

ECE124

Binary Func:

Outputs 0 or 1.

Jan 3, 2018

1) Truth table

x_0	x_1	x_2	$f(x_0, x_1, x_2)$
0	0	0	1
1	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Write rows in
increasing order

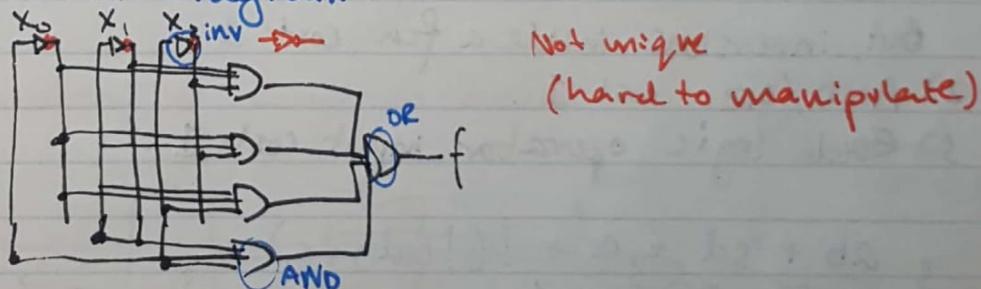
Unique

2) Logic Expressions

$$f = \overline{x_0} \overline{x_1} \overline{x_2} + \overline{x_0} x_1 \overline{x_2} + \overline{x_0} x_1 x_2 + x_0 x_1 x_2$$

Multi = and, add = or, \neg = inv
Not unique
(can manipulate)

3) Circuit Diagram:



Boolean Algebra:

1) Closure under + and \cdot .

precedence ↓
INV
AND
OR

2) Identity for + (0) and \cdot (1)

3) Commutative for both

4) Distributive for:

a) \cdot over +

$$x \cdot (y+z) = x \cdot y + x \cdot z$$

b) + over \cdot

$$x + (y \cdot z) = (x+y) \cdot (x+z)$$

5) $x + !x = 1$ and $x \cdot !x = 0$

6) \exists at least 2 elements where $x \neq y$.

Theorems:

- 1) $x+x=x$ and $x \cdot x = x$
- 2) $x+1=1$ and $x \cdot 0=0$
- 3) $\overline{\overline{x}}=x$
- 4) $x+(y+z)=(x+y)+z$ and $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- DMT 5) $!(x+y)=!x \cdot !y$ and $!(x \cdot y) = !x + !y$
- 6) $x+x \cdot y=x$ and $x \cdot (x+y)=x$

Example:

$$\begin{aligned}1) f &= ab + cd + a + !(\overline{cd}) + a \\&= a + cd + !(c + d + a) \\&= a + cd + cd \cdot !a \\&= a + cd\end{aligned}$$

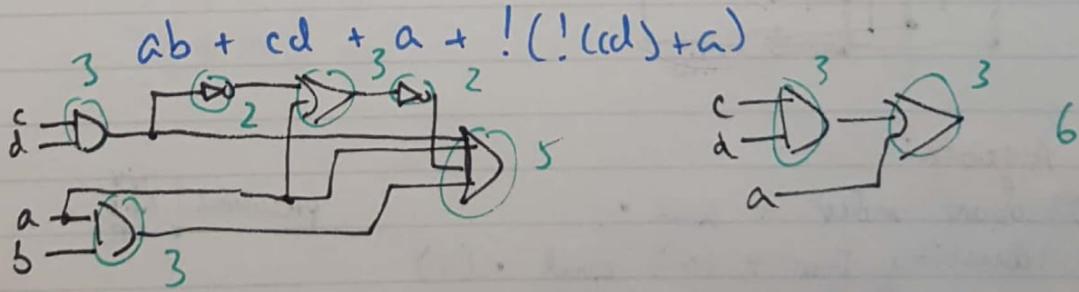
Cost:

1) Assume a and $!a$ available (inverters have 0 cost).

But, inversions inside a func cost 1.

2) AND/OR cost 1.

3) Each logic operator input costs 1.



Minterms:

Jan 8, 2018

Logical and for every input. 1 for current row, 0 for others.

x_0	x_1	x_2	minterms	f
0	0	0	$\bar{x}_0 \bar{x}_1 \bar{x}_2 = m_0$	0
0	0	1	$\bar{x}_0 \bar{x}_1 x_2$	1
0	1	0	$\bar{x}_0 x_1 \bar{x}_2$	0
0	1	1	$\bar{x}_0 x_1 x_2$	0
1	0	0	$x_0 \bar{x}_1 \bar{x}_2$	1
1	0	1	$x_0 \bar{x}_1 x_2$	0
1	1	0	$x_0 x_1 \bar{x}_2$	0
1	1	1	$x_0 x_1 x_2 = m_7$	1

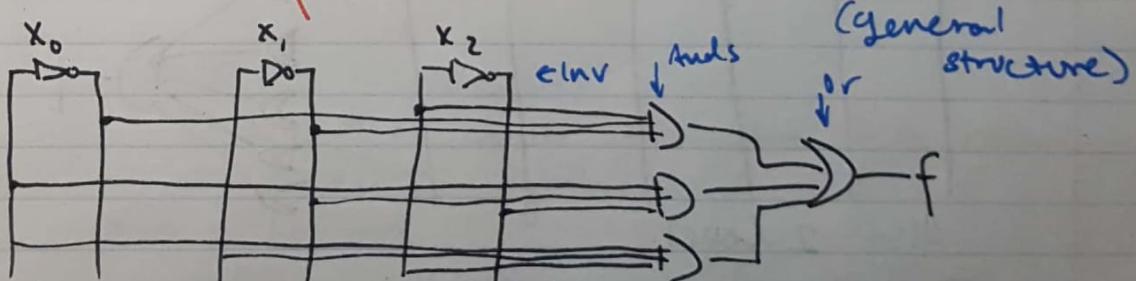
(turn function
on)

To write logic exp using table, OR the minterms for all $f(_) = 1$.

$$f = \bar{x}_0 \bar{x}_1 x_2 + x_0 \bar{x}_1 \bar{x}_2 + x_0 x_1 \bar{x}_2 = m_1 + m_4 + m_5$$

↪ "sum of minterms", "canonical sum of products"
↪ $\Sigma m(1, 4, 5)$

Implementation of Minterm Sum:



SOP's are 2-level circuits

① ②

Maxterms:

OR for every input

0 for current row, else 1.

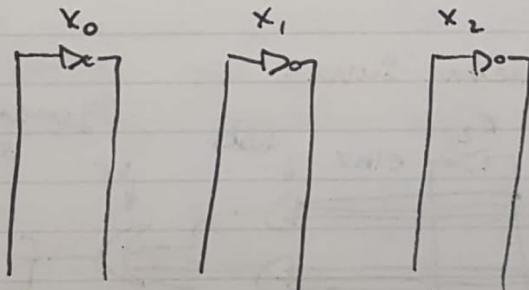
(Turn function off)

x_0	x_1	x_2	maxterms
0	0	0	$x_0 + x_1 + x_2$ = M_0
0	0	1	$x_0 + x_1 + \bar{x}_2$
0	1	0	$x_0 + \bar{x}_1 + x_2$
0	1	1	$x_0 + \bar{x}_1 + \bar{x}_2$
1	0	0	$\bar{x}_0 + x_1 + x_2$
1	0	1	$\bar{x}_0 + x_1 + \bar{x}_2$
1	1	0	$\bar{x}_0 + \bar{x}_1 + x_2$
1	1	1	$\bar{x}_0 + \bar{x}_1 + \bar{x}_2$ = M_7

To get logic expression, AND the maxterms for when $f(x) = 0$.

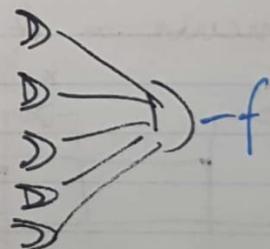
$$f = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 = \prod M(0, 2, 3, 5, 6)$$

Implementation:



Also 2 level.

General Case.



$$\bar{f} = \sum m(1, 4, 7) = \sum M(0, 2, 3, 5, 6)$$

$$\begin{aligned} \bar{f} &= \overline{(m_1 + m_4 + m_7)} \\ &= \overline{(m_1, m_4, m_7)} \leftarrow \bar{f} \\ &= \overline{(m_0 + m_2 + m_3 + m_5 + m_6)} \\ &= \overline{m_0 \bar{m}_2 \bar{m}_3 \bar{m}_5 \bar{m}_6} \\ &= \sum M(0, 2, 3, 5, 6) \end{aligned}$$

min/maxterms are not optimal.

Other Logic Gates:

Jan 10, 2018

NAND - AND then NOT

↪ $\overline{\square} \square$ symbol

NOR - OR then NOT

↪ $\overline{\square} \square$ symbol

XOR - exclusive OR

↪ 1 if inputs are diff. (for 2 input)

↪ 1 if # of inputs which are 1 is odd (for 3+ input)

$$\hookrightarrow f = \overline{x_0}x_1 + x_0\overline{x_1} = x_0 \oplus x_1$$

↪ $\overline{\square} \square$ symbol

x_0	x_1	x_2	F
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

Addition
fn (w/o carry)

NXOR - XOR then NOT

↪ 1 if # of inputs which are 1 is even

↪ $\overline{\square} \square$ symbol

BUF - buffer

↪ Does nothing

↪ $x \rightarrow \square$ symbol

x	f
0	0
1	1

→ Delays

→ increase power

TRI STATE BUF - tristate buffer

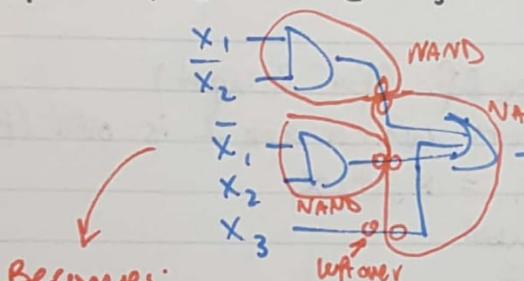
↪ $x \rightarrow \square^{en} \rightarrow f$ $f = x$ when $en = 1$, otherwise
 f is disconnected when $en = 0$.

Circuit Implementation of NAND:

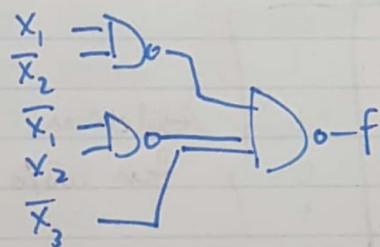
Easy to convert a 2-level SOP to an imp. Using only NANDs

Example:

$$1) F = x_1 \bar{x}_2 + \bar{x}_1 x_2 + x_3$$



Becomes:



Can do using only NAND

$$f = \overline{\overline{x_1 \bar{x}_2 + \bar{x}_1 x_2 + x_3}}$$

$$= \overline{[(x_1 \bar{x}_2) \cdot (\bar{x}_1 x_2) \cdot x_3]}$$

Basically put 2 inverters
on every wire.
Put 2 inv after
every NAND

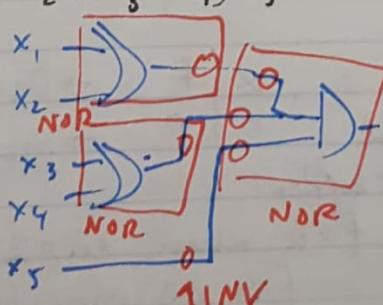
Circuit Implementation of NOR:

Jan 12, 2018

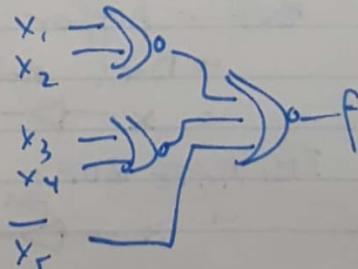
Easy to convert a 2-level POS to an imp. using only NORs

Example:

$$1) f = (x_1 + x_2)(x_3 + x_4) \cdot x_5$$



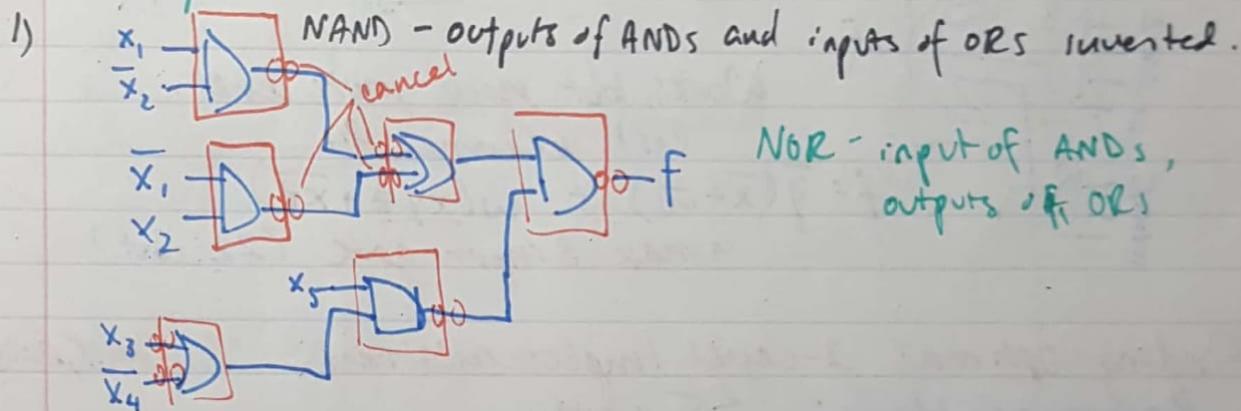
$$f = \overline{[(\overline{x_1 + x_2}) + (\overline{x_3 + x_4}) + \overline{x_5}]}$$



Multi-Leveled Circuits:

Can be done graphically. Structure doesn't change.

Example:

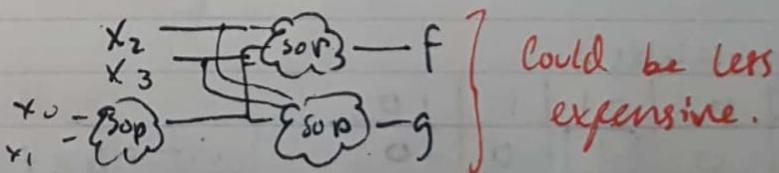
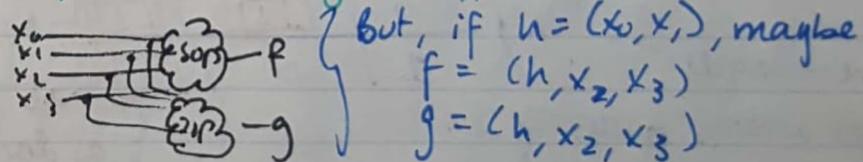


Can always write 2-level SOP/POS

→ not always practical (very large gates)

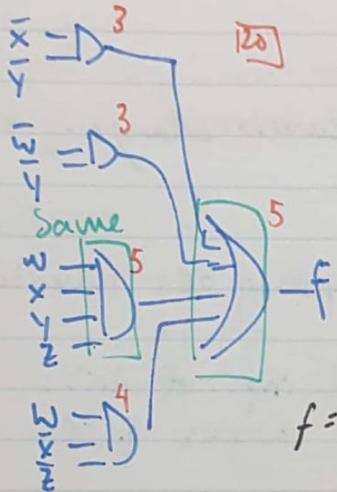
→ could have multiple fans that share a ab expr.

$$\Leftrightarrow f(x_0, x_1, x_2, x_3) \quad g(x_0, x_1, x_2, x_3)$$



Example:

1) $f = \bar{x}\bar{y} + \bar{w}\bar{y} + wxyz + w\bar{x}\bar{z}$ min. SOP.



120 4 input gates don't exist.

Use associativity
 $\Rightarrow = \Rightarrow \rightarrow$

Works, but now multi-level.

Cost becomes 24

$$f = \bar{y}(\bar{x} + \bar{w}) + w(xy\bar{z} + \bar{x}\bar{z})$$

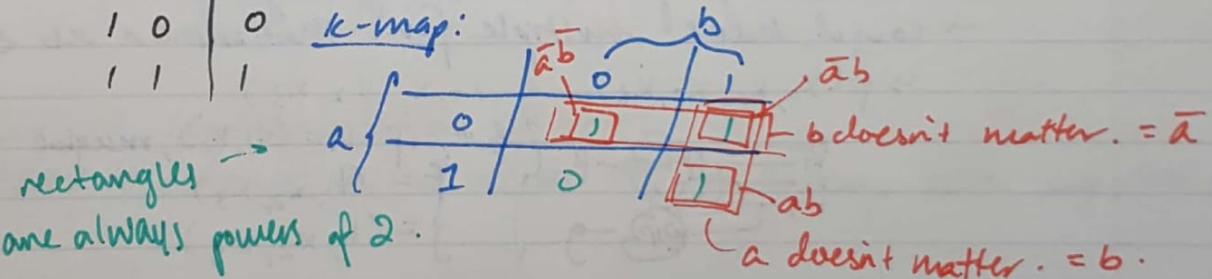
max 3 input gate (22 cost)

Finding Optimal 2-level Implementations: TAN15, 2018

Karnaugh Maps: ≤ 5 inputs.

a	b	f
0	0	1
0	1	1
1	0	0
1	1	1

(min SOP) $f = \bar{a}\bar{b} + \bar{a}b + ab$
 $= \bar{a}\bar{b} + \bar{a}b + \bar{a}b + ab$
 $= \bar{a} + b$.



3-variables:

a	bc	00	01	11	10
0	0	0	1	0	0
1	0	1	1	1	1
	\bar{bc}			ab	

$$\therefore f = \bar{b}c + ab$$

Ordered w/c adjacent columns only differ by 1 var.

More examples:

$a \backslash bc$	00	01	11	10
0	0	1	1	1
1	1	1	1	0

$$f = a\bar{b} + \bar{a}b + c$$

$a \backslash bc$	00	01	11	10
0	1	0	0	1
1	1	1	1	1

wraps around

$$f = a + \bar{c}$$

$a \backslash cd$	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	0	0

$$f = \bar{c} + \bar{a}\bar{d} + b\bar{d}$$

$a \backslash cd$	00	01	11	10
00	1	1	1	0
01	1	1	1	0
11	0	0	1	1
10	0	0	1	1

Jan 17, 2018

$$f = \bar{a}\bar{c} + cd + ac$$

$$\text{OR } f = \bar{a}\bar{c} + ac + \bar{a}cd$$

∴ More than one min SOP

5 variable Maps:

$b \backslash de$	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	0	0
10	0	1	0	0

$b \backslash de$	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	0	1	0	1

Join same rectangle

when $a=0$

$$f = \bar{a}ce + \bar{a}\bar{b}de + \bar{a}\bar{b}\bar{c}\bar{e} + ace + ab\bar{d}e + \bar{a}\bar{b}\bar{c}\bar{e} + bd\bar{e}$$

$$= ce + b\bar{d}e + \bar{b}\bar{c}\bar{e} + ab\bar{d}\bar{e}$$

Minimum POS:

$\bar{a} \backslash \bar{c}$	00	01	11	10
00	1	1	1	0
01	1	1	1	0
11	0	0	1	1
10	0	0	1	1

$a + \bar{c} + d$

$f = (\bar{a} + c)(a + \bar{c}d)$

* Another way:

Find $\text{not } f$ and SOP, invert SOP \rightarrow POS.

Function Don't Cares (Andy Take):

Denote with "x" (don't care about output)

How to handle? Cheapest way.

$\bar{a} \backslash \bar{c}$	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	2	0
10	0	0	1	0

$F = \bar{a}d + cd$. { Not equal
or $f = cd + \bar{a}b$ b/c x's role }

Multiple Functions:

$\bar{a} \backslash \bar{c}$	00	01	11	10
00	0	0	0	0
01	1	1	0	0
11	1	0	0	0
10	0	1	0	0

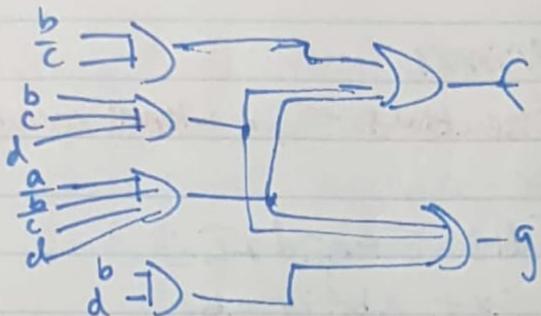
$\bar{a} \backslash \bar{c}$	00	01	11	10
00	0	0	0	0
01	1	0	1	1
11	1	0	0	1
10	0	1	0	0

Jan 19, 2018

$$f = b\bar{c} + bd + a\bar{c}d$$

$$g = bd + b\bar{c} + a\bar{b}\bar{c}d$$

To imp. f, g separately (individually optimal SOP) = 2g
 → make $f = b\bar{c} + bd + a\bar{b}\bar{c}d$ can be shared.
 → then $f = b\bar{c} + bd + a\bar{b}\bar{c}d$, $g = bd + b\bar{c} + a\bar{b}\bar{c}d$



cheaper than
individual fees.

Implicants:

A product term is an implicant: { ex: minterm
 \rightarrow term = 1 $\Rightarrow f = 1$. } when $f = 1$.

Prime Implicants:

Removing any variable \Rightarrow no longer implicant

Essential Prime:

Includes minterm not inside another prime.

Example:

1)

	bc			
	00	01	11	10
a	0	1	1	0
b	1	1	1	1

$\bar{a}\bar{b}\bar{c}$, $\bar{a}\bar{b}c$, $\bar{a}b\bar{c}$, $a\bar{b}c$, abc
 $\bar{b}c$, $\bar{a}b$, $a\bar{c}$, $\bar{a}\bar{b}$, $\bar{b}\bar{c}$
 b prime

x can't make
rect larger.

ac essential, same w/ b.

* Can always implement f w/ only prime implicants
 (largest rectangles)

- Want as few primes as possible, while getting all minterms.
- Must include e. primes (they have minterms not in any other primes).

2)

	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	1	1	1	0
10	0	0	1	1

Primes

Essentials —, minterms covered *

left over

$$f = \bar{a}\bar{b}d + ab\bar{c} + b\bar{c}d + \bar{b}c$$

$$\Leftrightarrow abd \text{ or } abc$$

2 of these.

Quine-McClusky:

More algorithmic for min SOP.

1) Get all primes

2) Get essentials and other needed primes.

Example:

i) $f = \sum m(0, 2, 3, 5, 10, 11, 12, 13, 15)$ x not prime (was merged) ✓ prime

# of 1s	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0000	x	(0, 2)	00-0												
1	0010	x	(2, 3)	001-	x											
2	0011	x	(2, 10)	-010	x											
3	0101	x	(3, 11)	-011	x											
4	1010	x	(5, 13)	-101	✓											
5	1100	x	(10, 11)	101-	x											
6	1011	x	(12, 13)	110-	✓											
7	1101	x	(11, 15)	1-11	✓											
8	1101	x	(13, 15)	11-1	✓											

$$f = \bar{a}\bar{b}d + b\bar{c}d + ab\bar{c} + \bar{b}c + (acd \text{ or } abd)$$

Found primes, now find essentials. "Covering Matrix"

0, 2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5, 13																
12, 13																
11, 15																
13, 15																
12, 3, 10, 11																

Columns
two are ✓
one essential!need 1 of
these four
12, 3, 10, 11✓ 15
left
over

If you have Dent Care:

Jan 22, 2018

- 1) Assume it's a 1 and check final exp.
- 2) DONT include the x in the matrix.
- 3) If your final soln includes the dent care term, it's optimal to make it a 1.

→ prev example: $\bar{bc} + \bar{b}\bar{c} + \begin{cases} \bar{a}\bar{c}\bar{d} & \text{if } a \\ \bar{a}\bar{b}\bar{d} & \text{if } b \end{cases}$

Unsigned Nums:

Jan 24, 2018

"positioned number notation" v in base r using n digits:

$$v = d_{n-1}r^{n-1} + \dots + d_0r^0$$

most significant least significant

$$= \sum_{i=0}^{n-1} d_i r^i$$

Base r :
Digits $0 \rightarrow r-1$.
(base 10, 0-9).

Converting:

Repeated division.

$$\frac{v}{r} = \sum_{i=0}^{n-1} d_i r^{i-1} = \underbrace{\sum_{i=1}^{n-1} d_i r^{i-1}}_{\text{Quotient}} + \underbrace{\frac{d_0}{r}}_{\text{remainder}}$$

Example:

1) 53 in base 2:

#	Q	R
53	26	1 d_0
26	13	0 d_1
13	6	1 d_2
6	3	0 d_3
3	1	1 d_4
1	0	1 d_5

stop!

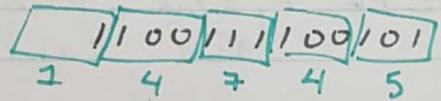
$$\therefore 53 = (110101)_2$$

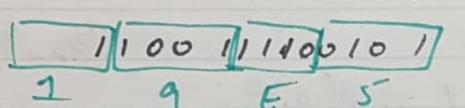
$$2) (23)_5 \rightarrow \text{base-3}$$

↪ Get $(23)_5$ in base 10 \rightarrow then base 3.

Powers of 2 Bases:

$$(1100111100101)_2 \Rightarrow \text{base } 8? 16?$$

 Base 8 - groups of 3 (2^3)

 Base 16 - groups of 4 (2^4)

Addition:

$$\begin{array}{r} 10101010 \\ + 00110000 \\ \hline 11011010 \end{array}$$

left over? Add digits ($1+0+b$)

↪ Would lose data as computer doe.

↪ "unsigned overflow"

↪ n-bits? $[0..2^n-1]$

Subtraction:

$$\begin{array}{r} 3122 \\ + 0401 \\ \hline 3523 \\ - 00110 \\ \hline 01111010 \end{array}$$

need to borrow? Unsigned underflow

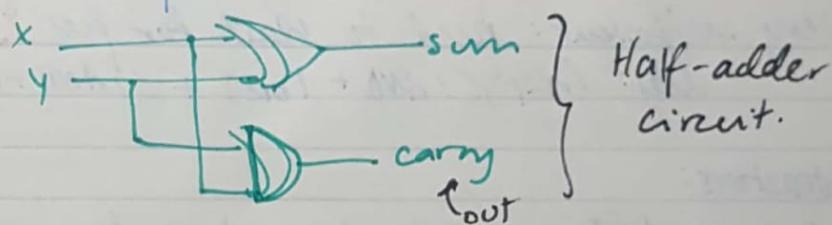
Arithmetic Circuits:
Half Adder:

Jan 26, 2018

Circuit to add 2 bits x, y ?

x	y	$f(\text{sum})$	carry out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum: $x \oplus y$
Carry: $\text{and } xy$.
 \uparrow
 c_{out}

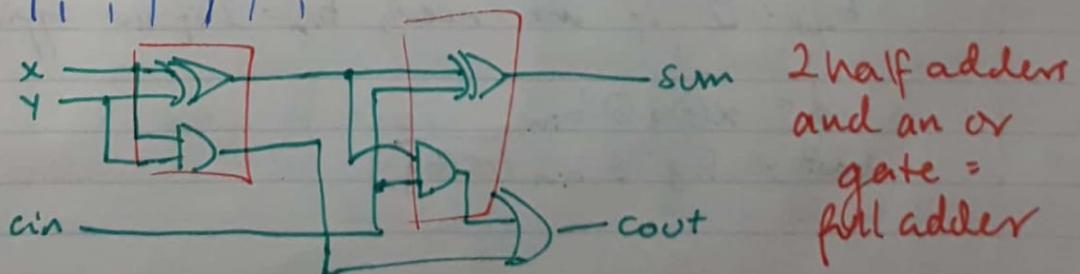


Full Adder:

Circuit to add 3 b.ts, $x + y + \text{carry in}$

x	y	c_{in}	sum	c_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

sum: $x \oplus y \oplus c_{\text{in}}$
cout: $x y + c_{\text{in}}(x \oplus y)$
= $g + c_{\text{in}}p$
generate propagate



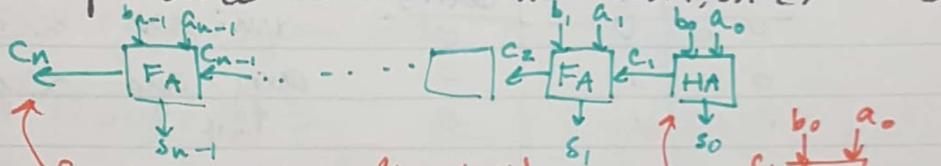
2 half adders
and an or
gate =
full adder

Ripple Adder:

One circuit to add 2 n-bit unsigned numbers

$A(a_{n-1}, a_{n-2}, \dots, a_0)$, $B(b_{n-1}, b_{n-2}, \dots, b_0)$

to produce n-bit sum $S(s_{n-1}, s_{n-2}, \dots, s_0)$ and carry



final cout (overflow bit)

Very inefficient: need to wait for full computation.

delay: $(n-1)(1\text{AND} + 1\text{OR}) + (1\text{AND} + 1\text{OR} + 1\text{XOR})$

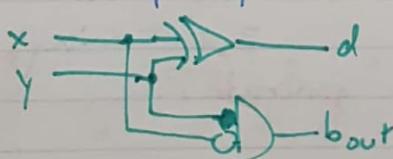
Half Subtractors:

$x - y$. difference d, borrow bit bout.

x	y	d	bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$d = x \oplus y$$

$$b_{out} = \overline{x}y$$



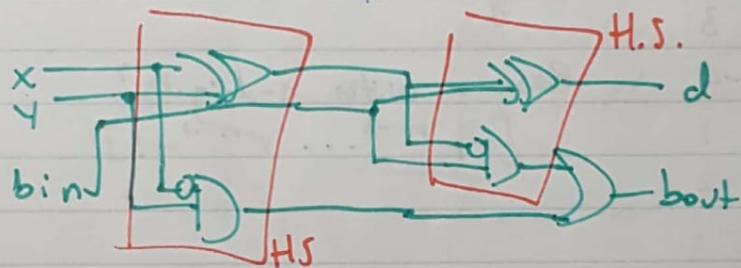
Full Subtraction:

b_{out} : 1 if we need, $b_{in} = 1$ if we supply.

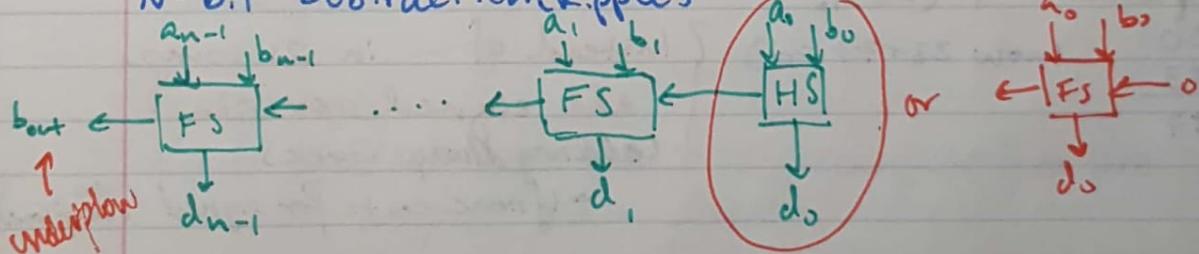
$$d = x \oplus y \oplus b_{in}$$

$$b_{out} = \overline{x}y + b_{in}(x \oplus y)$$

x	y	bin	d	bout
0	0	0	0	0
0	1	0	1	1
1	0	0	1	0
1	1	0	0	0
0	0	1	1	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	1



N-bit Subtraction (Ripple):



Jan 29, 2018

Signed Numbers:

Represent signed ints in base r with n digits?

✗ Can use sign bit (bad) \rightarrow 2 reps for 0, comp. circuits

✓ Radix Complements: + value V, base r, n-bits:

r is comp of V if:

$r^n - V$ if $V \neq 0$ flip all bits, add 1

0 if $V = 0$

✓ flip all bits left of 1

Example:

1) $V = 88, r = 2, n = 8$.

$$88: (01011000)_2 \rightarrow 2^8 - \text{this}$$

$$-88: (\underline{1}010\underline{1}000)_2$$

2) 3-bit 2's complement digits:

000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

Unsigned
Signed.

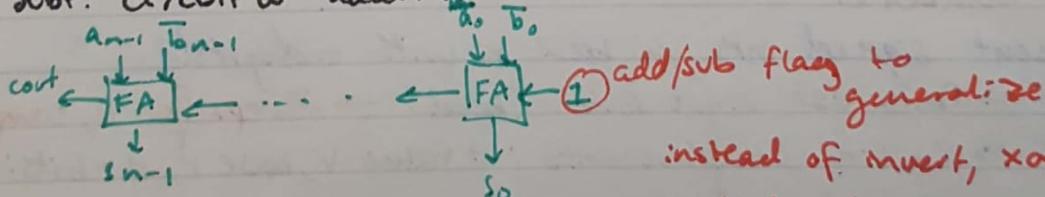
Base r with n -digits?
 $[-r^{n-1} \dots r^{n-1} - 1]$

3) 2's complement of $(23)_{10}$?

$$\begin{array}{r} 100 \\ - 23 \\ \hline 77 \end{array} \quad (\text{now } 23 + 77 = 0)$$

} Instead of - in 2's comp,
add signed version.
(adding always works)
- ignore carry for signed (overflow)

4) Subt. circuit w/ addition:



$$b_0 \oplus 0 = b_0$$

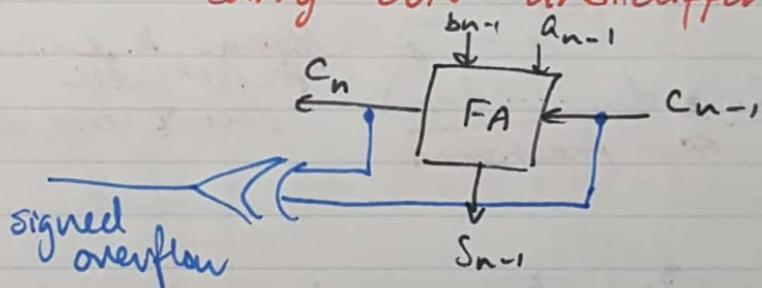
$$b_0 \oplus 1 = \bar{b}_0$$

Overflow Detection:

Jan 30, 2018

$$\begin{array}{r}
 \text{ignore } 0 \\
 + \begin{array}{r} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{array} \quad \begin{array}{r} 00 \\ 0000 \\ 0110 \\ 1001 \\ (+) \end{array} \quad \begin{array}{r} 0110 \\ 0000 \\ 1001 \\ 0110 \\ (+) + (+) = (-) \end{array} \quad \begin{array}{r} 70 \\ +80 \\ 150 \\ \text{Largest } + \text{ is } 127. \end{array} \\
 \hline
 \text{ignore } 1 \\
 - \begin{array}{r} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{array} \quad \begin{array}{r} 11 \\ 1011 \\ 1010 \\ 10110 \\ 000 \end{array} \quad \begin{array}{r} -70 \\ -80 \\ -150 \\ \text{Largest } - \text{ is } -128. \end{array}
 \end{array}$$

Never get overflow when last two carry outs aren't different.



Faster Adders:

Ripple Adders must wait + recompute for prev. adder.

↳ compute carrys faster

Before: $c_1 = g_1 + c_0 p_0$

Finishes at Time 1: $c_2 = g_1 + c_1 p_1$

Computed at time 1.

Time 0 Time 3

↳ Solution? Simplify.

$$c_2 = g_1 + p_1(g_0 + c_0 p_0) \leftarrow \text{same}$$

$$= \underline{\underline{g_1}} + \underline{\underline{g_0}} \underline{\underline{p_1}} + \underline{\underline{p_1}} \underline{\underline{p_0}} \underline{\underline{c_0}} \leftarrow \text{different}$$

\leftarrow now, c_2 finishes @ time 3.

↳ Similarly, $c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$.

→ Also finishes @ time 3

- $\sim O(N)$ {
 - Ripple adders:
 - ↳ Multi-level small gates
- $O(3)$ {
 - Carry look-ahead:
 - ↳ Collapses into 2-level SOP, large gates.
 - Much faster (kinda like concurrency)
 - Impractical (large gates don't exist)
 - Can merge the two, ex: use 17-CLA adders for 16-bits

Fixed Point Numbers

- When you have fixed range. (floating pt: changing rng).
- Int: positional notation.

$$D = \underbrace{(d_{n-1} \dots d_0)}_{\text{int}} \underbrace{\underline{\dots} \underline{d}}_{\text{fp}} \underbrace{\underline{\dots} \underline{d_{-k}}}_{\text{fractional}}$$

Always k decimals
"fixed" (add/
doesn't leave range)

Example:

$$\begin{aligned} 1) (1321.312)_4 &= 1 \times 4^3 + 3 \times 4^2 + 2 \times 4^1 + 1 \times 4^0 + 3 \times 4^{-1} + 1 \times 4^{-2} + 2 \times 4^{-3} \\ &= 121.84375 \end{aligned}$$

2) Go backwards.

→ Integer is obv (repeated div)

→ Frac: repeated mult:

$$0.625 \times 2 = 1.250 \quad (d_{-1} = 1)$$

$$0.250 \times 2 = 0.500 \quad (d_{-2} = 0)$$

$$0.500 \times 2 = 1.000 \quad (d_{-3} = 1)$$

When dealing in fracs, 2's comp doesn't really work.
→ Use sign bit instead.

Multiplexers:

Jan 31, 2018

Inputs, Outputs, and select line.

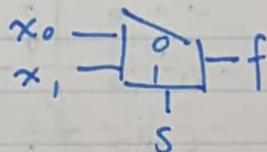
$\rightarrow n$ inputs $\Rightarrow \log(n)$ select lines.

\rightarrow Output is one of the inputs based on select line
(can mix)

2-to-1 Multiplexer:

If statement	S	F
	0	x_0
x_0, x_1 , input	1	x_1

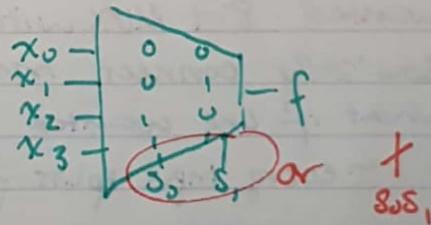
\rightarrow Symbol:



4-to-1 MUX:

S ₀	S ₁	F
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

$$f = \bar{s}_0 \bar{s}_1 x_0 + \bar{s}_0 s_1 x_1 + s_0 \bar{s}_1 x_2 + s_0 s_1 x_3$$

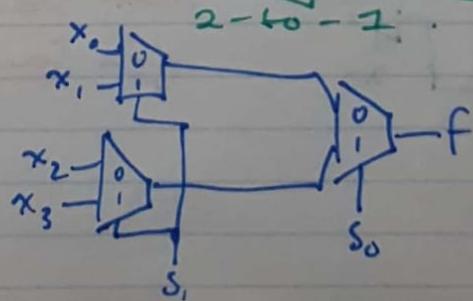


$$+ s_0 s_1$$

Example:

4-to-1 MUX \approx 2-to-1 Mux's?

$$f = \underbrace{\bar{s}_0 (\bar{s}_1 x_0 + s_1 x_1)}_{2\text{-to-1}} + \underbrace{s_0 (\bar{s}_1 x_2 + s_1 x_3)}_{2\text{-to-1}}$$

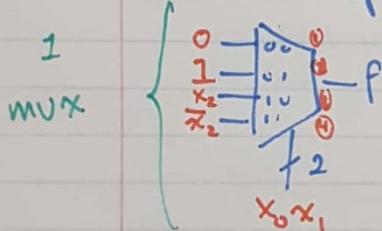


Uses:

- Steer data and implement logic funcs
- N-input func can be done in a MUX in $n-1$ select lines.
 - Connect $n-1$ inputs to select lines,
 - then last input to mux data.

Example:

1) $f = \bar{x}_0 x_1 + x_0 \bar{x}_1 x_2 + x_0 x_1 \bar{x}_2$ in 1 MUX?
3 inputs \Rightarrow 2 select lines.



$x_0 x_1$	x_2	f
00	0	0
01	0	1
01	1	0
10	0	1
011	1	1
000	0	0
101	1	1
110	0	1
111	0	0

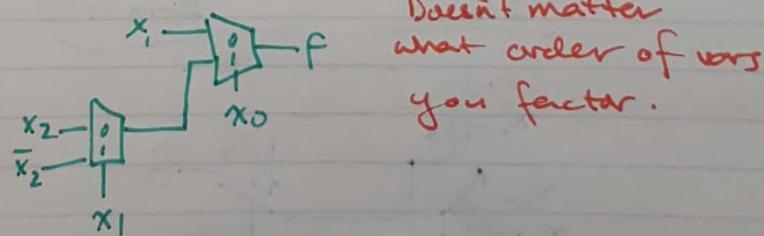
* Note: If wanted 8-1 MUX with 8 select lines:

→ Basically connect truth table.

→ What if one wanted smallest MUX over largest?

Shannon Fitting → → every input splits func into 2 halves ($x(\dots) + \bar{x}(\dots)$)

2) $f = \bar{x}_0 x_1 + x_0 \bar{x}_1 x_2 + x_0 x_1 \bar{x}_2$
 $= \bar{x}_0 (x_1) + x_0 (\bar{x}_1 x_2 + x_1 \bar{x}_2)$
 $= \bar{x}_0 (x_1) + x_0 [\bar{x}_1 (x_2) + x_1 (\bar{x}_2)]$



$$3) f = \bar{x}_0 x_1 + x_0 \bar{x}_1 x_2 + x_0 x_1 \bar{x}_2$$

$$x = \bar{x}_2 (x_0 x_1) + x_2 (x_0 \bar{x}_1 + \dots) \text{ whoops } (\text{no } \bar{x}_2).$$

$$\checkmark = \bar{x}_2 (x_0 x_1 + \bar{x}_0 \bar{x}_1) + x_2 (x_0 \bar{x}_1 + \bar{x}_0 x_1)$$

↳ Basically or with x_2, \bar{x}_2 in original fn

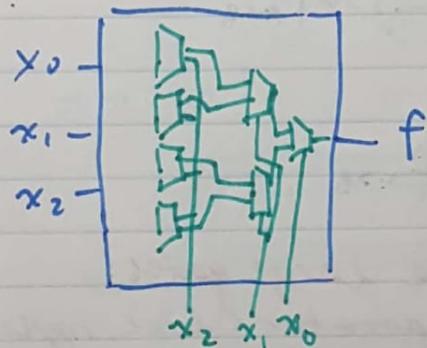
Lookup Table:

→ Built from MUXES.

↳ Generic programmable block used in FPGAs

Example:

1) 3-LUT:



Inputs are programmable
(can implement any
3-input fn)

Comparators:

Feb 2, 2018

Circuit to compare two unsigned numbers.

↳ Equality? $a_i \oplus b_i$

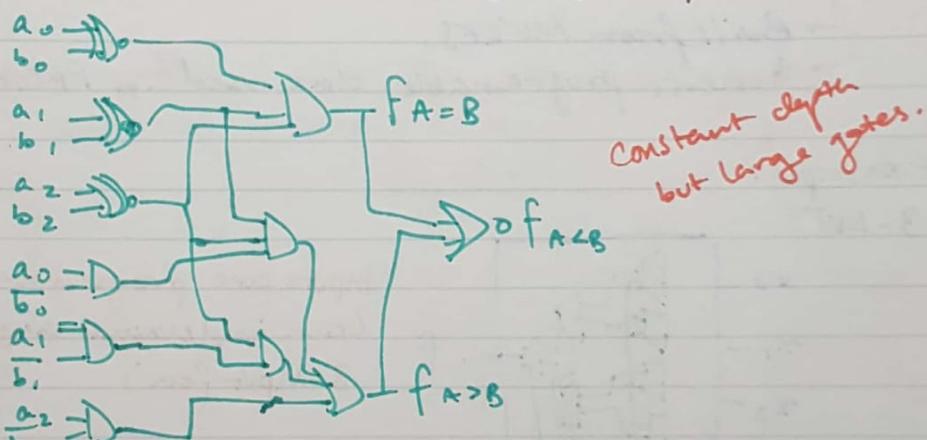
↳ 3 bits:



$\rightarrow A > B ? A < B ?$

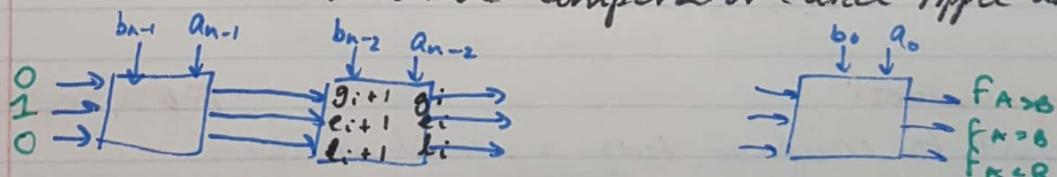
$a_i b_i$	$a_i > b_i$	$a_i \leq b_i$
0 0	0	1
0 1	0	1
1 0	1	0
1 1	0	0

$$f = a_{n-1} \overline{b_{n-1}} + e_{n-1} a_{n-2} \overline{b_{n-2}} + \\ e_{n-2} e_{n-3} a_{n-3} \overline{b_{n-3}} + \dots + \\ e_{n-1} e_{n-2} e_{n-3} \dots e_1 a_0 b_0$$



what if we didn't have large gates?

→ Iterative comparator (slice ripple adder)



For block i :

$$A = B \ L_i := e_{i+1} \cdot (a_i \oplus b_i)$$

$$A > B \ g_i := g_{i+1} + e_{i+1} a_i \overline{b_i}$$

$$A < B \ l_i := l_{i+1} + e_{i+1} \overline{a_i} b_i$$

Decoders:

n inputs, 2^n outputs

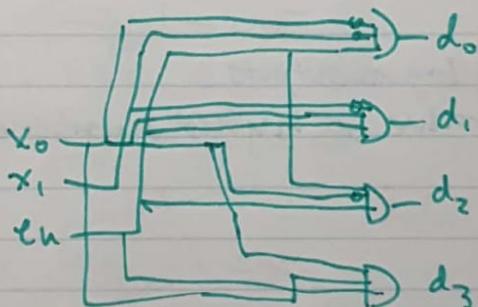
→ recognize input and set 1 output to 1.

→ Enable input: if 0, output all 0's.

Example:

1) 2-to-4 decoder.

x_0	x_1	en	d_0	d_1	d_2	d_3
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
xx	0	0	0	0	0	0

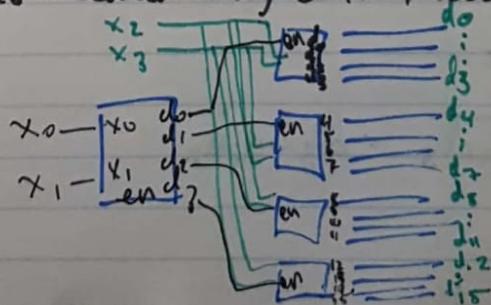


Decoder Trees:

Big decoders from small.

Example:

1) 4-to-16 decoder using 2-to-4 decoders



Basically, enable each decoder as needed, w initial decoder.

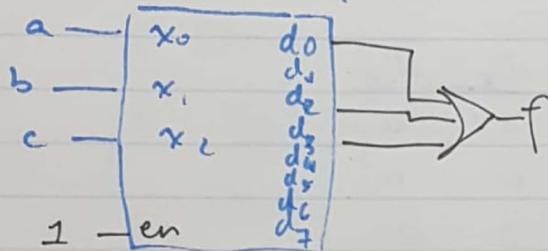
Logic Functions with Decoders:

Feb 5, 2018

B/c decoder outputs represent truth table rows, they can be used to pick minterms.

Example:

1) $F = \sum m(0, 2, 3) = f(a, b, c)$



Priority Encoders:

Opposite to decoder.

↳ Takes n -inputs, log n outputs

↳ Outputs binary value of highest numbered input.

Example:

1) 4 to 2 p. encoder:

$d_3\ d_2\ d_1\ d_0$	$x_0\ x_1$	valid
0 0 0 0	0 0	0
1 0 0 0	0 0	1
x 1 0 0	0 1	1
xx 1 0	1 0	1
xx x 1	1 1	1

$$\text{valid} = \overline{d_3} \overline{d_2} \overline{d_1} \overline{d_0}$$

$$x_0 = d_2 + d_3$$

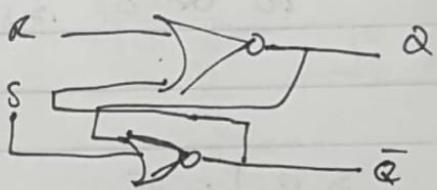
$$x_1 = d_3 + d_1 \overline{d_2} \overline{d_3} = d_3 + d_1 \overline{d_2}$$

Latche:

→ Concept of memory

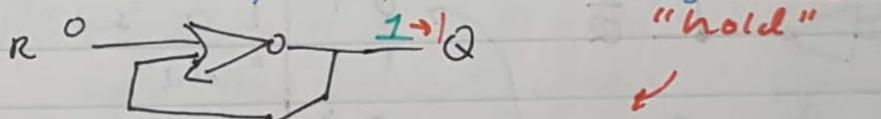
→ SR Latch:

- Can set (1) output, reset (0) output, or "hold".
- Should have 2-complementary outputs.



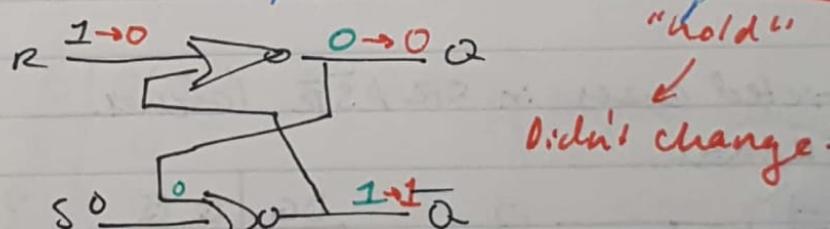
NOR	F
00	1
01	0
10	0
11	0

Case 1) $S=1, R=0$ then change S to 0.



↓
Didn't change.

Case 2) $S=0, R=1$ then change R to 0.



↓
Didn't change.

Case 3) $S=1, R=1$ then change both to 0.

Breaks logic (Q and \bar{Q} 0).

→ both Q and \bar{Q} change to 1, then feedback breaks nor logic and the process repeats (oscillates indefinitely).

→ Or, if time isn't exact, delay goes to case 1/2.

More on SR Latches:

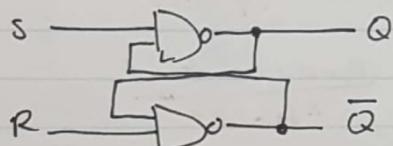
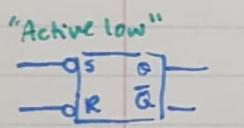
Feb 7, 2018

SR	Q \bar{Q}
1 0	1 0 "set"
0 1	0 1 "reset"
0 0	"hold"
1 1	0 0 "restricted"

$$Q(t+1) = \bar{R}(Q(t)) + S$$

can safely check any change to
10 or 01 instead.

SR Latches w NANDs:



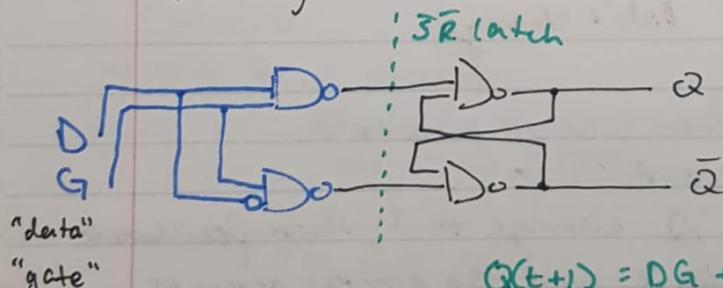
S, R are flipped.

SR	Q \bar{Q}
1 0	0 1 "reset"
0 1	1 0 "set"
1 1	"hold"
0 0	"restricted"

$$Q(t+1) = R(Q(t)) + \bar{S}$$

D-Latch:

Avoiding restricted cases in SR / $\bar{S}\bar{R}$ latches.



DG	Q \bar{Q}
X 0	hold
1 1	1 0 set
0 1	0 1 reset

No glitchy chat

$$Q(t+1) = DG + Q(t)\bar{G} \quad \# \text{ changes when } G \text{ is 1.}$$



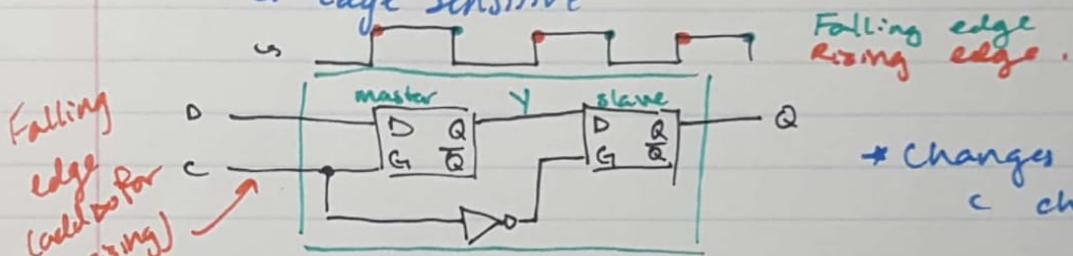
Latches are "level sensitive." Output changes based on values of inputs

- Change can happen when inputs change.
- Value depends on prev. state.

Flip Flops: (DFF)

Output depends on input, but only changes when one input changes from $0 \rightarrow 1$ or $1 \rightarrow 0$

→ "Edge sensitive"

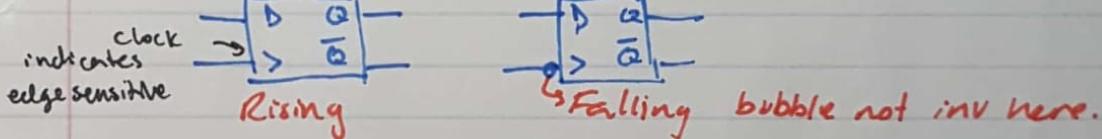


Falling edge

Rising edge

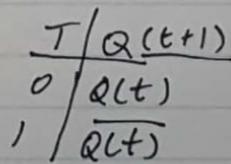
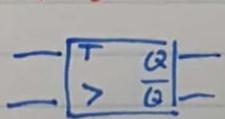
* changes when c changes

c:
 $I \rightarrow 0 \left\{ \begin{array}{l} \text{If } c=1, Y=D, \text{ slave holds, } Q \text{ doesn't change} \\ \text{If } c \text{ changes to a } 0, Q \text{ gets value of } D \\ \text{which was present when } c \text{ changed.} \end{array} \right.$



D	$Q(t+1)$	$D = Q(t+1)$	Implies when c is active (the right edge)
0	0		
1	1		

TFF's :



$$Q(t+1) = Q(t) \oplus T$$

"Toggle"es"