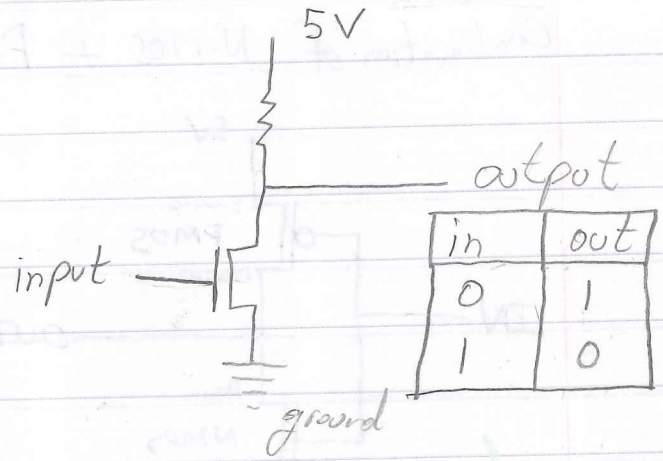
Inverters

N-Type Transistors

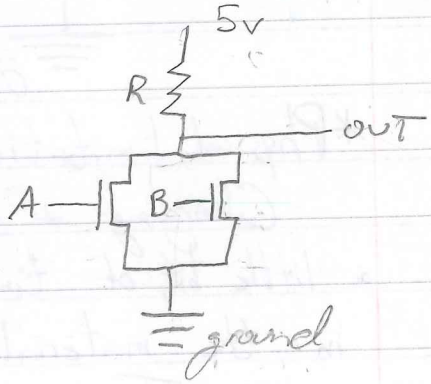


in	out
0	1
1	0

NOR Gate

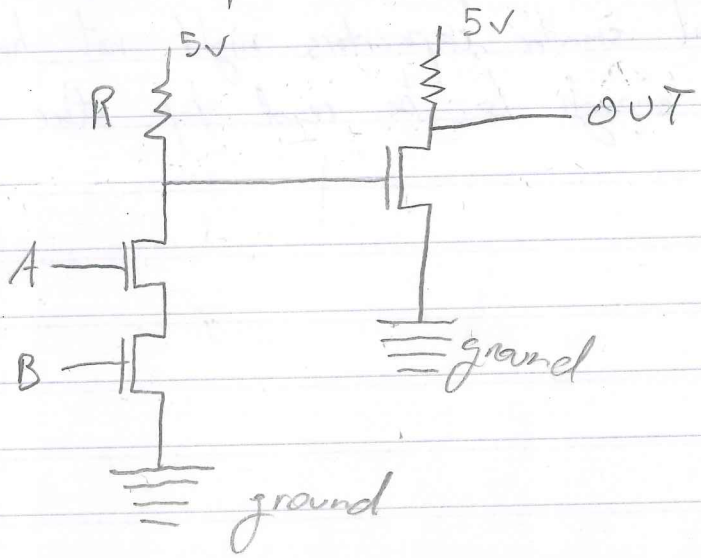
N-Type Transistors in Parallel

A	B	OUT
0	0	1
0	1	0
1	0	0
1	1	0



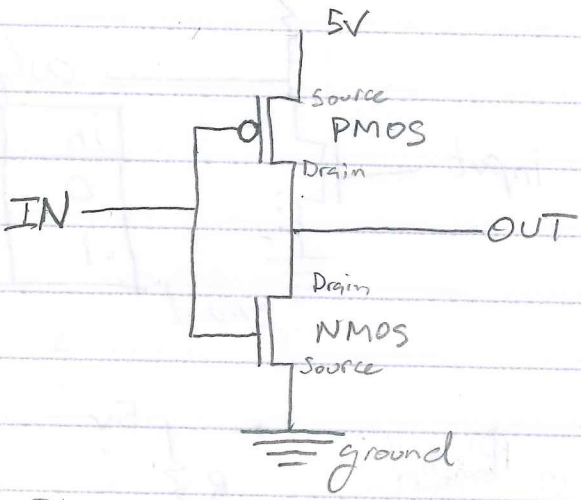
NAND Gate

N-Type in series plus inverter at OUT



CMOS

Combination of N-MOS & P-MOS (replaces resistance)



0V input will create a channel at PMOS gate allowing current to run from 5v source to OUT

*Physical limitations to speed:

Charging & discharging of capacitors require a little bit of time. This time increases heat in the materials due to the flow of current. To minimize heat dissipation we need to make capacitors smaller. The trade-off is that smaller capacitors might not hold charge long enough to be read by other components

Boolean Algebra

Basic operations: AND, OR, NOT

Switching Circuit:  open: 0 – off, Closed: 1 – ON

Laws of Boolean Algebra:

$X+0=X$ $\quad X+X'=1$

$X+1=1$ $\quad X+Y=Y+X$

$X+X=X$ $\quad (X+Y)+Z=X+(Y+Z)$

$X''=X$ $\quad X(Y+Z)=XY+XZ$

DeMorgan's Law:

$$(X+Y)'=X'\cdot Y'$$

Dual:

Interchange 0 & 1, AND & OR

No change in variables or complements allowed

Ex:

$X+0=X \rightarrow X\cdot 1=X$

$X+1=1 \rightarrow X\cdot 0=0$

$X+X=X \rightarrow X\cdot X=X$

$X+X'=1 \rightarrow X\cdot X'=0$

$X(Y+Z)=XY+XZ \rightarrow X+(YZ)=(X+Y)\cdot(X+Z)$

$(X+Y)'=X'\cdot Y' \rightarrow (X\cdot Y)'=X'+Y'$

Uniting Theorem:

$XY + XY' = X \longrightarrow (X+Y)\cdot(X+Y') = X$ (Dual)

Absorbtion Law:

$X + (XY) = X \longrightarrow X\cdot(X+Y) = X$

Elimination:

$X + (X'Y) = X + Y$

expanded $\longrightarrow (X+X')(X+Y) \longrightarrow X\cdot X + XY + X'X + X'Y$

$= X + \underbrace{XY + 0 + X'Y}_{Y} \longrightarrow \boxed{X + Y}$

Duality: $\xrightarrow{D}$ Dual

$(X + Y + \ldots + Z)^D \Longrightarrow (X \cdot Y \cdot \ldots \cdot Z)$

Consensus:

$XY + YZ + X'Z = XY + X'Z$

Simplification Examples:

$[(A'+B)C']'$ De Morgans

$(A'+B)'' + C''$ De Morgans

$A\cdot B' + C$

$[(AB'+C)D' + E]'$ (OR The whole expression on left) De Morgans

$[(AB'+C)' + D]E'$ De Morgans

$[(\overline{AB'}\cdot C') + D]\cdot E'$ De Morgans

$\{[(A'+B)\cdot C'] + D\}\cdot E'$ Distribute

$(A'C' + BC' + D)E'$ Distribute

$A'C'E' + BC'E' + DE'$

Proving the Validity of an equation

$f(x,y,z) = g(x,y,z)$

Ex:

$(xy' + x'y) \stackrel{?}{=} \overline{(xy + x'y')}$ True

Methods of Proving

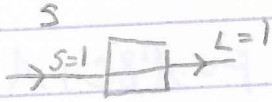
- Boolean algebra & show equivalence
- Construct a truth table & show equivalence

English → Boolean Expression

Do laundry on Saturdays

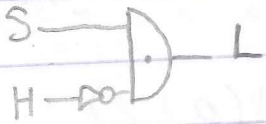
(if it is Saturday then do laundry)

S L

circuit 

Change: Unless it is a USF home game

$L = S \text{ AND } \overline{H}$ $H = 1$



Change: Unless USF Game or Exam on Monday
$E = 1$

$L = S \text{ AND } (\overline{H} \text{ OR } \overline{E})$

$L = S(\overline{H} + \overline{E})$ X No construct truth table to see. Need AND

$L = S(\overline{H} \cdot \overline{E})$

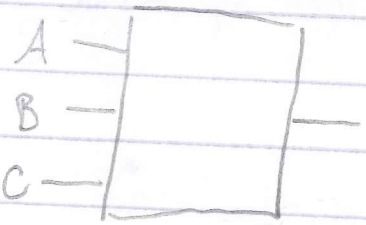


figure 4.1

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

A

SOP

$F = A'BC + \underbrace{AB'C' + AB'C + ABC' + ABC}_{A}$

$F = A'BC + A$

$\sum m(3,4,5,6,7)$

where m = minterm

POS

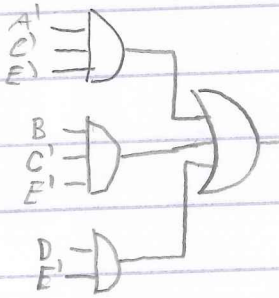
$F = (A+B+C)(A+B+\overline{C})(A+\overline{B}+C) \cdot O$
$\quad\quad 0\ 0\ 0 \quad\quad 0\ 0\ 1 \quad\quad 0\ 1\ 0$

$F = \prod M(0,1,2)$

Input variables & its complements are readily available. No inverter is needed. D-flipflops contain both. Ex: $A, A', B, B' \ldots$

$A'C'E' + BC'E' + DE'$ is a 3 literal term

Literal: Variable or its compliment in an expression



output = $A'C'E' + BE' + DE'$

Example:

$A'B + A'B'C'D' + ABCD'$ Simplify for less literals

$A'(B + B'C'D') + ABCD'$ Take out A'

$A'(B + C'D') + ABCD'$ Elimination

$B(A' + CD') + A'C'D'$ from $\underline{A'B} + A'C'D' + \underline{ABCD'}$

$A'B + BCD' + A'C'D'$

AND → OR Structures

Also known as Programmable Arrays.

Simplifying Expressions Cont.

$F = AB' + CD'E + AC'E'$

Sum → Sum of Products (SoP)

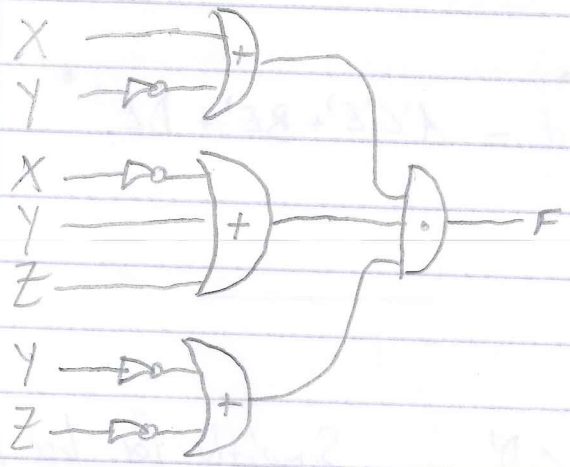
Products

↳ Also Known as minterms

$F = (X + \bar{Y})(\bar{X} + Y + Z)(\bar{Y} + \bar{Z})$

Product of Sums (POS)

Maxterms



Note: $\overline{(X \oplus Y)} = (X \equiv Y)$

$(X \oplus Y) = XY' + X'Y$

$(X \equiv Y) = XY + X'Y'$

Missed Notes:

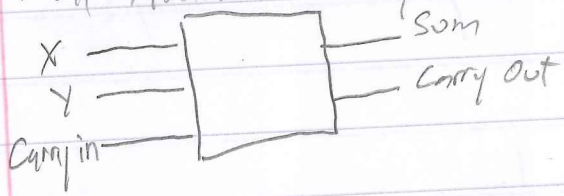
Adders: Half Adder - 2 bits

X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$Sum = X'Y + XY'$

$Carry = XY$

Full Adder: Carry in 1 bit



$Sum = XY + X C_{in} + Y C_{in} = X \oplus Y \oplus C_{in}$

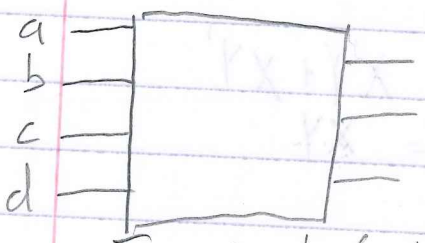
X	Y	C_{in}	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$Sum = X'Y'C_{in} + X'YC_{in}' + XY'C_{in}' + XYC_{in} = \boxed{X \oplus Y \oplus C_{in}}$

$Carry\ out = X'YC_{in} + XY'C_{in} + XYC_{in} + XYC_{in}$

$Carry\ out = \boxed{XY + YC_{in} + C_{in}X}$

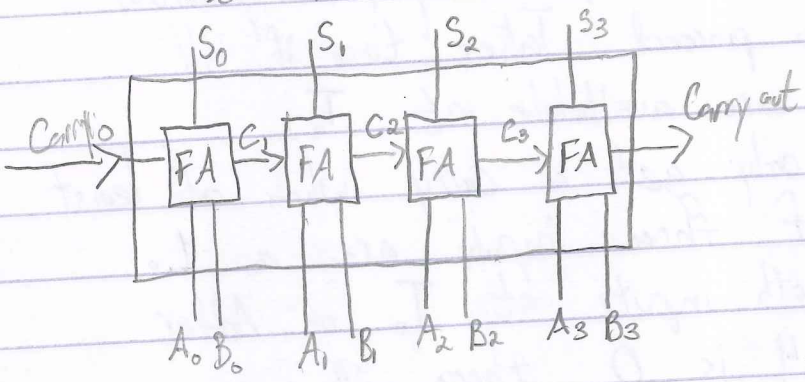
2-bit full Adder:



Two inputs (ab, cd) each one 2-bits

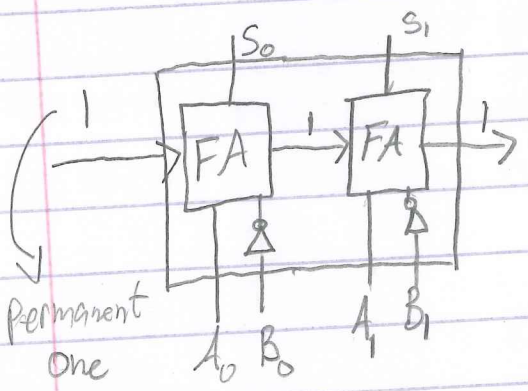
ab	cd	Sum	Carry → 2bit carry?
0	0	0	0
0	0	0	1
0	0	0	1 1
0	0	0	0
0	1	1	0
0	1	1	1
0	1	1	1
0	1	1	0 1
1	0	0	1 0
1	0	0	1 1
1	0	1	1 1
1	0	1	1 0 1
1	1	0	0 0 1
1	1	0	0 1 1
1	1	1	0 1 1
1	1	1	0 0 ?

Adding a 4-bit number.
Use 4 1-bit adders



Subtract:
Take 2's Comp of the number to be subtracted & add both numbers (no carry)

6 → 0110
-3 → 1101 (2's Comp)
3 0011



Subtractor
permanent one & inverter for 2's Comp

Carry Look Ahead (CLA) hardware:
Full Adders wait for carry from previous input to proceed. Takes time if all inputs are available at T_0.
You only get a carry when at least Two of three inputs are a 1.
if both inputs at T_0 on Adder number 4 is 0, there will never be a carry & that calculation can proceed

Book Uses
G: Generate } Carry
P: Propagate }

Pg 106

Value	Code 6 3 1 1 (A B C D)	Error?
0	0 0 0 0	0
1	0 0 0 1	0
X	0 0 1 0	1
2	0 0 1 1	0
3	0 1 0 0	0
4	0 1 0 1	0
X	0 1 1 0	1
	0 1 1 1	0
	1 0 0 0	0
	1 0 0 1	0
X	1 0 1 0	1
	1 0 1 1	0
	1 1 0 0	0
X	1 1 0 1	1
X	1 1 1 0	1
X	1 1 1 1	1

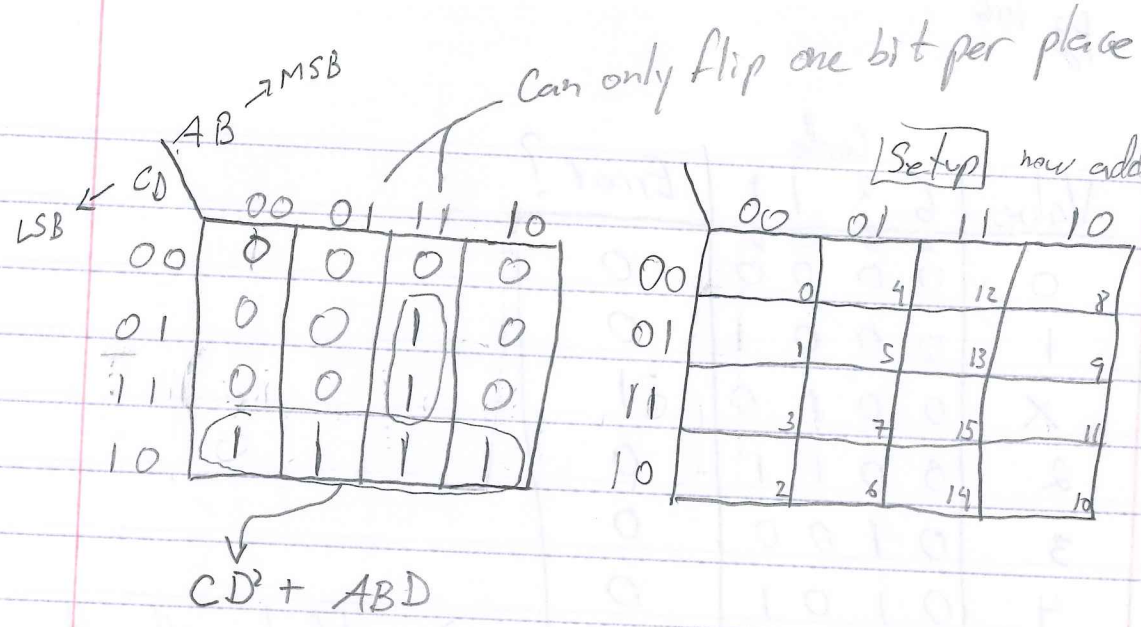
→ would be another 4

Error Check Circuit

$F = \Sigma m(2, 6, 10, 13, 14, 15)$

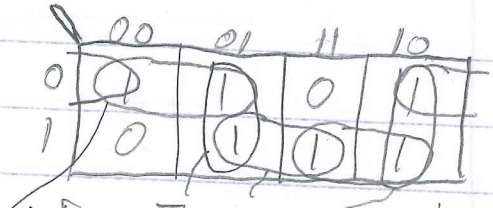
$F = A'B'CD' + A'BCD' + AB'CD' + ABC'D + ABCD' + ABCD$

KMAPS →



Prime Implicant - Are groupings in KMAPS

They become essential Prime's when a box is essential to cover a straggling (1)



↳ Prime Implicants but no Essentials

Groups can only be made in power of 2's

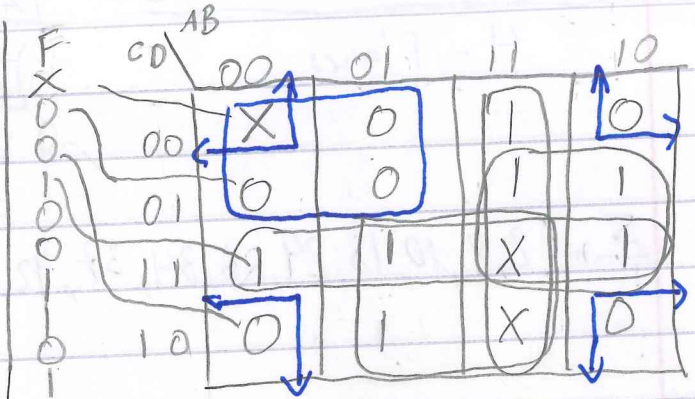
2^0, 2^1 2^2 2^3

1 2 4 8

↳ No 6

Examples:

A	B	C	D	F
0	0	0	0	X
}	}	}	}	0
1	1	1	1	0
				1
				0
				1
				0
				1
				0
				1
				0
				1
1	1	1	0	X
1	1	1	1	X



All boxes Essential Prime (EPI)

$F = CD + AB + BC + AD$

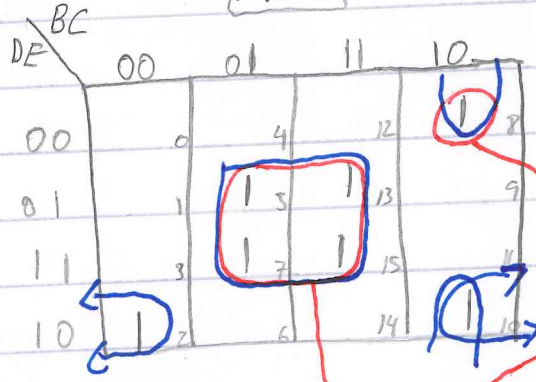
$POS = (A+C)(B+D)$

5 - Variable KMAPS #rows = 32

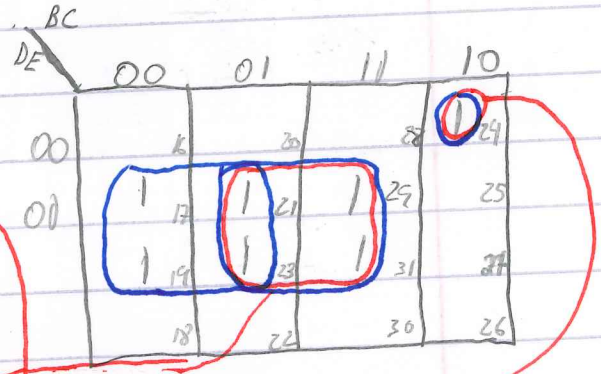
$\Sigma m(2,5,7,8,10,13,15,17,19,21,23,24,29,31)$

$2 = A'B'C'DE'$

A = 0

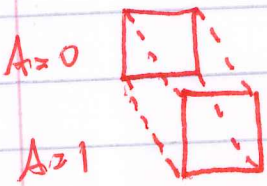


A = 1



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

$AB\bar{C}\bar{D}\bar{E}$



Box of 8

$$F = CE + A\bar{B}E + \bar{A}\bar{C}DE + B\bar{C}\bar{D}\bar{E}$$

Group of: 8 4 2 2

6 - Variable KMAPS

4 - Planes

AB = 00

AB = 01

AB = 11

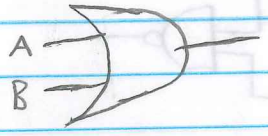
AB = 10

$\Sigma m(2,8,10,18,24,26,34,37,42,45,50,53,58,61$

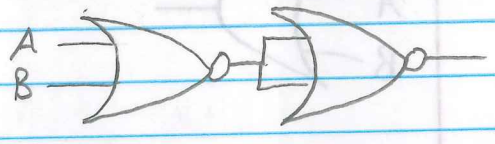
Universal Gates

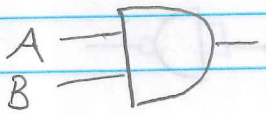
NOR Gates

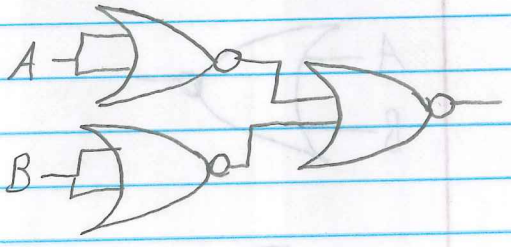
Desired Gate

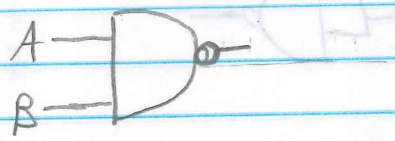


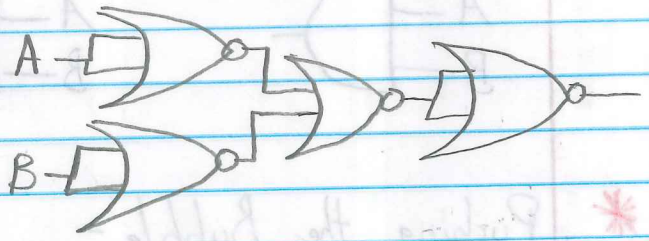
NOR Implementation

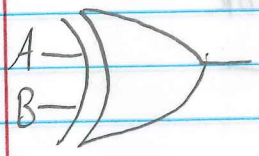


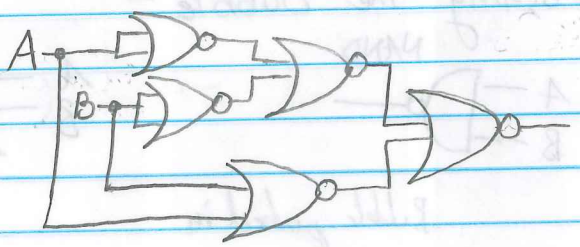






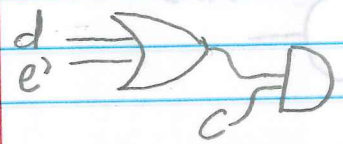




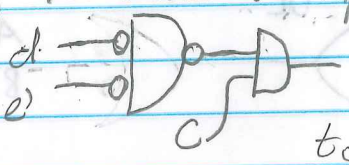


*

① More Bubbles:



② Put bubbles on input & output



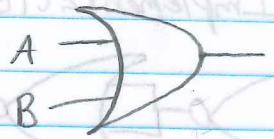
to change gates

Push bubbles to inputs ←

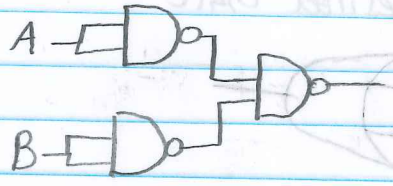
↳ Changes input Keep gates

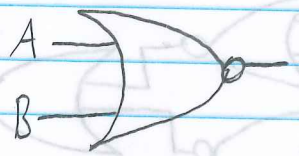
NAND Gates

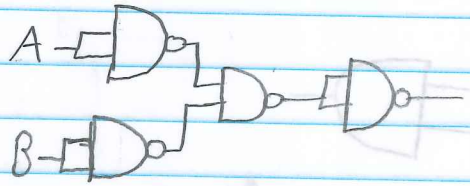
Desired Gate

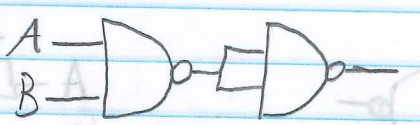


NAND Implementation

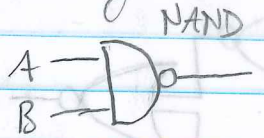


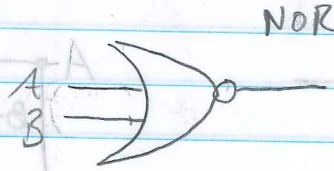






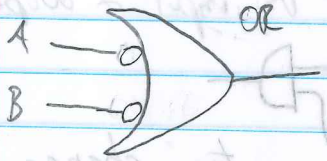
* Pushing the Bubble -

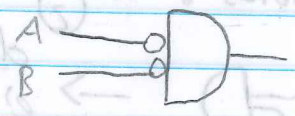




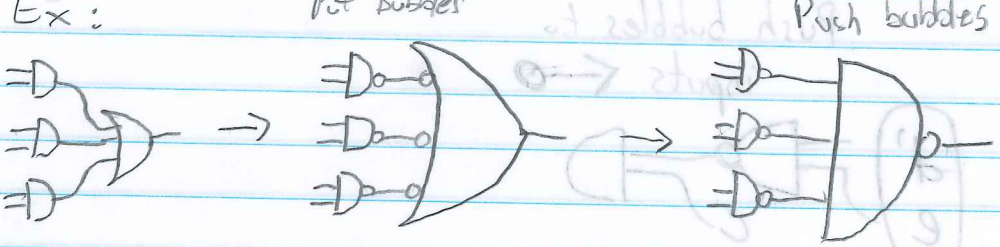
Bubble pushed in ←

Pushing in ←





Ex:



doesn't change

Quine-McCluskey Method

Skip: Petrick's Method & Map Entered Variable

$F(w,x,y,z) = \Sigma m(0,5,9,10,15) + \Sigma d(2,8,11)$

yz \ wx	00	01	11	10
00	1	0	0	X
01	0	1	0	1
11	0	0	1	X
10	X	0	0	1

Σm: 0000, 0101, 1001, 1010, 1111 — Σd: 0010, 1000, 1011

1's: 1, 2, 4, 8
2 1's: 3, 5, 6, 9, 10, 12
3 1's: 7, 11, 13, 14
} Groups

Groups	m/d	Column 1	Combined	Combined 4
0	0	0000 ✓	0,2: 00_0 ✓	check _
1	2	0010 ✓	0,8: _000 ✓	0,2,8,10: _0_0 }
	8	1000 ✓		0,8,2,10: _0_0 } → _0_0 *
2	5	0101 *	2,5: Can't Comb	
	9	1001 ✓	2,10: _010 ✓	
	10	1010 ✓	8,9: 100_ ✓	
3	11	1011 ✓	8,10: 10_0 ✓	
4	15	1111 ✓	9,11: 10_1 ✓	8,9,10,11: 10__ } 10__ *
			10,11: 101_ ✓	8,10,9,11: 10__ }
			11,15: 1_11 *	

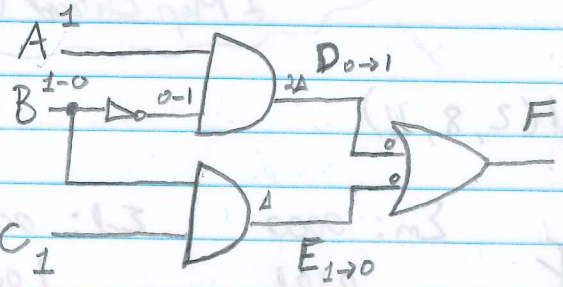
* Prime Implicant

* Prime Implicant Chart

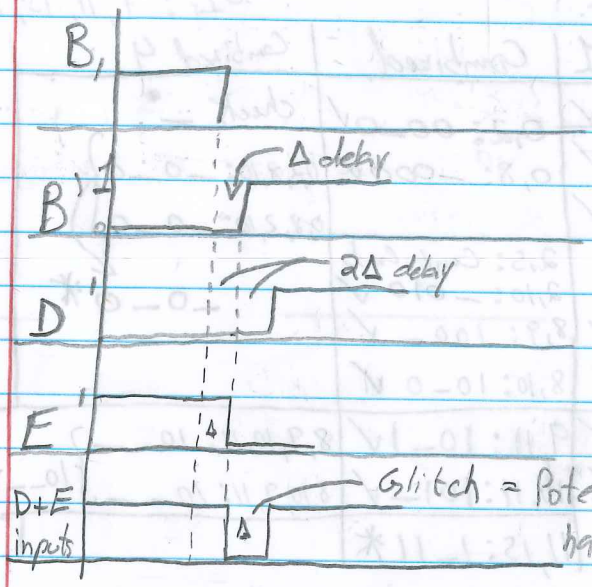
→ (m) minterms	0 ✓	5 ✓	9 ✓	10 ✓	15 ✓	
↓ Prime Implicants						
5 0101 --		X				EPI = 0101 = $\bar{w}x\bar{y}z$
(11,15) 1_11 --					X	EPI = 1_11 = wyz
(0,2,8,10) _0_0	X					EPI = _0_0 = $x\bar{z}$
(8,9,10,11) 10--				X		EPI = 10__ = $w\bar{x}$

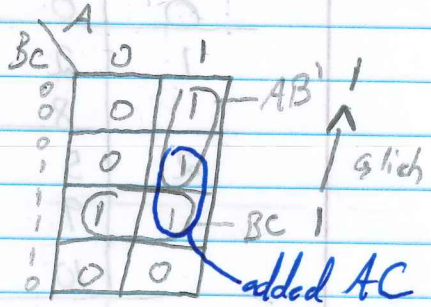
Final Answer: $F(w,x,y,z) = \bar{w}x\bar{y}z + wyz + x\bar{z} + w\bar{x}$

Time Delays



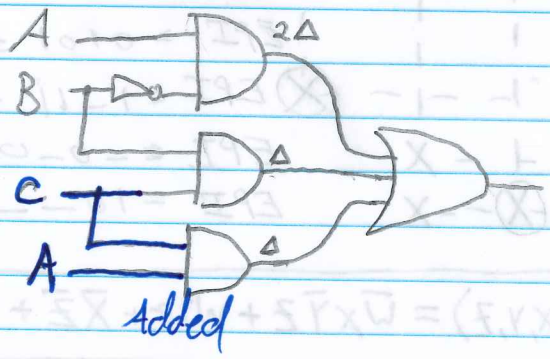
A = 1, C = 1, Out always 1
Static Output

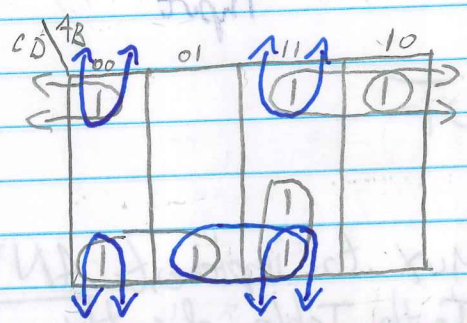




Add another term to take care of Glitch

- Glitch caused by 1 var change





Glitches b/w 1 variable changes
ie: adjacent bubbles

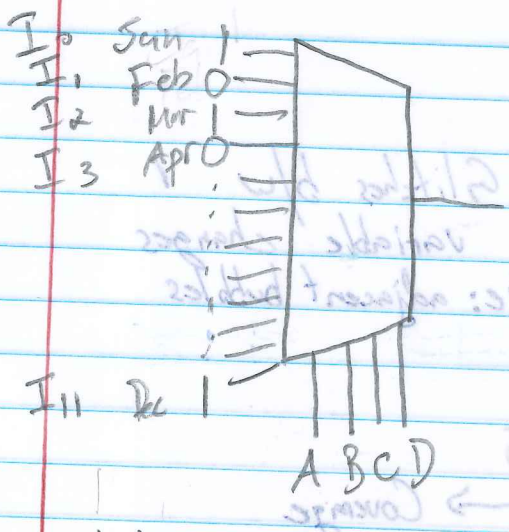
$F = \bar{B}\bar{C}\bar{D} + A\bar{C}\bar{D} + ABC + \bar{A}C\bar{D}$
$+ \bar{A}\bar{B}\bar{D} + AB\bar{D} + BC\bar{D}$ → Coverage

Demux

Build logic circuit that outputs 1 when number of days in the month is 31

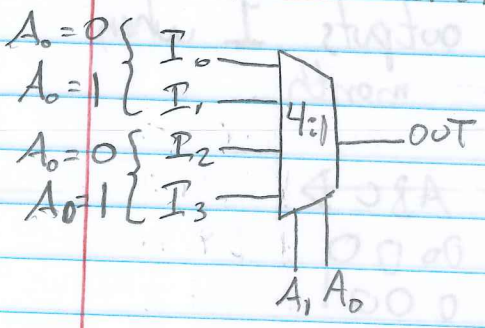
*Truth Table			ABCD
Jan	31	1	0000
Feb	28/29	0	0001
Mar	31	1	0010
Apr	30	0	0011
⋮	⋮	⋮	⋮
Dec	31	1	1011

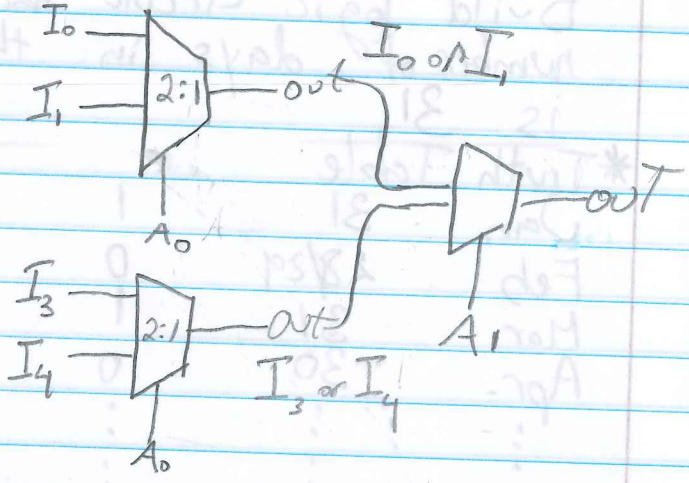
MSB is last input



- Mux to implement ANY Truth Table directly

Works via SoP logic where inputs ABCD get inverted then ANDed with the selector on the left.

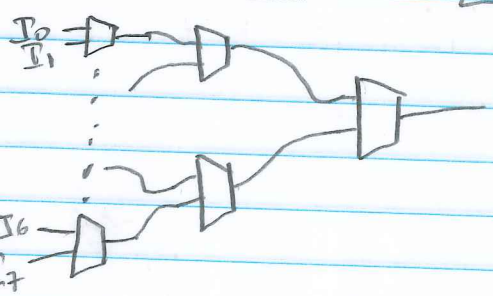




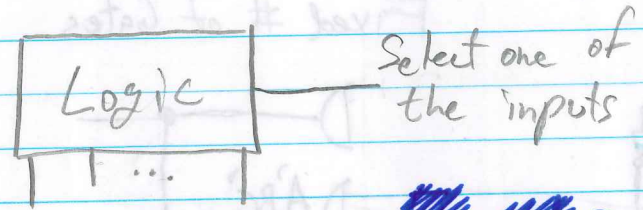
A_1	A_0	
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

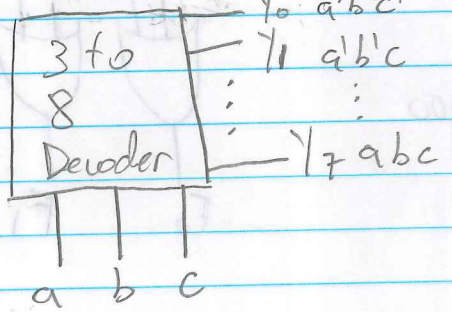
4_x [2:1] + 2_x [2:1] + 1_x [2:1]

8:1 Mux using 2:1

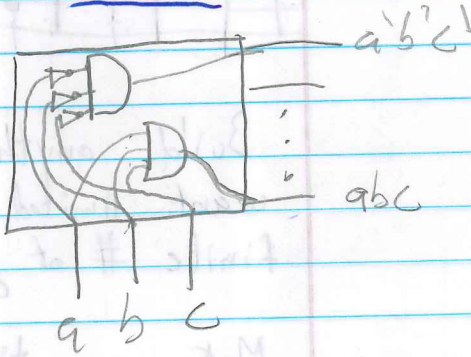


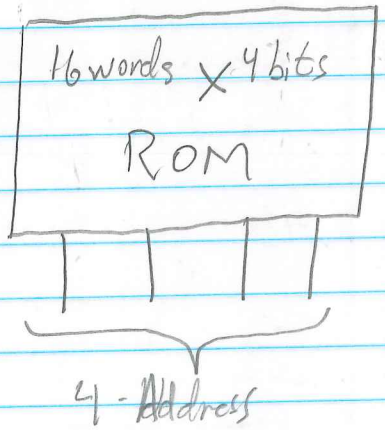
- N Address Lines

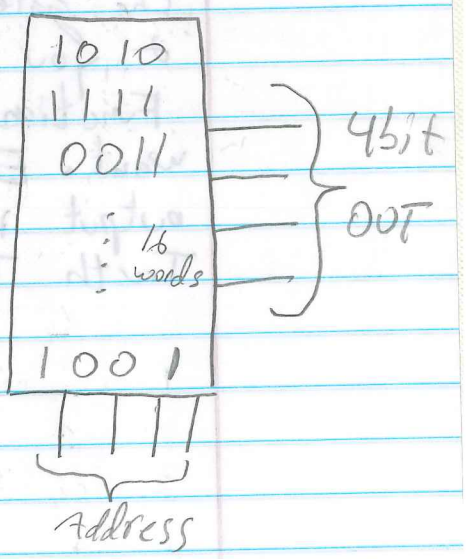




Decoder





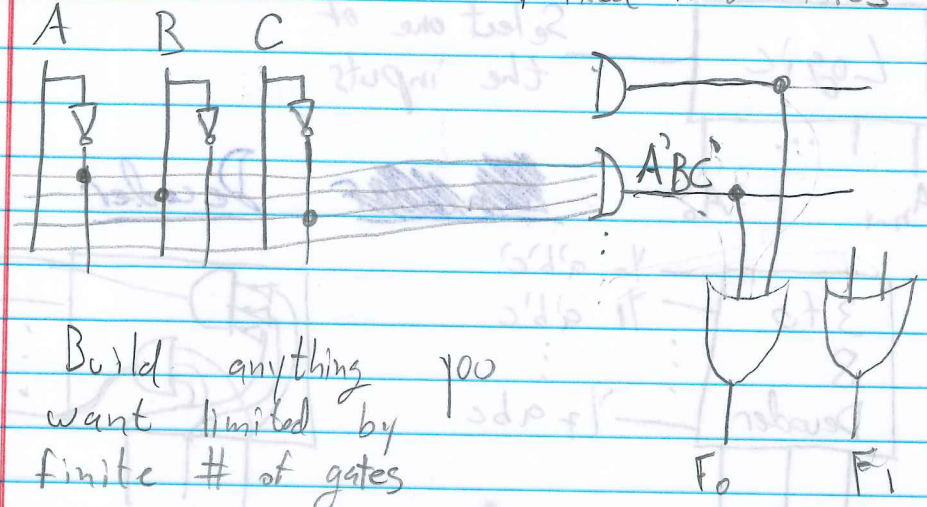


n address bits, m bits each word
is 2^n words each of m-bits wide

PLA

Programmable Logic Array

Fixed # of Gates

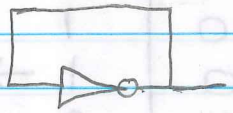


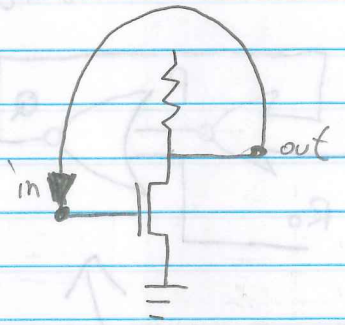
Build anything you want limited by finite # of gates

Make connections to the gates you want & you get whatever Function outputs you want. Each 'F' output is an individual Truth Table

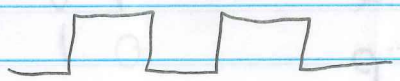
Sequential Circuits

Chapter 11

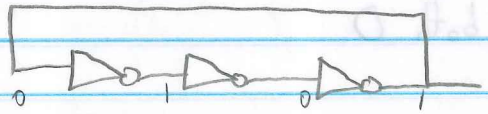




Creating a system that operates as a function of time

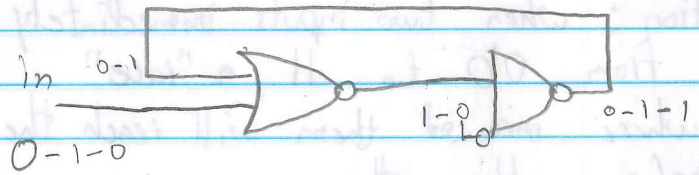
Clock 

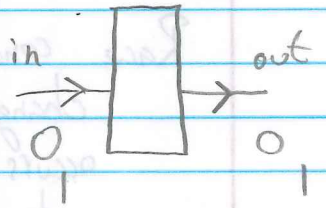
Use odd # of inverters



If we wish to control that oscillating circuit we can do the following

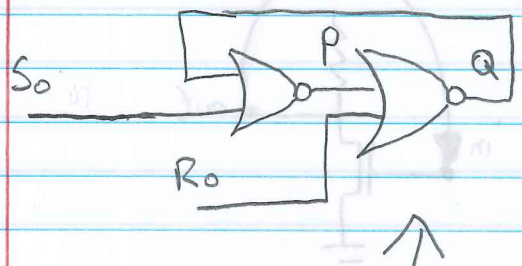
Latch





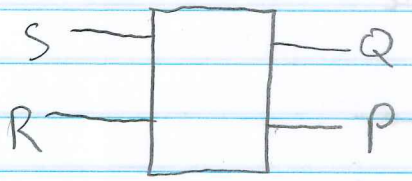
	in	out
	0	0
	1	1
3rd land	0	1
Latched	1	1
forever	0	1

But how to reset it?



S	R	OUT
0	0	0
1	0	1
0	0	1 → Latch 1 | Set
0	1	0 → Reset

S-R Latch



S	R	P	Q	
1	0	0	1	
0	1	1	0	
1	1	0	0	→ Not inverse each other
0	0	1	0	

SR = 1,1 P & Q are both 0

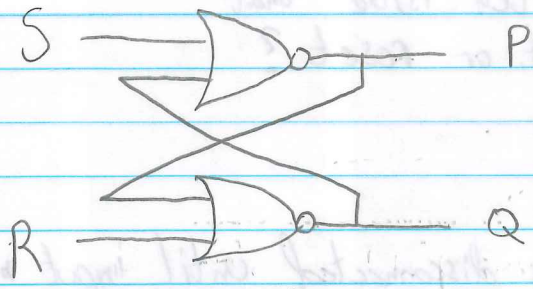
↓

SR = 0,0 Output (may Oscillate) race condition

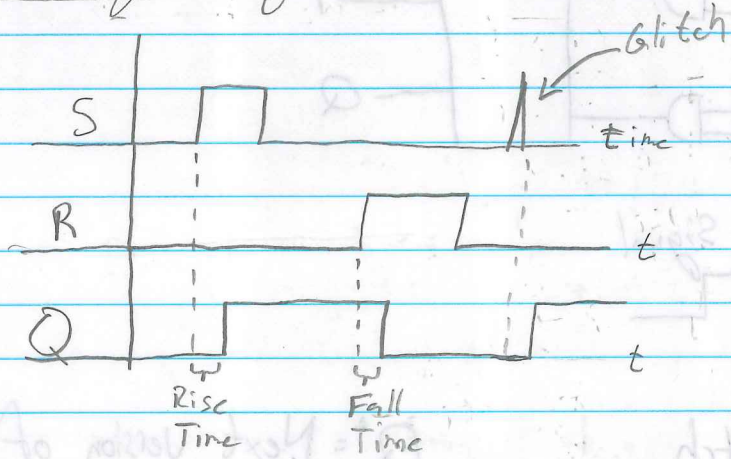
SR = should not be set to 1,1 (uncertain)

Race condition: when two inputs immediately change from 00 to 11, a "race" occurs where one of them will reach the gate before the other.

Same S-R Latch Circuit



Timing Diagram



S	R	Q^t (Present t)	Q^{t+1} (Next t)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	forbidden	
1	1	forbidden	

Q^+ = function of (S, R, Q)
Characteristic function

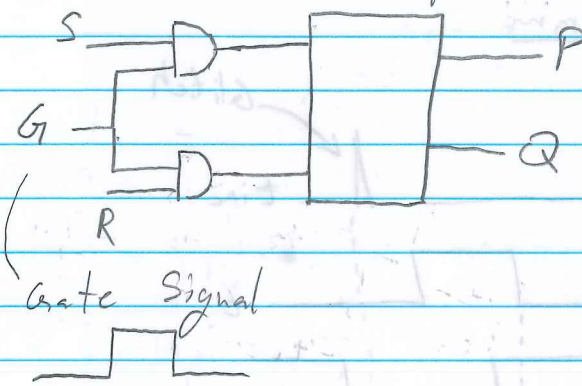
RQ \ S	0	1
00		1
01	1	1
11		X
10		X

$Q^+ = S + \bar{R}Q$

What about the glitch issue that can permanently set or reset?

Gated Latch

Keep S & R always disconnected until input made



SR Latch

Q^+ = Next version of Q

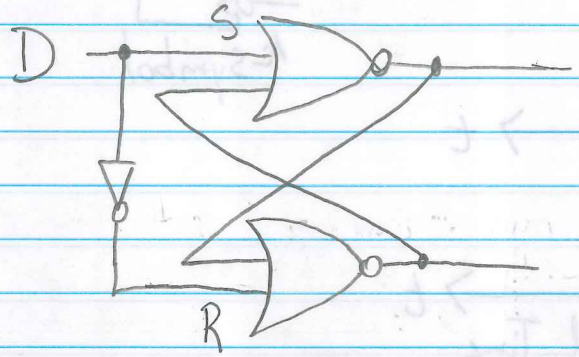
$Q^+ = S + \bar{R}Q$

$S=0$,	$R=0$	Hold $Q^+ = Q$
$S=1$,	$R=0$	Set $Q=1$
$S=0$,	$R=1$	Reset $Q=0$
$S=1$,	$R=1$	Unpredictable

With Gate

$Q^+ = SG + Q(\bar{R}+\bar{G})$ Race condition still exists

D Latch

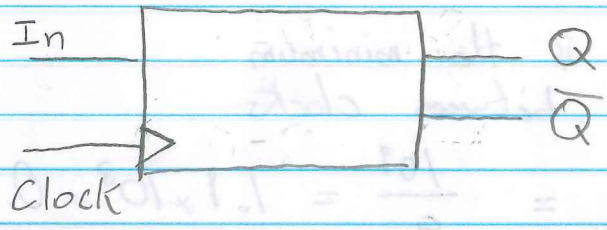


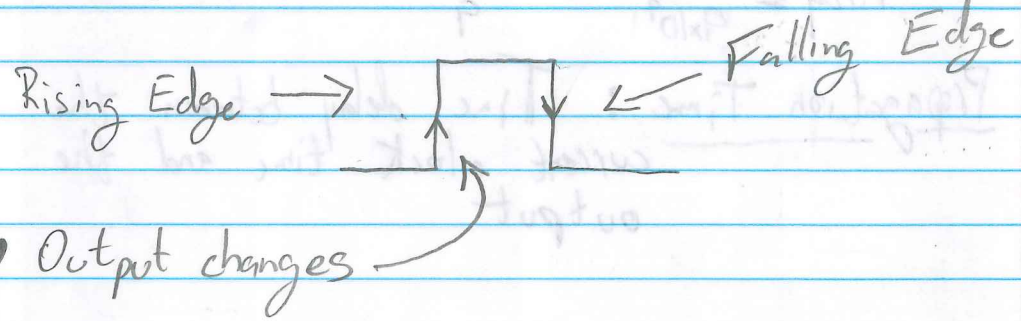
D	Q^+	
0	0	Reset
1	1	Set

Advantage:
Both S & R inputs can no longer be 1 simultaneously
Also no Race condition

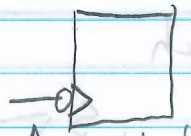
Issue:
Can no longer be 0 Simultaneously. No Hold condition. Lost memory capability

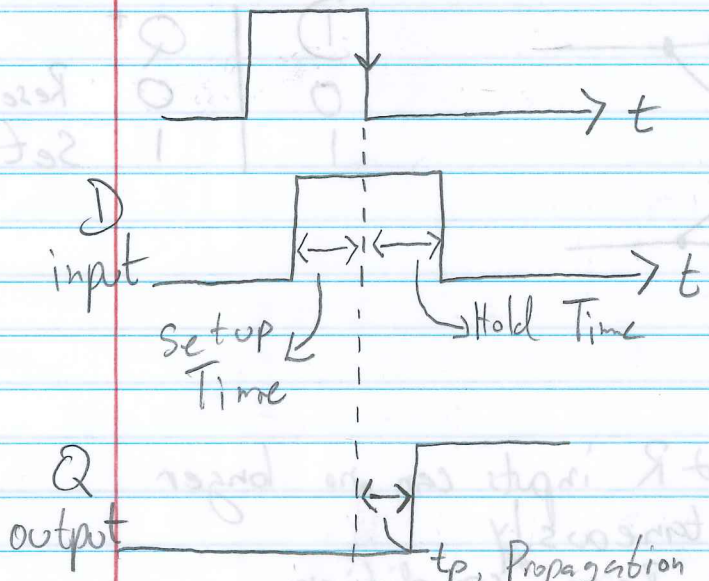
Edge Triggered Flip-Flop

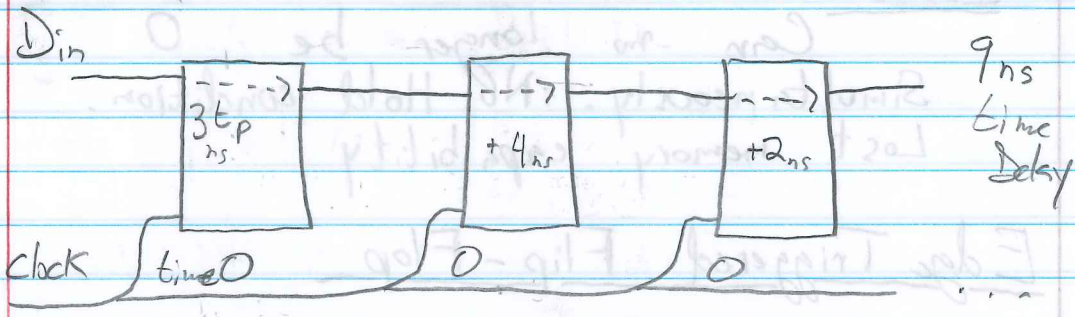




Falling Edge Clock





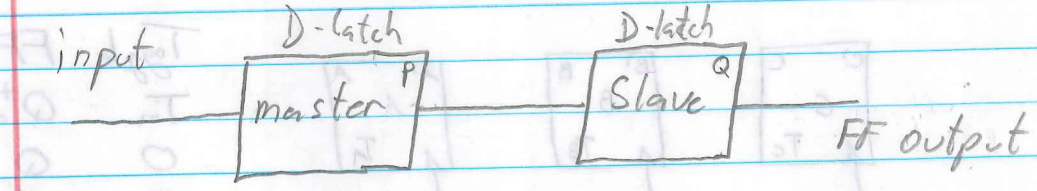


9 ns is the minimum period between clocks

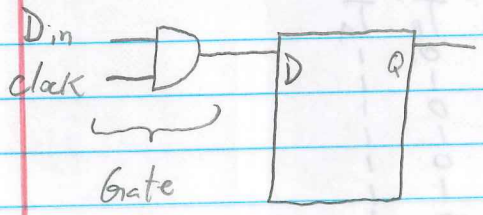
$$\text{Freq} = \frac{1}{9\times10^{-9}} = \frac{10^9}{9} = 1.1\times10^8 = 0.11\,\text{GHz}$$

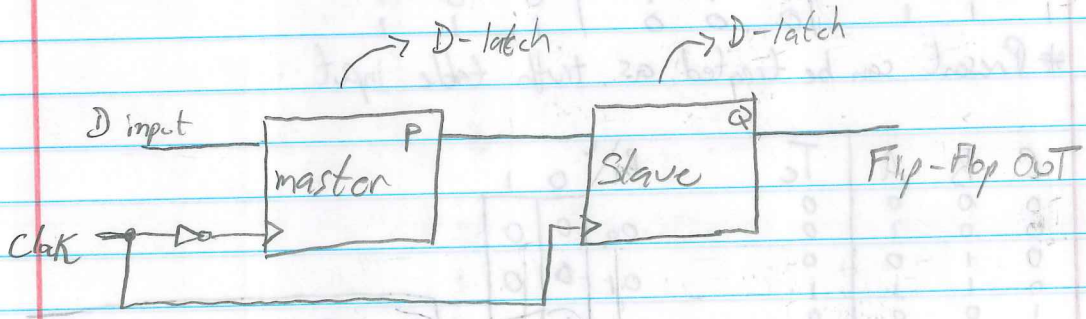
Propagation Time: Time delay between the current clock time and the output

Construction of D-Flip Flop



Gated D-Flip Flop



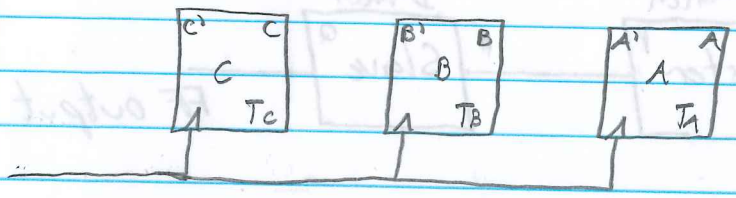


Clock: 1	Clock: 0
Master: OFF	Master: ON
Slave: ON	Slave: OFF

At the transition from 0 → 1, master keeps the last value it received before the switched. Slave turns on & reads the output value of Master and outputs the new result.

Rising Edge: Inverter on Master
Falling Edge: Inverter on Slave

Binary Counter



Toggle FF

T_{in}	Q^+_{out}
0	Q
1	Q'

Present State			Next State			Flip Flop In		
C	B	A	C^+	B^+	A^+	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

* Present can be treated as truth table input

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

BA \ C	0	1
00	0	0
01	0	0
11	1	1
10	0	0

$T_C = BA$
So AND B, A together

C	B	A	C^+	B^+	A^+	D_C	D_B	D_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

$D_C = C^+$
$D_B = B^+$
$D_A = A^+$
For DFF just use the Next state

BA \ C	0	1
00	1	1
01		
11		
10	1	1

$D_A = \bar{A}$

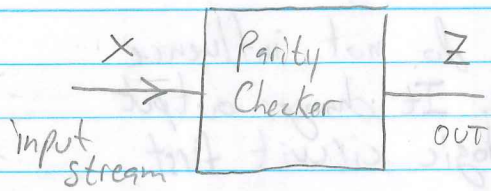
BA \ C	0	1
00		
01	1	1
11		
10	1	1

$D_B = \bar{B}A + B\bar{A}$
$= B \oplus A$

BA \ C	0	1
00		1
01	1	1
11		
10	1	

$C'BA + CB' + CA'$
$= C'BA + C(BA)'$
$\downarrow$
$C \oplus BA$

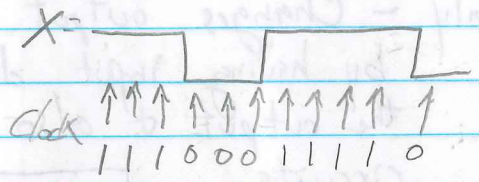
Parity Checker

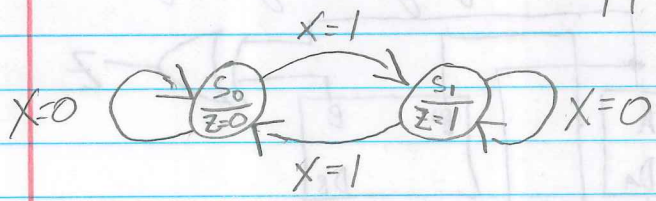


1001100111110...

Even # of 1's at input then Parity = 0
while odd, Parity = 1

$Z^+ = Z \oplus X$



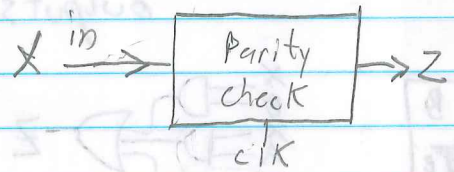


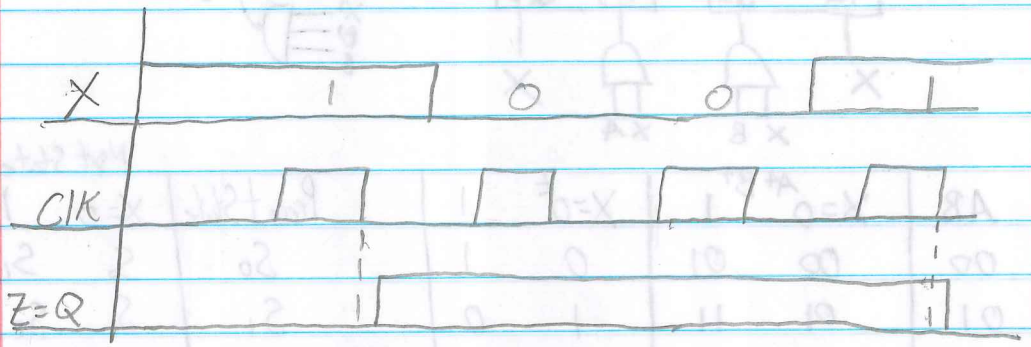
State Graph

Every time input = 1, flip atput
Can be implemented with a TFF

Q	Q^+ X=0	Q^+ X=1	T X=0	T X=1	Z
0	0	1	0	1	0
1	1	0	0	1	1

State Table 13-1



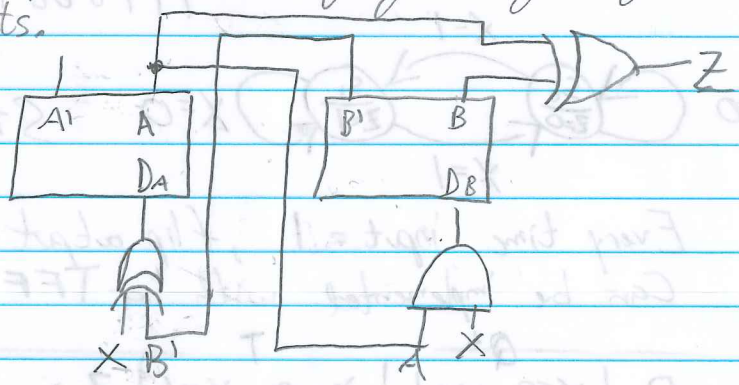


Moore & Mealy Machines

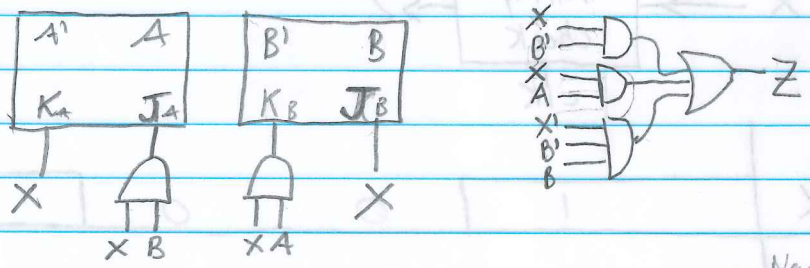
Moore - Changes to input do not influence the output directly. It changes output by going through logic circuit first

Mealy - Changes output directly from input by having input directly connected to the output & also going through logic circuits.

Moore Machine →



Mealy Machine → Figure 13-7 — Watch out for Figure 13-8 — possible false outputs



AB	$X=0$ A^+B^+ $X=1$		$X=0$ Z 1	
00	00	01	0	1
01	01	11	1	0
11	11	00	0	1
10	10	01	0	1

	Next State		Present out	
Present State	X=0	1	X=0	1
S_0	S_0	S_1	0	1
S_1	S_1	S_2	1	0
S_2	S_2	S_0	0	1
S_3	S_3	S_1	0	1

$$A^+ = J_A A' + K_A' A$$

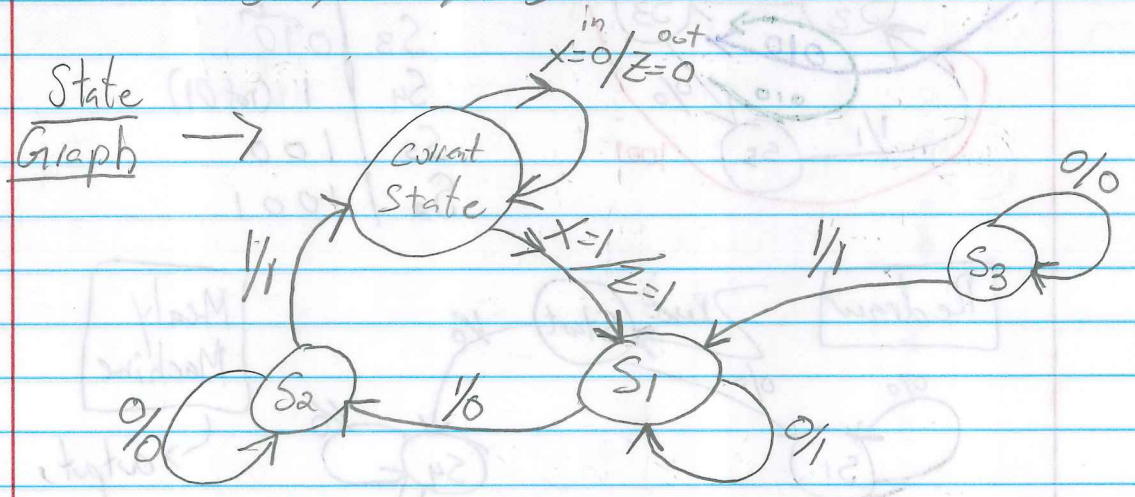
$$= XBA' + X'A$$

$$B^+ = J_B B' + K_B' B$$

$$= XB' + (AX)'B = XB' + XB' + A'B$$

$$Z = XB' + XA + X'A'B$$

State Graph →

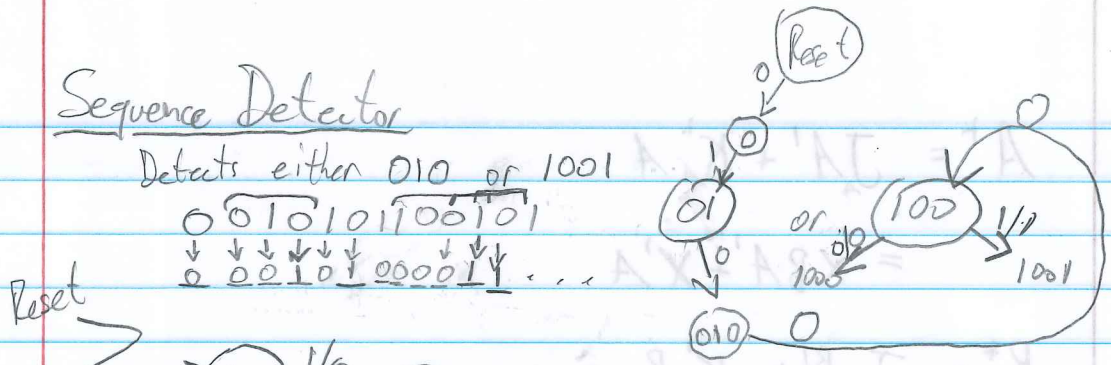


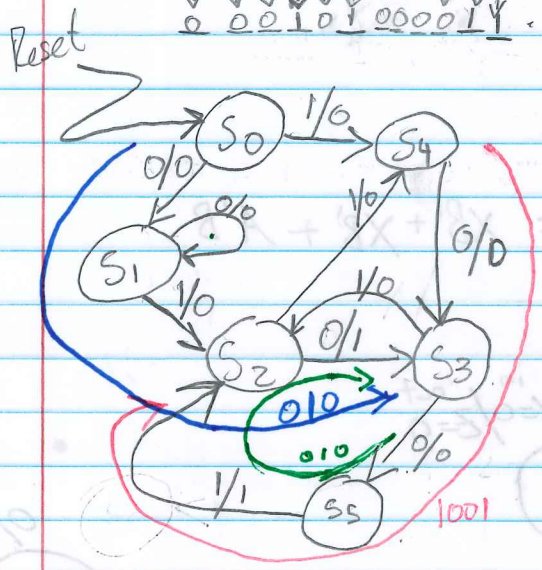
Sequence Detector

Detects either 010 or 1001

0 0 1 0 1 0 1 1 0 0 1 0 1

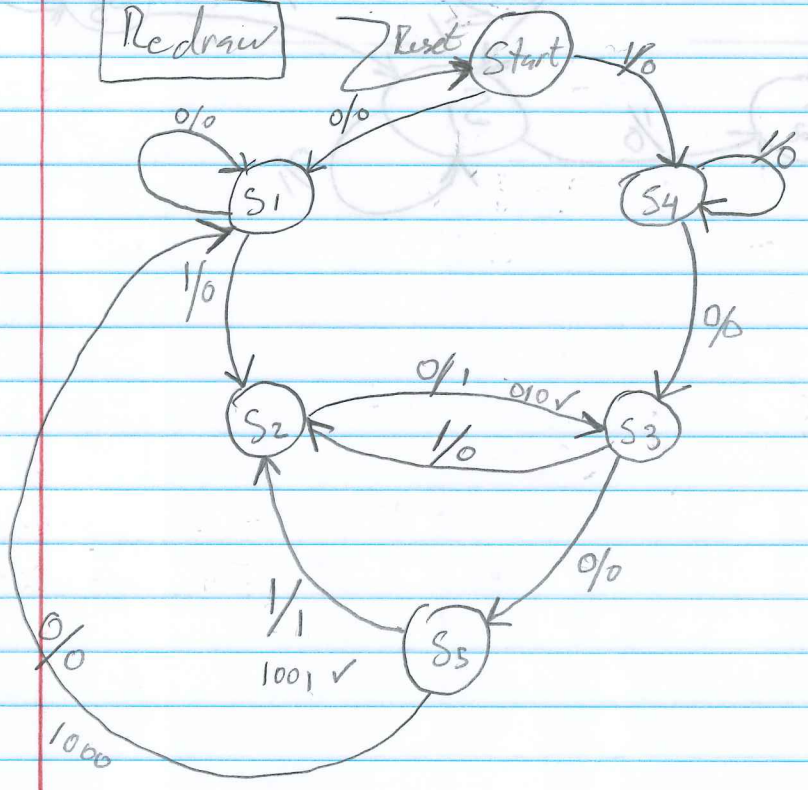
0 0 0 1 0 1 0 0 0 0 1 1 . . .





State	Sequence Ends in
S_0	Reset
S_1	0
S_2	01
S_3	010
S_4	1 (not 01)
S_5	100
S_2	1001

Redraw



Mealy Machine

→ outputs written on arcs

(2)

MOORE MACHINE

Output: 1 if odd # of 1's received

0 1 1 0 1 0 0 1 ...

↓ ↓ ↓ ↓ ...

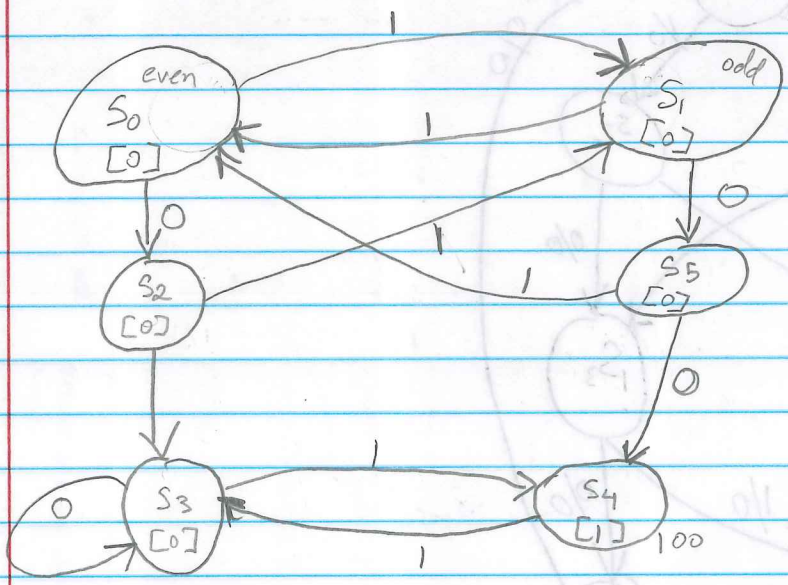
1 0 0 1 ... → lag b/w out/in because needs to change state

New condition: now only 1 after sequence of 00 received

0 1 1 0 1 0 0 1 1

↓ ↓ ↓ ↓ ... ↓

0 0 0 0 ... 1



Disjoint

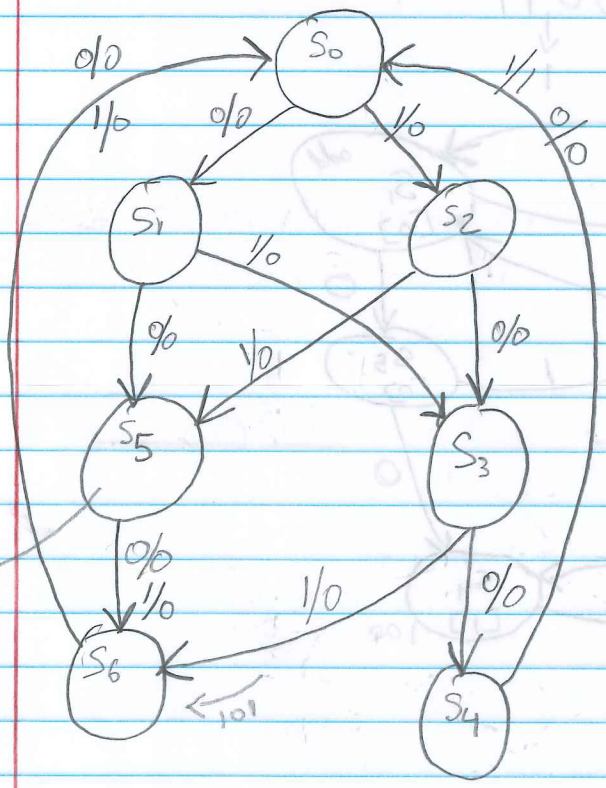
* Not overlapping

only differs

Block of 4 bits : 0101 or 1001

can't use partial bits

Output : 1 if either block of whole 4-bits seen



Overlapping Sequence

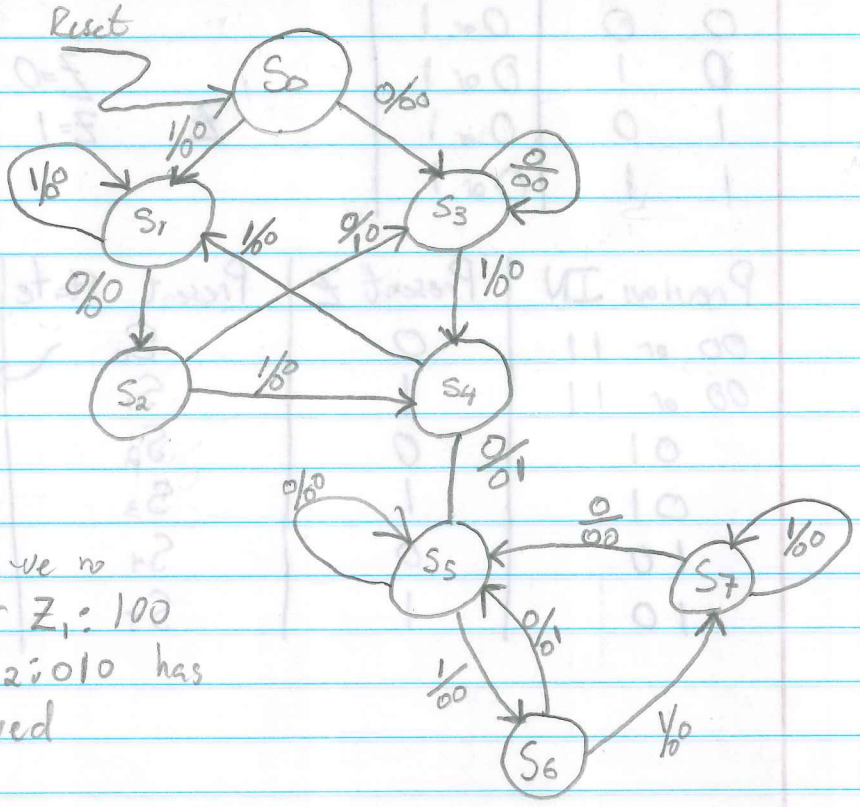
X: 1 bit Input
Z_1, Z_2: 2 bit output
Z_1: 1 when 100 received
Z_2: 1 when 010 received

1 0 0 1 0 1 0 0 ...
Z_1: 0 0 1 0 0 0 0 1
Z_2: 0 0 0 0 1 0 1 0

Z_1: Now ONLY until 010 received (after that $Z_1 = 0$)

1 1 0 0 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 0 ...
Z_1: ? ? ↓ ↓ ?
Z_2: ↓ ↓ 1 1
Z_1 updated: 1 1 0 0 0 ...

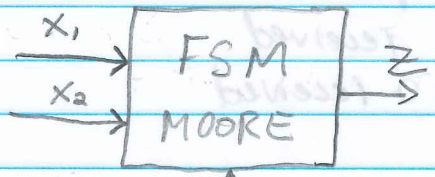
S_0 -
S_1, S_3 1 bit
S_2, S_4 2 bit



$S_4 \rightarrow S_5$ we no longer consider Z_1: 100 because a Z_2: 010 has been received

Input
X_1 X_2

Output
Z



Previous X_1 X_2	Present X_1 X_2	New Z
0 1	1 1	0
1 0	1 1	1
1 0	0 1	Z

Previous IN X_1 X_2	Output Z
0 0	0 or 1
0 1	0 or 1
1 0	0 or 1
1 1	0 or 1

Next Input X_1 X_2: 00 01 11 10

Z $Z=0$ $Z=1$

Previous IN	Present Z	Present state	Next State (X_1X_2 = 00 01 11 10)
00 or 11	0	S_0	S_0 S_2 S_0 S_4
00 or 11	1	S_1	S_1 S_3 S_1 S_5
01	0	S_2	S_0 S_2 S_0 S_4
01	1	S_3	S_1 S_3 S_0 S_5
10	0	S_4	S_0 S_2 S_1 S_4
10	1	S_5	S_1 S_2 S_1 S_5

Ch 15 Matching Method

1. Row matching method

Same next states & same outputs among rows independent of the input sequence

Ex:

C.S. input	Present State	N.S X=0	N.S X=1	OUT X=0	OUT X=1
000	H	A	A	0	0
001	I	A	A	0	0
010	J	A	A	0	1

merge [H, I

Simply strike out the row to be merged & keep or rename the remaining row

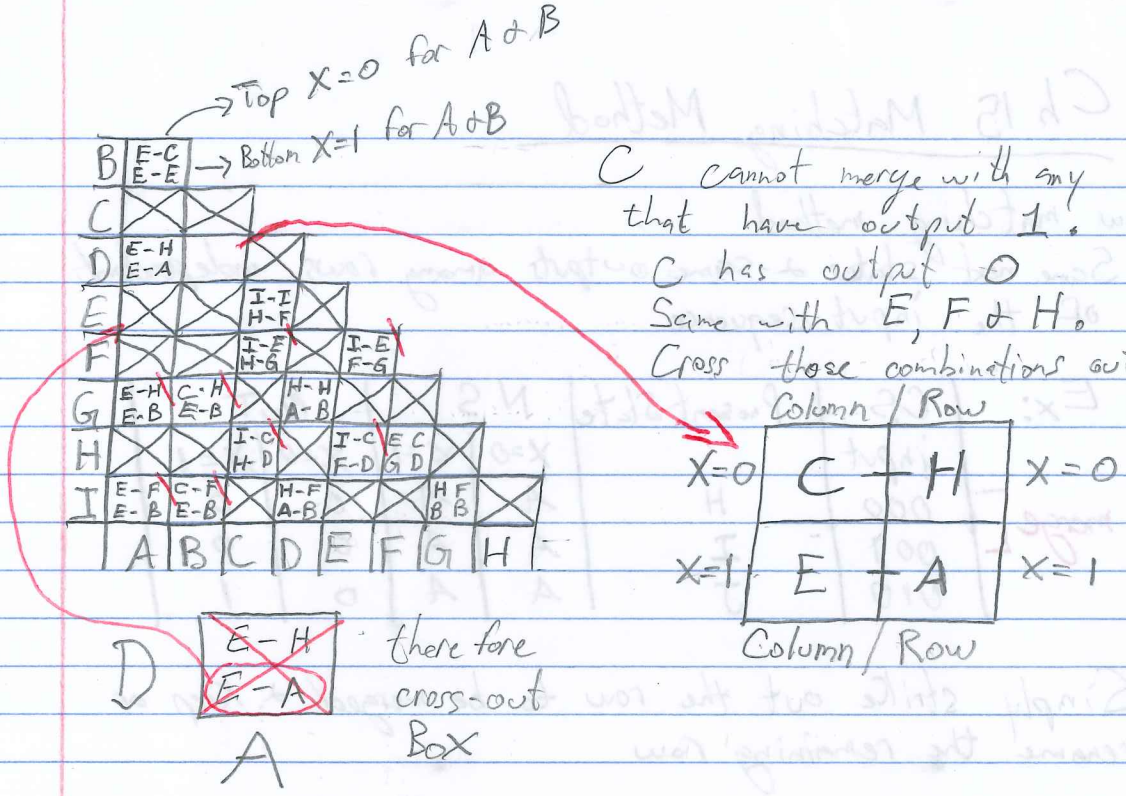
~~001 I A A 0 0~~ row striked out

Upon merging, the new remaining row might now have similar next states & outputs with other rows where it didn't share similarities with. Search again for more matching rows & eliminate repetitions again.

2. Implication Chart Method

Reduce the following State Table

Present State	Next State X=0	Next State X=1	Present Output
A	E	E	1
B	C	E	1
C	I	H	0
D	H	A	1
E	I	F	0
F	E	G	0
G	H	B	1
H	C	D	0
I	F	B	1



Red slash marks in boxes above are crossed out

- Now Merge remaining boxes

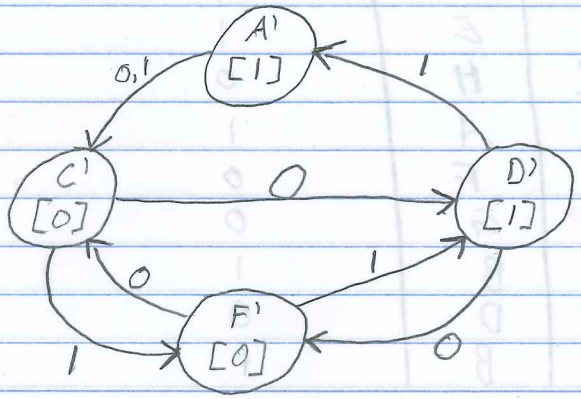
$A \equiv B \rightarrow A'$

$C \equiv E \rightarrow C'$

$D \equiv I \equiv G \rightarrow D'$

$F \equiv H \rightarrow F'$

Present State	Next State		Present Output	
	X=0	X=1		
A'	C'	C'	1	Merged A & B
C'	D'	F'	0	merged C & E
D'	F'	A'	1	merged D & I & G
F'	C'	D'	0	merged F & H



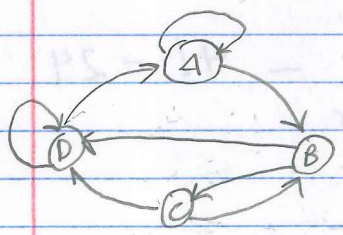
Reducing within multiple machines

Similar machines have the same next state & same output for the same input. It is the same & equal by having the same length of states & same outputs.

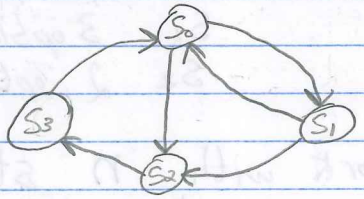
The machines need to be functionally equal. Machines having different number of states can still be equal if one of the machines have not been minimized yet.

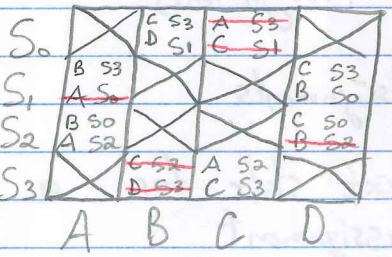
	X=0	X=1	X=0	X=1
A	B	A	0	0
B	C	D	0	1
C	A	C	0	1
D	C	B	0	0

	X=0	X=1	X=0	X=1
S_0	S_3	S_1	0	1
S_1	S_3	S_0	0	0
S_2	S_0	S_2	0	0
S_3	S_2	S_3	0	1



Not the Same?





Match outputs according to inputs inside the box & cross out non-compatability. Then match transitions.

Only consider boxes that don't have anything crossed out

B S_0
A S_2

Both top & Bottom check out

C S_3
B S_0

Top & Bottom ✓

C S_3
D S_1

Top ✓ Bottom ✓

A = S_2	C = S_3
B = S_0	D = S_1

Machines are the Same

15.6

	X=0	X=1
S_0	S_1	S_2
S_1	S_3	S_2
S_2	S_1	S_4
S_3	S_5	S_2
S_4	S_1	S_6
S_5	S_5	S_2
S_6	S_1	S_6

State Table

0 0 → How Was this Assigned? → 1 0

ABC	X=0	X=1	X=0	X=1
000	110	001	0	0
110	111	001	0	0
001	110	011	0	0
111	101	001	0	0
011	110	010	0	0
101	101	001	1	0
010	110	010	0	1

Transition Table

State Assignment

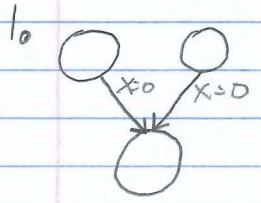
Lets work with 3 states, only need 2-bits

bits: 00, 01, 10, 11

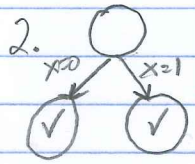
States: S_0 can have 4-options, S_1 3 options, S_2 2 options } $4 \times 3 \times 2$ 24 possible assignments $- 4! = 24$

Work with n states.
We can have $n!$ possible assignments

Heuristics - Guidelines for State Assignments

1. 

* 1. States which have same next state for a given input ⇒ Adjacent assignment

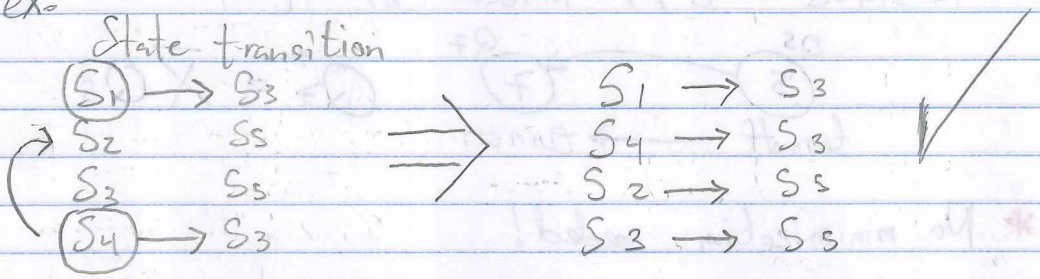
2. 

* 2. States which are the next states (children) of a given state ⇒ Adjacent Assignment

* 3. States with same output for a given input ⇒ adjacent assignment
doesn't have to go to same state

1. States which have the same next state for a given input should be assigned to adjacent boxes

ex.



Ex: 5 - 15

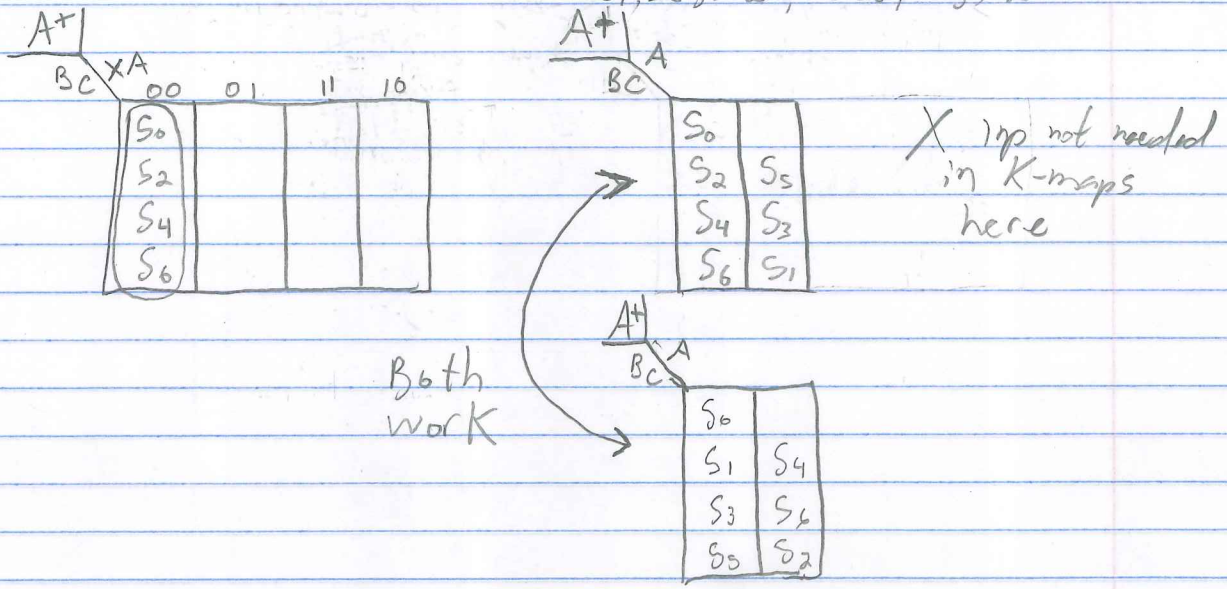
ABC		X=0	X=1	X=0	X=1
0 0 0	S_0	S_1	S_2	0	0
1 1 0	S_1	S_3	S_2	0	0
0 0 1	S_2	S_1	S_4	0	0
1 1 1	S_3	S_5	S_2	0	0
0 1 1	S_4	S_1	S_6	0	0
1 0 1	S_5	S_5	S_2	1	0
0 1 0	S_6	S_1	S_6	0	1

By rule 1: X=0 $\{S_0, S_2, S_4, S_6\}$, $\{S_3, S_5\}$

X=1 $\{S_0, S_1, S_3, S_5\}$, $\{S_4, S_6\}$

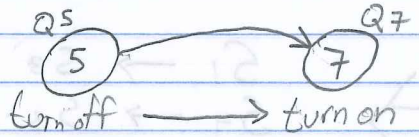
By rule 2: $\{S_1, S_2\}$, $\{S_2, S_3\}$, $\{S_1, S_4\}$ ...

$\{S_1, S_6\} \times 2$, $\{S_5, S_2\} \times 2$



One hot Encoding

- One FF for each state
  13 states = 13 FF instead of 4



$$Q_7^+ = \bar{X} Q_5$$

* No minimization needed!

BCD → Excess 3 Code

0000 + 0011 → 0011
1001 + 0011 → 1100

See example 16-1 in Book for rest