $2^2 = 4$. So our box size would be 4×1.





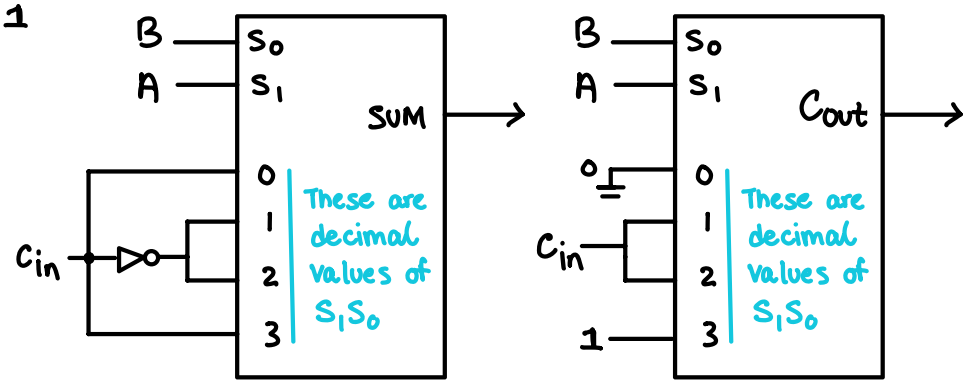
Implement a full adder, using two 4x1 multiplexer.

A	B	C_{in}	Sum	C_{out}
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

this means that we will have two selection lines which are A and B

Blue: C_{in}
Red: $\bar{C}_{in}$
Green: $\bar{c}_{in}$
Purple: C_{in}

Blue: 0
Red: C_{in}
Green: C_{in}
Purple: 1

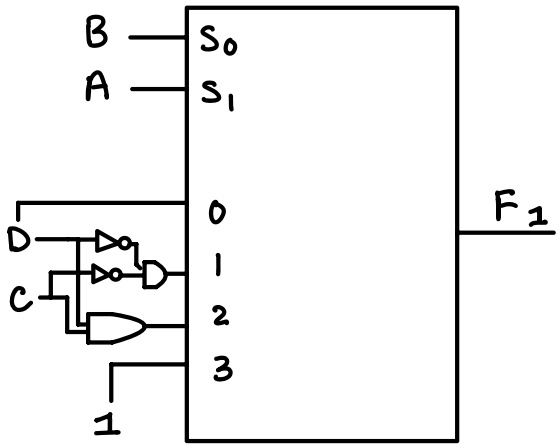


Ex #2 Implement the following boolean function with a a 4x1 multiplexer:

$$F_1(A,B,C,D) = \sum(1,3,4,11,12,13,14,15)$$

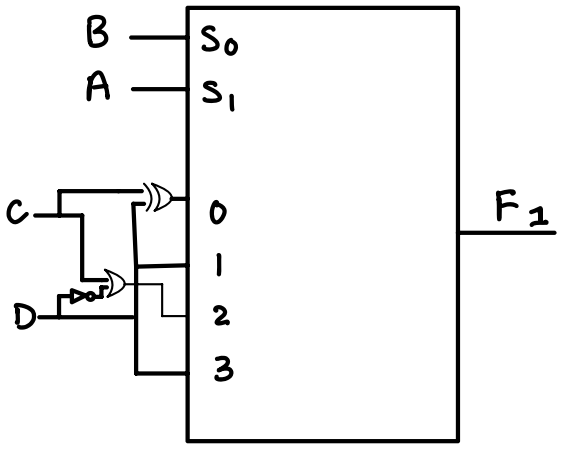
A	B	C	D	F_1	F_2
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	0	1	1	1
1	1	1	0	1	0
1	1	1	1	1	1

Blue: AB = 00; F = D
Purple: AB = 01; F = $\bar{C}\bar{D}$
Green: AB = 10; F = CD
Orange: AB = 11; F = 1



What about if $F_2 = \sum(1,2,5,7,8,10,11,13,15)$:

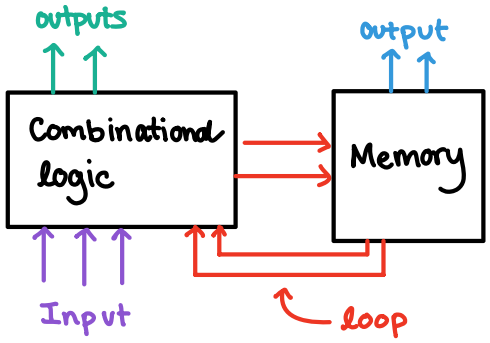
Blue: AB = 00; $F_2 = \bar{C}D + \bar{D}C = C \oplus D$
Purple: AB = 01; $F_2 = D$
Green: AB = 10; $F_2 = C + D'$
Orange: AB = 11; $F_2 = D$



Sequential logic circuits:

Unlike combinational circuits, the output not only depends on current inputs but also on the past sequence of inputs.

Constructed using Combinational logic and a number of memory elements with some or all of the memory outputs fed back into the combinational logic forming a feedback path or loop.

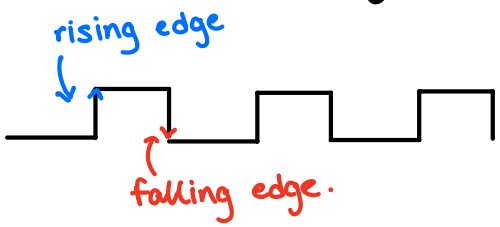


State transition: A change in the stored values in memory elements ⇒ changing the sequential ckt from one state to another state.

Suppose we have a 1 bit memory element Q. It can either hold a one or zero.

- Set : Q = 1
- Reset : Q = 0
- Flip: Q = 0 → 1 or Q = 1 → 0
- Hold value : don't change

Clock:



The time interval between two rising edges is known a the clock period c. and clock frequency is 1/c.

Sychronous Sequential Circuit:

Circuits that have a clock signal as one of the inputs

Latches and flip-flops are the basic single bit memory elements used to build a sequential circuit.

Latches: the change of state of a latch happens whenever the input changes

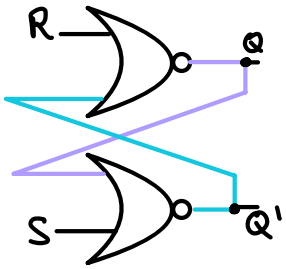
Flip-Flop: the change of state occurs based on the clock input.

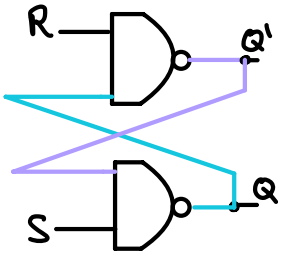
Latches:

Set-Reset (SR) Latch:

This is built using NOR gates or NAND gates

NOR gates use active high, NAND gates use active low.





R: reset and S: set.

Only one input can be active at a time to avoid undefined behaviour.

State table:

R	S	Q	Q'	function
0	0	Q	Q'	Storage
0	1	1	0	Set
1	0	0	1	Reset
1	1	0	0	Undefined

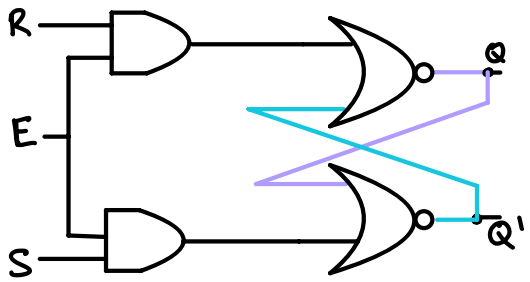
↑ NOR Gate

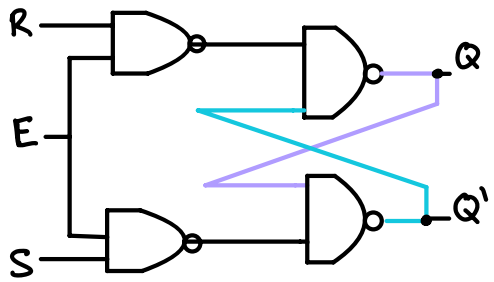
R	S	Q	Q'	function
0	0	1?	1?	undefined
0	1	1	0	Reset
1	0	0	1	Set
1	1	Q	Q'	Storage

↑ NAND Gate

Since we cannot have the same value for q and q', the inputs that result of inputs is not allowed.

Enabled SR Latch (NOR):





State table:

E	R	S	Q	Q'	function
0	x	x	Q	Q'	Storage
1	0	1	1	0	Set
1	1	0	0	1	Reset
1	1	1	0	0	Undefined

NOR Gate

State table:

E	R	S	Q	Q'	function
0	x	x	Q	Q'	Storage
1	0	0	Q	Q'	Storage
1	0	1	1	0	Reset
1	1	0	0	1	Set
1	1	1	0	0	Undefined

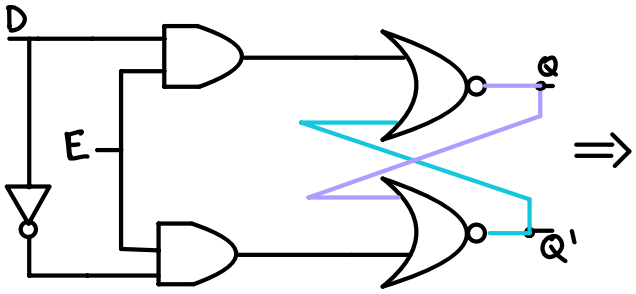
NAND Gates

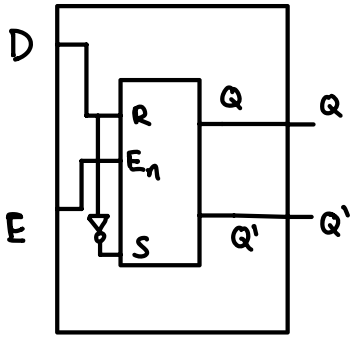
When e is inactive the RS latch will automatically be forced into storage.

D-LATCH: (Data or Delay Latch)

Used to overcome the undefined state.
Active high enable for NOR Latch

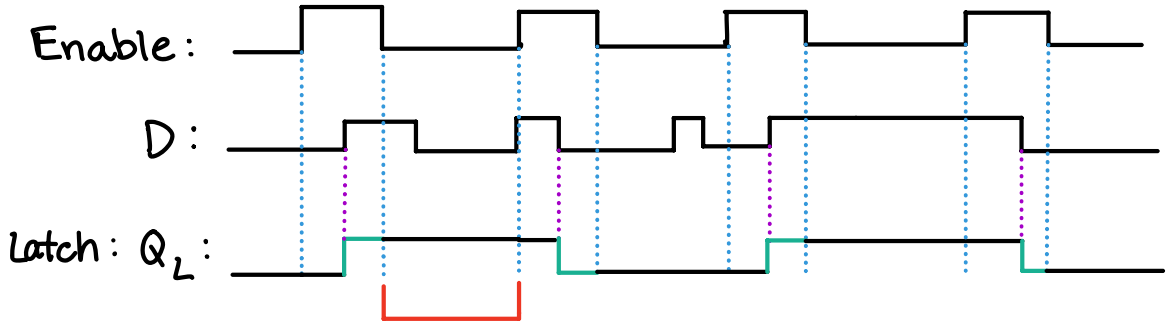
E	D	Q	Q'	Storage
0	x	Q	Q'	Storage
1	0	0	1	transparent
1	1	1	0	transparent





It is referred to as transparent because D determines the value.

Timing Diagrm for D-latch:



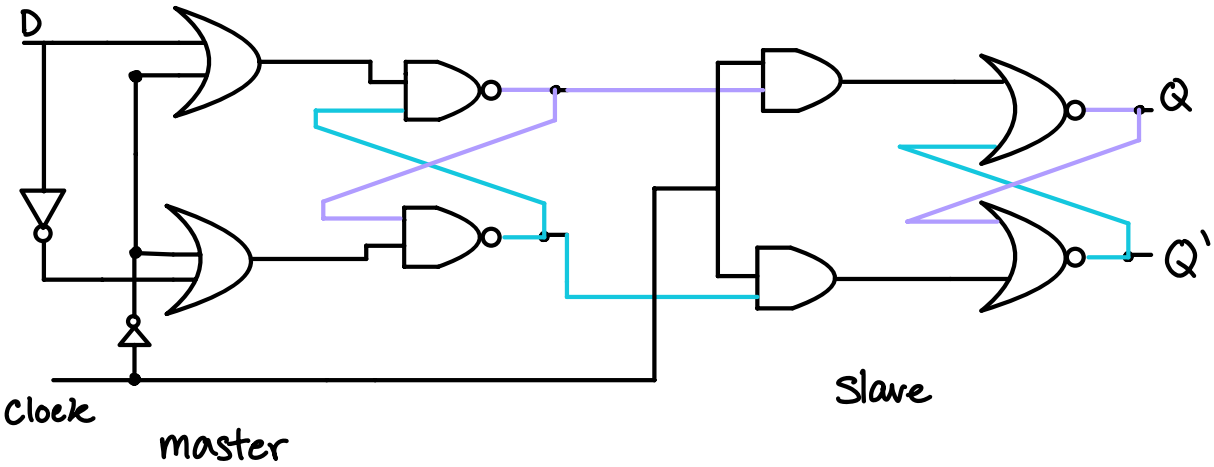
*D will only change when e is enabled otherwise it holds it's value.

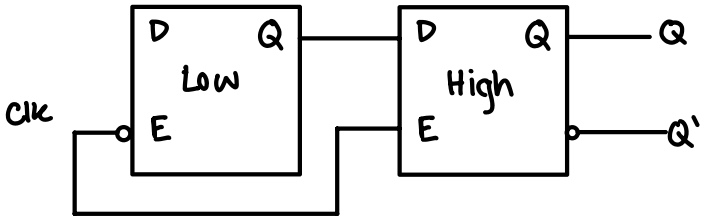
D-flip-flop or Master-Slave flip-flop:

One value is always in storage and other is transparent. It is sensitive to the oppossite levels of the clock. The data moves through on clock transition

Rising edge-triggered: Active low followed by Active high
Falling edge-triggered: Active high followed by Active low

Active low builds based on NAND
Active high builds based on NOR



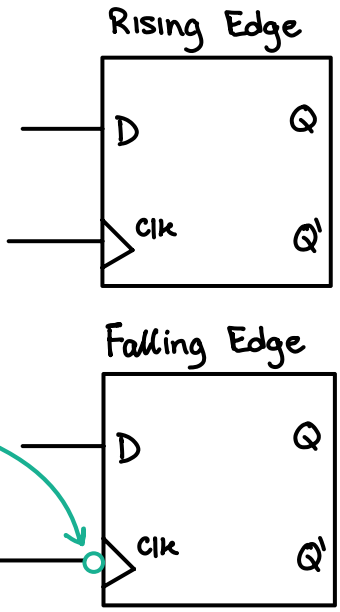


D-Flip Flops continued:

falling_edge = negative_edge
rising_edge = positive_edge

The ▷ symbol represents the clock

The circle represents a falling edge



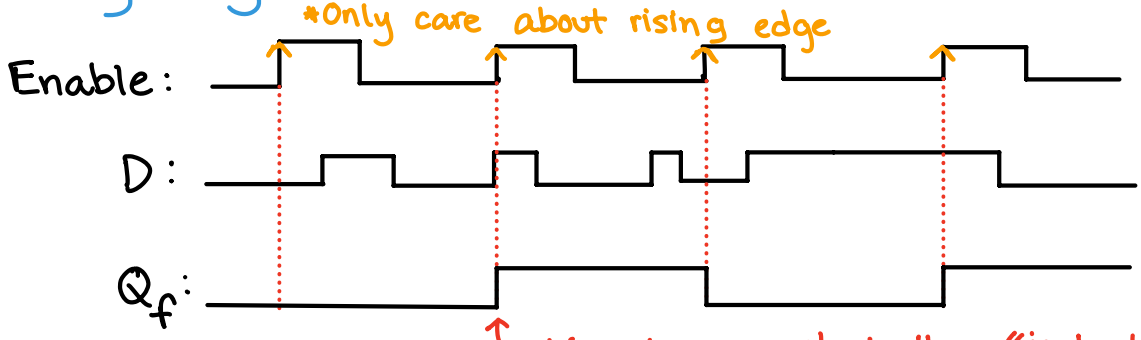
Rising Edge:

D	Clock	Q
0	↑	0
1	↑	1
x	↓	Q ⇒ storage

Falling Edge

D	Clock	Q
0	↓	0
1	↓	1
x	↑	Q ⇒ storage

Timing Diagram for D-flip-flop:



We only care about the "instantaneous" value of d at the rising edge. And that value of Q_f stays the same until the next rising edge.

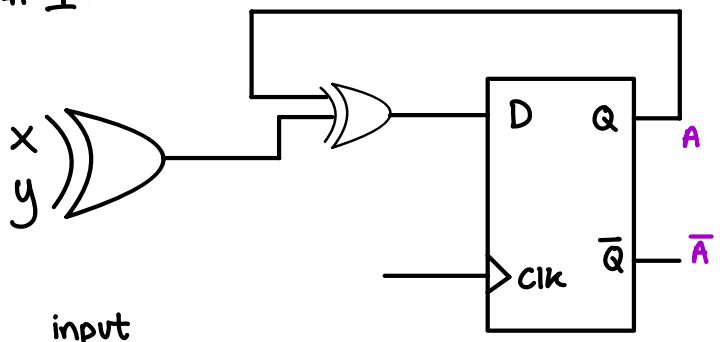
State Table: consists of three sections present state, next state, inputs, and outputs.

Present State: designates the state of a flip flop before the clock pulse:

Next State: shows the states after clock pulse

Output: output variable values during the present state

Ex # 1:



Find the the following for the circuit:

a). State equation
b). state table
c). State diagrm

input

Present State

Next State

A_{t+1} , x, y, A_t

One flip flop gives $2^1 = 2$ states 0 or 1.

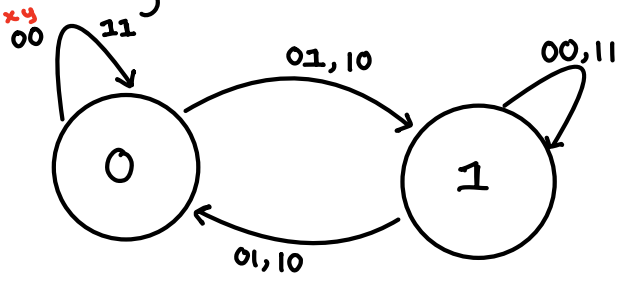
Present State	Inputs		Next-State	Outputs
A_t	x	y	A_{t+1}	
0	0	0	0	
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	
1	0	1	0	
1	1	0	0	
1	1	1	1	

It is important to note that we have a feedback loop so that means A_t is an input:

State Eq:

$A_{t+1} = x \oplus y \oplus A_t$

State Diagrm: there will be a circle for each state.



Ex #2:

For the following state equations, derive the state table, state diagrm show the sequential circuit.

$A_{t+1} = A_t x + B_{(t)} x$

$B_{t+1} = \overline{A}_t x$

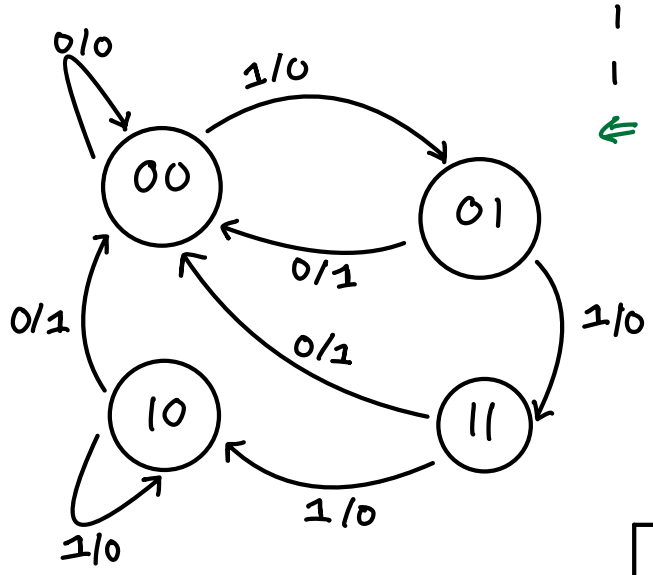
$y = [B_t + A_t]\,\overline{x}$

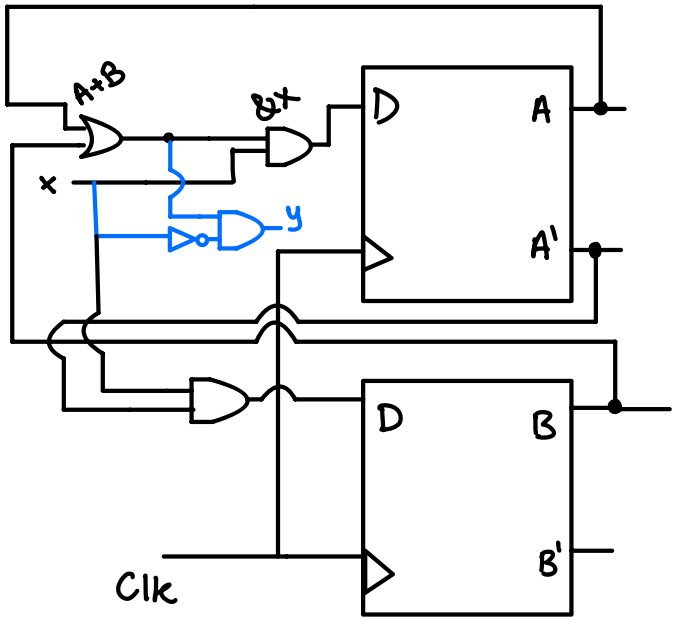
2 Flip Flops

$2^2 = 4$ states

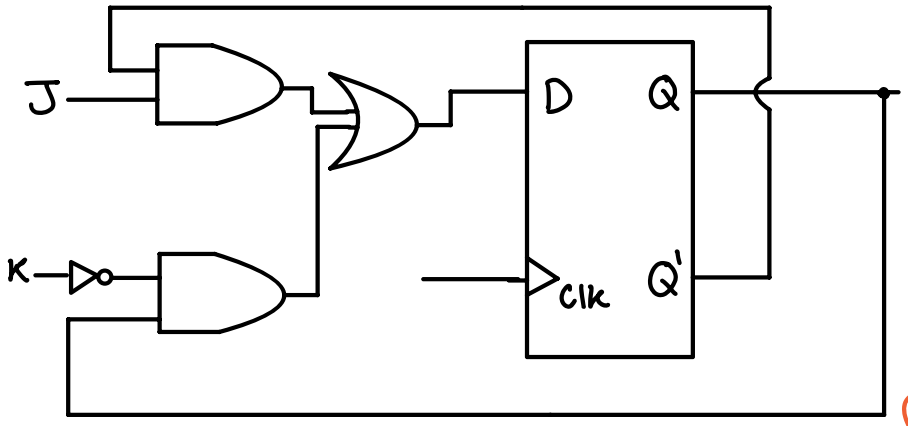
Present State		Inputs	Next-State		Outputs
A_t	B_t	x	A_{t+1}	B_{t+1}	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

⇐ Input / Output ⇒ x/y





JK Flip Flop:

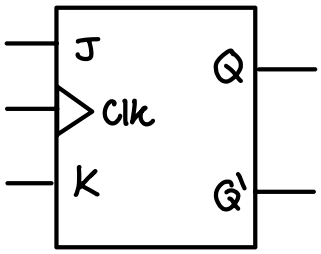


$D = JQ' + K'Q$

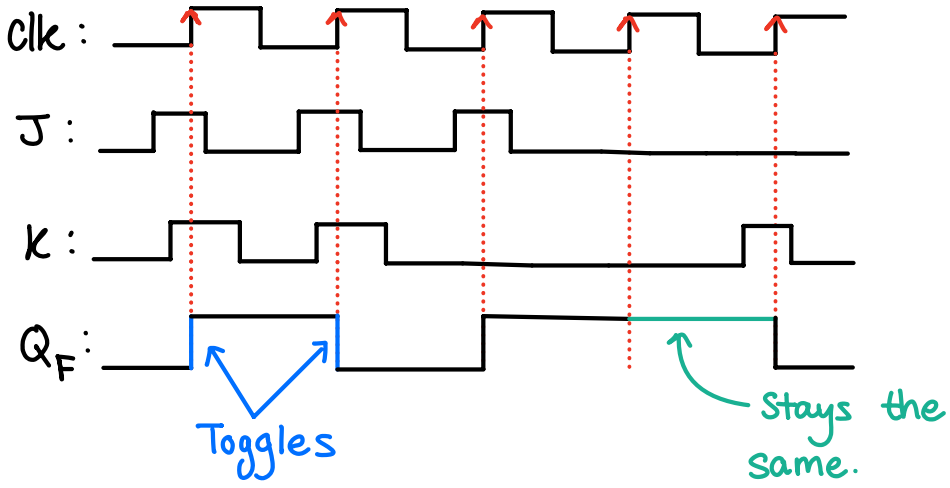
J	K	Clk	Q	Q'
0	0	↑	Q	Q'
0	1	↑	0	1
1	0	↑	1	0
1	1	↑	Q'	Q
x	x	↓	Q	Q'

Also called toggle

Block diagram:



Jk Flip Flop Timing Diagram:



Sequential Circuit for JK Flip Flop:

$K_A = B\bar{x}$; $J_A = B$

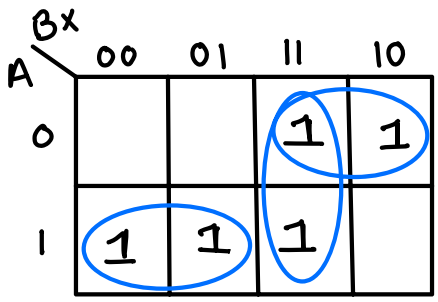
$K_B = A \oplus x$; $J_B = \bar{x}$

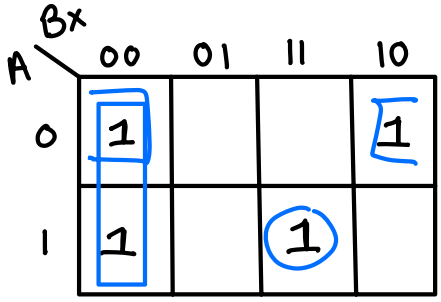
we have two flip flops J_a and J_B with inputs of K_a and K_B respectively

Present State		Input	Next-State		Input Equations			
A_t	B_t	x	A_{t+1}	B_{t+1}	J_A	K_A	J_B	K_B
0	0	0	0	1 m_0	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1 m_2	1 m_2	1	1	1	0
0	1	1	1 m_3	0	1	0	0	1
1	0	0	1 m_4	1 m_4	0	0	1	1
1	0	1	1 m_5	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1 m_7	1 m_7	1	0	0	0

To determine the next state values we use the truth table for the JK Flip Flop

To find the state equations we use a K map





$A_{t+1} = A\bar{B} + BX + \bar{A}X$

$\Rightarrow A \oplus B + AX$

$B_{t+1} = \bar{B}\bar{x} + \bar{A}\bar{x} + ABx$

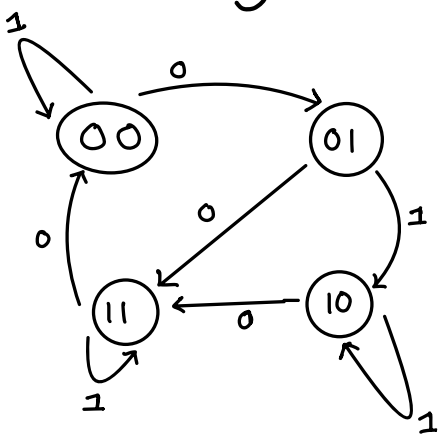
Recall that $D = JQ' + K'Q \Rightarrow A_{t+1} = J_A A' + K_A' A$

$B_{t+1} = J_B B' + K_B B$

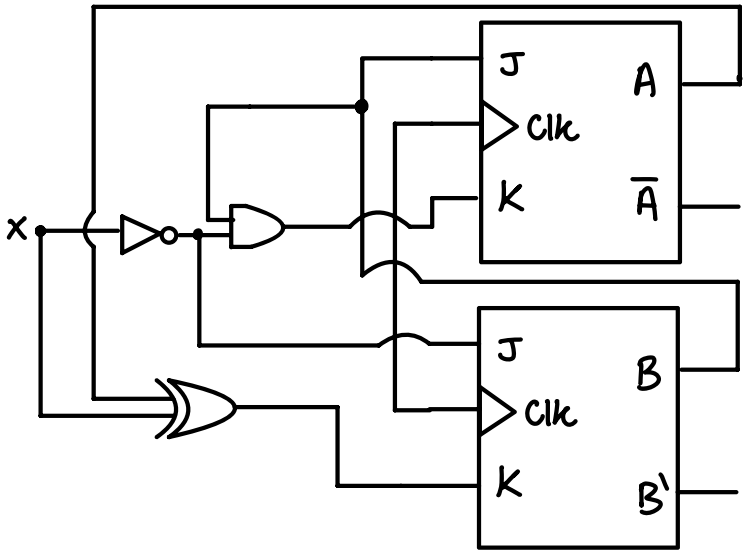
Using substitution we see that we get the same answer: For the state equations.

Ex continued:

State Diagrm:



Circuit Diagrm:

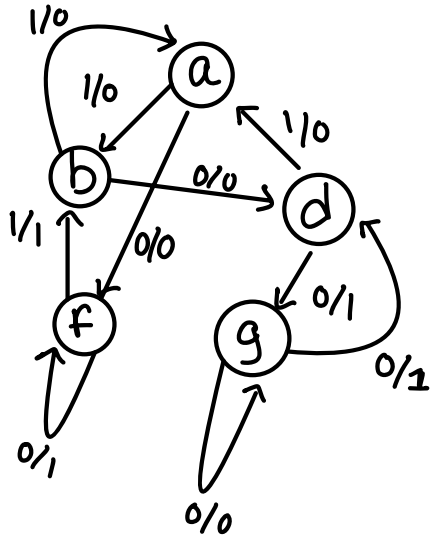


Ex: State Table Reduction:

When a present state has the same next state and outputs we can keep exactly one of those duplicate states. and each of these is replaced

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
a	f	b	0	0
b	d	~~c~~ a	0	0
Remove c	f	~~e~~ b	0	0
d	g	a	1	0
Remove e	d	c	0	0
f	f	b	1	1
g	g	~~h~~ d	0	1
Remove h	g	a	1	0

h = d
e = b
c = a



Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

JK Flip-Flop More Complex Example:

$J_A = Bx + B'y'$ $\quad K_A = B'xy'$

$J_B = A'x$ $\quad K_B = A + xy'$

$z = Ax'y' + Bx'y'$

Present State		Inputs		Next State		Input Equ				Output
A_t	B_t	x	y	A_{t+1}	B_{t+1}	J_A	K_A	J_B	K_B	z
0	0	0	0	1	0	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	1	1	1	1	1	0
0	0	1	1	0	1	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	1
0	1	0	1	0	0	0	0	0	0	0
0	1	1	0	1	1	1	0	1	1	0
0	1	1	1	1	1	1	0	1	0	0
1	0	0	0	1	0	1	0	0	1	1
1	0	0	1	1	0	0	0	0	1	0
1	0	1	0	0	0	1	1	0	1	0
1	0	1	1	1	0	0	0	0	1	0
1	1	0	0	1	0	0	0	0	1	1
1	1	0	1	1	0	0	0	0	1	0
1	1	1	0	1	0	1	0	0	1	0
1	1	1	1	1	0	1	0	0	1	0

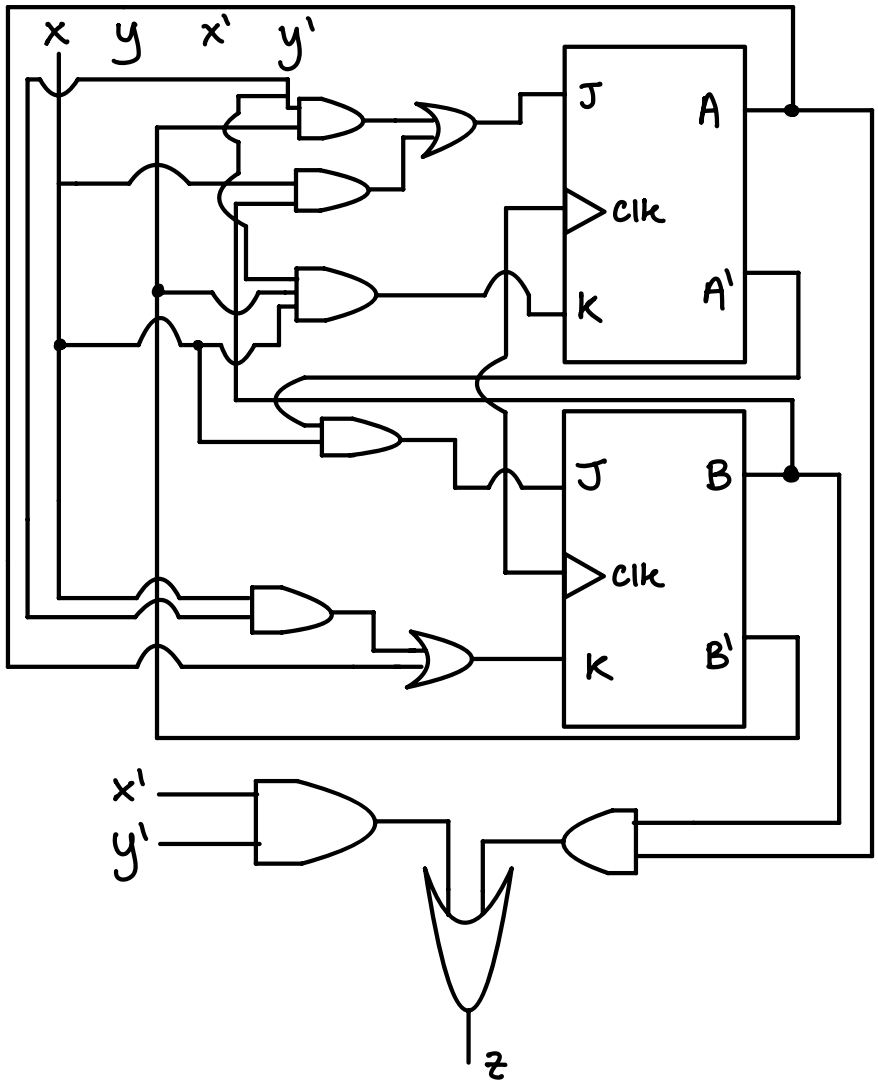
State Equations for A & B

$$A_{t+1} = (Bx + B'y')(A') + (B'xy')'(A)$$

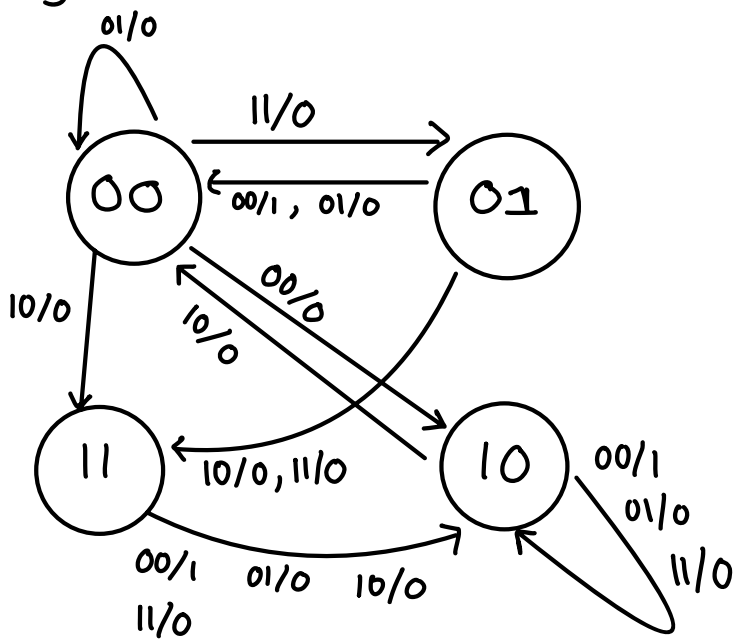
$$B_{t+1} = (A'x)(B)' + (A + xy')'(B)$$

Ex Continued:

Circuit Diagrm:



State Diagrm:

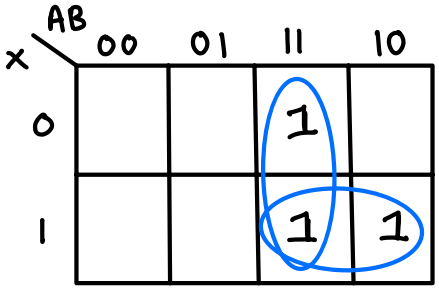


Example Using D-Flip-Flops:

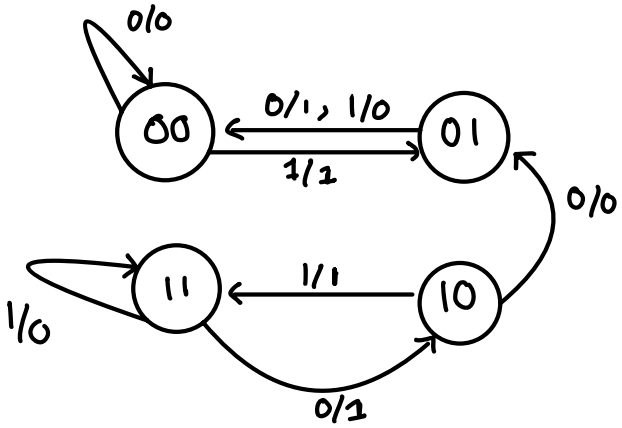
x	A	B	A(t+1)	B(t+1)	z
0	0	0	0	0	0
0	0	1	0	0	1 m_1
0	1	0	0	1 m_2	0
0	1	1	1 m_3	0	1 m_3
1	0	0	0	1 m_4	1 m_4
1	0	1	0	0	0
1	1	0	1 m_6	1 m_6	1 m_6
1	1	1	1 m_7	1 m_7	0

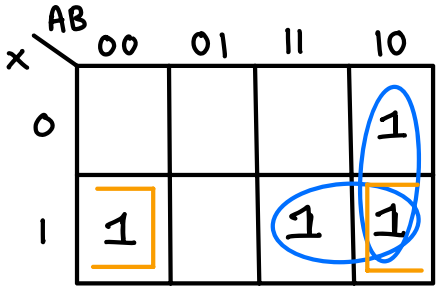
2 states so two d-flip-flops

State Equations:

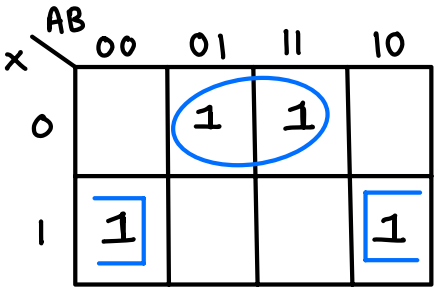


$A_{t+1} = AB + xA$



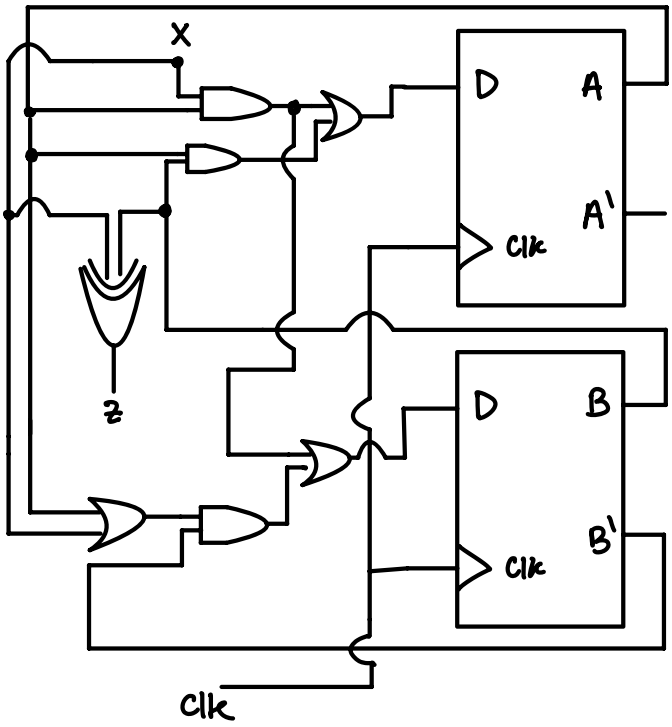


$B_{t+1} = x\bar{B} + xA + A\bar{B}$

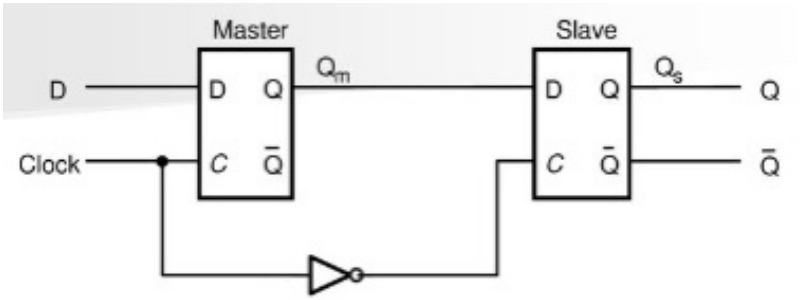


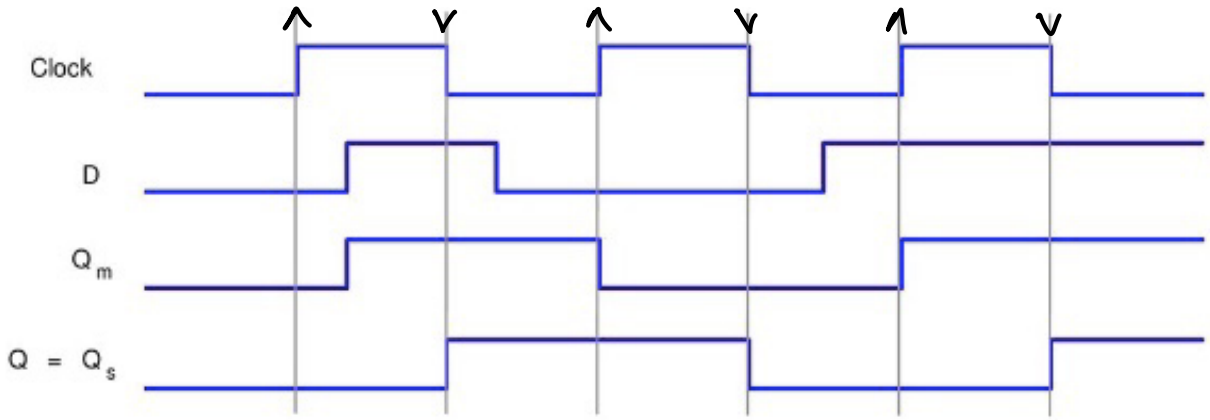
$z = \bar{x}B + x\bar{B} = x \oplus B$

Ex continued:



Master-Slave Timing Diagram: (Rising Edge)





Finite State Machine (FSM)

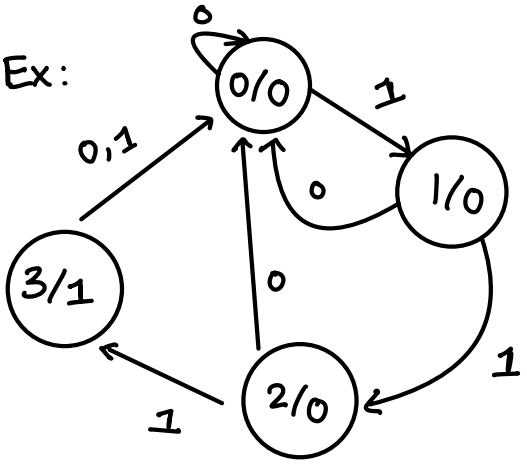
Set of inputs, outputs, states and transitions.
State graph, defines input/output relationship.
It has inputs (sensors), outputs, controllers, and state graph.

Meely Machine: are any of the above state diagrams. while the output depends on present state and input.

Moore FSM:

The output of a moore machine depends only on the current state; output written on inside:

Ex:

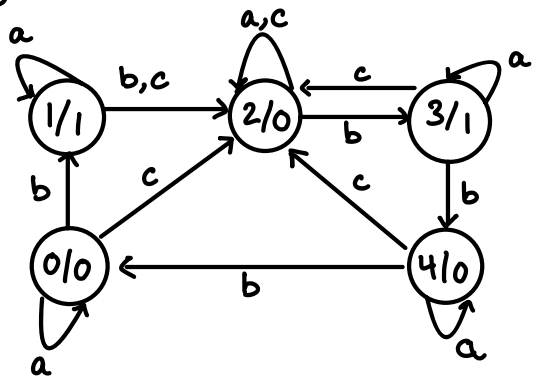


A_t	B_t	x	A_{t+1}	B_{t+1}	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	0	0	1

Example: We have 5 states going from 0, 1, 2, 3, 4.
Draw the state diagram

$$S(t+1) = \begin{cases} S(t) & \text{if } x = a \\ (S(t)+1) \bmod 5 & \text{if } x = b \\ 2 & \text{if } x = c \end{cases}$$

$$z(t) = \begin{cases} 0 & \text{is } s(t) \text{ is even} \\ 1 & \text{is } s(t) \text{ odd.} \end{cases}$$



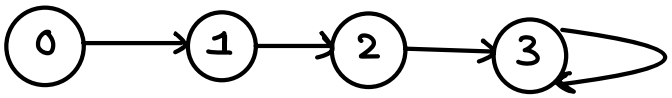
Registers are very fast, expensive, small memory cells:

Ex: Intel x86-64 has 16 registers each with 64 bits For each bit we would use 1 flip-flop. So to build 1 register we need 64 bits. So to build 16 register we get $2^4 \times 2^6 = 2^{10}$ flip-flops.

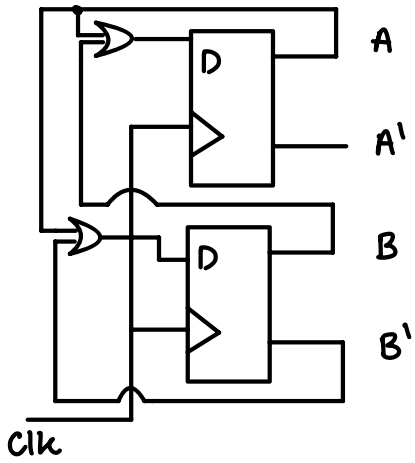
Counter:

a type of sequential circuit that repeats a sequence

Ex: will count from 0 to N sequence is 0-1-2-3-3-3



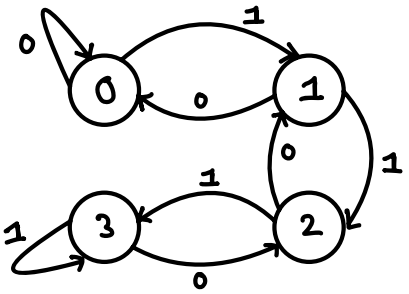
PS AB	NS A_{+1} B_{+1}
0 0	0 1
0 1	1 0
1 0	1 1
1 1	1 1



$A_{t+1} = A + B$; $B_{t+1} = \overline{B} + A$

Ex: Simple up/down counter, has input to indicate if counting up or down. up = 1 count from 0 to N and saturate at N. if up=0 will count down from N to 0 and saturates at zero. Ex: 3-2-1-0-0-0

0-1-2-3-3-3



A	B	Up	A_{+1}	B_{+1}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1

⇒

AB	up	A_{+1}	B_{+1}
1 1	0	1	0
1 1	1	1	1