EE2001 - DIGITAL SYSTEMS

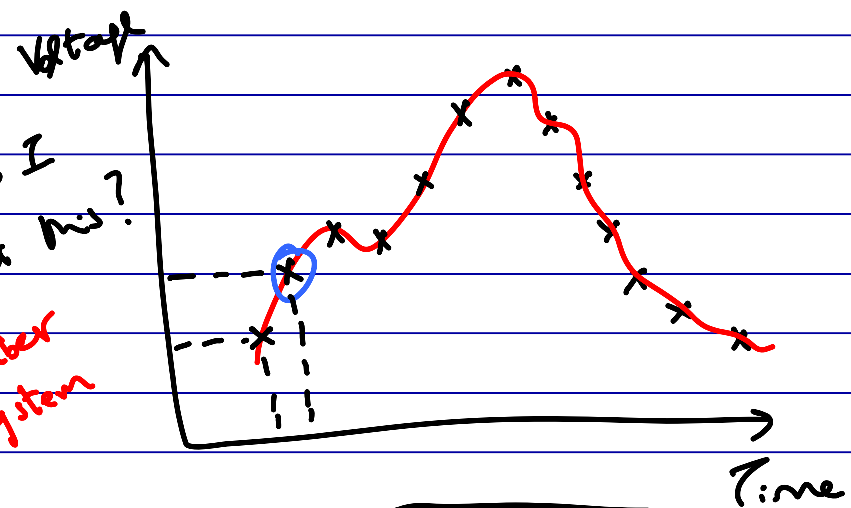
Note Title 1/16/2018

Learning objective: Carry out conversion between different number systems

Real world signals are analog in nature

How do I represent this?

⇒ Number system



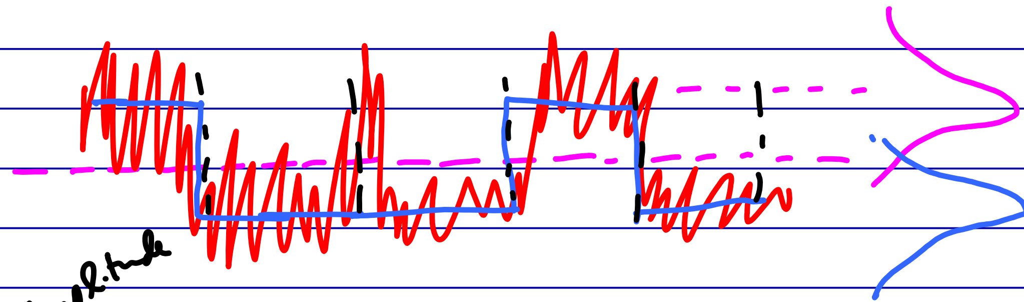
Analog in → Analog to Digital → Discrete → Store/Process/Transmit → Digital to Analog → Analog Signal

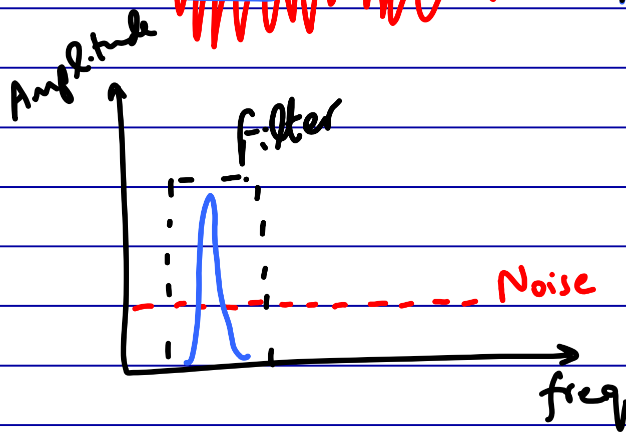
Prone to errors; more expensive

Energy consumption

End / Start	Binary	Decimal	Hexa-D
Binary	✗	$\times 2^{n-1}$	4 binary → Hexa
Decimal	÷ 2 (left) × 2 (right)	✗	÷ 16 (left) × 16 (right)
Hexa-D	Each dec → 4 binary	$\times 16^{n-1}$	✗

$$\begin{matrix} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \times & \times & \times & \times & \times \\ 1 & 0 & 0 & 1 & 0 \end{matrix} \rightarrow (18)_{10}$$





PC → CPU → ALU → Adders, Comparators, Multipliers

I/O   Registers

Memory   Control

$$(11010.11)_2 \rightarrow (26.75)_{10}$$

$$1 \times 2^{-2} = 0.25$$

$$1 \times 2^{-1} = 0.5$$

$r \rightarrow$ base

$$(a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2})_r$$

$$a_4 r^4 + a_3 r^3 + \dots + a_{-1} r^{-1} + a_{-2} r^{-2}$$

Binary	Decimal	Hexa-D
0001		1
0010		2
		3
		4
		5
		6
		7
		8
		9
	10	A
	11	B
	12	C
	13	D
	14	E
1111	15	F

$11 \times 16^3 + 6 \times 16^2 + 5 \times 16 + 15$

$\uparrow$

$(B65F)_{16}$

$= (46,687)_{10}$

$(41.6875)_{10} \rightarrow (101001.1011)_2$

2	41	
2	20	1
2	10	0
2	5	0
2	2	1
	1	0

$.6875 \times 2 = 1 + 0.375$

$0.375 \times 2 = 0 + 0.75$

$0.75 \times 2 = 1 + 0.5$

$0.5 \times 2 = 1$

$(\underline{10}\,\underline{1100}\,\underline{0110}\,\underline{1011}.\underline{1111}\,\underline{0000}\,\underline{0110})_2$

$(2\ C\ 6\ B.F\ 0\ 6)_{16}$

	Addition	Subtraction	Multiply
11	1011	1011	1011
5	0101	0101	× 0101
	16 ← 10000	6 ← 0110	

110111 = 55

18/1/18

LO: Carry out calculations involving signed binary numbers

How do we represent signed numbers?

1 Byte → 8 bits

Complementary

$+20 \rightarrow 00010100$

Toggle

1s complement → 11101011 (−20)

2s complement → 11101100

Extra sign bit

'0' → +ve

'1' → −ve

Sign

$(-20)_{10} \rightarrow 1\ 001\ 0100$

Decimal	Sign & Magnitude	Signed 1s complement	Signed 2s complement
+3	0011	0011	0011
+2	0010	→ 0010	→ 0010
+1	0001	→ 0001	→ 0001
0	0000	→ 0000	→ 0000
−0	1000	1111	—
−1	1001	1110	1111
−2	1010	1101	1110
−3	1011	1100	1101
−4	1100	1011	1100

0000

Arithmetic Operations

$+6 \rightarrow 0000\ 0110$

$+13 \rightarrow 0000\ 1101$

$+19 \leftarrow 0001\ 0011$

$-6 \rightarrow 1111\ 1010$

$+13 \rightarrow 0000\ 1101$

$+7 \quad \cancel{1}\ 0000\ 0111$

$+6 \rightarrow 0000\ 0110$

$-13 \rightarrow 1111\ 0011$

$-7 \leftarrow 1111\ 1001$

$-6 \rightarrow 1111\ 1010$

$-13 \rightarrow 1111\ 0011$

$-19 \leftarrow \cancel{1}\ 1110\ 1101$

Subtract: $11101 - 10101$

(a) direct

(b) 1s complement

(c) 2s complement

Confirm your result in decimal

(a) Direct subtraction

$$\begin{array}{r} 11101 \\ -\ 10101 \\ \hline (01000)_2 \end{array}$$

$8 \leftarrow (01000)_2$

(b) 1s complement

$$\begin{array}{r} 11101 \\ +\ 01010 \\ \hline 1\ 00111 \end{array}$$

Add carry $(01000)_2$

(c) 2s complement

$$\begin{array}{r} 11101 \\ +\ 01011 \\ \hline (01000)_2 \end{array}$$

Binary coded decimal:

$(267)_{10} = (0010\ 0110\ 0111)_{BCD}$

$(100001011)_2$

Excess-3 (Add 3)

011

$0000 \rightarrow 0011$

$\vdots \qquad \vdots$

$1001 \rightarrow 1100$

0	0000
1	0001
2	
3	
4	
5	
6	
7	
8	1000
9	1001

19/1/18

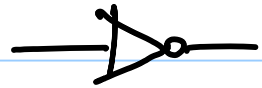
BOOLEAN ALGEBRA

Mathematical method that simplify circuits relying primarily on Boolean algebra

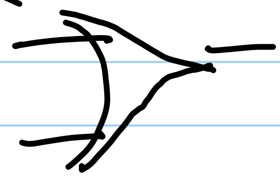
x	y	$x.y$	$x+y$
0	0	0	0
0	1	0	1
	0	0	1
		1	1

x	x'
0	1
1	0

NOT



OR



Postulates & Theorems

Post. 2	$x + 0 = x$	$x . 1 = x$
Post. 5 (Inverse)	$x + x' = 1$	$x . x' = 0$
Theorem 1	$x + x = x$	$x . x = x$
Theorem 2	$x + 1 = 1$	$x . 0 = 0$
Theorem 3 (Involution)	$(x')' = x$	
Post. 3, (Commutative)	$x + y = y + x$	$xy = yx$
Th. 4 (Associative)	$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$
Post. 4 (Distributive)	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
Th. 5 (DeMorgan)	$(x + y)' = x'y'$	$(xy)' = x' + y'$
Th. 6 (Absorption)	$x + xy = x$	$x(x + y) = x$

$x . x = xx + 0$

$= xx + xx'$

$= x(x + x')$

$= x . 1 = x$

$x(y+z)$

$\uparrow$

$\rightarrow x + xy + xz$

$+ yz$

Simplify :

(a) $AB + A'B = B(A+A') = B.1 = B$

(b) $xy + x'z + yz = xy + x'z + yz(x+x')$

$= xy + xyz + x'z + x'yz$

$= xy + x'z$

(c) $(ABC + A'BC' + A'B'C)' = (A'+B'+C')(A+B'+C)(A+B+C')$

Ex 2.2

$F_1' = (x'yz' + x'y'z)'$ Find dual

$= (x + y' + z) . (x + y + z')$ Take complement of each literal

22/1/18

LO : Express a Boolean function as Sum of Products/Minterms

or Product of Sums / Maxterms

Dual

* Consider Boolean function depending on 3 variables (x, y, z)

$\Rightarrow$ 8 combinations

x	y	z		Product yielding '1' Minterm	$\longleftrightarrow$ Dual	Sum yielding '0' Maxterms
0	0	0	0	$x'y'z'$ (m_0)		$x+y+z$ (M_0)
0	0	1	1	$x'y'z$ (m_1)		$x+y+z'$ (M_1)
0	1	0	0	$x'yz'$ (m_2)		$x+y'+z$ (M_2)
0	1	1	0	$x'yz$		$x+y'+z'$
1	0	0	1	$xy'z'$		$x'+y+z$
1	0	1	1	$xy'z$		$x'+y+z'$
1	1	0	1	xyz'		$x'+y'+z$
1	1	1	1	xyz		$x'+y'+z'$

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

$$f_1' = (x+y+z')\cdot(x'+y+z)\cdot(x'+y'+z') = M_1 M_4 M_7$$

Ex 2.4

$$F = A + B'C$$

$$A = A(B+B') = AB + AB'$$

$$= AB(C+C') + AB'(C+C')$$

$$= ABC + ABC' + AB'C + AB'C'$$

$$B'C = B'C(A+A') = AB'C + A'B'C$$

$$F = m_1 + m_4 + m_5 + m_6 + m_7$$

$$F = \Sigma(1, 4, 5, 6, 7)$$

Can this be expressed in maxterms?

$$F = (F')' = (\Sigma(0, 2, 3))' = (m_0 + m_2 + m_3)'$$

$$= M_0 \cdot M_2 \cdot M_3 = \Pi(0, 2, 3)$$

$$\Sigma(1, 4, 5, 6, 7) = \Pi(0, 2, 3)$$

Ex

$$F = (A + BC)(B + D)(C + AD)$$

25/1/18

LO : Gate-level minimization using Karnaugh map

3-variable Boolean function (A, B, C)

A \ BC	00	01	11	10
0	$A'B'C'$ (0)	$A'B'C$ (1)	$A'BC$ (3)	$A'BC'$ (2)
1	$AB'C'$ (4)	$AB'C$ (5)	ABC (7)	ABC' (6)

$F = \Sigma(3, 4, 6, 7)$

A \ BC	00	01	11	10
0			1	
1	1		1	1

$A'BC + ABC$
$= BC(A + A')$
$= BC \cdot 1$
$= BC$

AC'

$F = BC + AC'$

$F = \Sigma(0, 2, 4, 5, 6)$

A \ BC	00	01	11	10
0	1			1
1	1	1		1

C'

AB'

$B'C'$

BC'

$F = AB' + B'C' + BC'$

$= B'(A + C') + BC'$

$F = C' + AB'$

3 var K-map

3 → 1 square → 3 literals

2 → 2 squares → 2 "

1 → 4 " → 1 "

$$F = \underset{(1,3)}{A'C} + \underset{(2,3)}{A'B} + \underset{(5)}{AB'C} + \underset{(3,7)}{BC}$$

Ex

$$A'C(B+B') = \underset{(3)}{A'BC} + \underset{(1)}{A'B'C}$$

A \ BC	00	01	11	10
0		1	1	1
1		1	1	

A'B

C

$$F = A'B + C$$

$$F(x,y,z) = \Sigma(1,2,5,6,7)$$

Simplify through (a) Sum of products (Minterms)

(b) Product of sums (Maxterms)

x \ yz	00	01	11	10
0	0	1	0	1
1	0	1	1	1

$y'z$ $\quad xz \quad$ yz'

$\cancel{xy}$ Redundant

$$F = y'z + xz + yz'$$

$$= z(x+y') + yz'$$

$$\bar{F} = \Sigma(0,3,4) = y'z' + x'yz$$

$$F = (F')' = \overline{y'z' + x'yz}$$

$$\Pi(0,3,4) = (y+z).(x+y'+z')$$

$$= xy + xz + y'z + yz'$$

$$F = z(x+y') + yz' + xy$$

4 - variable K-map :

1 square → 4 literals

2 squares → 3 "

4 " → 2 "

8 " → 1 "

16 " → always 1

wx \ yz	00	01	11	10
00	1 (0)	1 (1)	(3)	1 (2)
01	1 (4)	1 (5)	(7)	1 (6)
11	1 (12)	1 (13)	(15)	1 (14)
10	1 (8)	1 (9)	(11)	(10)

$F = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

$F = y' + w'z' + xz'$

$= y' + z'(w' + x)$

$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

Simplify

AB \ CD	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

$B'C' + B'D' + A'CD'$

$F = B'(C' + D') + A'CD'$

Prime implicant → product term obtained by combining the max. possible number of adj squares

$$F = \Sigma(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$$

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	
11		1	1	
10	1	1	1	1

Essential prime implicants

$B'D' + BD + AD + CD$

$B'D' + BD + CD + AB'$

Procedure: (1) Identify essential prime implicants

(2) Cover the remaining minterms

(3) Ensure that there are no redundant terms

Don't care conditions.

* Some applications do not use all 16 4-bit combinations

$\Rightarrow$ don't care conditions

$\Rightarrow$ can be used as 0 or 1 conveniently

$$F = \Sigma(1, 3, 7, 11, 15)$$

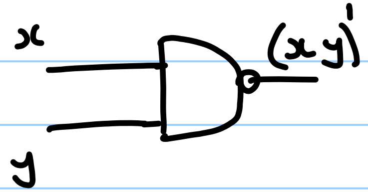
$$d = \Sigma(0, 2, 5)$$

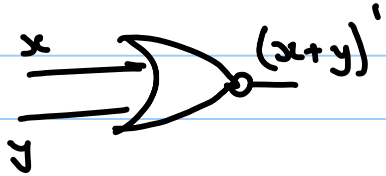
wx \ yz	00	01	11	10
00	X	1	1	X
01		X	1	
11			1	
10			1	

$\cancel{w'x'z}$

$yz + w'z$

$+ w'x'$





x	y	NAND	NOR
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

Simplify

$$F = \Sigma(0, 1, 3, 4, 6, 9, 10, 11)$$

$$d = \Sigma(5, 7, 12, 13, 14, 15)$$

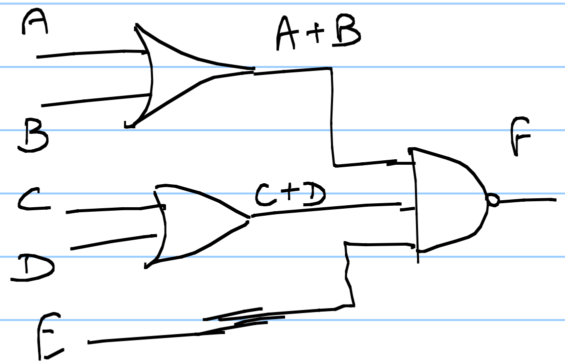
5/2/18

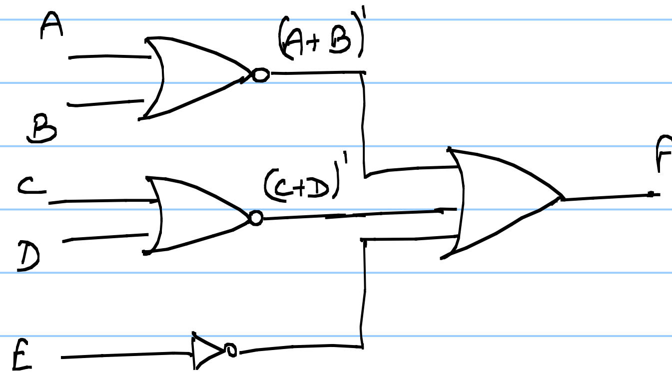
NAND / NOR Implementation:

(1) Obtain a simplified Boolean function

(2) Convert the function to NAND/NOR logic

$$F = [(A+B)(C+D)E]' = (A+B)' + (C+D)' + E'$$





In general, AND-NOR (or) NAND-AND → SOP

OR-NAND (or) NOR-OR → POS

Ex 3-10

$x \backslash yz$	00	01	11	10
0	1	0	0	0
1	0	0	0	1

OR-NAND   AND-NOR

NOR-OR   NAND-AND

$$F = x'y'z' + xyz'$$

$$= [(x+y+z).(x'+y'+z)]'$$

OR-NAND

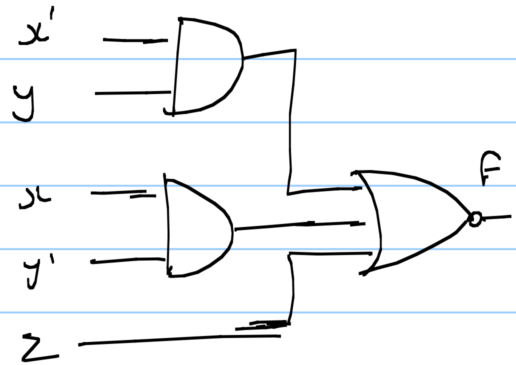
$$= (x+y+z)' + (x'+y'+z)'$$

NOR-OR

$$F = (F')' = (z + x'y + xy')' = (xy')'.(x'y)'z'$$

AND-NOR

NAND-AND



Ex-OR Gates: Very useful for error detection/correction

$$F = x \oplus y = xy' + x'y$$

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

Identities:

$$x \oplus 0 = x$$

$$x \oplus 1 = x'$$

$$x \oplus x = 0$$

$$x \oplus x' = 1$$

$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

$F = A \oplus B$ —

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$F = A \oplus B \oplus C$

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

$F = A \oplus B \oplus C \oplus D$

AB \ CD	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

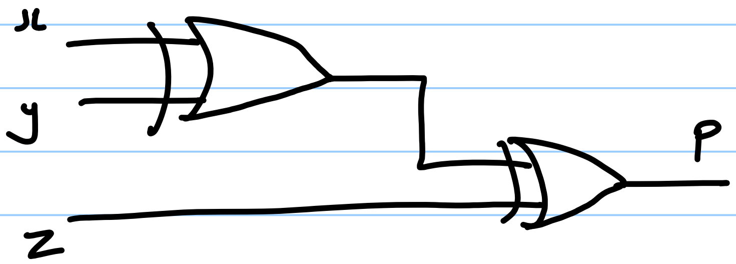
Parity Detection / Correction :

At transmitter

x	y	z	Parity (Even)
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

→ such that total # of 1s is even

$P = x \oplus y \oplus z$



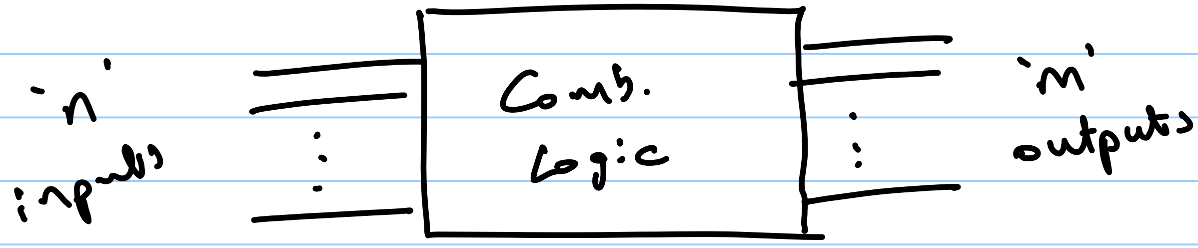
At Destination

$C = x \oplus y \oplus z \oplus P$

x	y	z	P	c
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
1	1	0	1	1

Errors

COMBINATIONAL LOGIC CIRCUITS



Design Procedure:

1) From the given spec., # of inputs/outputs are determined, assign symbol

2) Derive the truth table that defines the required relationship between input and output

3) Obtain simplified Boolean func. for each output

4) Draw the logic diagram & verify functionality

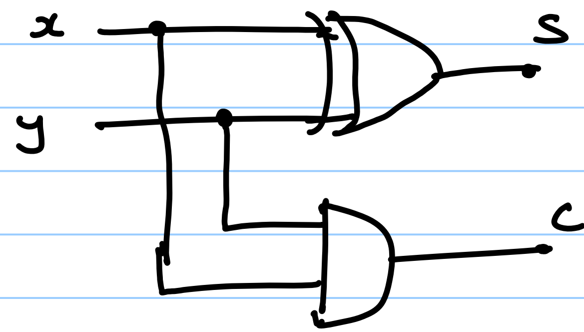
Half Adder

$x, y \rightarrow S, c$

Inputs		Outputs	
x	y	c	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = xy' + x'y = x \oplus y$$

$$c = xy$$



Full Adder

$x, y, z \rightarrow s, c$

x	y	z	c	s
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

S

x \ yz	00	01	11	10
0		1		1
1	1		1	

$$S = x \oplus y \oplus z$$

C

x \ yz	00	01	11	10
0			1	
1		1	1	1

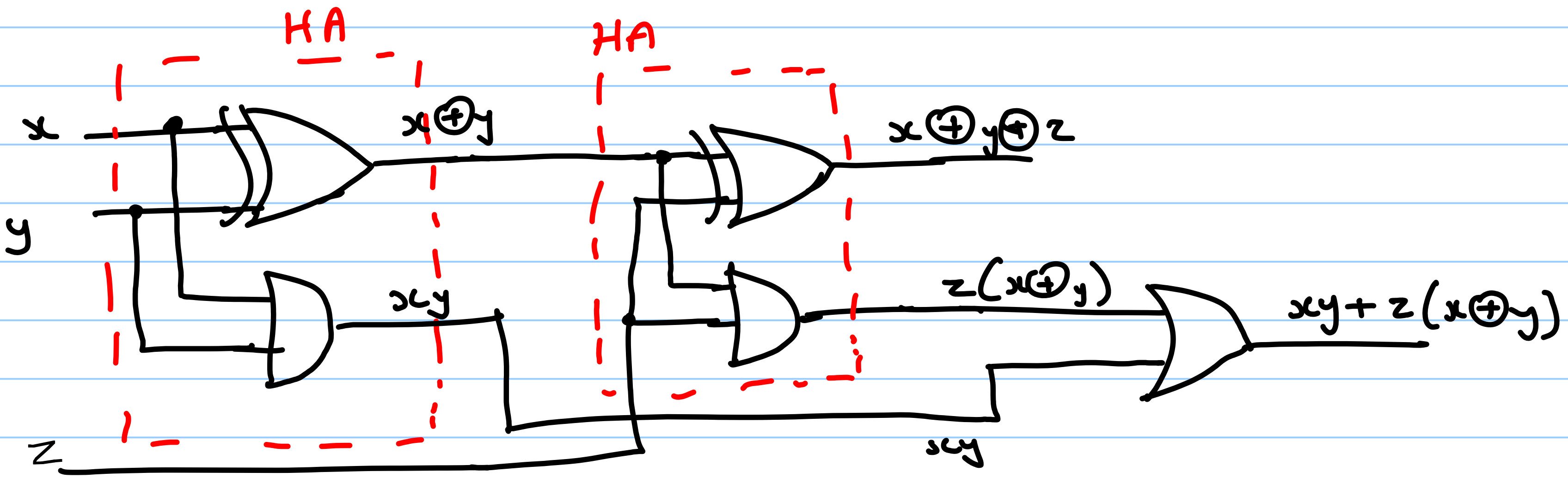
$$C = xy + xz + yz$$

$$= xy + z(x+y)$$

$$= xy + z(xy' + x'y)$$

$$= xy + z(x \oplus y)$$

$S = x \oplus y \oplus z \qquad C = xy + z(x \oplus y)$



FA → 2 HAs + OR

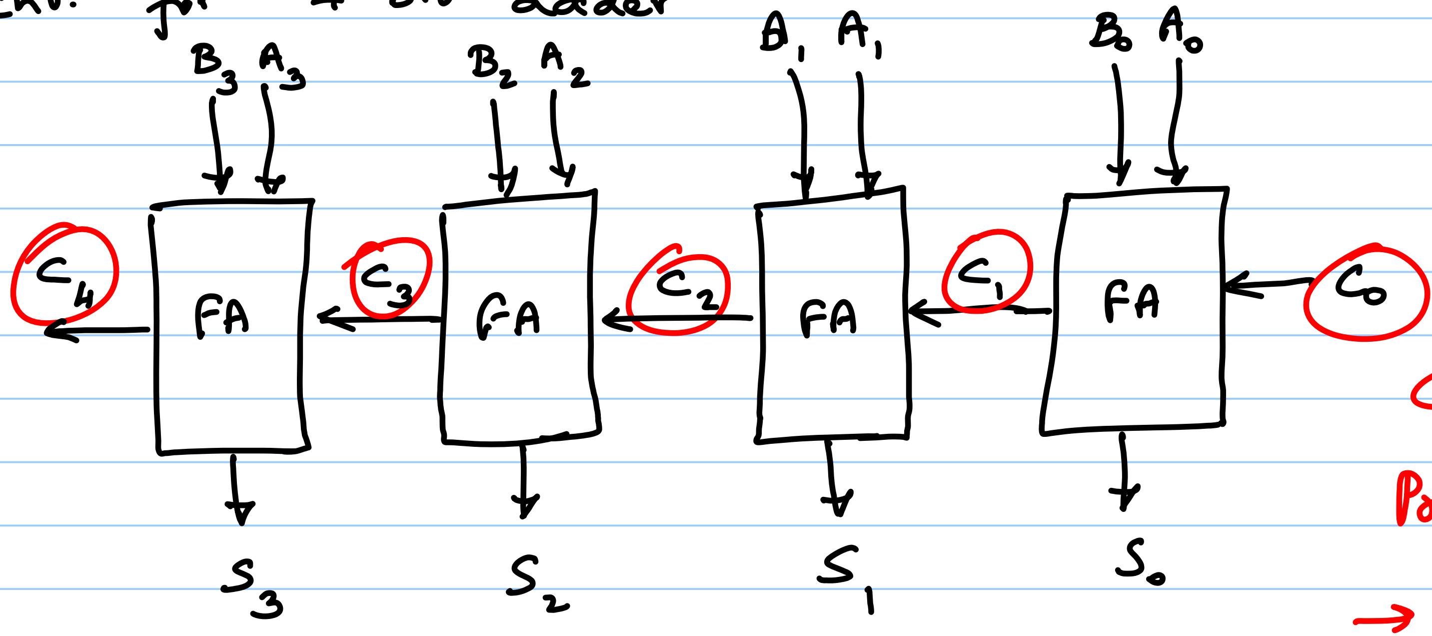
Suppose we want to add $A = 1011 \quad B = 0011$

Ripple Carry Adder

	0	1	1	0 (C_0)	C_t
Augend	1	0	1	1	A_t
Addend	0	0	1	1	B_t
	1	1	1	0	S_t
	0	0	1	1	C_{t+1}

$$C_{t+1} = A_t B_t + C_t (A_t \oplus B_t)$$

Comb. ckt. for 4-bit adder



Carry Propagation → Delay

Reduce delay → Carry Look Ahead Logic

Let us define $P_t = A_t \oplus B_t$

$G_t = A_t B_t$ $\Rightarrow S_t = P_t \oplus C_t$

$C_{t+1} = G_t + P_t C_t$

$C_0 \rightarrow$ input carry

$C_1 = G_0 + P_0 C_0$

Carry Look Ahead

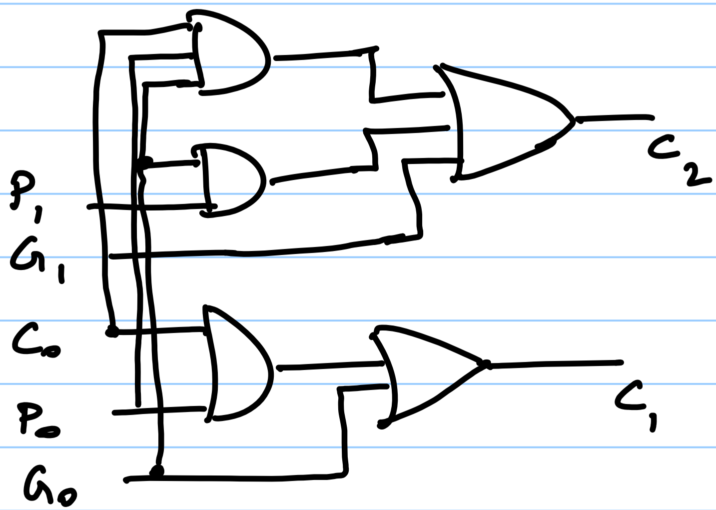
$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$

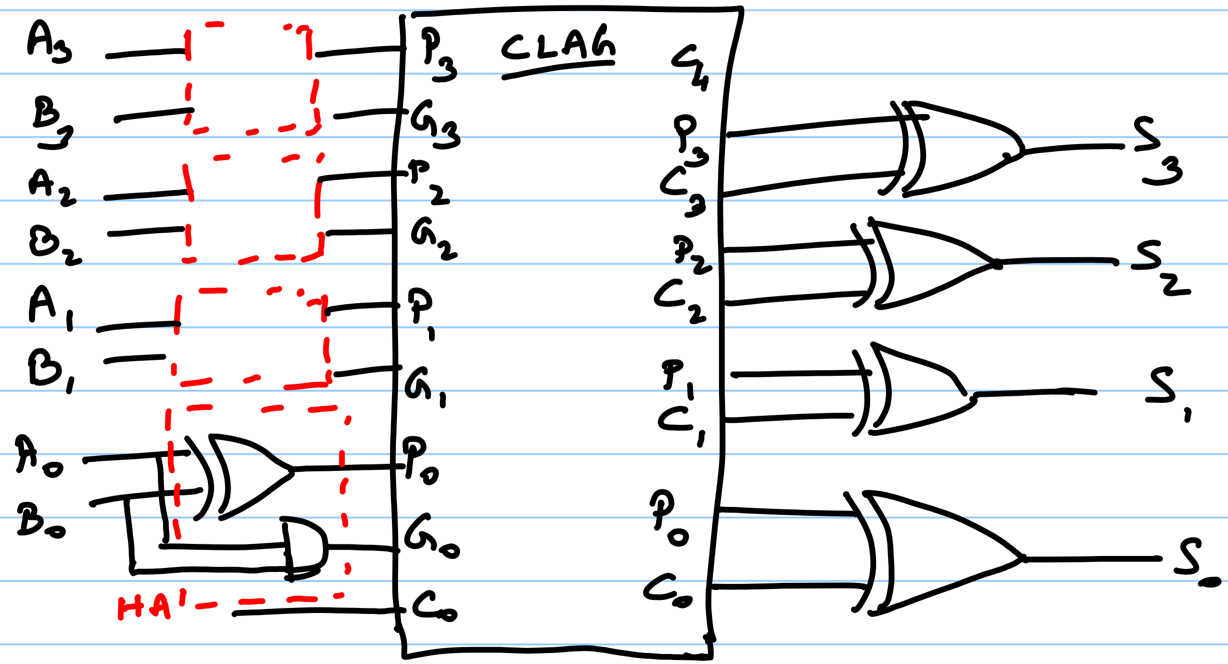
$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$C_4 = G_3 + P_3 C_3 =$

$\Rightarrow$ All carries determined simultaneously

Speed gained at the expense of additional hardware





CLAG
HA'
A3
B3
A2
B2
A1
B1
A0
B0
P3
G3
P2
G2
P1
G1
P0
G0
C0
G4
P3
C3
P2
C2
P1
C1
P0
C0
S3
S2
S1
S0

BCD Adder: How to add two decimals?

$(11)_{10} \rightarrow (1\,0001)_{BCD}$

$(5)_{10} \rightarrow (00\,101)_{BCD}$

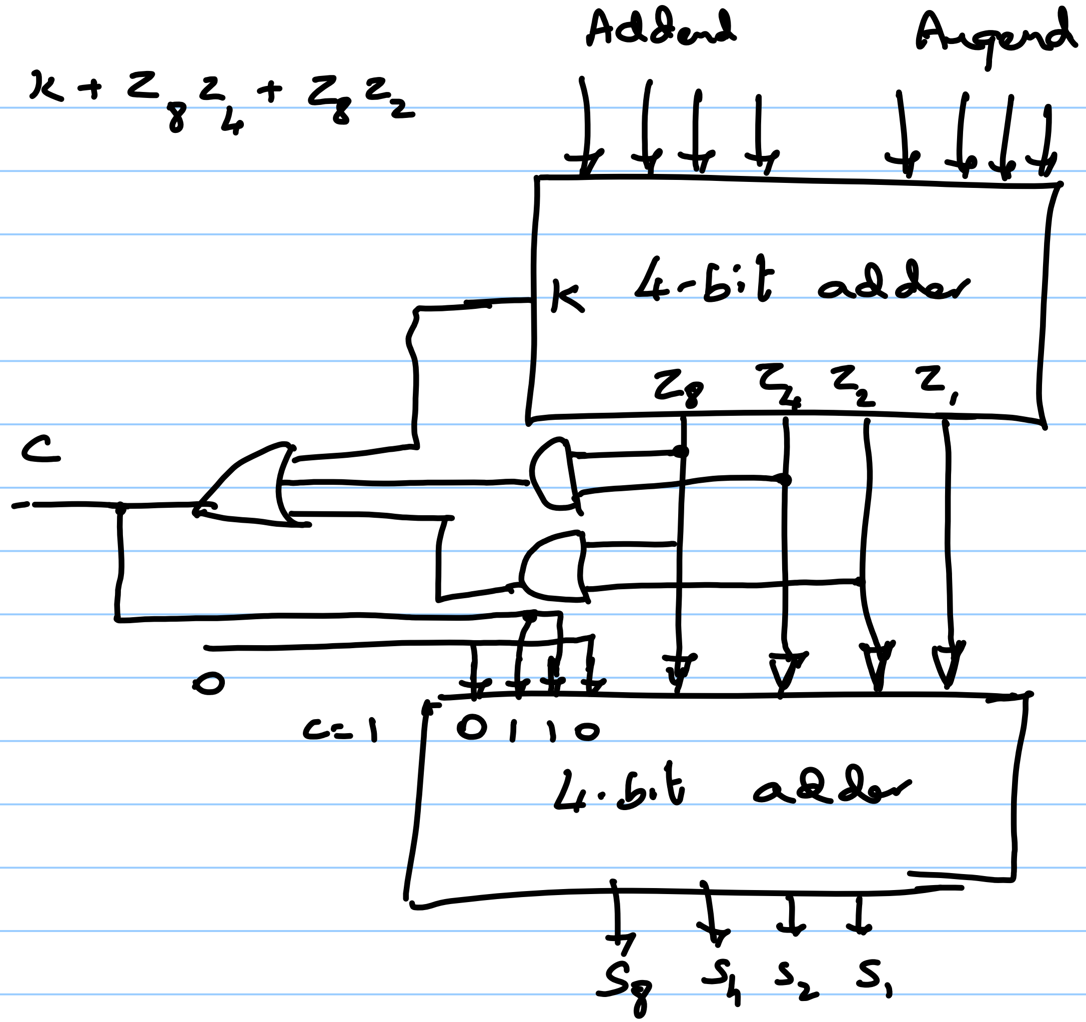
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0			1	0	0					2
0			1	1						3
0		1	0	0						4
0		1	0	1						5
0		1	1	0						6
0		1	1	1						7
0	1	0	0	0						8
K 0	1	0	0	1	C 0	1	0	0	1	9
0	1	0	1	0 → (0110)	1	0	0	0	0	10
0	1	0	1	1 →	1	0	0	0	1	11
0	1	1	0	0 →	1	0	0	1	0	12
0	1	1	0	1 →	1	0	0	1	1	13
0	1	1	1	0 →	1	0	1	0	0	14
0	1	1	1	1 →	1	0	1	0	1	15
1	0	0	0	0 →	1	0	1	1	0	16

$C = K + Z_8 Z_4 + Z_8 Z_2$

C

Z_8Z_4 \ Z_2Z_1	00	01	11	10
00				
01				
11	1	1	1	1
10			1	1

$$C = K + Z_8 Z_4 + Z_8 Z_2$$


Addend
Augend
K
4-bit adder
Z_8 Z_4 Z_2 Z_1
C
0
C=1
0 1 1 0
4-bit adder
S_8 S_4 S_2 S_1

Binary Subtraction

$(49)_{10} \rightarrow (00110001)_2 \longrightarrow -(49)_{10} \rightarrow (11001111)_2$

$(29)_{10} \rightarrow (00011101)_2 \rightarrow -(29)_{10} \rightarrow (11100011)_2$

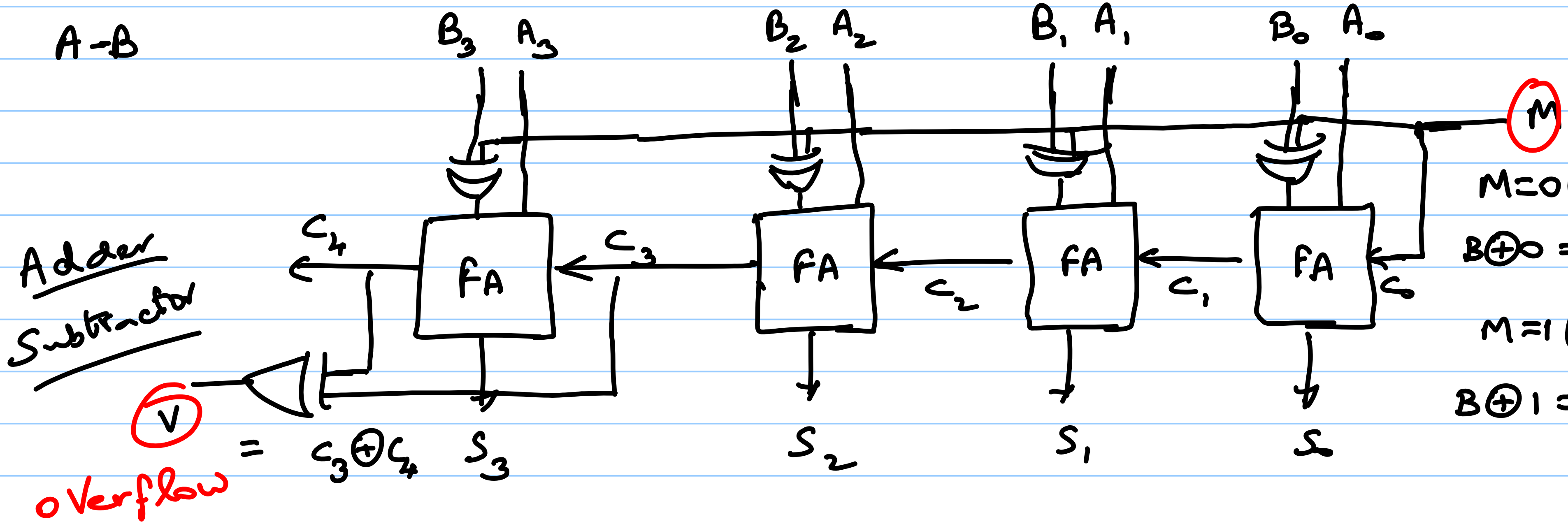
$$
\begin{array}{rcr}
 & & 111111 \\
+29 & \rightarrow & 00011101 \\
-49 & \rightarrow & 11001111 \\
\hline
 & & (11101100)_2
\end{array}
$$

$(11101100)_2 \rightarrow -(00010100)_2$

$-(20)_{10}$

Suppose you have two 4-bit unsigned binary numbers (A, B)

A − B



$M=0$ $(A+B)$

$B \oplus 0 = B$

$M=1$ $(A-B)$

$B \oplus 1 = B'$

$V = C_3 \oplus C_4$

If A & B are of same sign & high value

0 1 → overflow

$(+70)_{10} \rightarrow (01000110)_2$

$(+80)_{10} \rightarrow (01010000)_2$

$(+150)_{10} \leftarrow$ (0) 10010110

1 0 1 1

$(-70)_{10} \rightarrow (10111010)_2$

$(-80)_{10} \rightarrow (10110000)_2$

$(-150)_{10} \leftarrow$ (1) 11101010

Lets consider 4-bit binary numbers

2 (b)

M	A	B	
0	1000	1001	
	↓	↓	
	$(-8)_{10}$	$(-7)_{10}$	

$$
\begin{array}{r}
1\ 0 \quad \text{Overflow} \\
1000 \\
+\ 1001 \\
\hline
(1)\ 0001
\end{array}
$$

$(-15)_{10}$

$V = C_3 \oplus C_4 = 1$

$\Rightarrow$ overflow

2(c)

M	A	B
1	1100	1000
	↓	↓
	$(-4)_{10}$	$(-8)_{10}$

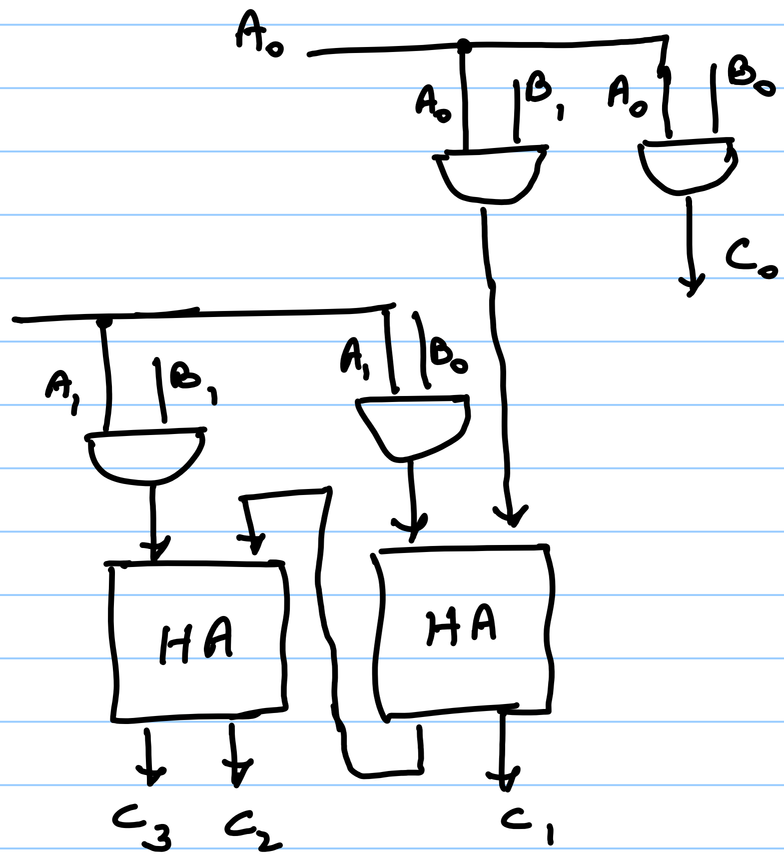
$$
\begin{array}{rr}
 & 1\,1\,1\,1 \\
A \rightarrow & 1100 \\
B' \rightarrow & 0111 \\
 & 1 \;\; C_0 \\
\hline
(+4)_{10} \leftarrow & 0100
\end{array}
$$

$V = 0$

$\Rightarrow$ no overflow

Binary Multiplier :

Multiplicand (B) $\rightarrow$			B_1	B_0
Multiplier (A) $\rightarrow$			A_1	A_0
			A_0B_1	A_0B_0
	A_1B_1	A_1B_0		$\downarrow$
	A_1B_1	$A_0B_1+A_1B_0$		A_0B_0
C_3	C_2	C_1		C_0

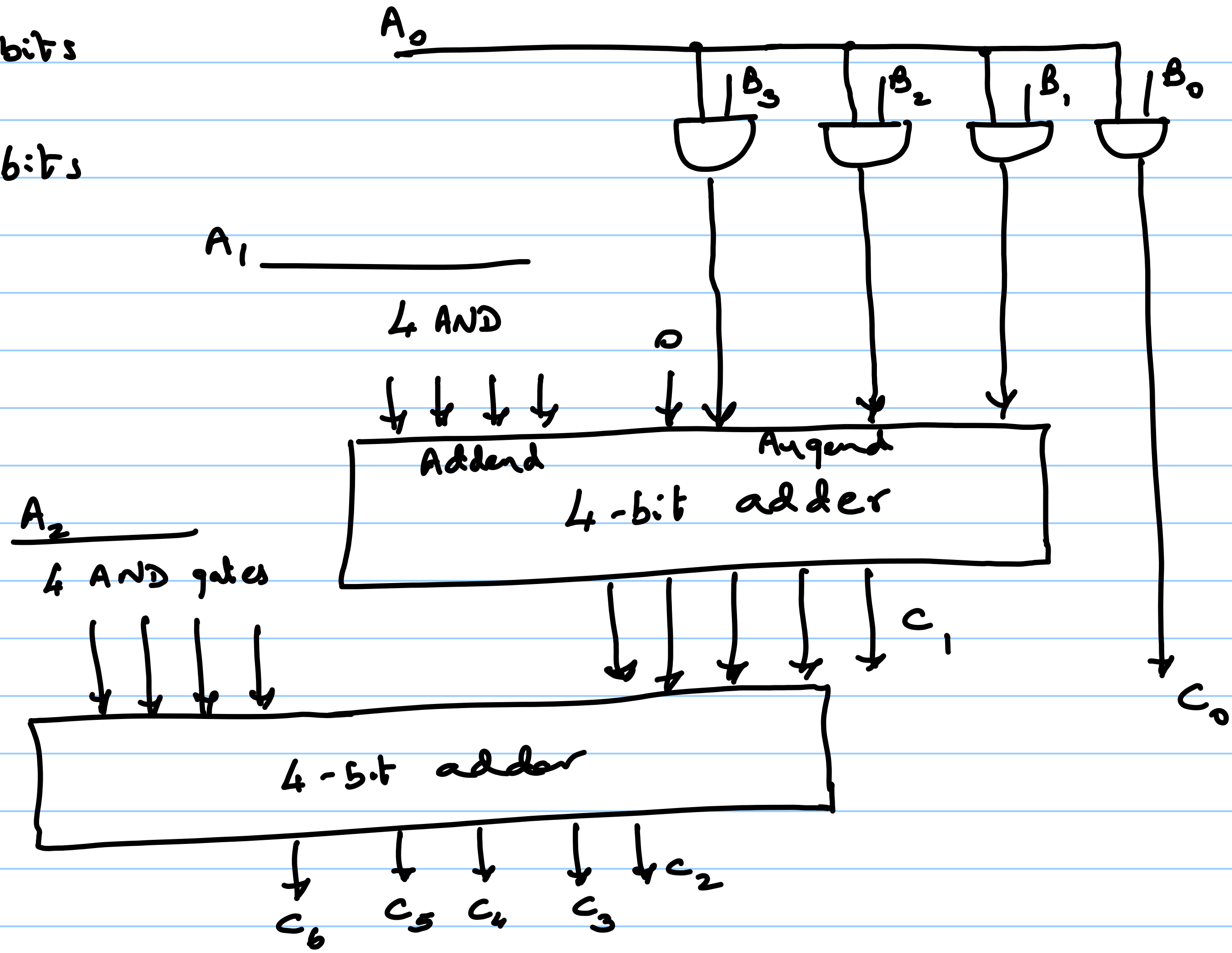


$A \rightarrow J$ bits $\Rightarrow$ $J \times K$ AND gates

$B \rightarrow K$ bits $\quad$ $(J-1)$ K-bit adders

A → 3 bits

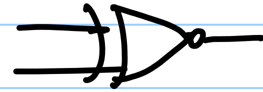
B → 4 bits



Magnitude Comparator :

$A = A_3 \quad A_2 \quad A_1 \quad A_0$

$B = B_3 \quad B_2 \quad B_1 \quad B_0$



ex-NOR

		x_t
0	0	1
0	1	0
1	0	0
1	1	1

$(A=B) = $

$A_3 = B_3 \quad x_3$

$A_2 = B_2 \quad x_2$

$A_1 = B_1 \quad x_1$

$A_0 = B_0 \quad x_0$

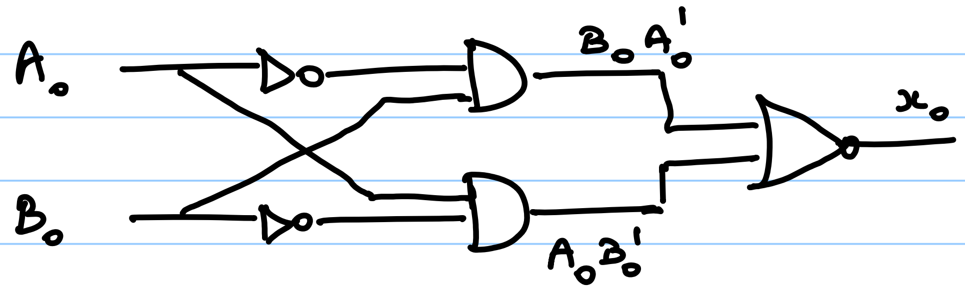
$x_t = A_t B_t + A_t' B_t'$

$(x_0 x_1 x_2 x_3) = 1$

AND gate

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$$



Decoders:

n coded inputs $\rightarrow$ 2^n unique outputs

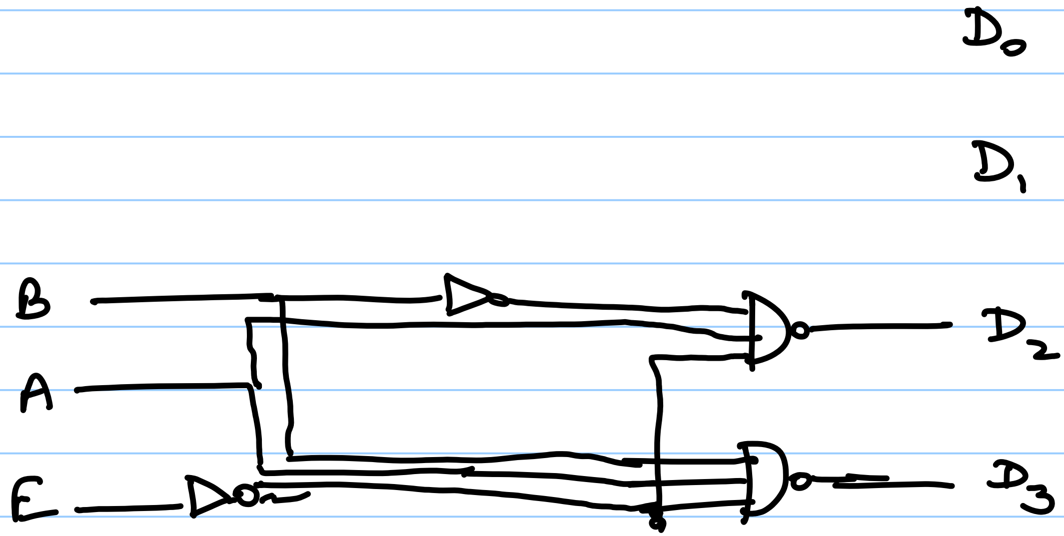
$\rightarrow$ Typically deployed with ENABLE (0 or 1)
(0 → Active Low; 1 → Active HIGH)

$\rightarrow$ NAND gates are preferred

Active LOW $\rightarrow$

E	A	B	D_0	D_1	D_2	D_3
1	x	x	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

NAND $\rightarrow (AB)'$



Encoder :

2^n inputs $\longrightarrow$ n outputs

Priority Encoder

Validity bit

D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Due to priority assigned

$V = D_0 + D_1 + D_2 + D_3$

$x = D_3 + D_2$

$y = D_3 + D_1 D_2'$

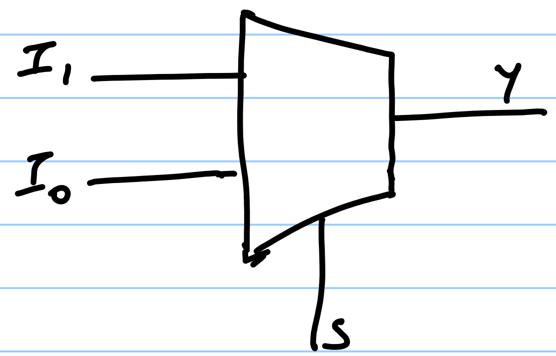
x

$D_0D_1 \backslash D_2D_3$	00	01	11	10
00	X	1	1	1
01		1	1	1
11		1	1	1
10		1	1	1

y

$D_0D_1 \backslash D_2D_3$	00	01	11	10
00	X	1	1	
01	1	1	1	
11	1	1	1	
10		1	1	

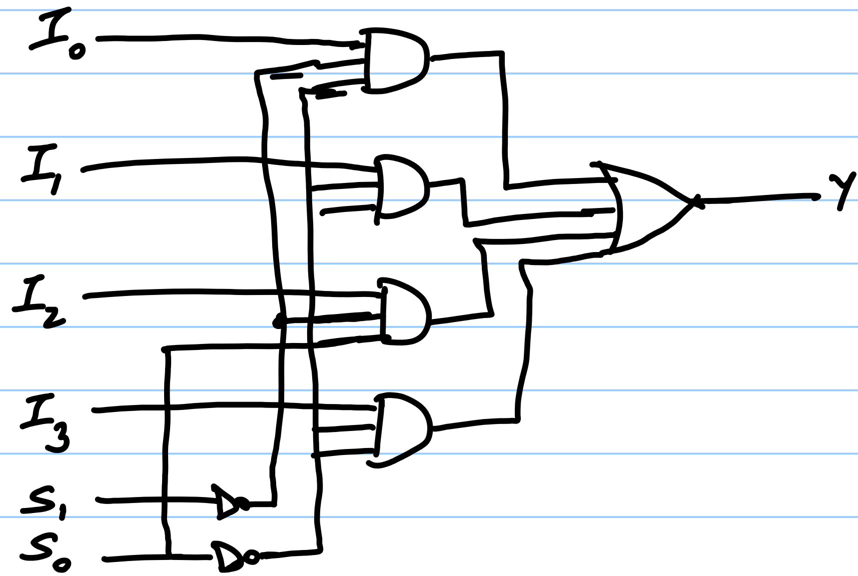
Multiplexers :



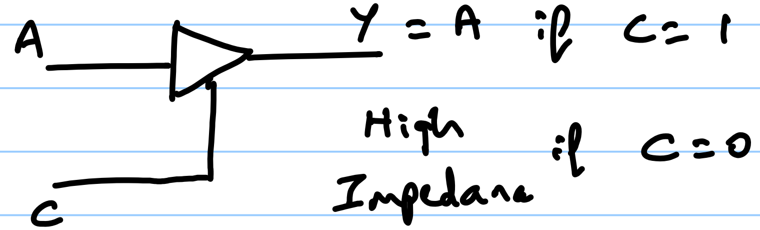
* 4 input streams → I_0 I_1 I_2 I_3

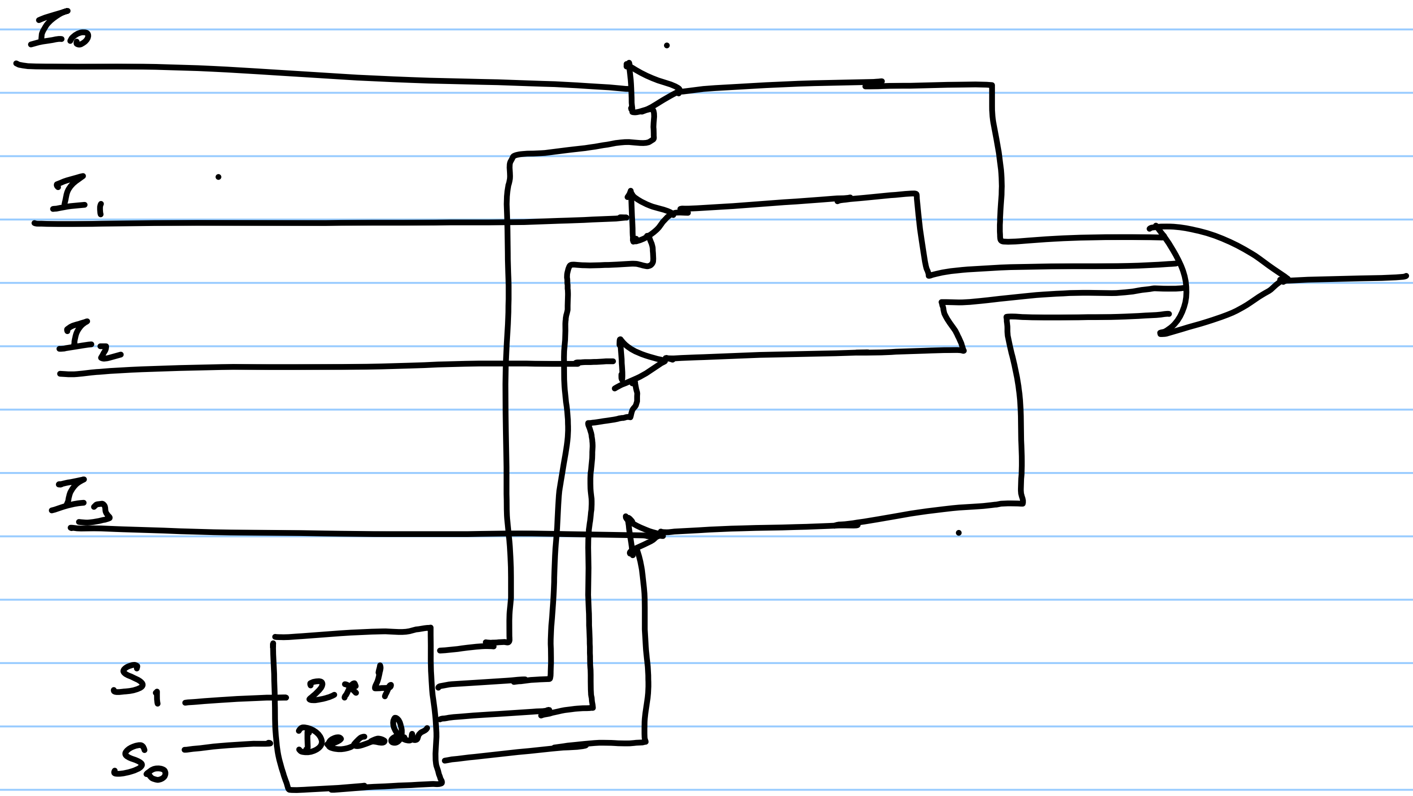
Select → S_0 S_1

S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



Tri-State Gate:



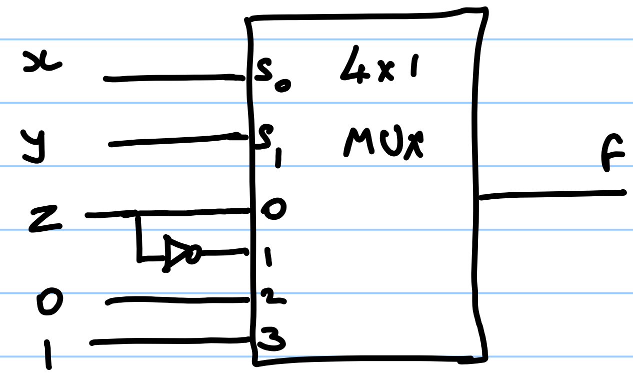


Boolean Function Implementation:

$$F(x, y, z) = \Sigma\ 1, 2, 6, 7$$

n - variables $\rightarrow$ $n-1$ select bits, 2^{n-1} inputs

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	



Ex:

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

→ Represent using 8x1 MUX

A	B	C	D	F	
0	0	0	0	0	f=D
0	0	0	1	1	
0	0	1	0	0	f=D
		1	1	1	
	1	0	0	1	f=D'
	1	0	1	0	
	1	1	0	0	f=0
	1	1	1	0	
1	0	0	0	0	f=0
1	0	0	1	0	
1	0	1	0	D	f=D
1	0	1	1	1	
1	1	0	0	1	

Construct 2 - 4 line decoder w/ Active HIGH Enable

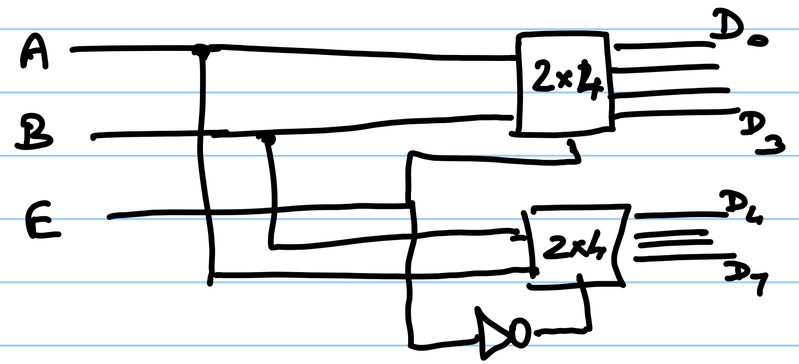
& with two such decoders, construct 3x8 decoder

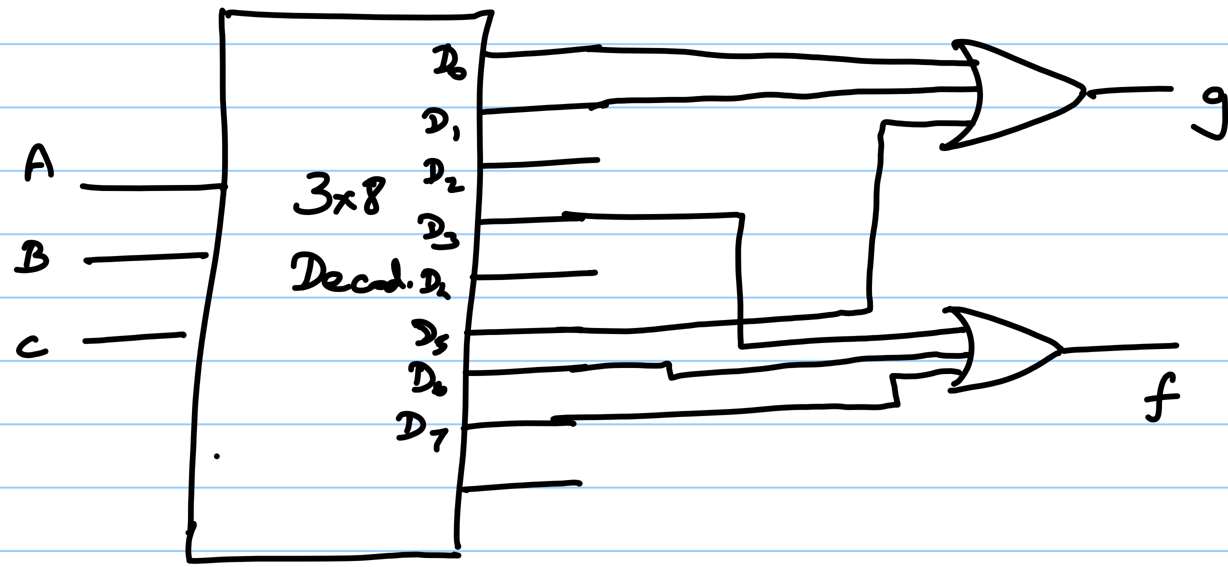
Realize $f(A, B, C) = \bar{A}BC + AB\bar{C} + ABC$

$g(A, B, C) = A\bar{B}C + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$

A	B	E	D_0	D_1	D_2	D_3
x	x	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

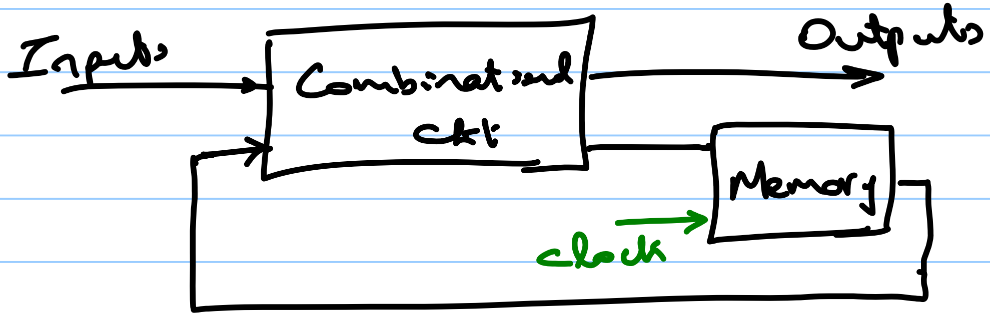
E ———





A
B
C
3x8
Decod.
D0
D1
D2
D3
D4
D5
D6
D7
g
f

Sequential Circuits



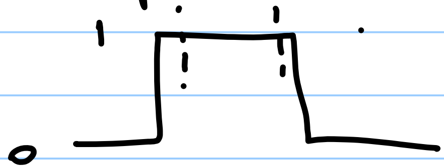
Synchronous
- clock

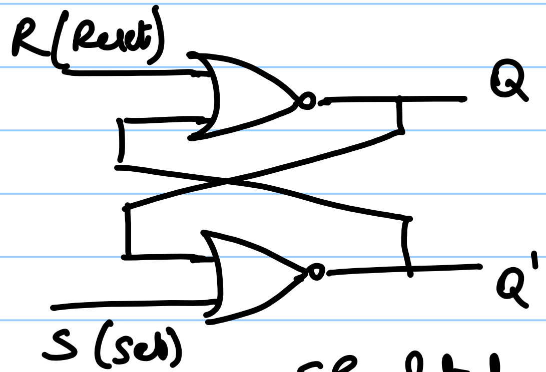
Asynchronous
- reliability

Set Reset

S-R Latch

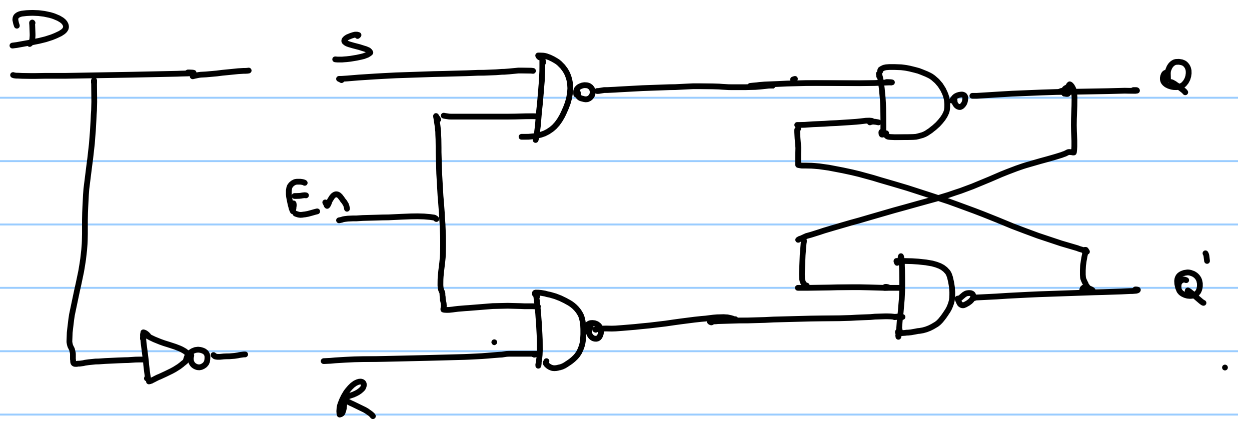
NOR Implementation





SR latch

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(S=1, R=0)
0	1	0	1	
0	0	0	1	(S=0, R=1)
1	1	0	0	(forbidden)



$\overline{S}\,\overline{R}$ latch

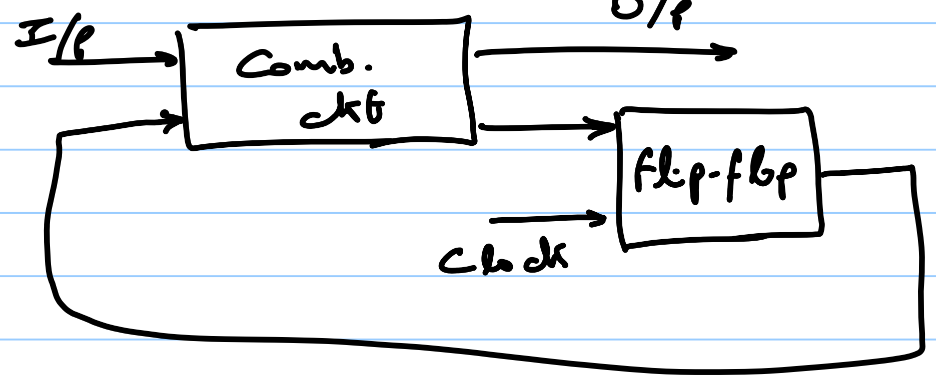
S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(S=1, R=0)
0	1	1	0	
1	1	1	0	(S=0, R=1)
0	0	1	1	(forbidden)

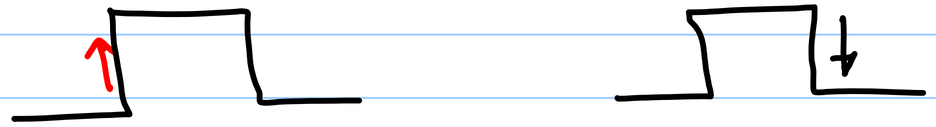
En	S	R	Q Q'
0	x	x	No change
1	0	0	No change
1	0	1	Q=0 ; reset state
1	1	0	Q=1 ; set state
1	1	1	Forbidden

D - Latch
(Avoids forbidden state)

En	D	Q	Q'
0	x		
1	0	Q =0 (Reset)	
1	1	Q =1 (Set)	

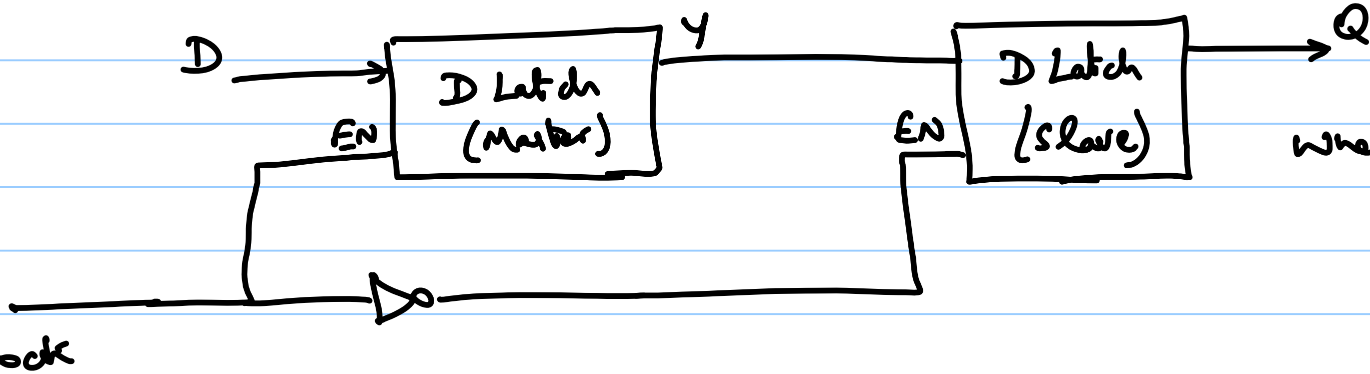
Flip-Flops:





Positive trigger   Negative trigger

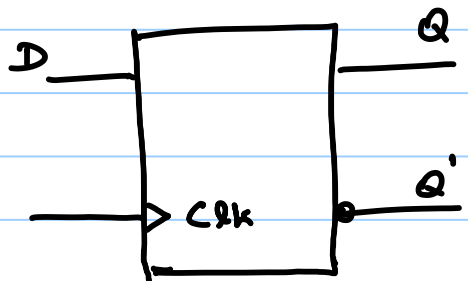
Edge Triggered D Flip-Flop: (Negative trigger)



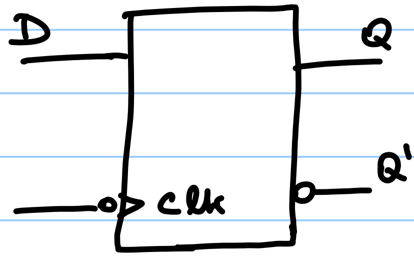
when clk → 0, Q → Y

clk → 1, Y → D

clk → 0, Q → Y = D



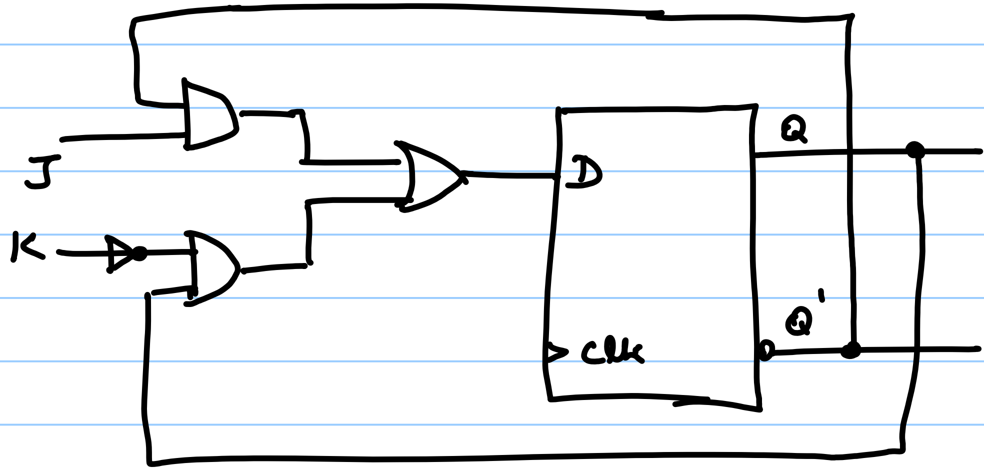
Positive



Negative

JK flip-flop:

$D = JQ' + QK'$



J	K	Q_{t+1}
0	0	Q_t
0	1	0 (R)
1	0	1 (S)
1	1	Q_t'

23/2/18

LO: Analyze clocked sequential circuits

↓

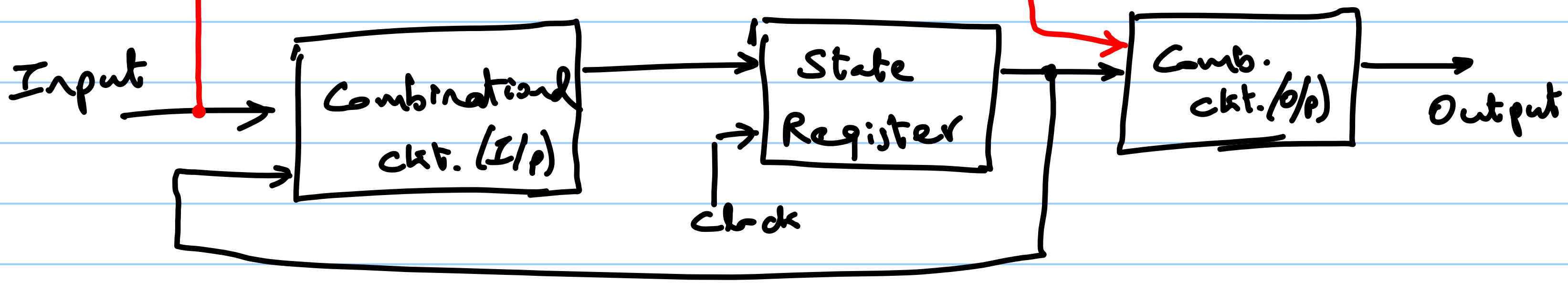
Finite State Machines

Mealy Model

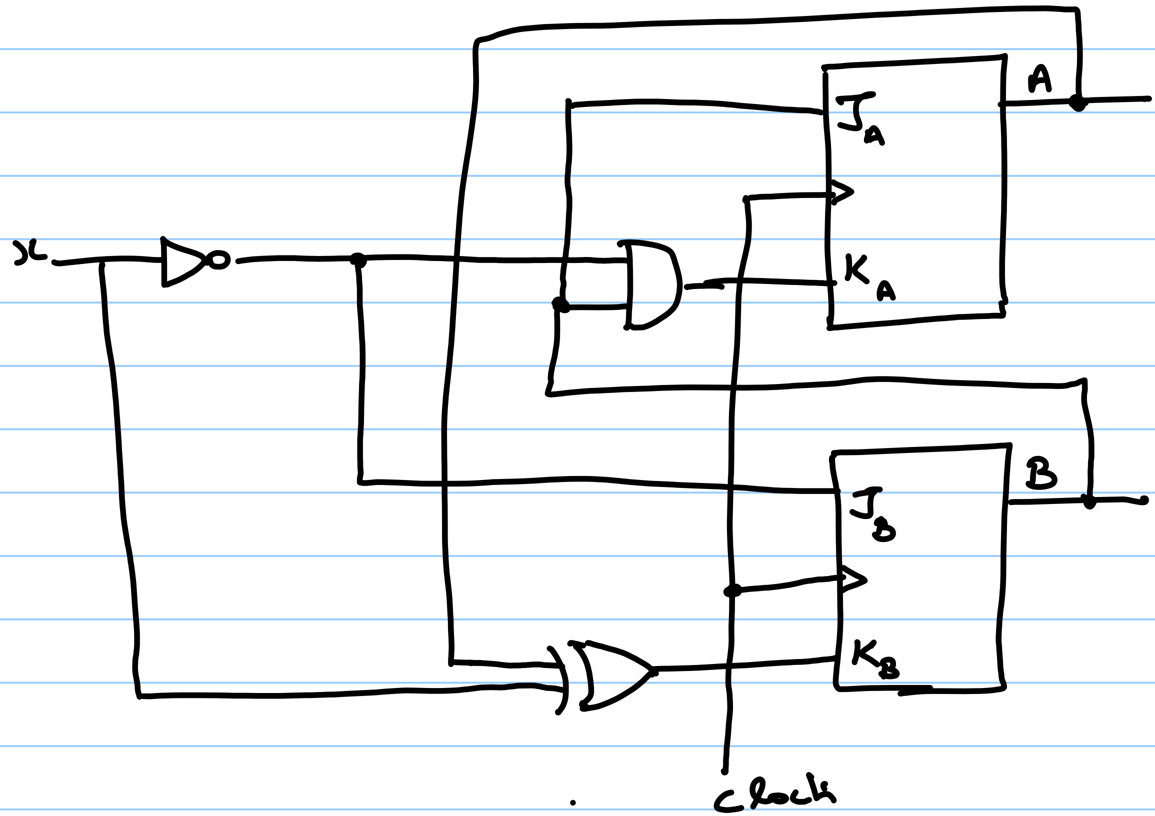
Moore Model

Input is synchronous to clock

$y = (A+B)x'$



Ex: JK Flip-flop ckt.



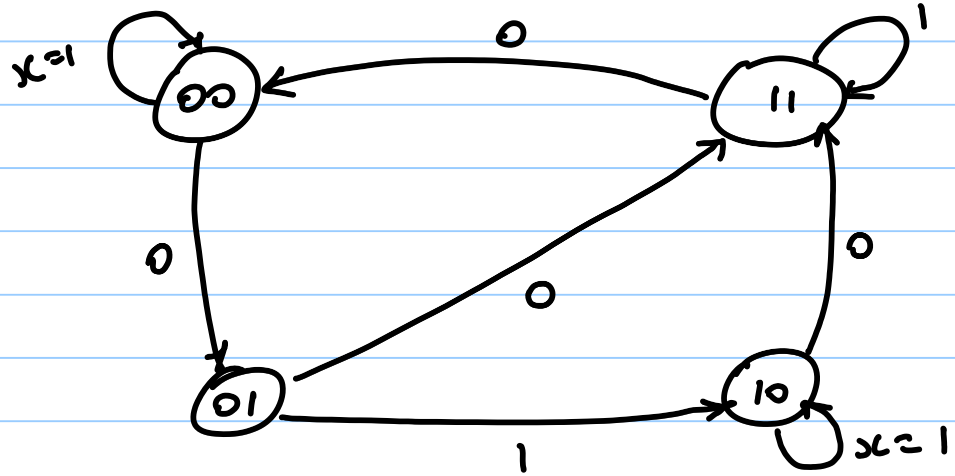
$J_A = B$

$K_A = Bx'$

$J_B = x'$

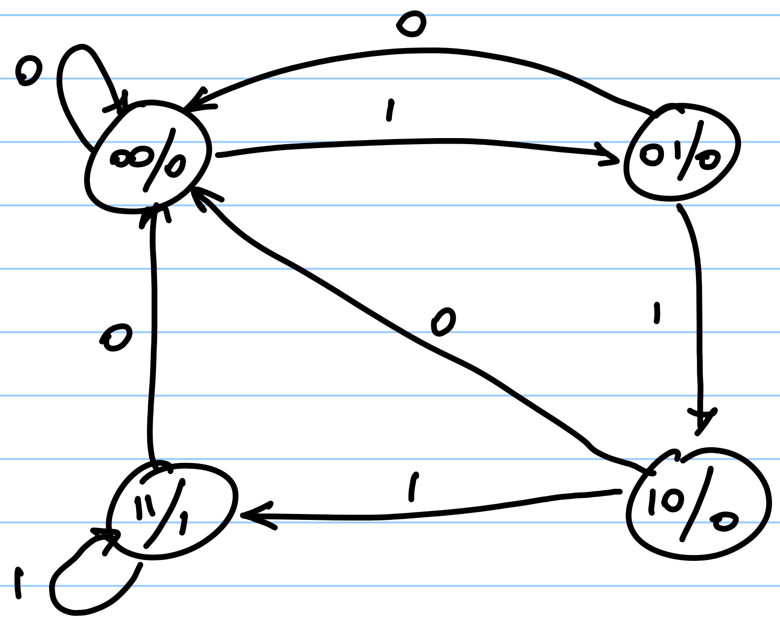
$K_B = A \oplus x$

Present State		Input	Next State		Flip-flop inputs			
					$(=B)$	(Bx')	(x')	$(A\oplus x)$
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0



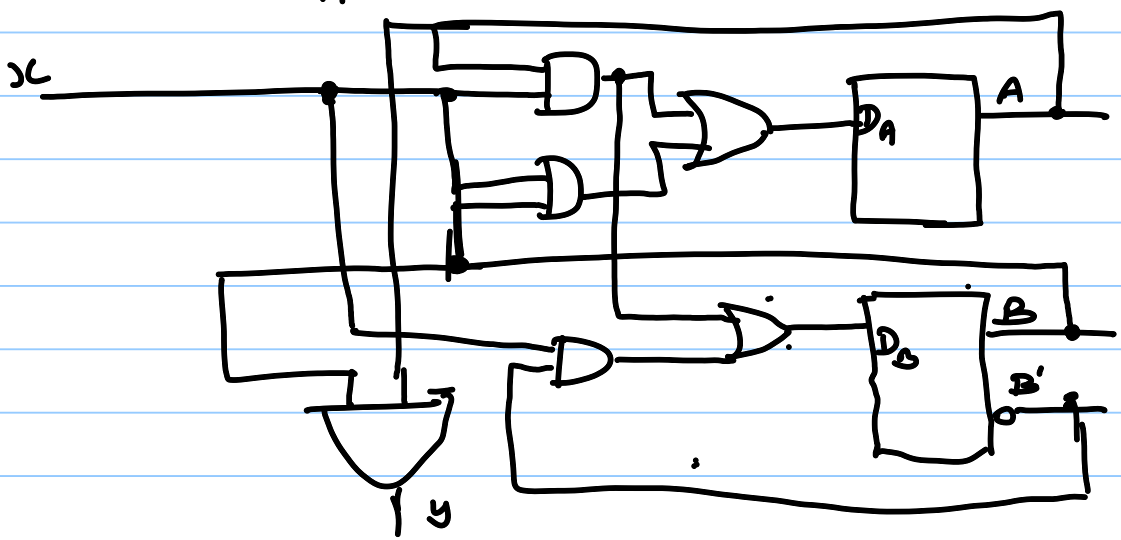
1/3/18

Design of Sequential Circuits using flip-flops :

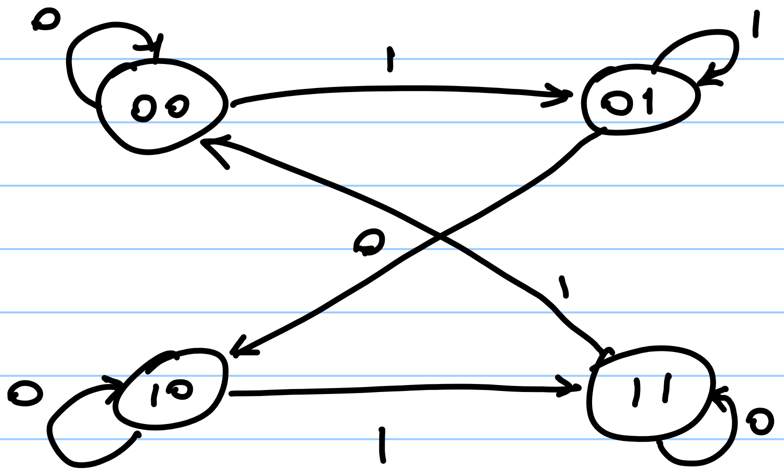


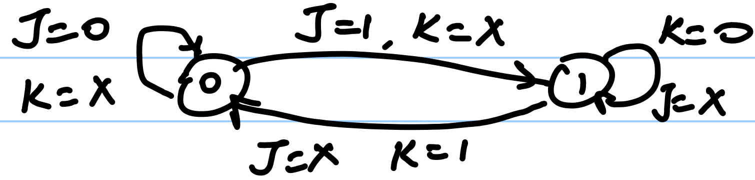
Present A B	Input x	Next A B	Output y
1 0	0	0 0	0
1 0	1	1 1	0
1 1	0	0 0	1
1 1	1	1 1	1

$D_A = Ax + Bx \qquad D_B = Ax + B'x \qquad y = AB$



Sequential Circuit w/ JK flip-flop





Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Present		Input	Next		J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	1	0	1	x	x	1
0	1	1	0	1	0	x	x	0
1	0	0	1	0	x	0	0	x
1	0	1	1	1	x	0	1	x
1	1	0	1	1	x	0	x	0
1	1	1	0	0	x	1	x	1

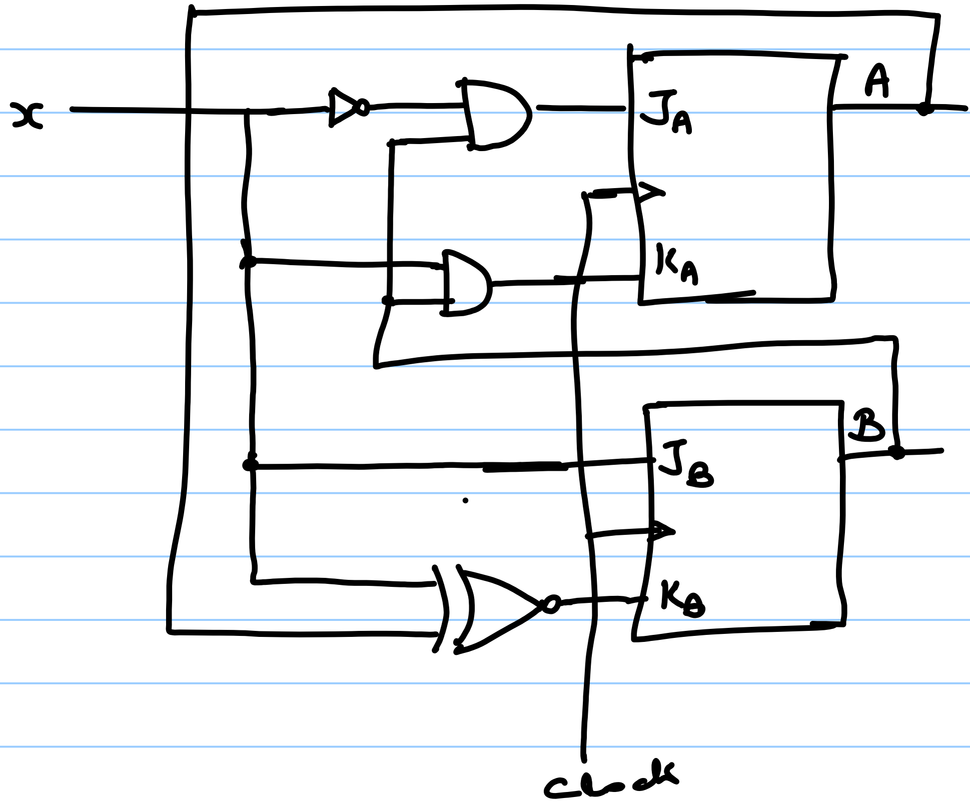
$J_A = Bx'$

A \ Bx	00	01	11	10
0				1
1	x	x	x	x

$K_A = Bx$

$J_B = x$

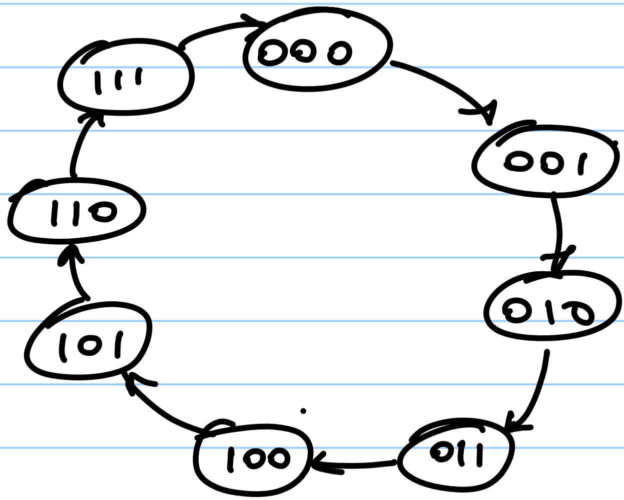
$K_B = Ax + A'x' = (A \oplus x)'$



x
A
J_A
K_A
B
J_B
K_B
clock

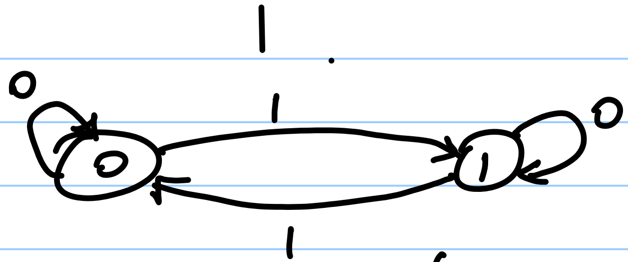
Synthesis using T flip-flop:

Three-bit counter



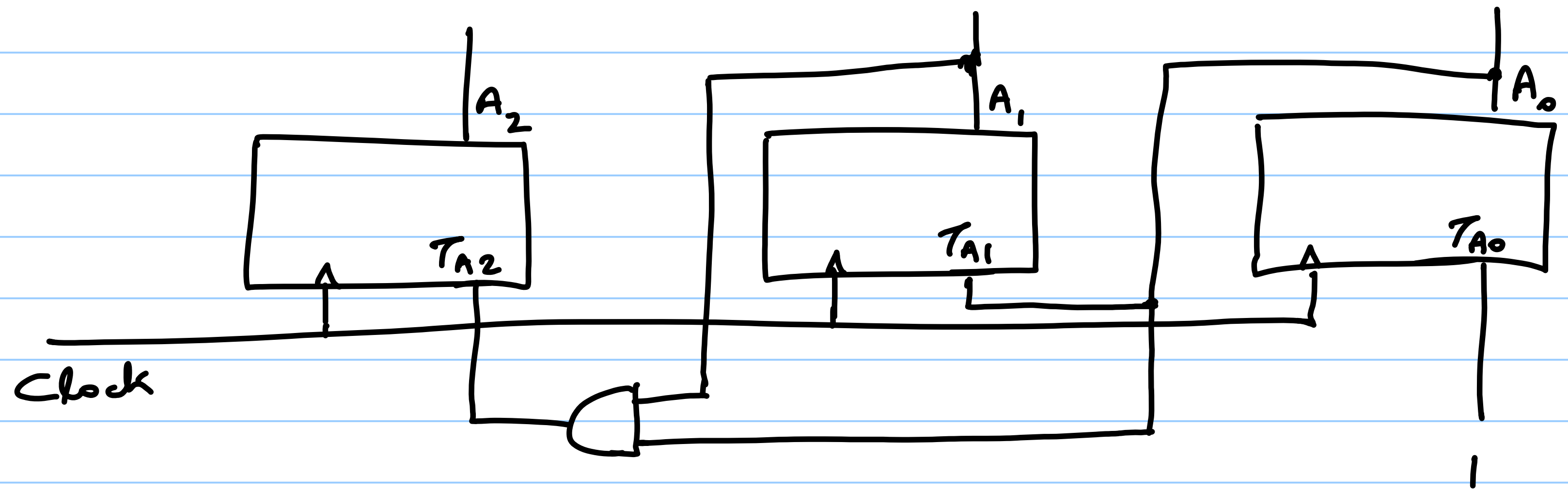
Present A_2	Present A_1	Present A_0	Next A_2	Next A_1	Next A_0	Flip-flop T_{A2}	Flip-flop T_{A1}	Flip-flop T_{A0}
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

$T_{A2} = A_1 A_0$ $\quad$ $T_{A1} = A_0$ $\quad$ $T_{A0} = 1$



Excitation Table

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0



A2
TA2
A1
TA1
A0
TA0
Clock
1

5/3/18 LO: Design of Shift Registers & Synchronous Counters

Shift Registers:

Group of flip-flops synchronized by a common clock

('n' bit register - 'n' flip-flops)

Synchronous Counters:

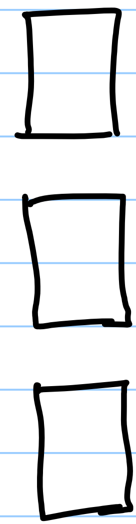
Register that goes through a pre-determined sequence of binary states

D flip-flop

→ storage element

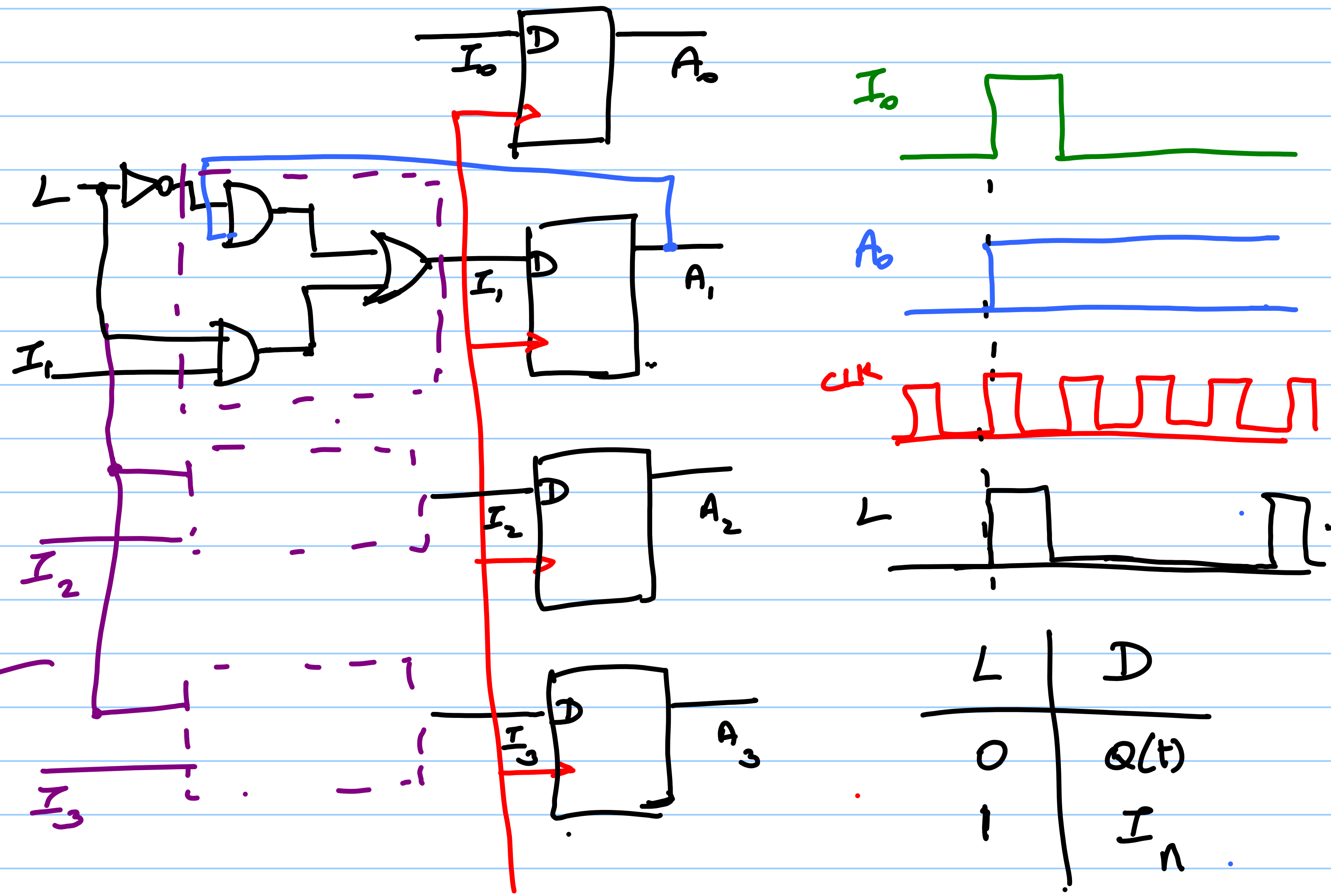
D	$Q(t+1)$
0	0
1	1

Ex: 4-bit register



Loading
(or) Updating

~~Gating clock~~ Gate the input

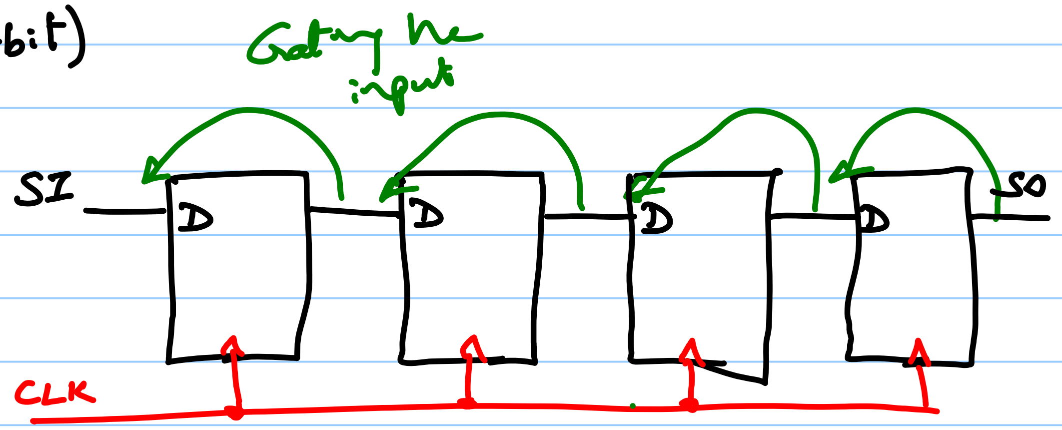


L	D
0	$Q(t)$
1	I_n

Shift Register

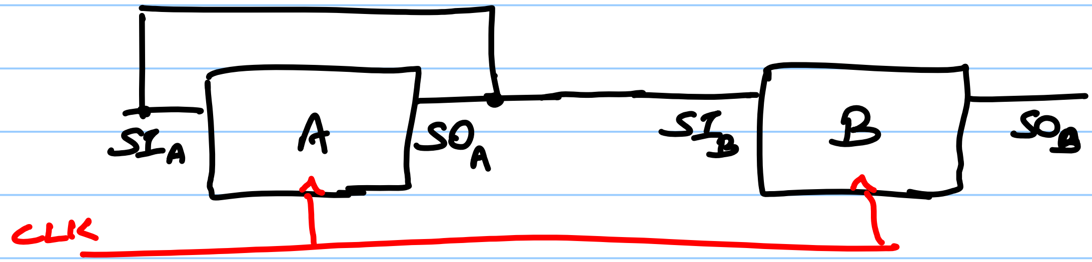
Left - to - right shift register (4-bit)

	SI	Flip-flop states			
Initial	1	0	0	0	0
After T_1	1	1	0	0	0
After T_2	0	1	1	0	0
After T_3	1	0	1	1	0
After T_4		1	0	1	1



Copying contents of one 4-bit shift register to another

	Shift Register A				Shift Register B			
Initial.	1	0	1	1	0	0	1	0
After T_1	1	1	0	1	1	0	0	1
After T_2	1	1	1	0	1	1	0	0
After T_3	0	1	1	1	0	1	1	0
After T_4	1	0	1	1	1	0	1	1



Universal Shift Register :

Shift Register → Left to Right

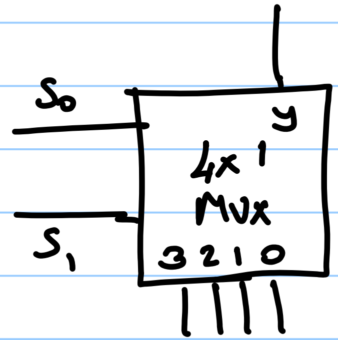
Shift Register → Serial IN - Serial OUT

Parallel IN - Parallel OUT

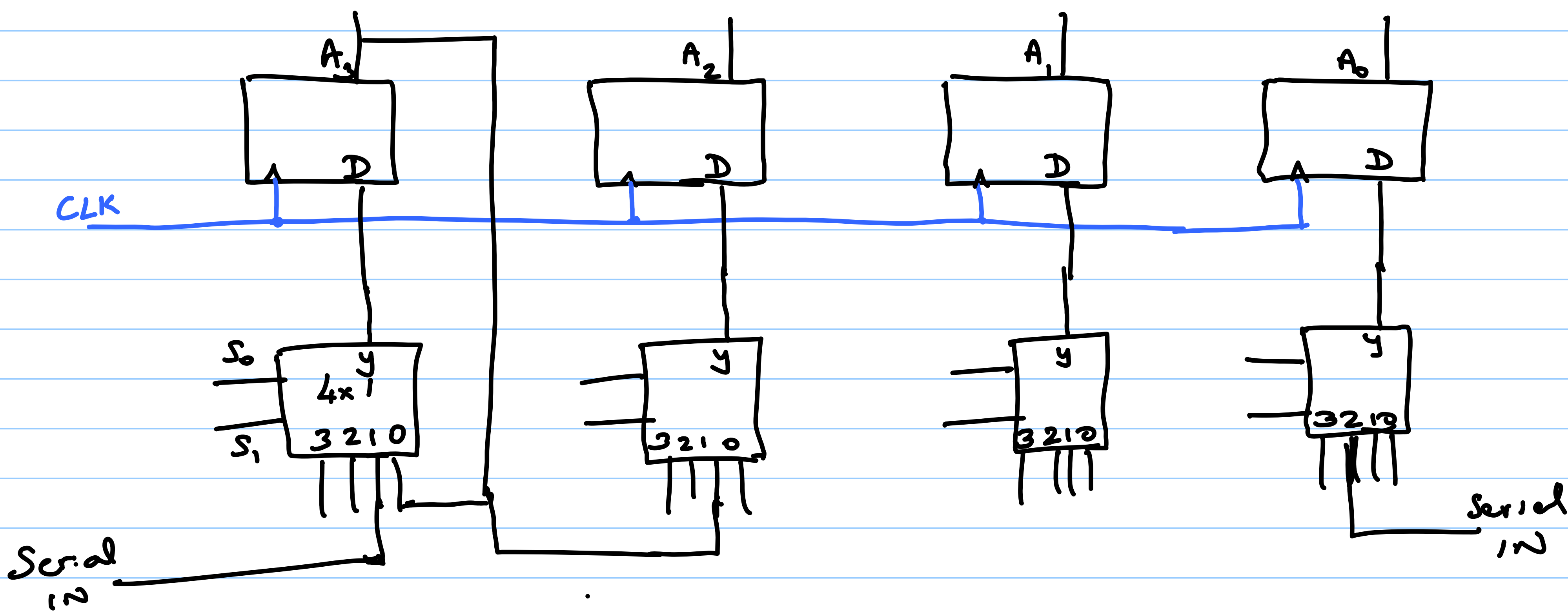
Serial IN - Parallel OUT

Parallel IN - Serial OUT

S_0	S_1	Operations
0	0	No change
0	1	Shift Right $(A_3 \to A_2 \to A_1 \to A_0)$
1	0	Shift Left $(A_0 \to A_1 \to A_2 \to A_3)$
1	1	Parallel Load

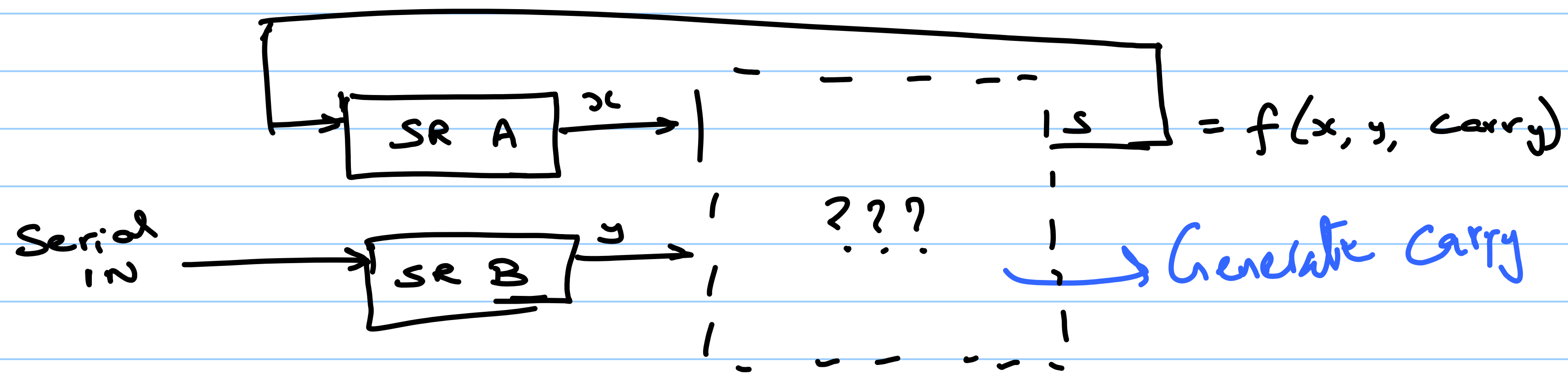


Parallel outputs



9/3/18 **Serial Adders :**

Need to load two binary numbers (4-bit) through a shift register and store the sum in the other shift register.

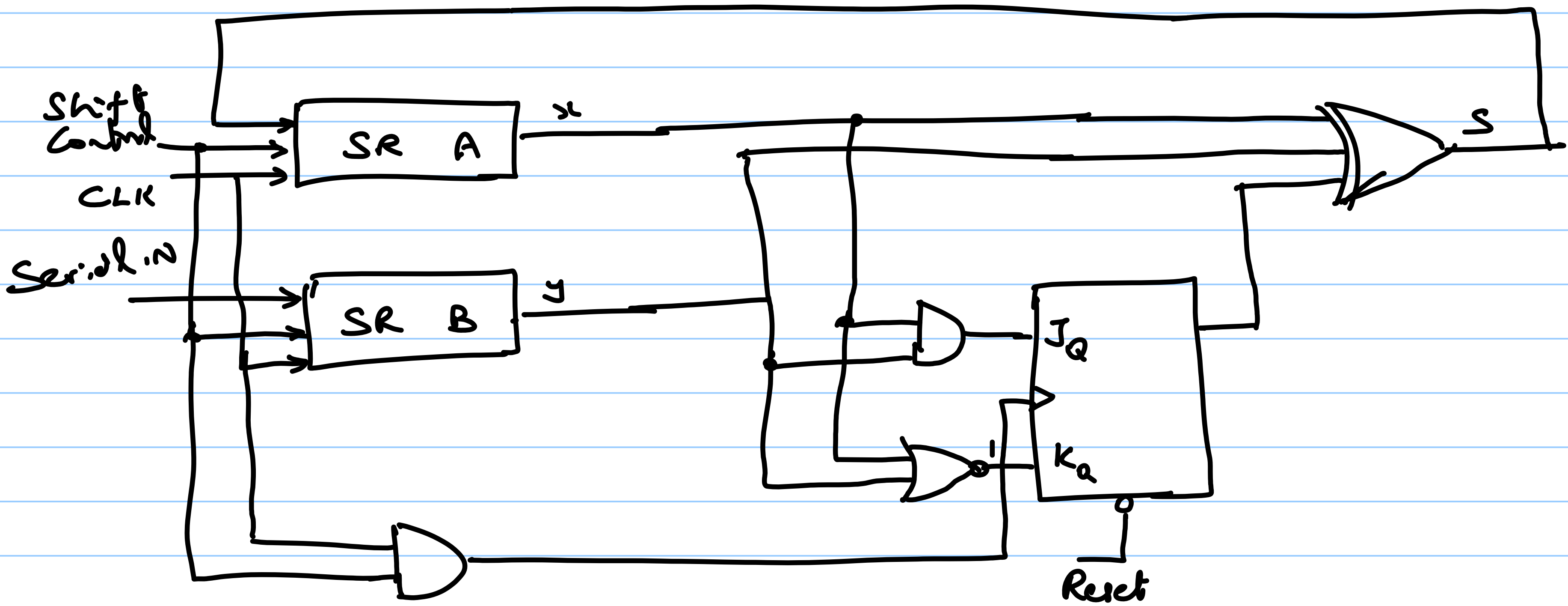


State Table

Present State	Inputs		Next State	Sum	Flip-flop inputs	
Q	x	y	Q	S	J_Q	K_Q
0	0	0	0	0	0	x
0	0	1	0	1	0	x
0	1	0	0	1	0	x
0	1	1	1	0	1	x
1	0	0	0	1	x	1
1	0	1	1	0	x	0
1	1	0	1	0	x	0
1	1	1	1	1	x	0

$J_Q = xy$ $\qquad$ $K_Q = x'y' = (x+y)'$ $\qquad$ $S = x \oplus y \oplus Q$

$Q \backslash xy$	00	01	11	10
0			1	
1	x	x	x	x



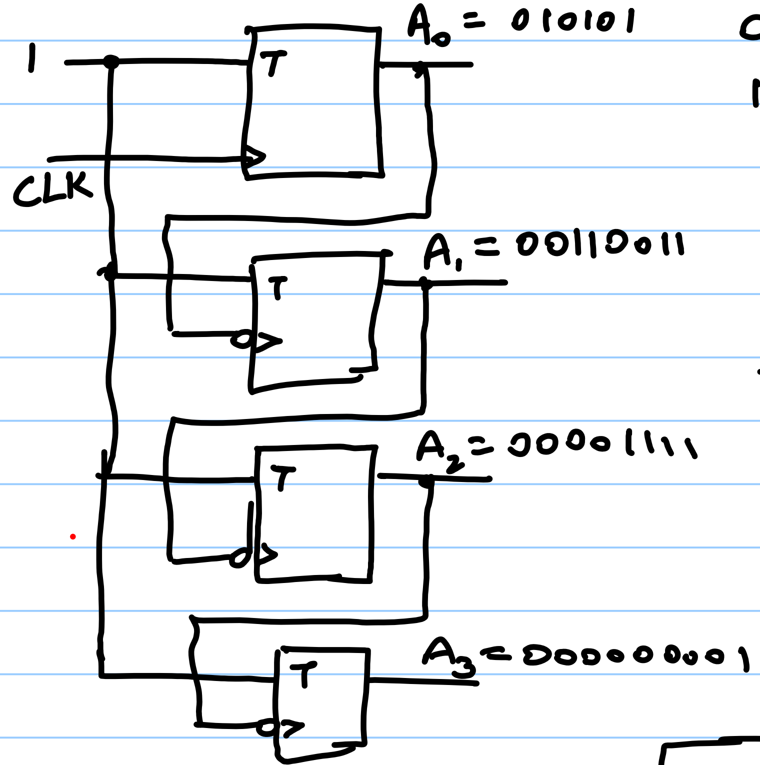
Counters:

Binary counter → Ripple (Flip-flop transitions decide the state of other flip-flops)

→ Synchronous (clock is directly applied to all the flip-flops)

Negative trigger $1 \rightarrow 0$

Clock cycle	A_3	A_2	A_1	A_0
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0 ↓
4	0	0	1	1
5	0	1	0 ↓	0
6	0	1	0	1 ↓
7	0	1	1	0
8	0	1	1	1
9	1	0 ↓	0 ↓	0 ↓



T	Q(t+1)
0	Q(t)
1	Q'(t)

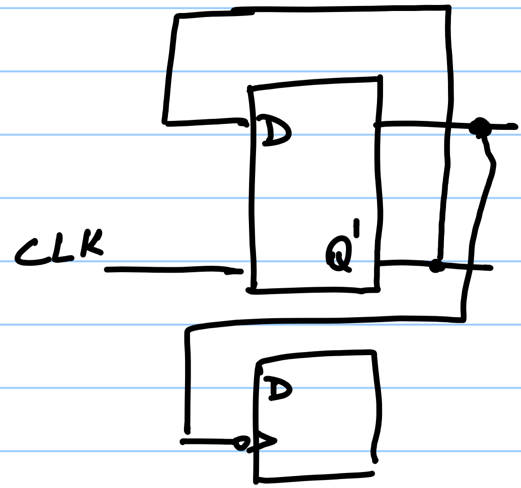
D	Q(t+1)	Q'
0	0	1
1	1	0

Down Counter?

1 1 1 1

1 1 1 0 ↓ Positive trigger

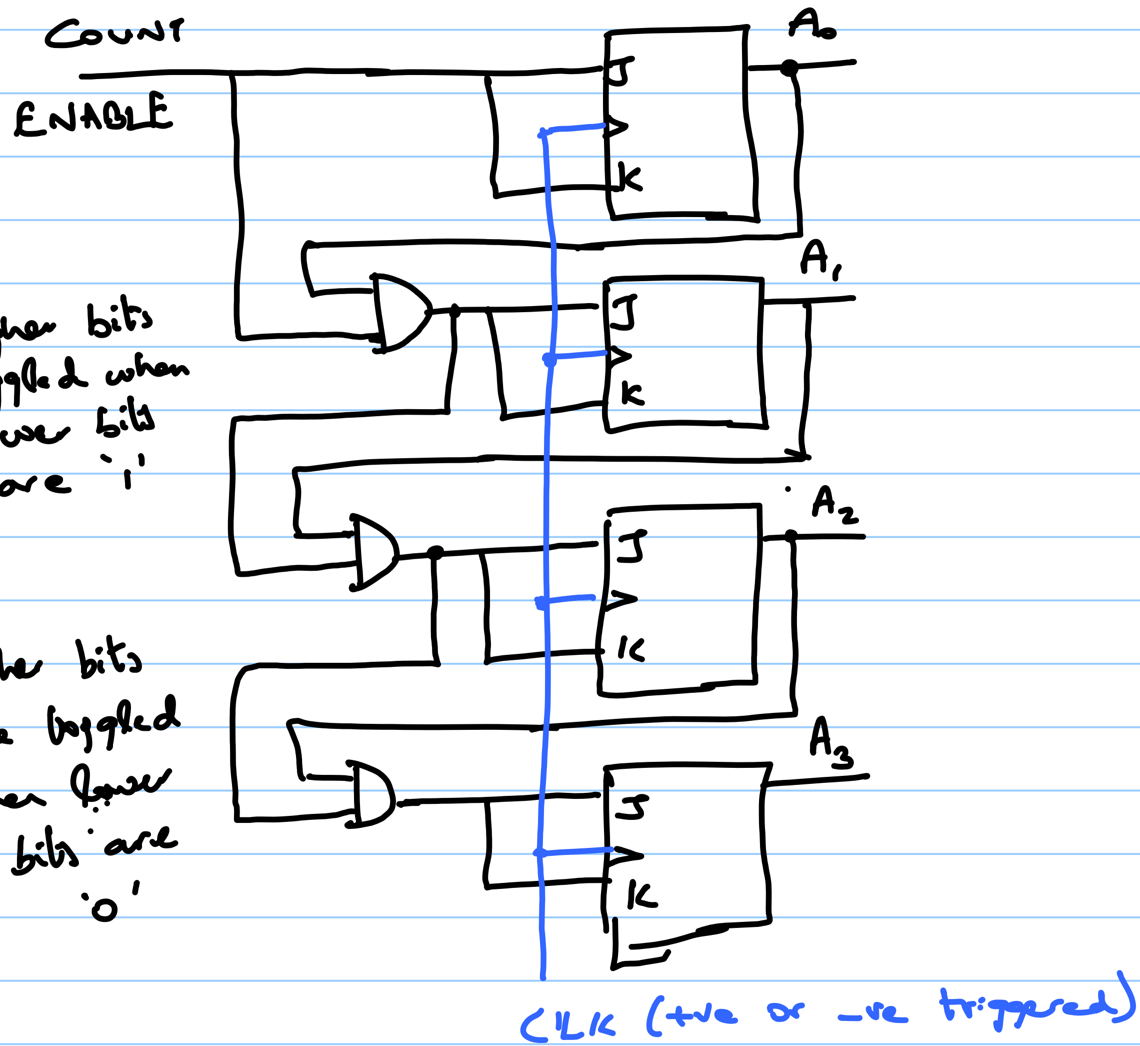
1 1 0 1



12/3/18 <u>LO</u>: Design synchronous counters using flip-flops

<u>Example</u>: Synchronous counter using JK flip-flop

T	Q(t+1)	J	K
0	Q(t)	0	0
1	Q'(t)	1	1



UP COUNTER

Ex: 0 1 1 1
1 0 0 0

Higher bits toggled when lower bits are '1'

DOWN COUNTER

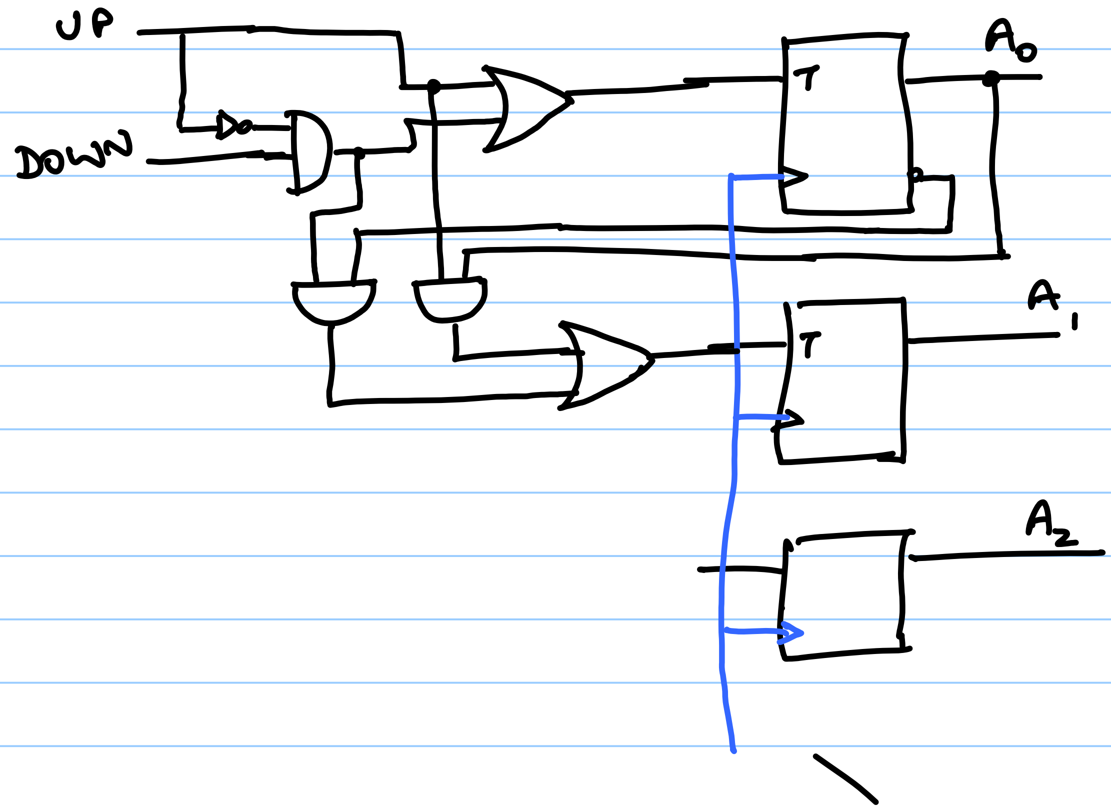
$A_3 A_2 A_1 A_0$

<u>Ex</u>: 0 1 0 0
0 0 1 1

Higher bits are toggled when lower bits are '0'

UP/DOWN COUNTER :

(Universal)

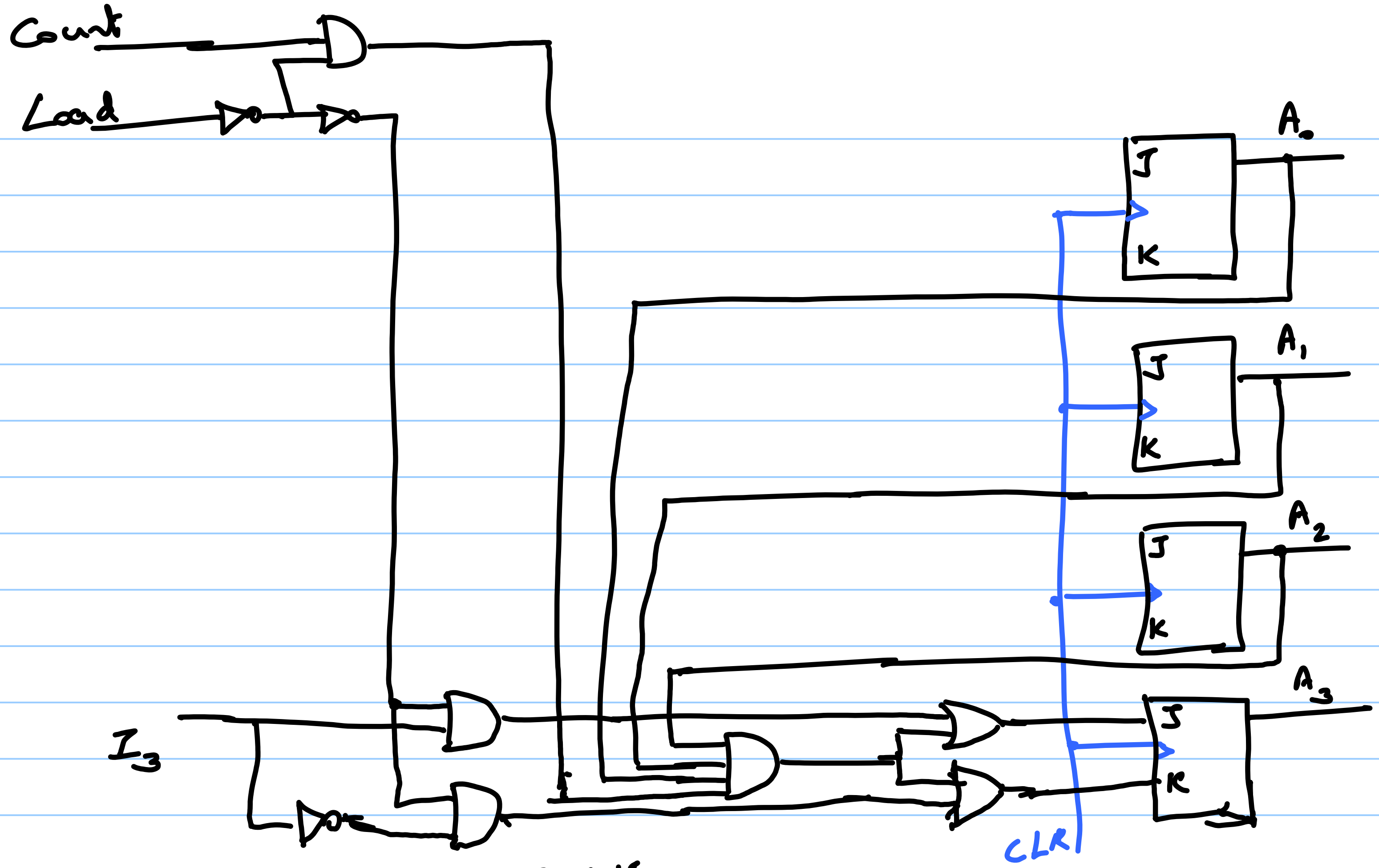


BCD Counter: (T flip-flop)

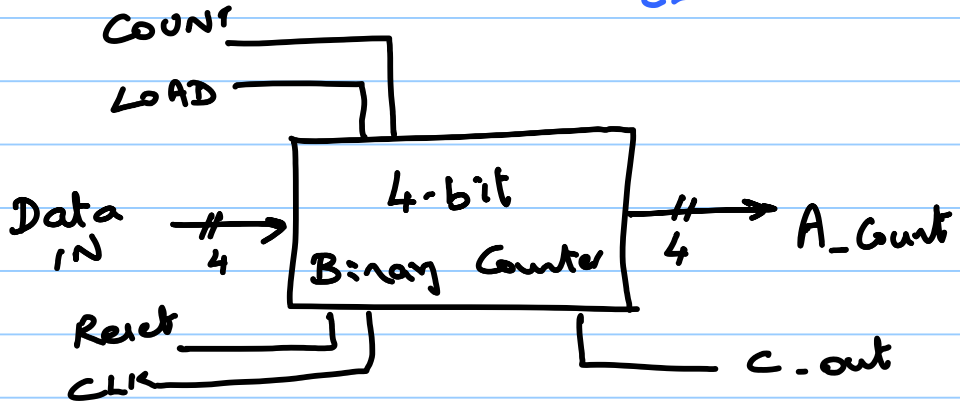
	Present State				Next state				Output	Flip-flop inputs			
	A_3	A_2	A_1	A_0	A_3	A_2	A_1	A_0	y	T_{A3}	T_{A2}	T_{A1}	T_{A0}
0	0	0	0	0	0	0	0	1	0	0	0	0	1
⋮		⋮							⋮				
9	1	0	0	1	0	0	0	0	1	1	0	0	1

$T_{A0} = 1$ $\quad T_{A1} = A_3' A_0$ $\quad T_{A2} = A_1 A_0$ $\quad T_{A3} = A_3 A_0 + A_2 A_1 A_0$

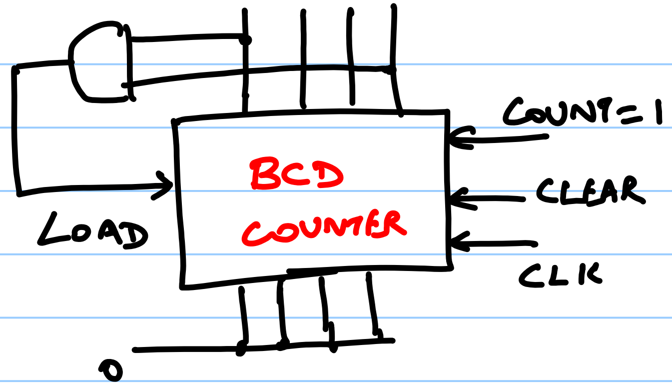
$y = A_3 A_0$

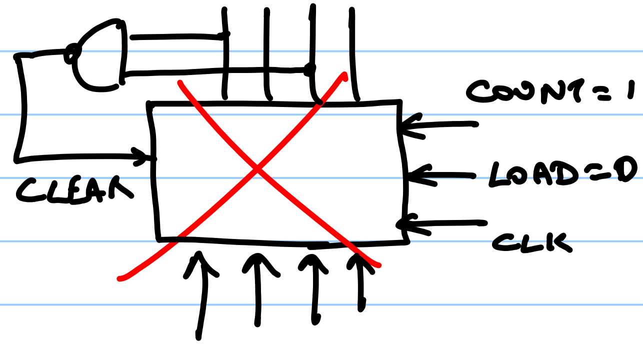


Count
Load
A0
A1
A2
A3
J
K
I3
CLR



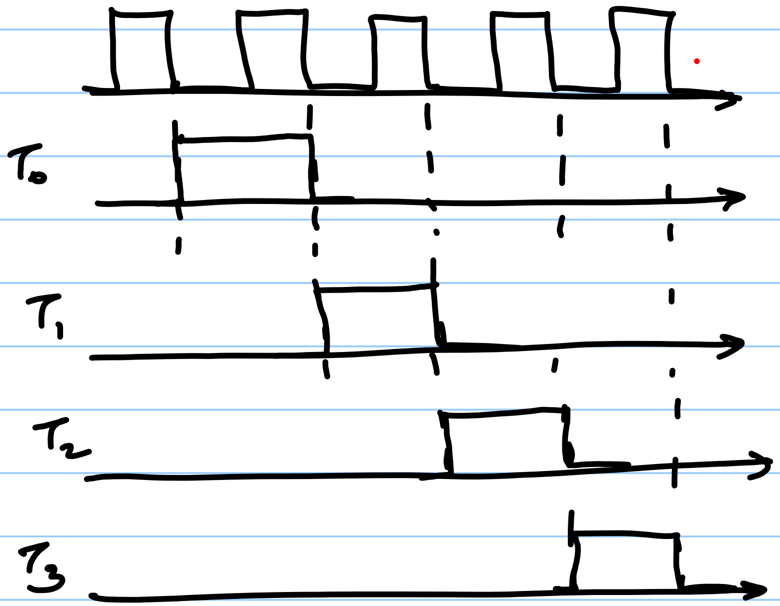
COUNT
LOAD
Data IN
4
4-bit Binary Counter
A_Count
Reset
CLK
C_out

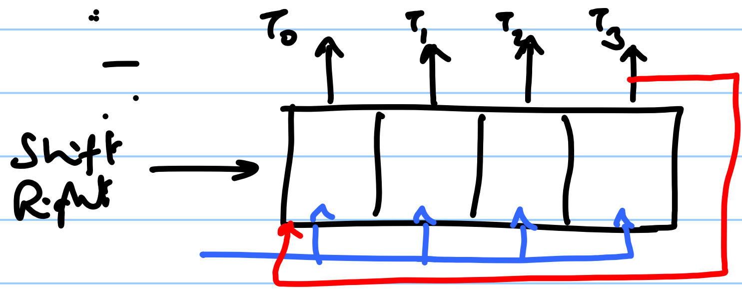




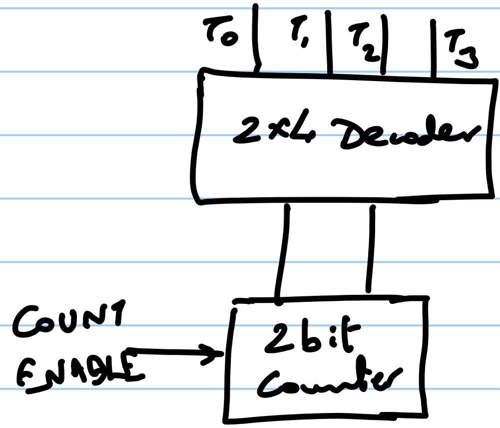
Not recommended

Ring Counter:





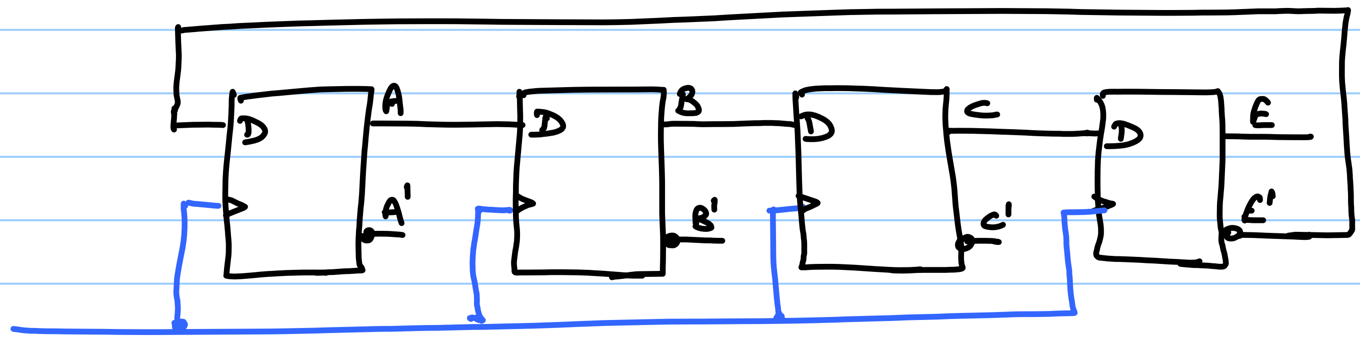
Initial → 1000



4 bits
→ 4 states

Johnson Counter:

k bits → '2k' distinguishable states



Seq #	A	B	C	E	AND gate for o/p
	Flip-flop o/ps				
1	0	0	0	0	A'E'
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	A'B
7	0	0	1	1	B'C
8	0	0	0	1	C'E

16/3/18

Programmable Logic Devices.

↳ ?

↳ Gates / flip-flops / Registers / Encoder / Decoders)

MUX / DeMUX

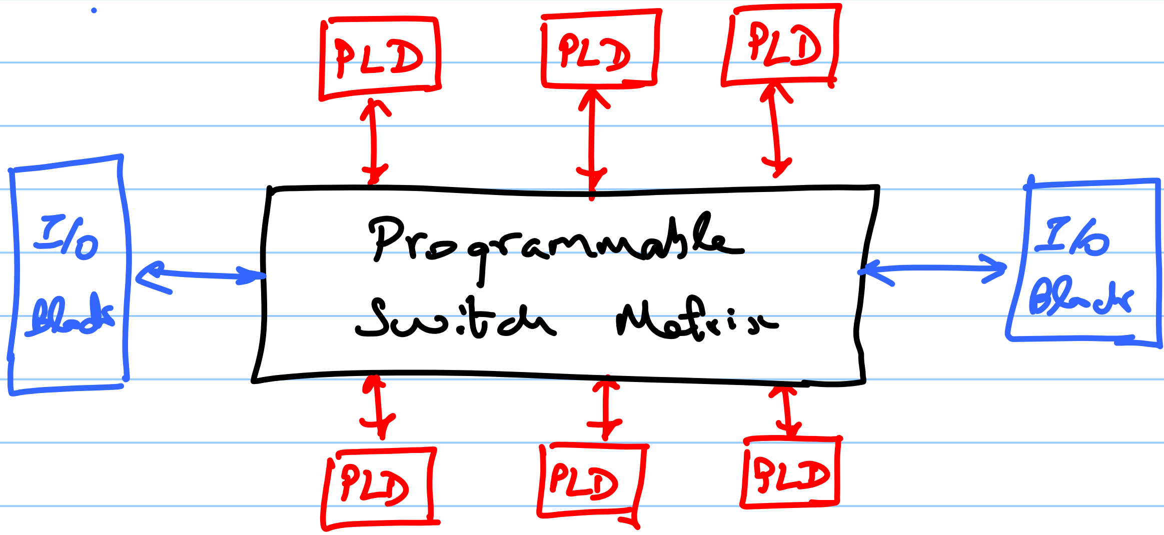
Hardware Description Language (HDL)

Hardware

VHDL / Verilog

→ buses / gates / logic devices

Complex Programmable Logic Device (CPLD)



Field Programmable Gate Array (FPGA) → "Soft" Microprocessor

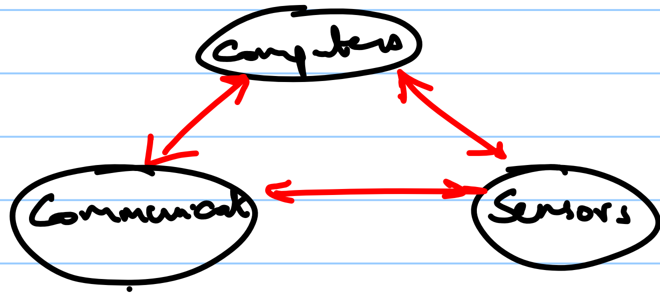
- Machine Learning
- 5G Wireless
- Embedded Vision
- IoT
- Cloud Computing

Example

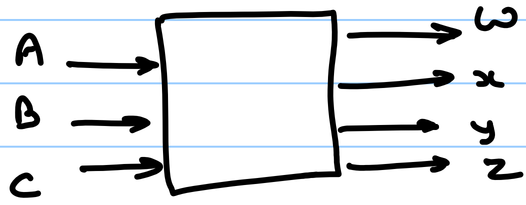
Memory

Non-volatile	Volatile
Read-Only Memory (ROM)	Random Access Memory (RAM)
Typically slow	Faster than ROM

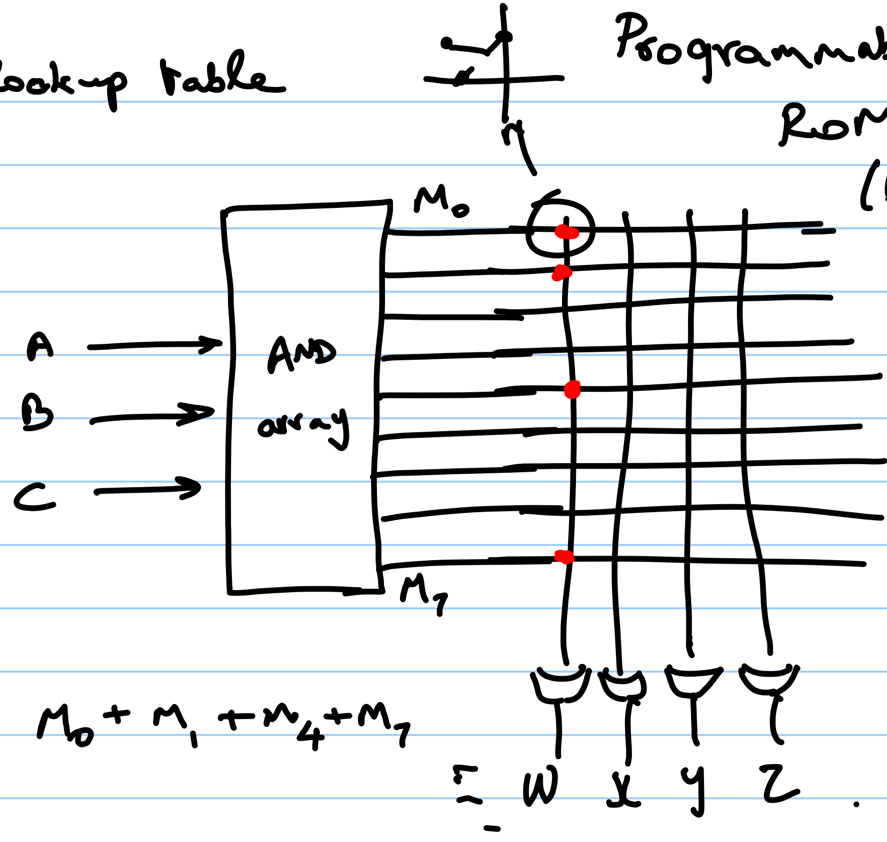
Cyber-Physical Systems



· ROM → Hard-wired truth table / look-up table



Electrically Erasable Programmable ROM (EEPROM)



$M_0 + M_1 + M_4 + M_7$ = W

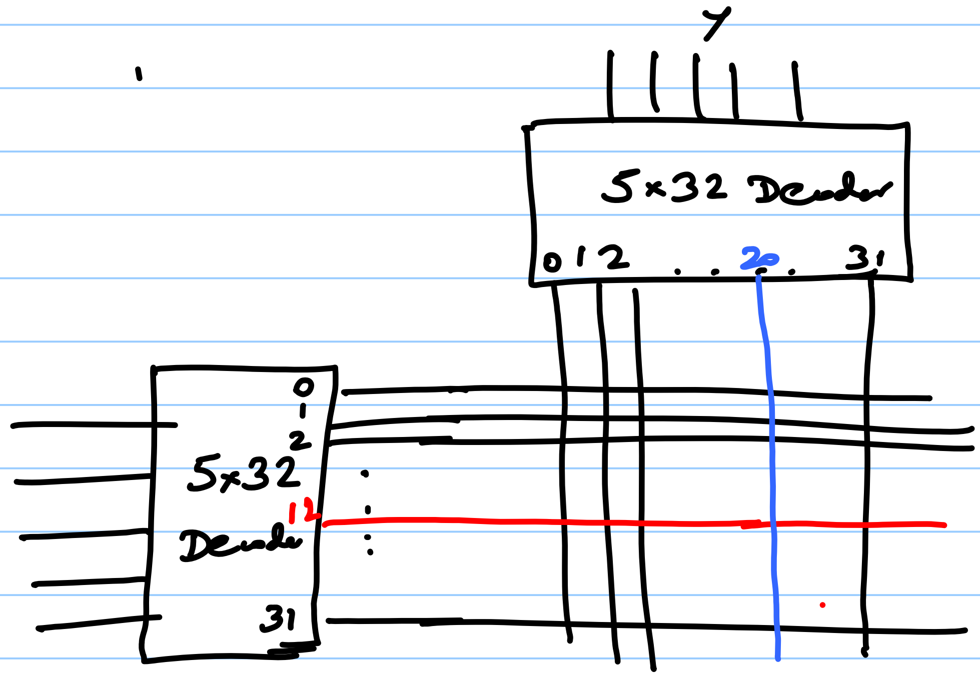
22/3/18

Address Multiplexing

* Decoder w/ K inputs to address 2^K words

  $\rightarrow$ 2^K AND gates w/ K inputs per gate

* Two-dimensional addressing $\rightarrow$ Coincident Decoding



1K words

$\Rightarrow$ 10 bit addressing

1024 AND w/ 10 inputs

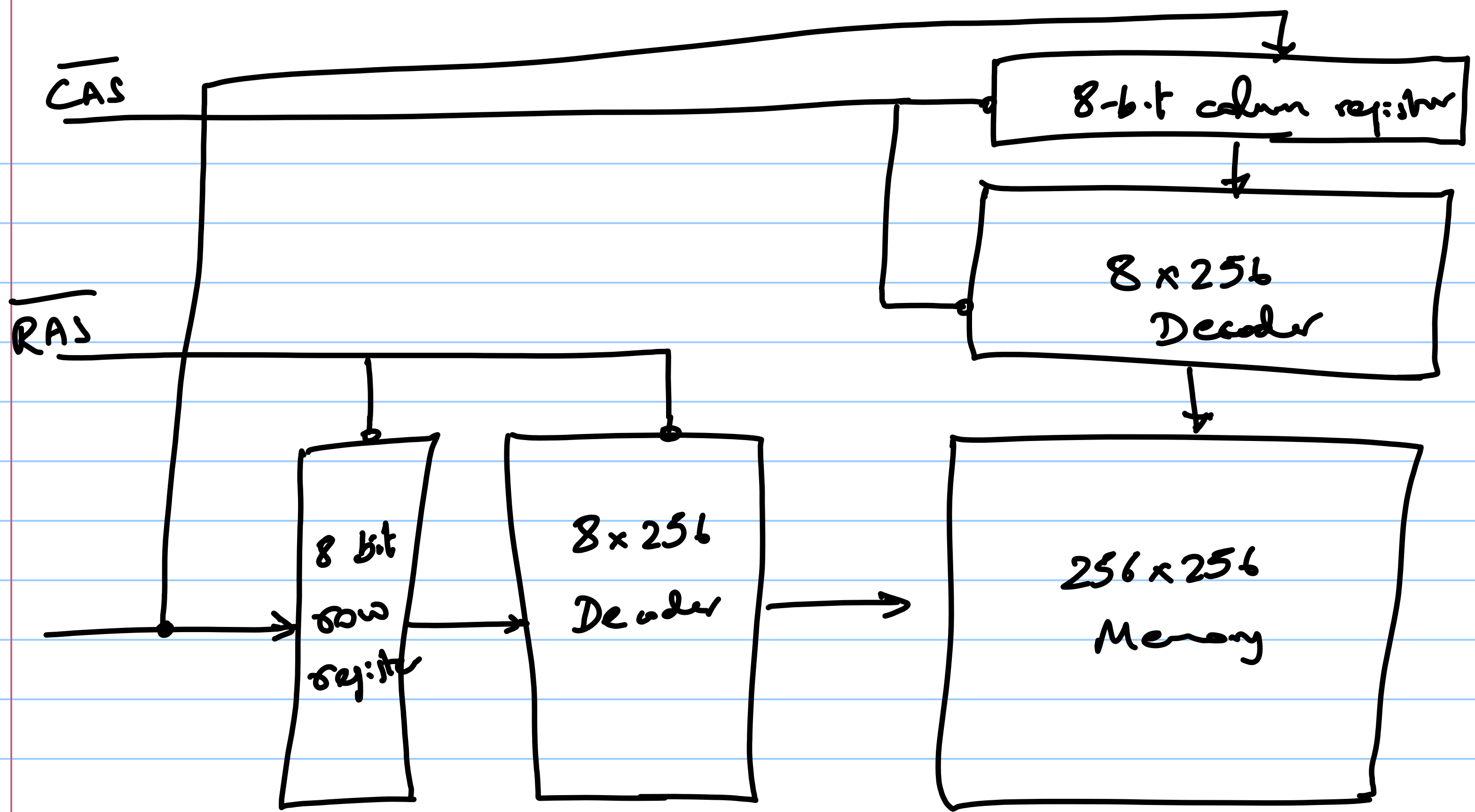
2D addressing

$\Rightarrow$ 64 AND w/ 5 inputs.

404 $\rightarrow$ 0110010100

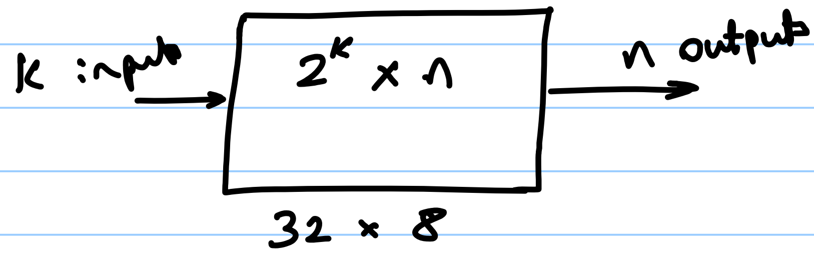
$\underbrace{01100}_{12}\ \underbrace{10100}_{20}$

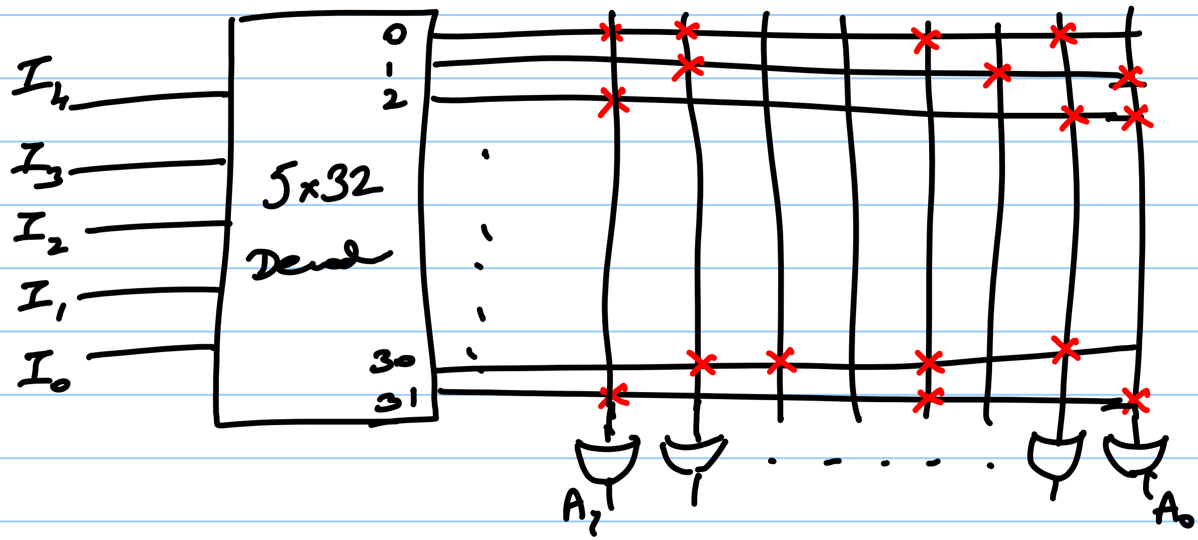
SRAM	DRAM	
Flipflop / 6 transistors	transistor + Capacitor	$2^{16} \Rightarrow 16$ bits
	4x more density	$\uparrow$
	$\Rightarrow$ 4x more capacity	64 K Memory
	$\Rightarrow$ 4x less cost	$\Rightarrow$ Address multiplexing
	Lower power consumption	


CAS
RAS
8-bit column register
8 x 256 Decoder
8 bit row register
8 x 256 Decoder
256 x 256 Memory

Read Only Memory (ROM)

→ Only READ





$A_0 = \Sigma(1, 2, \ldots \ldots 31)$

$\vdots$

$A_7 = \Sigma(0, 2, \ldots \ldots 31)$

ROM → Minterm generator

(Look-up Table)

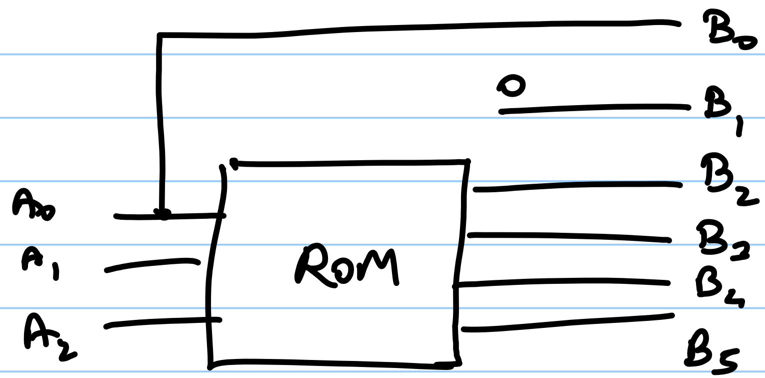
	I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0	1	1	0	0	1	0	1	0
1	0	0	0	0	1	0	1	0	0	0	1	0	1
2	0	0	0	1	0	1	0	0	0	0	0	1	1
⋮													
30	1	1	1	1	0								
31	1	1	1	1	1								

Example

To accept a 3-bit number and output a binary number equal to square of the input number

Inputs			Outputs						Decimal
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

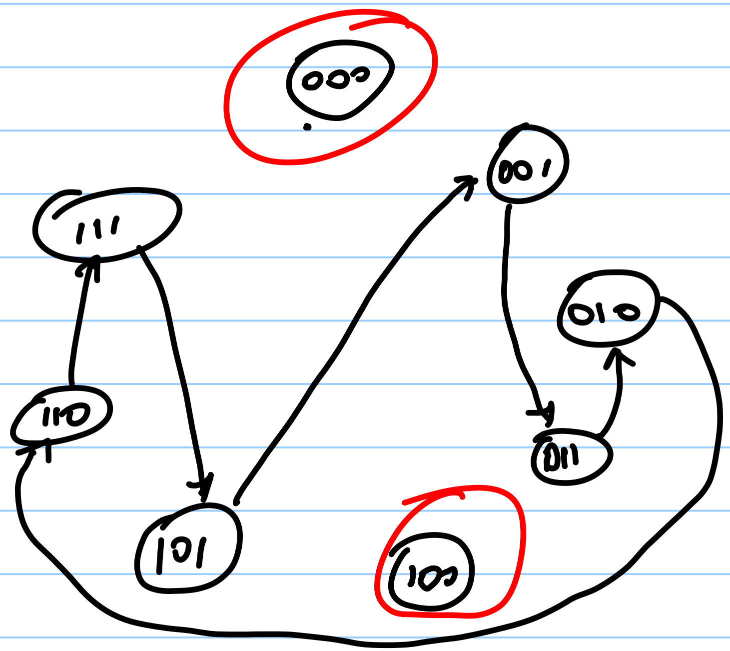
Truth Table for ROM



ROM
A_0
A_1
A_2
0
B_0
B_1
B_2
B_3
B_4
B_5

Example Design a 3-bit counter which counts in the sequence

001 → 011 → 010 → 110 → 111 → 101 → 001 . . .



Unused states

Present State Q_2	Q_1	Q_0	Next State Q_2	Q_1	Q_0	Flip-flop inputs J_2	K_2	J_1	K_1	J_0	K_0
0	0	0				x	x	x	x	x	x
0	0	1	0	1	1	0	x	1	x	x	0
0	1	0	1	1	0	1	x	x	0	0	x
0	1	1	0	1	0	0	x	x	0	x	1
1	0	0				x	x	x	x	x	x
1	0	1	0	0	1	x	1	0	x	x	0
1	1	0	1	1	1	x	0	x	0	1	x
1	1	1	1	0	1	x	0	x	1	x	0

Truth Table

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

Transition Table

Q(t)		Q(t+1)	J	K
0	→	0	0	x
0	→	1	1	x
1	→	0	x	1
1	→	1	x	0

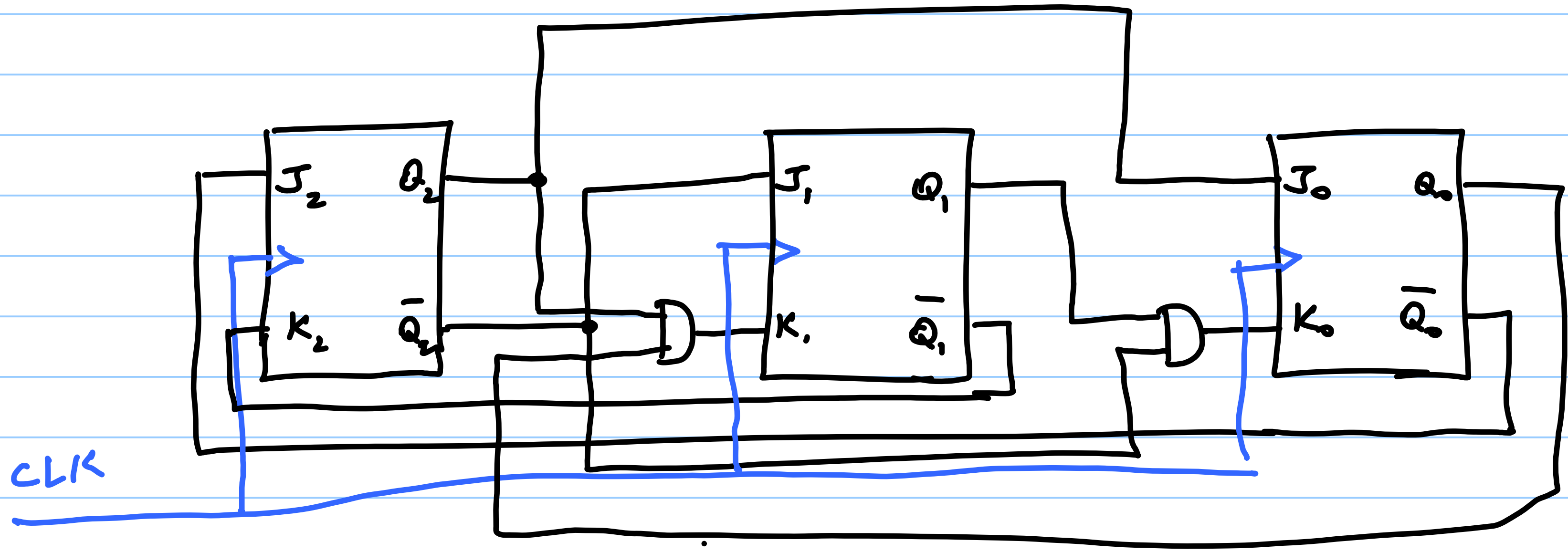
J_2

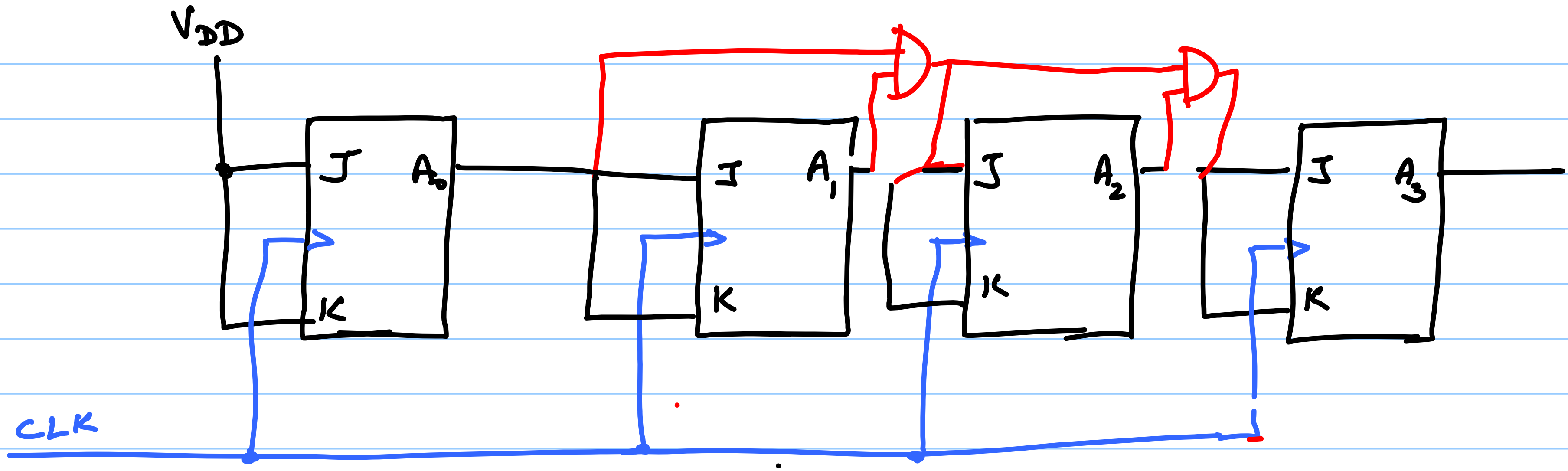
$Q_2 \backslash Q_1Q_0$	00	01	11	10
0	x			1
1	x	x	x	x

$J_2 = \bar{Q}_0$ $\quad K_2 = \bar{Q}_1$

$J_1 = \bar{Q}_2$ $\quad K_1 = Q_2 Q_0$

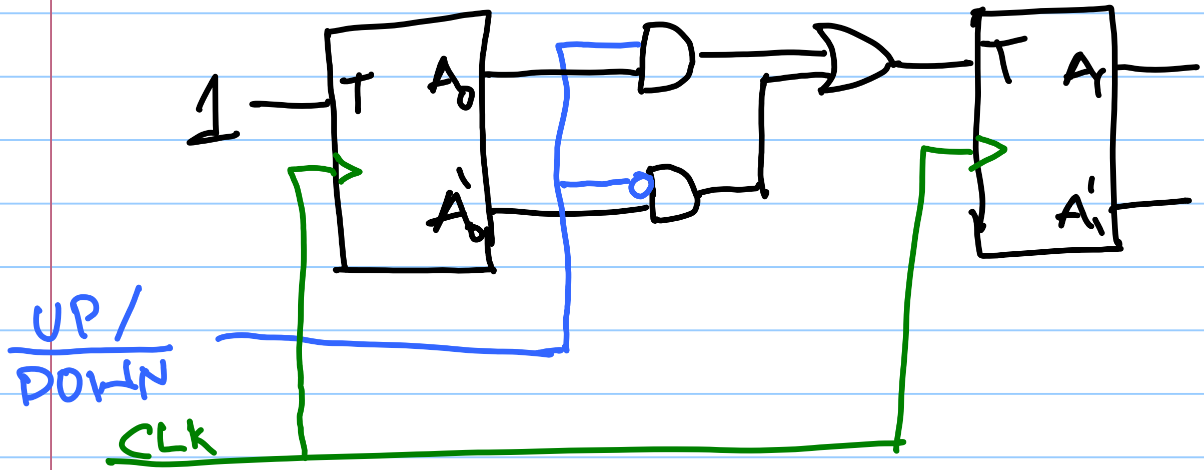
$J_0 = Q_2$ $\quad K_0 = \bar{Q}_2 Q_1$





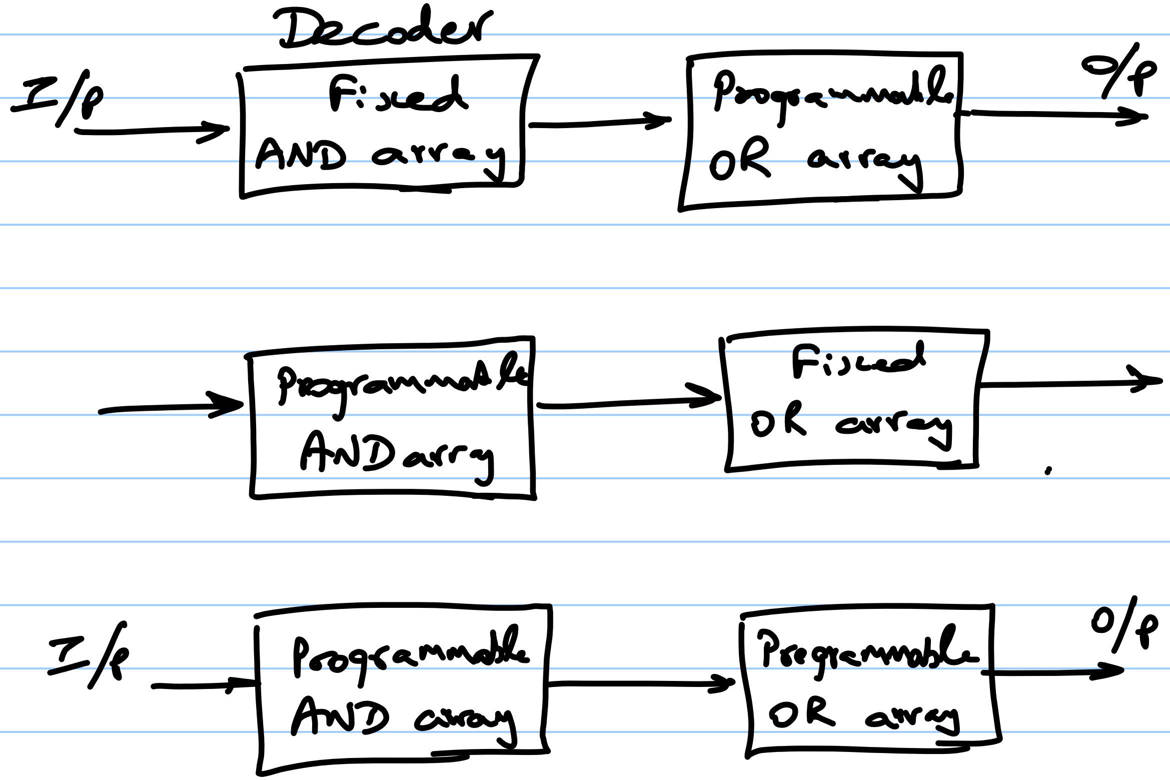
	A_3	A_2	A_1	A_0
Start →	0	0	0	0
	0	0	0	1
	0	0	1	0
	0	1	1	1
	1	0	0	0
	1	0	0	1
	1	0	1	0
	1	1	1	1

Only 8 counts



1
T
A_0
A_0'
UP/$\overline{\text{DOWN}}$
CLK
T
A_1
A_1'

Programmable Logic Devices :



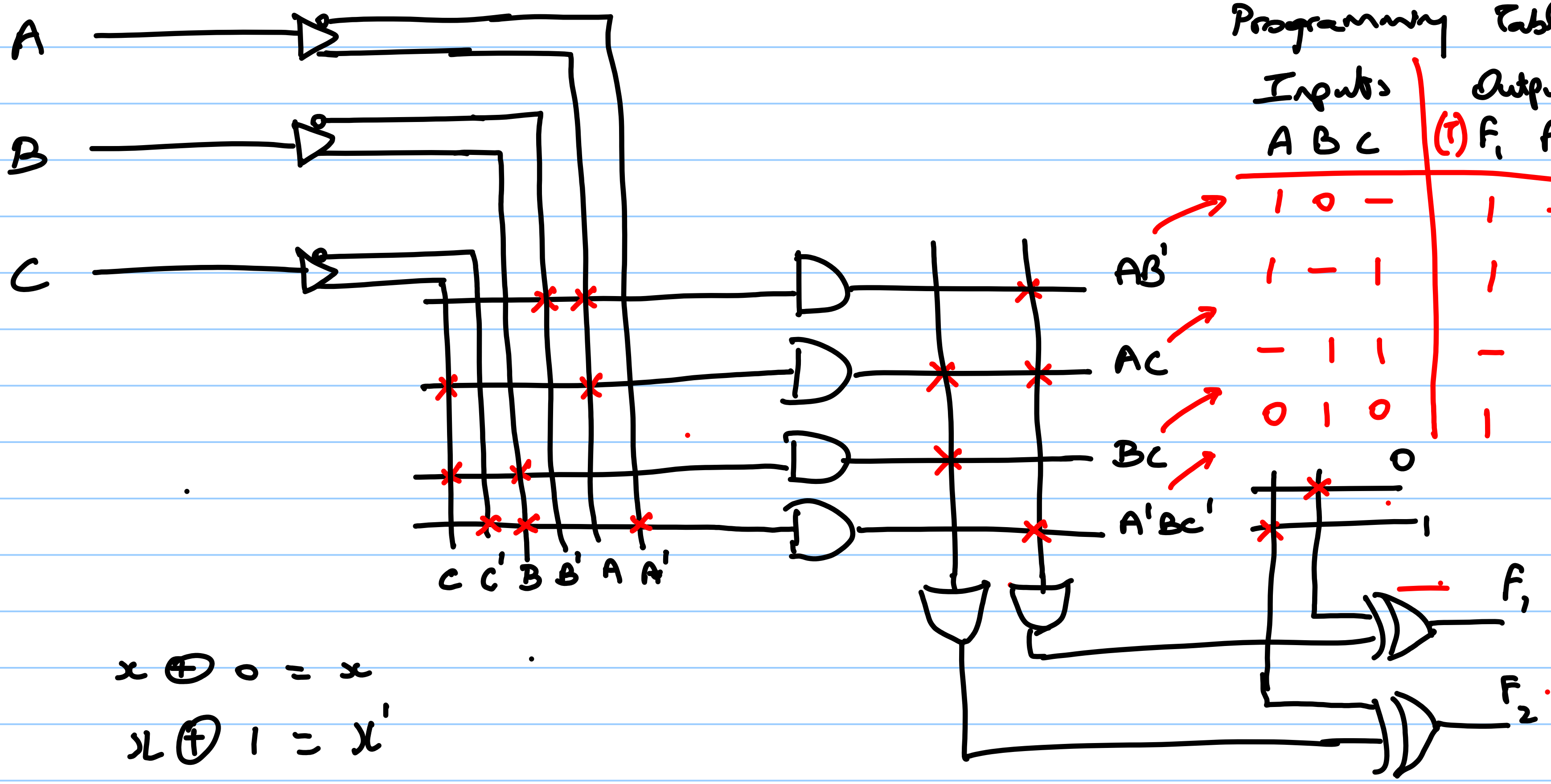
PROM

Programmable Array Logic (PAL)

Programmable Logic Array (PLA)

Ex:

$$F_1 = AB' + AC + A'BC' \qquad F_2 = (AC + BC)'$$



Programming Table

Inputs			Outputs	
A	B	C	(T) F_1	F_2 (C)
1	0	–	1	–
1	–	1	1	1
–	1	1	–	1
0	1	0	1	–

$$x \oplus 0 = x$$

$$x \oplus 1 = x'$$

Size of PLA

\# of inputs (n) → buffer/inverters
\# of product terms (K) → AND
\# of outputs (m) → OR

$2n \times K$ connections between inputs & AND array

$m \times K$ connections between AND & OR array

m connections allocated w/ XOR gates

Typical → 16, 48, 8
i/p, AND, o/p

<u>Ex</u>

$$F_1 = \Sigma(0, 1, 2, 4) \qquad F_2 = \Sigma(0, 5, 6, 7)$$

⇒ 3 variables A, B, C

$F_1 = (AB + BC + AC)'$

F_1

A \ BC	00	01	11	10
0	1	1	0	1
1	1	0	0	0

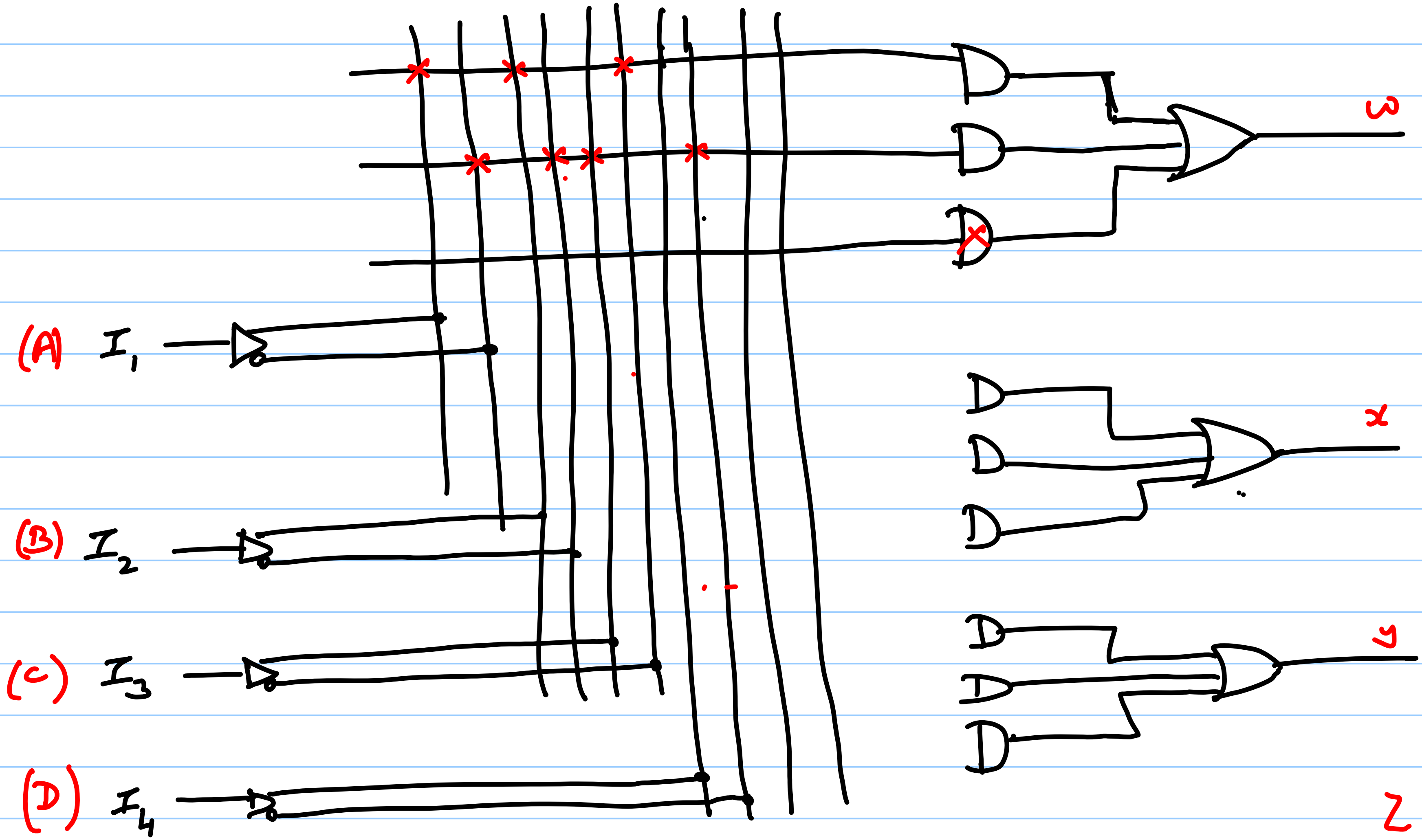
$F_2 = AB + AC + A'B'C'$

A \ BC				
	1			
		1	1	1

Prod. Term	Inputs			Outputs	
	A	B	C	F_1 (C)	F_2 (T)
AB	1	1	–	1	1
AC	1	–	1	1	1
BC	–	1	1	1	–
$A'B'C'$	0	0	0	–	1

Programmable Array Logic (PAL):

- Programmable AND array + fixed OR array



$$w = \sum(2, 12, 13)$$

$$x = \sum(7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$y = \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$

$$z = \sum(1, 2, 8, 12, 13)$$

After simplification with K-maps

$$w = ABC' + A'B'CD'$$

$$x = A + BCD$$

$$y = A'B + CD + B'D'$$

$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$

$$= w + AC'D' + A'B'C'D$$

Product Term	A	B	C	D	W	Outputs
1	1	1	0	–	–	$w = ABC' + A'B'CD'$
2	0	0	1	0	–	
3	–	–	–	–	–	
4						x
5						
6						
						y

(The columns A, B, C, D, W are headed "AND inputs".)

1

1-bit Full adder using 3 WIDE AND-OR PAL

Truth Table

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

$SUM = X \oplus Y \oplus C_{in}$

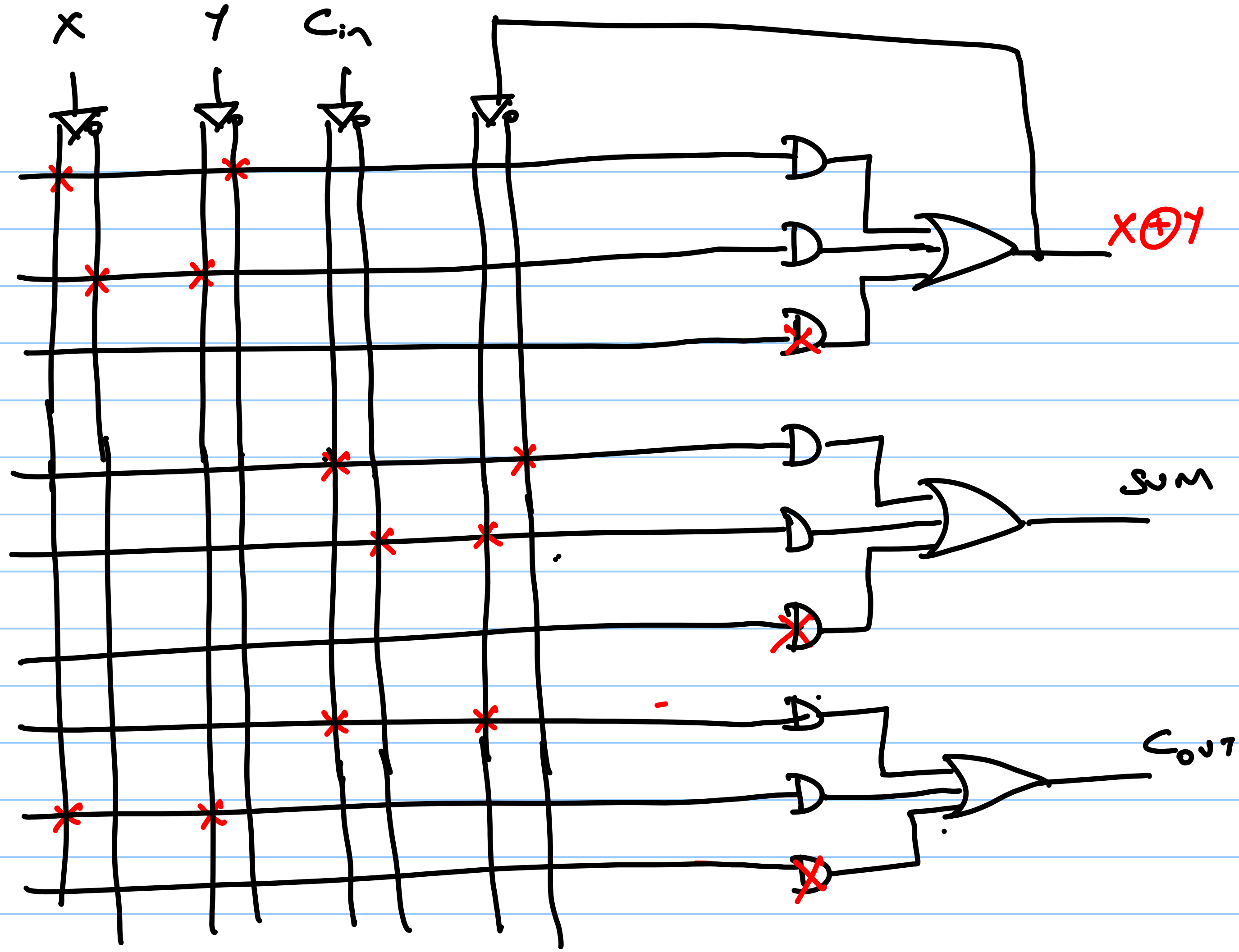
$= (X \oplus Y) C_{in}'$

$+ (X \oplus Y)' C_{in}$

$C_{OUT} = C_{in}(X \oplus Y)$

$+ XY$

X	Y	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



X
Y
Cin
X⊕Y
SUM
Cout

$A \rightarrow A_1 A_0 \qquad B \rightarrow B_1 B_0$

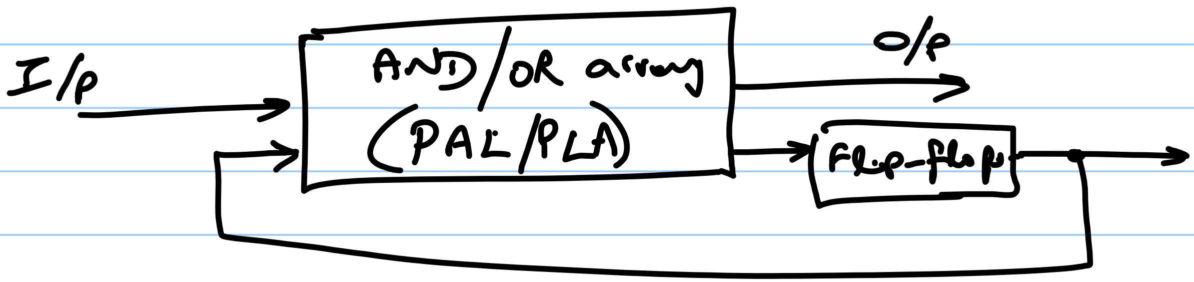
2-bit Magnitude comparator using 3 WIDE AND-OR PAL

$$A > B = A_1 B_1' + A_1 A_0 B_0' + A_0 B_1' B_0'$$

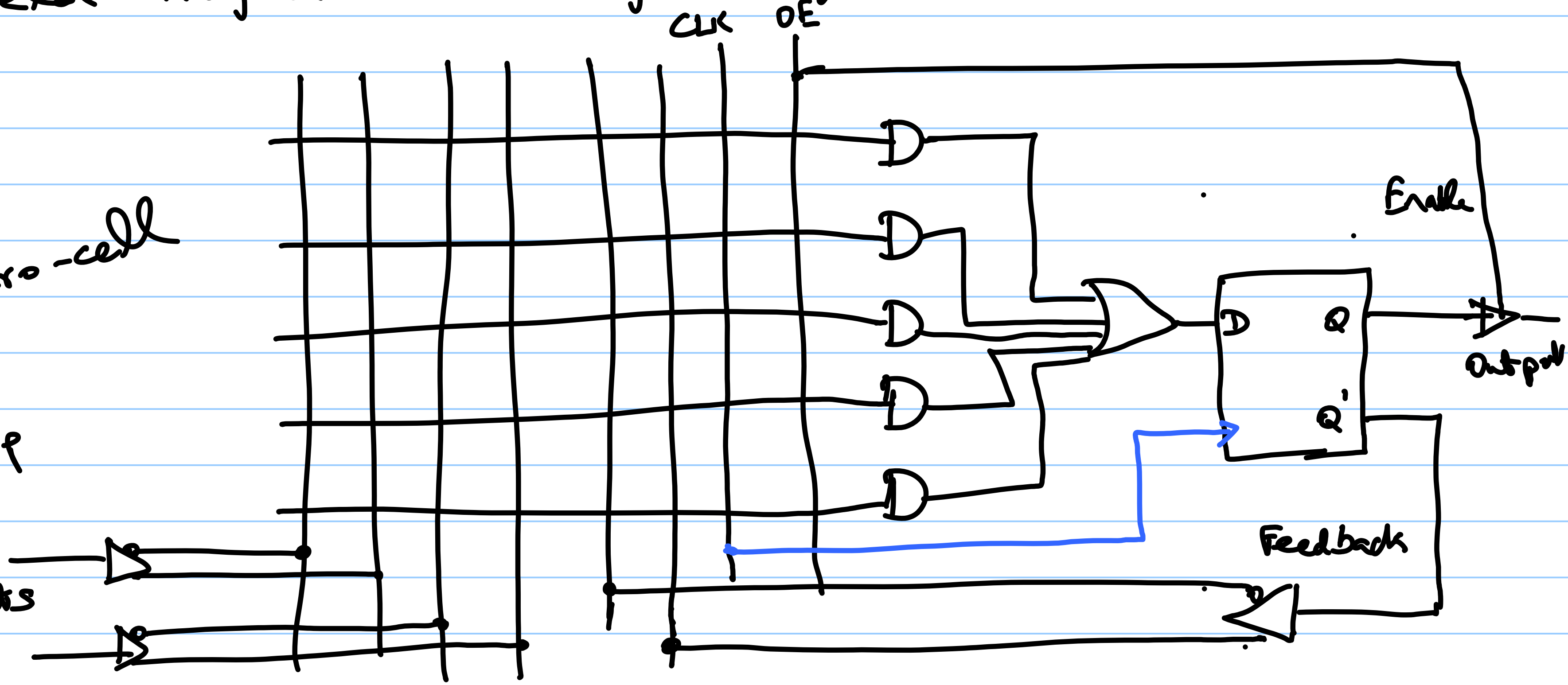
$$A < B = A_1' B_1 + B_1 B_0 A_0' + A_1' A_0' B_0$$

$$(A = B) = (A > B)' (A < B)'$$

13/4/18 LO: **Sequential PAL/PLA Design**



Field Programmable Logic Sequencer



Macro-cell

8-10 MCs in a chip

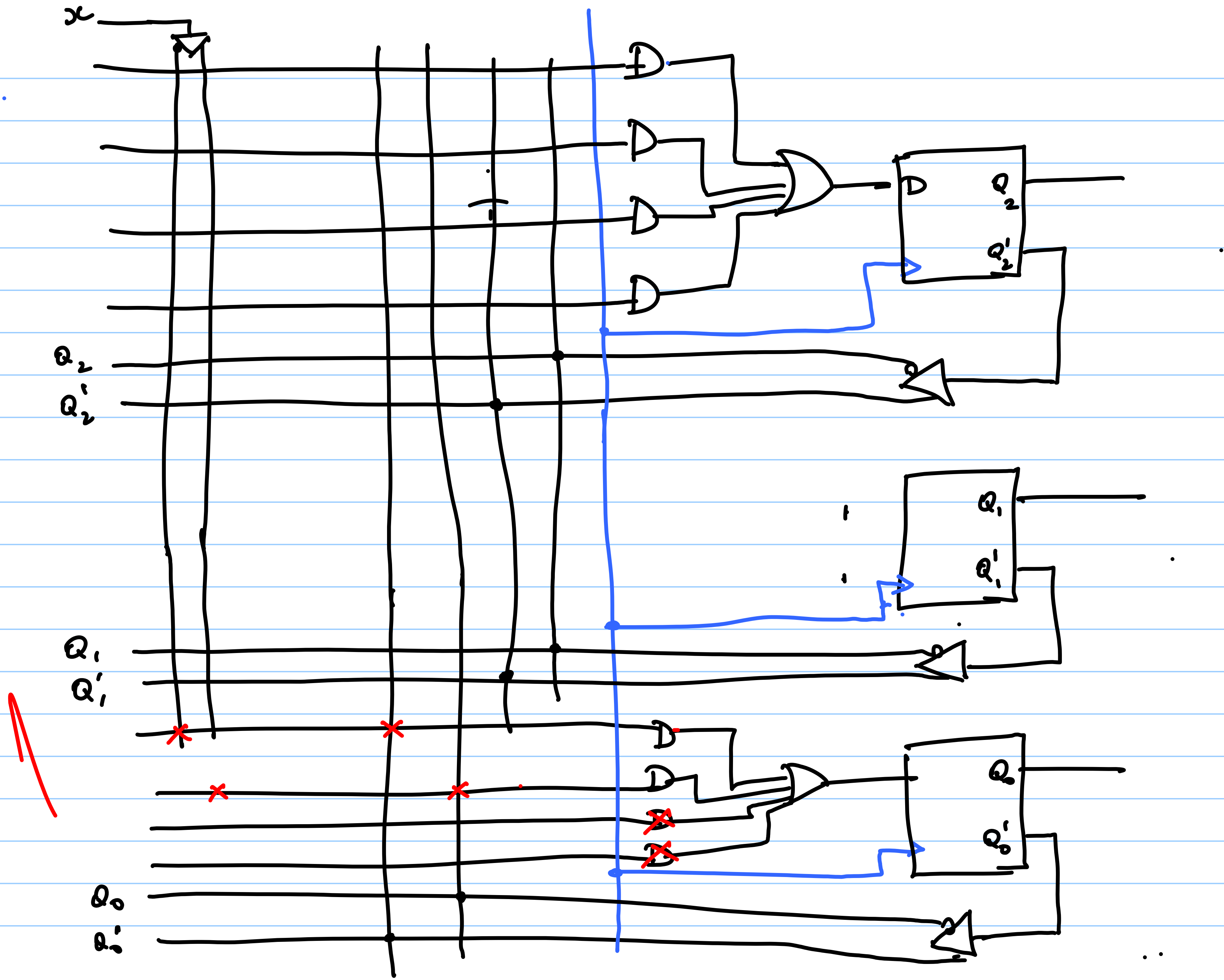
Ex Design 3-bit UP counter which counts when input = 1 and remains in same state when input = 0 using sequential PLA

Present State			Input	Next State		
Q_2	Q_1	Q_0	x	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	1	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	1	0	0
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	0	0	0

$$Q_2^+ = Q_2Q_0' + Q_2Q_1' + Q_2x' + Q_2'Q_1Q_0x$$

$$Q_1^+ = Q_1Q_0' + Q_1x' + Q_1'Q_0x$$

$$Q_0^+ = Q_0'x + Q_0x'$$

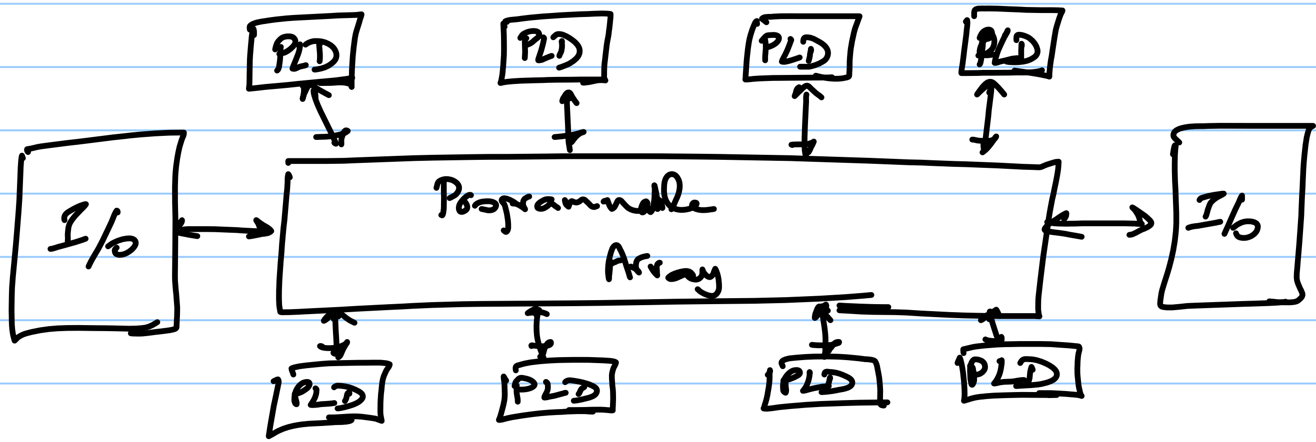

x
Q2
Q'2
Q1
Q'1
Q0
Q'0
D
Q2
Q'2
Q1
Q'1
Q0
Q'0

Eg:- Design 1101 sequence detection using PAL

SPLD → Sequential (Simple) Programmable Logic Devices

CPLD → Complex Prog. Logic Device

FPGA

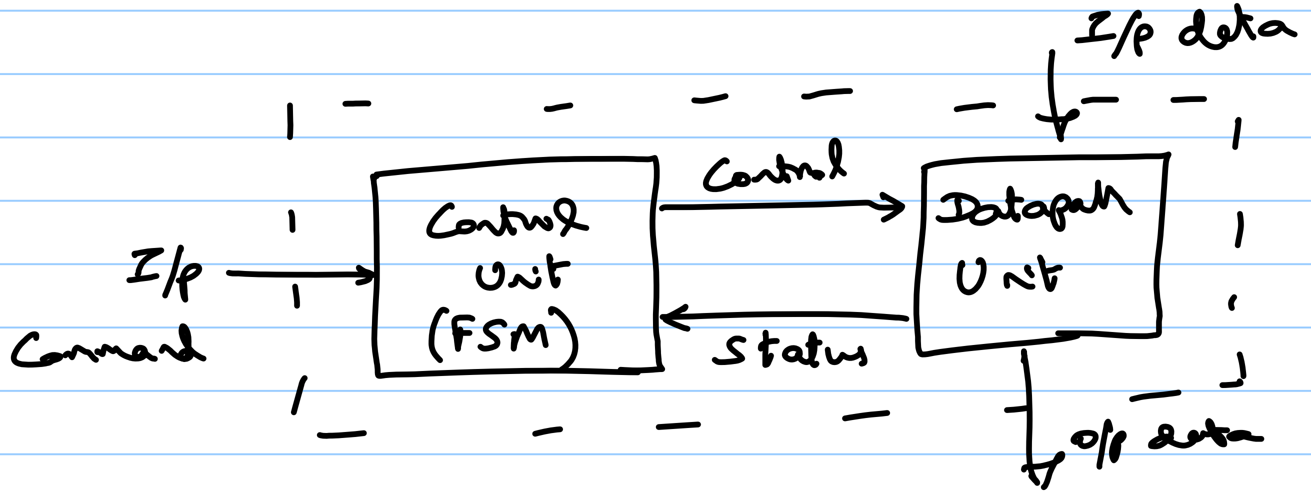


Binary information → Data
→ Control

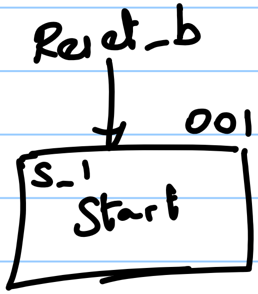
} specified by hardware algorithm

↓ Flowchart

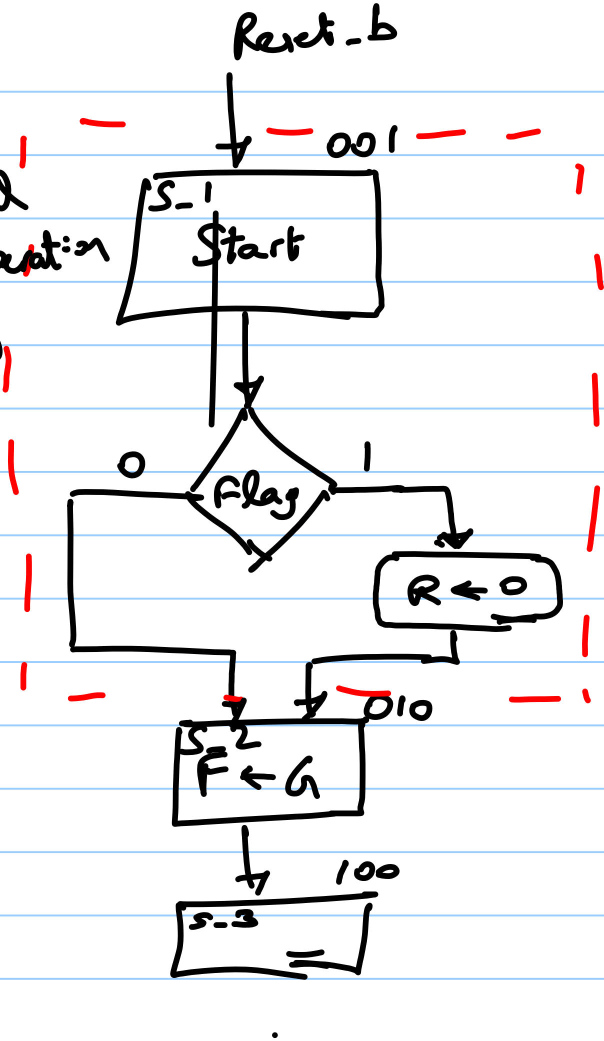
Algorithmic State Machine (ASM).



- Register Transfer (R1 ← R2)



Unconditional Register operation (Moore)



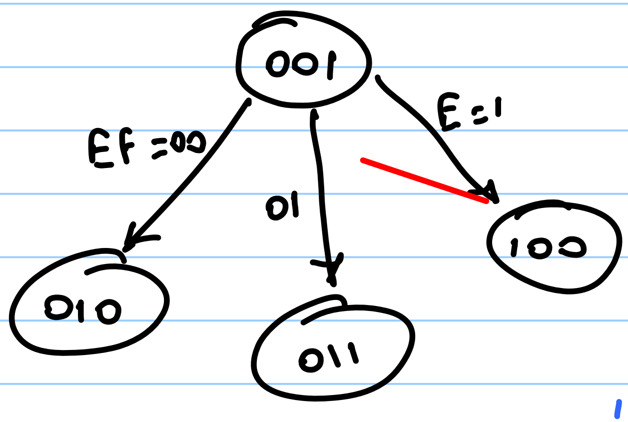
ASM Block (executed within one clock cycle)

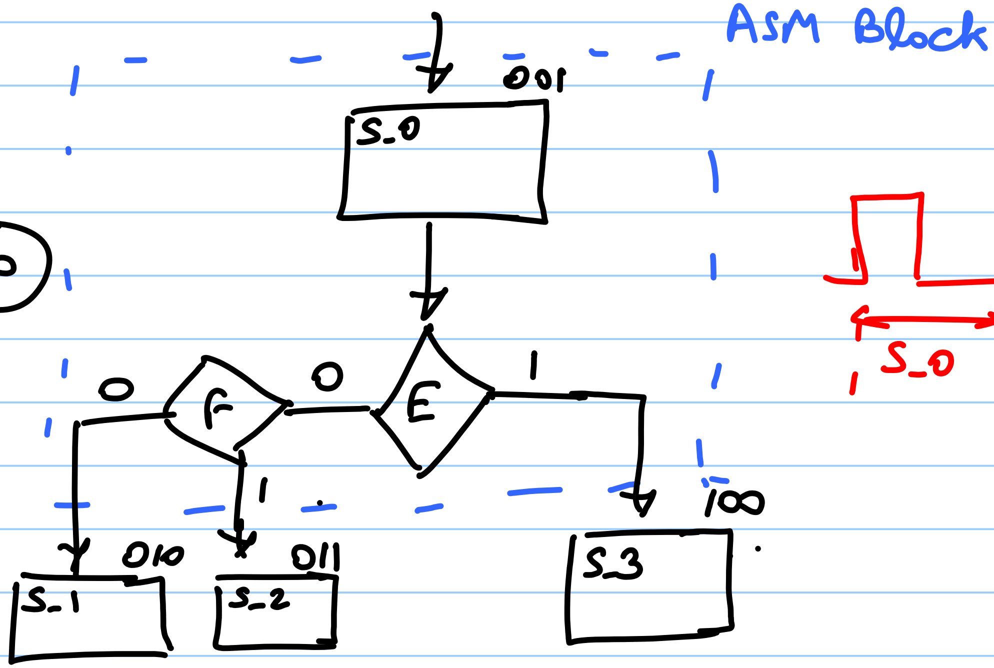
Conditional (Mealy)

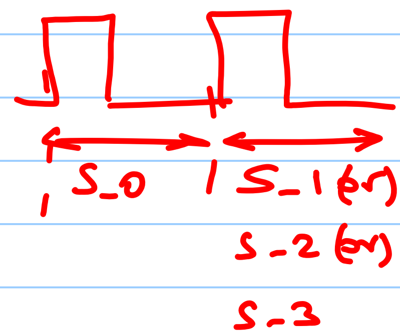
Control

Flash R

Load F ← G







Algorithmic State Machine & Datapath (ASMD)

ASMD Chart Design Example:

Datapath unit consists of two J-K flip-flops E & F

one 4-bit counter $\overset{MSB}{A_3}, A_2, A_1, \overset{LSB}{A_0}$

A signal START initiates the system's operation by clearing the counter A and flip-flop F

At each subsequent clock pulse, the counter is incremented by 1 until the operation stops

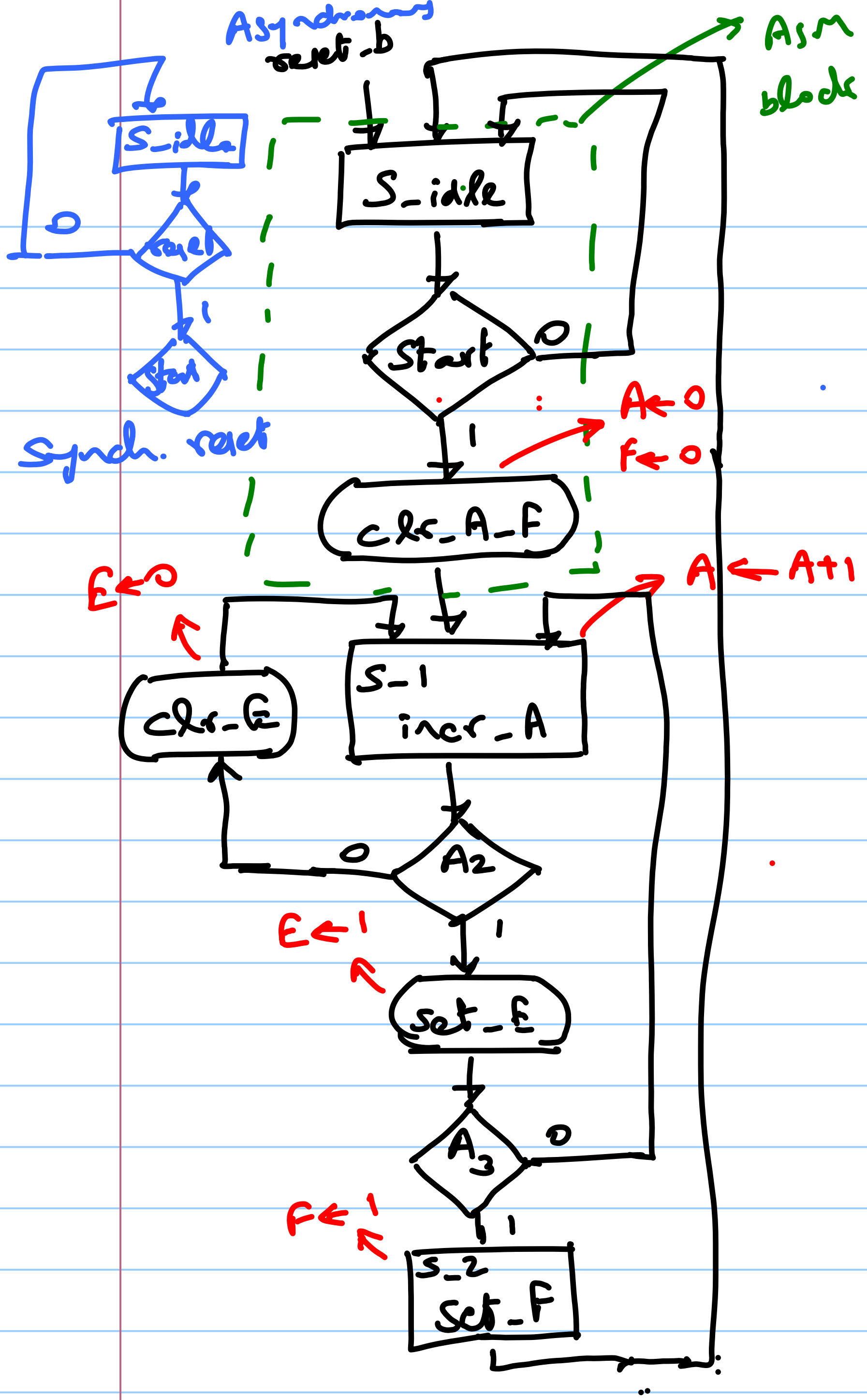
Counter bits A_2 and A_3 determine the sequence of operation.

→ If $A_2 = 0$, E is cleared to 0 and the count continues

If $A_2 = 1$, E is set to 1; then if $A_3 = 0$ count continues

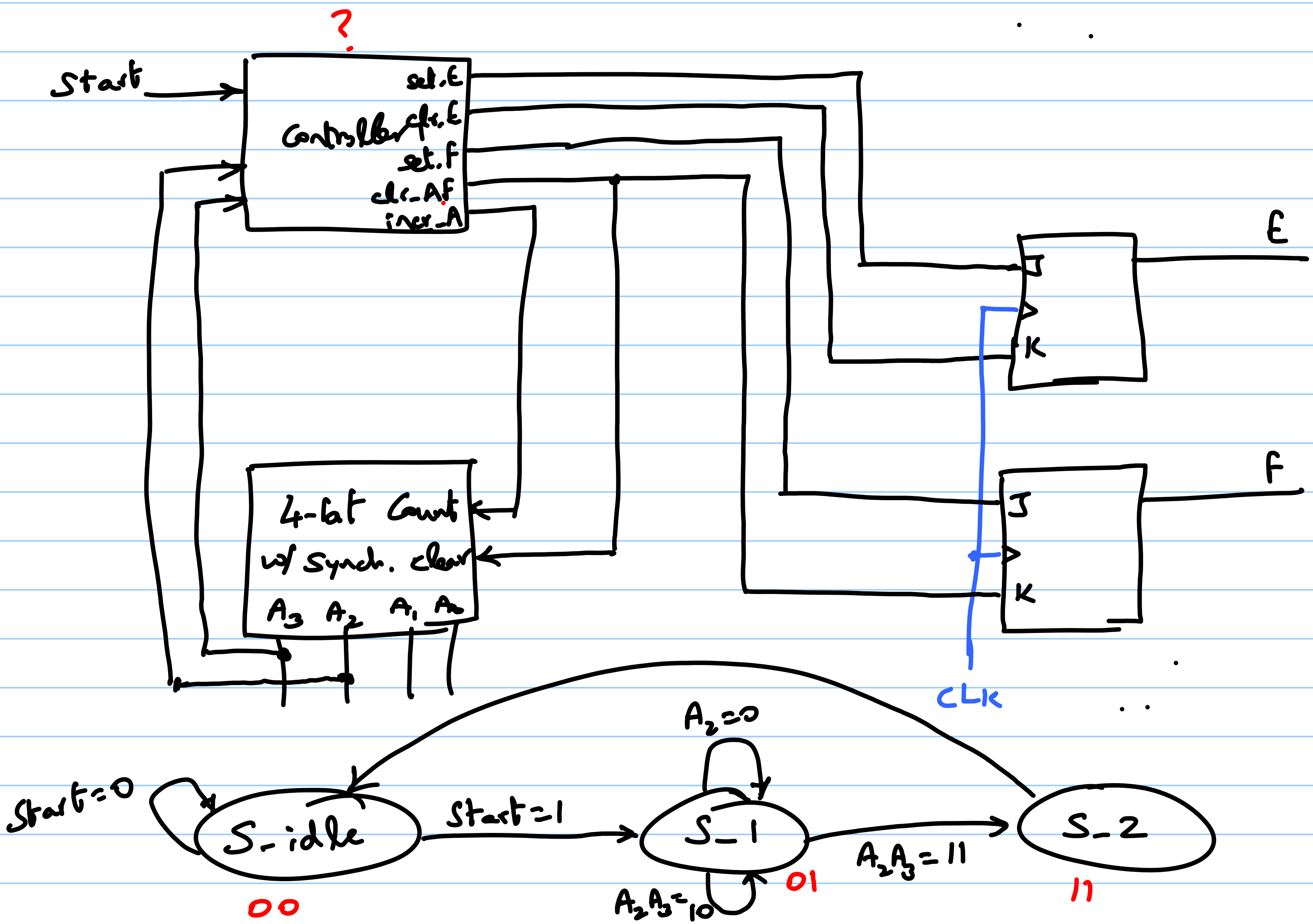
but if $A_3 = 1$, F is set to 1

counter stops on next clock pulse



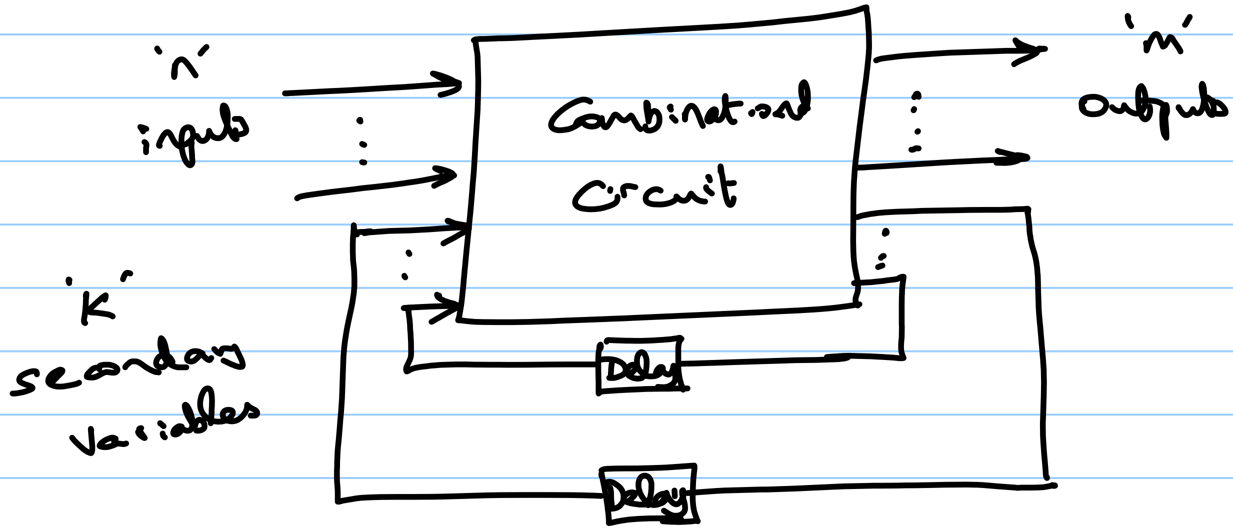
Sequence of operations

Counter				Flip-flops		Conditions	States
A_3	A_2	A_1	A_0	E	F		
0	0	0	0	1	0		S_idle
0	0	0	1	0	0	$A_2=0$	S_1
0	0	1	0	0	0	$A_3=0$	
0	0	1	1	0	0		
0	1	0	0	0	0		
0	1	0	1	1	0	$A_2=1$	S_1
0	1	1	0	1	0	$A_3=0$	
0	1	1	1	1	0		
1	0	0	0	1	0		
1	0	0	1	0	0	$A_2=0$	S_1
1	0	1	0	0	0	$A_3=1$	
1	0	1	1	0	0		
1	1	0	0	0	0		
1	1	0	1	1	0	$A_2=1$	S_2
1	1	0	1	1	1	$A_3=1$	S_idle


?
Start
Controller
set_E
clr_E
set_F
clr_AF
incr_A
4-bit Count
w/ synch. clear
A3 A2 A1 A0
J
K
E
F
CLK
Start=0
S_idle
Start=1
S_1
A2=0
A2A3=10
A2A3=11
S_2
00
01
11

PS		Inputs			NS		Outputs				
Symbol	G_1	G_0	Start	A_2	A_3	G_1^+	G_0^+	set_E	clr_E	set_F	clr_AF	incr_A
S_idle	0	0	0	x	x	0	0	0	0	0	0	0
	0	0	1	x	x	0	1	0	0	0	1	0
S_1	0	1	x	0	x	0	1	0	1	0	0	1
	0	1	x	1	0	0	1	1	0	0	0	1
	0	1	x	1	1	1	1	1	0	0	0	1
S_2	1	1	x	x	x	0	0	0	0	1	0	0

Asynchronous Sequential Circuits.

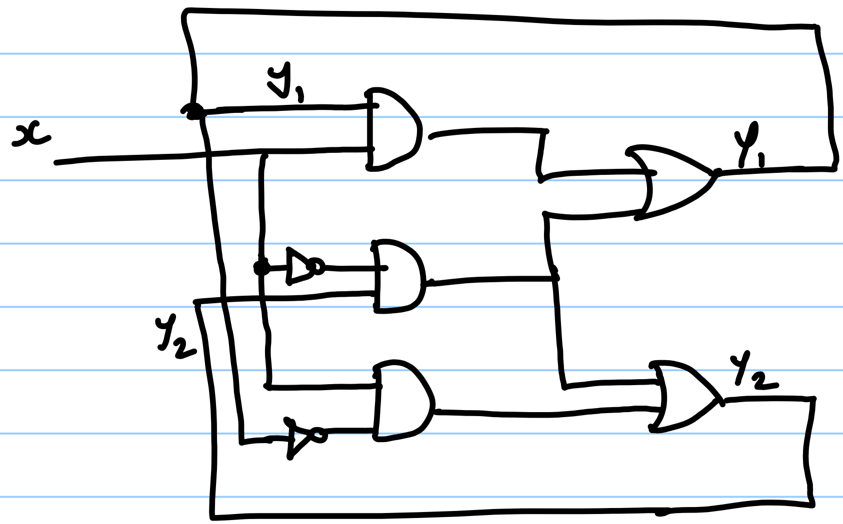


→ speed/energy/complexity in clock distribution

→ transition happens only in response to input variable

→ circuit must be allowed to attain stable state before input is changed

→ cannot have more than one input change @ any instant



$$Y_1 = xy_1 + x'y_2$$

$$Y_2 = xy_1' + x'y_2$$

State Table :

Y_1

y_1y_2 \ x	0	1
00	0	0
01	1	0
11	1	1
10	0	1

Y_2

y_1y_2 \ x	0	1
00	0	1
01	1	1
11	1	0
10	0	0

Y_1Y_2

y_1y_2 \ x	0	1
00	00	01
01	11	01
11	11	10
10	00	10

PS		NS $x=0$	NS $x=1$
0	0	0 0	0 1
0	1	1 1	0 1
1	0	0 0	1 0
1	1	1 1	1 0

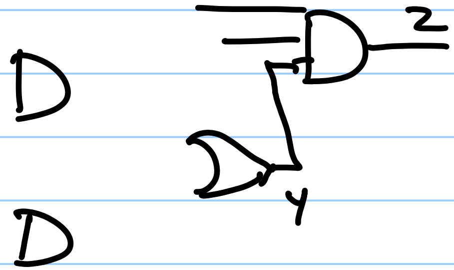
} : Atleast one state is same as prev. state

Flow Table

y \ x	0	1
a	a	b
b	c	b
c	c	d
d	a	d

Example

y \ x_1x_2	00	01	11	10
a	a,0	a,0	a,0	b,0
b	a,0	a,0	b,1	b,0



What is the corresponding circuit?

Y

y \ x_1x_2	00	01	11	10
0	0	0	0	1
1	0	0	1	1

$$Y = x_1x_2' + x_1y$$

z

			11	
	0	0	0	0
1	0	0	1	0

$$z = x_1x_2y$$

Race Conditions

y_1y_2 \ x	0	1
00	00 →	11
01		11
11		11
10		11

00 → 11

00 → 01 → 11

00 → 10 → 11

Non-critical race condition

y_1y_2 \ x	0	1
00	00	11
01		01
11		11
10		10

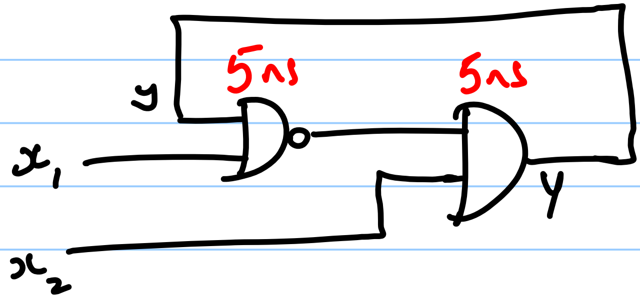
$\left.\begin{array}{l} 00 \to 11 \\ 00 \to 01 \\ 00 \to 10 \end{array}\right\}$ Different states $\Rightarrow$ Critical race condition

	0	1
00	00	01
01		11
11		10
10		01

$x \,.\; 0 \to 1 \quad (y = 00)$

$00 \to 01 \to 11 \to 10$

Stability Considerations

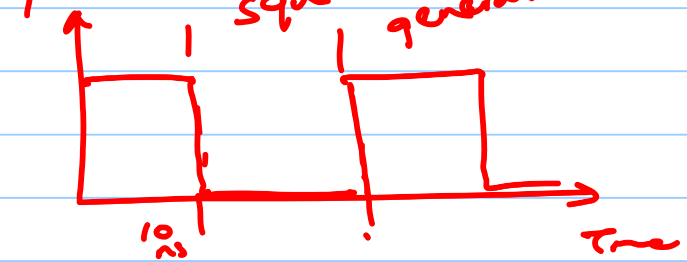


$$Y = (x_1 y)' x_2$$
$$= (x_1' + y') x_2$$
$$= x_1' x_2 + y' x_2$$

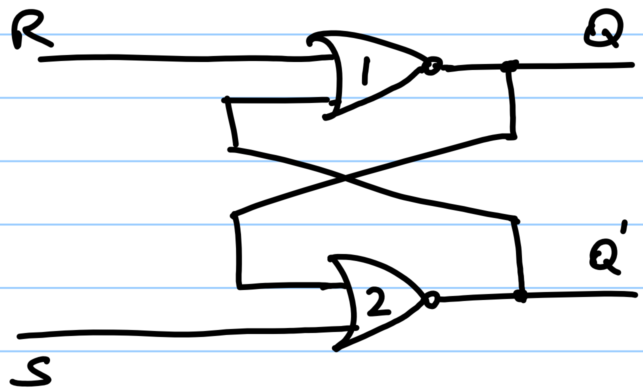
$y \backslash x_1x_2$	00	01	11	10
0	0	1	1	0
1	0	1	0	0

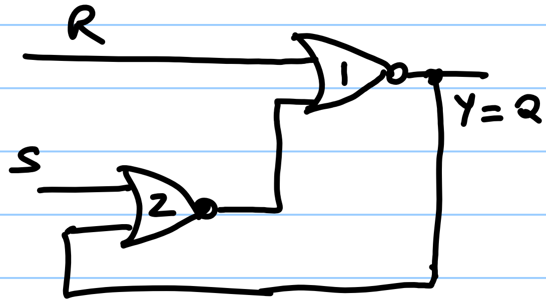
$Y = y \to$ stable state

50 MHz square wave generator



Circuits w/ Latches (S-R) :





S R	Q Q'
1 0	1 0
0 0	1 0
0 1	0 1
0 0	0 1
1 1	0 0

$$Y = \left((S+y)' + R\right)'$$

$$Y = (S+y)R' = SR' + yR'$$

y \ SR	00	01	11	10
0	0	0	0	1
1	1	0	0	1

→ Critical race

If SR = 11 → 00

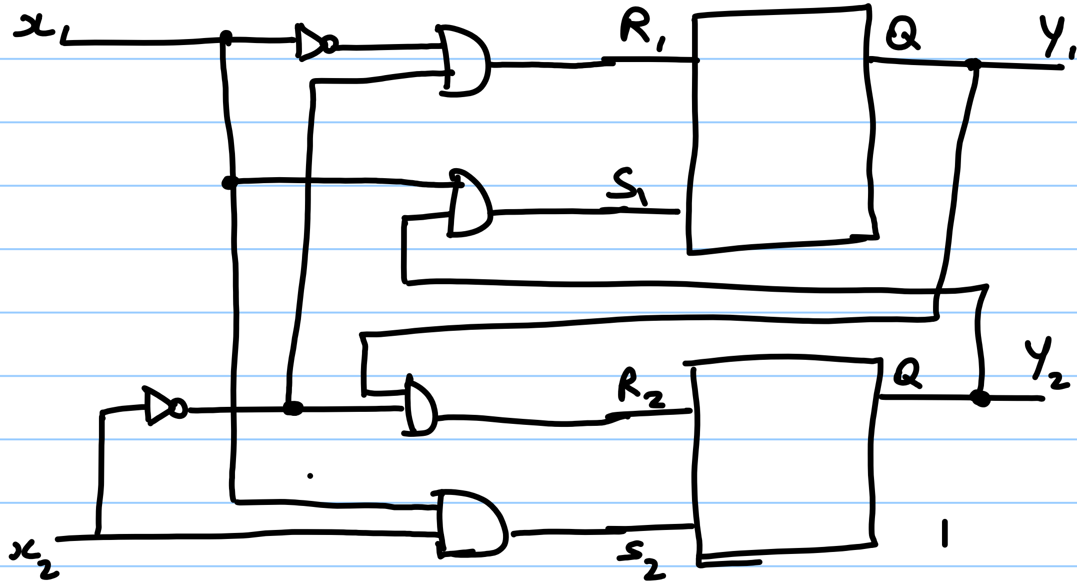
⇒ Needs to be avoided

$SR = 0$

$$SR + SR' = S$$

$\Rightarrow$ $\boxed{Y = S + yR'}$ when $SR=0$

Analysis of asynchronous seq. circuit w/ SR Latch:



$S_1R_1 = x_1y_2x_1'x_2$
$= 0$

$S_1 = x_1y_2$

$R_1 = x_1'x_2'$

$S_2R_2 = x_1x_2x_2'y_1 = 0$

$S_2 = x_1x_2$

$R_2 = x_2'y_1$

$$Y_1 = S_1 + R_1'y_1 = x_1y_2 + x_1y_1 + x_2y_1$$

$$Y_2 = S_2 + R_2'y_2 = x_1x_2 + x_2y_2 + y_1'y_2$$

y_1y_2 \ x_1x_2	00	01	11	10
00	00	00	01	00
01	01	01	11	11
11	00	11	11	10
10	00	10	11	10

Stable circuit

Critical race

$y_1y_2 = 11$, $x_1x_2 = 01 \to 00$
$x_2 \quad 1 \to 0$

} If Y_1 changes to zero before Y_2
Final state goes to 01 instead of 00

Implementation Example:

$$Y = x_1 x_2' + x_1 y$$

$y \backslash x_1x_2$	00	01	11	10
0	0	0	0	1
1	0	0	1	1

Excitation table

$y \rightarrow Y$		S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

S

$y \backslash x_1x_2$	00	01	11	10
0	0	0	0	1
1	0	0	X	X

R

$y \backslash x_1x_2$	00	01	11	10
0	X	X	X	0
1	1	1	0	0

$$S = x_1 x_2'$$

$$R = x_1'$$