ECE124

Binary Fcns: Jan 3, 2018

Outputs 0 or 1.

1) Truth table

	X_0	X_1	X_2	$f(X_0, X_1, X_2)$
0	0	0	0	1
1	0	0	1	0
:	0	1	0	1
:	0	1	1	1
:	1	0	0	0
:	1	0	1	0
:	1	1	0	0
7	1	1	1	1

Write rows in increasing order

Unique

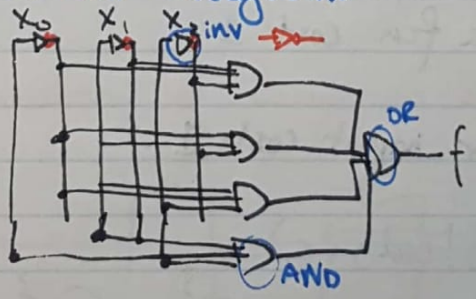
2) Logic Expressions

$f = \overline{X_0}\,\overline{X_1}\,\overline{X_2} + \overline{X_0}X_1\overline{X_2} + \overline{X_0}X_1X_2 + X_0X_1X_2$

Not unique (can manipulate)

↳ Multi = and, add = or, $\overline{\ }$ = inv

3) Circuit Diagram:



Not unique (hard to manipulate)

Boolean Algebra:

precedence ↓ INV, AND, OR

1) Closure under + and ·.
2) Identity for + (0) and · (1)
3) Commutative for both
4) Distributive for:
   a) · over + : $x \cdot (y+z) = x \cdot y + x \cdot z$
   b) + over · : $x + (y \cdot z) = (x+y) \cdot (x+z)$
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 $!(x \cdot y) = !x + !y$

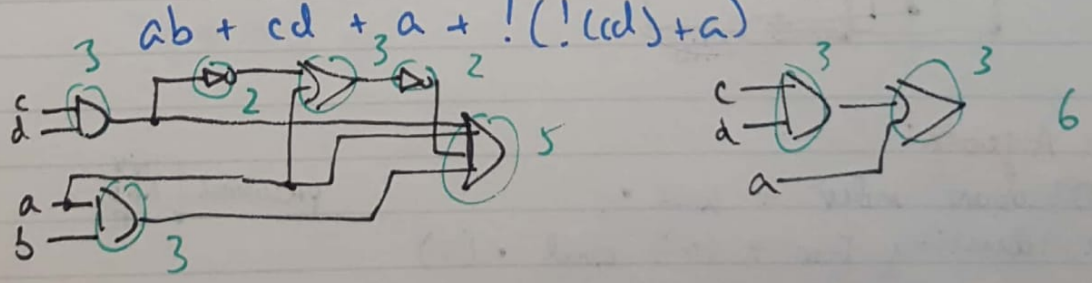
6) $x + x \cdot y = x$ and $x \cdot (x+y) = x$

Example:

1) $f = ab + cd + a + !(!(cd) + a)$
$= a + cd + !(!c + !d + a)$
$= a + cd + cd!a$
$= a + cd$

Cost:

1) Assume a and !a available (inverters have 0 cost).
But, inversions inside a fcn 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 other.

(truth function on)

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

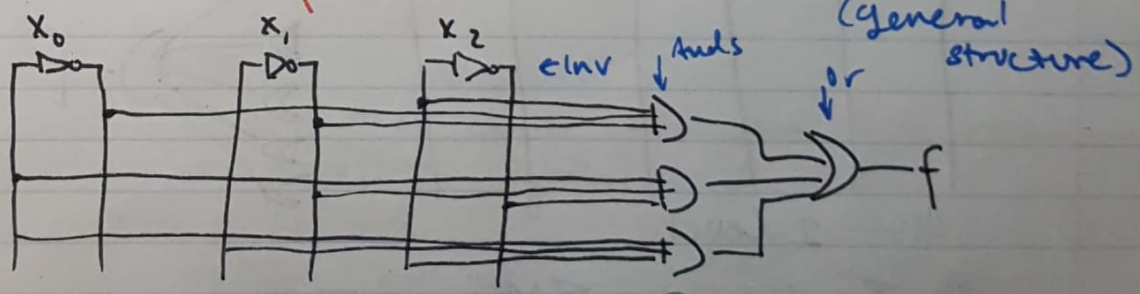
To write logic exp using table, OR the minterms for all $f(\_\_) = 1$.

$f = \bar{x}_0\bar{x}_1 x_2 + x_0\bar{x}_1\bar{x}_2 + x_0 x_1 x_2 = m_1 + m_4 + m_7$

↳ "sum of minterms", "canonical sum of products"

↳ $\Sigma m(1,4,7)$

Implementation of Minterm Sums:



SOP's are 2-level circuits ① ②

Maxterms:

OR for every input — Ø for current row, else 1.

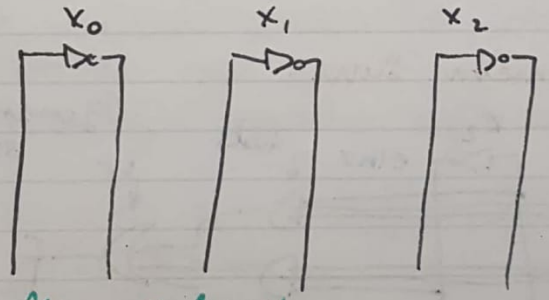
(Turn function off)

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	$\bar{x}_0 + \bar{x}_1 + \bar{x}_2$	$= M_7$

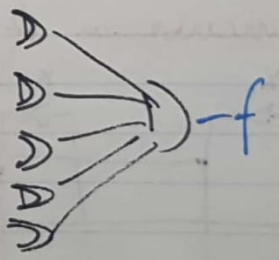
To get logic expression, AND the maxterms for when $f(\_) = 0$.

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

Implementation:



General Case.



Also 2 level.

$f = \Sigma m(1,4,7) = \Sigma M(0,2,3,5,6)$

$\bar{f} = (\overline{m_1 + m_4 + m_7})$

$= (\bar{m}_1 \bar{m}_4 \bar{m}_7) \leftarrow \bar{f}$

$= (m_0 + m_2 + m_3 + m_5 + m_6)$

$= \bar{m}_0 \bar{m}_2 \bar{m}_3 \bar{m}_5 \bar{m}_6$

$= \Sigma M(0,2,3,5,6)$

*min/maxterms are not optimal.

Other Logic Gates: Jan 10, 2018

NAND - AND then NOT

↳  symbol

NOR - OR then NOT

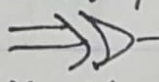
↳  symbol

XOR - exclusive OR

↳ 1 if inputs are dif. (for 2 input)

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

↳ $f = \bar{x}_0 x_1 + x_0 \bar{x}_1 = x_0 \oplus x_1$

↳  symbol

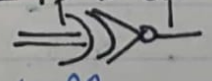
↳

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 fcn (w/o carry)

NXOR - XOR then NOT

↳ 1 if # of inputs which are 1 is even

↳  symbol

BUF - buffer

↳ Does nothing

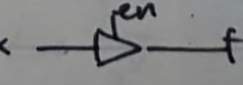
↳ x → f symbol

x	f
0	0
1	1

→ Delays

→ Increase power

TRI STATE BUF - tristate buffer

↳ 

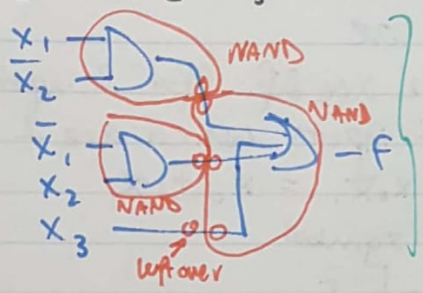
f = x when en = 1, otherwise f is disconnected when en = 0.

Circuit Implementation of NAND:

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

Example:

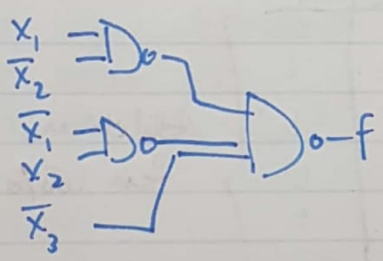
1) $f = x_1\bar{x}_2 + \bar{x}_1x_2 + x_3$



Can do using only NAND

$$f = x_1\bar{x}_2 + \bar{x}_1x_2 + x_3 = !\left[\overline{(x_1\bar{x}_2)} \cdot \overline{(\bar{x}_1x_2)} \cdot \bar{x}_3\right]$$

Becomes:



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

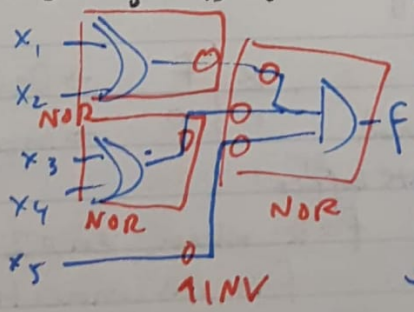
Circuit Implementation of NOR:

Jan 12, 2018

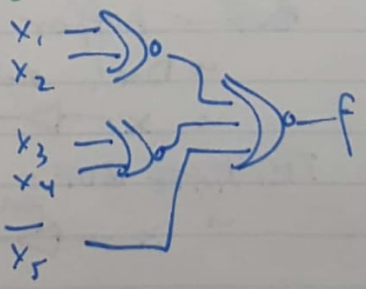
Easy to convert a 2-level POS to an imp. using only NORs

Example:

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



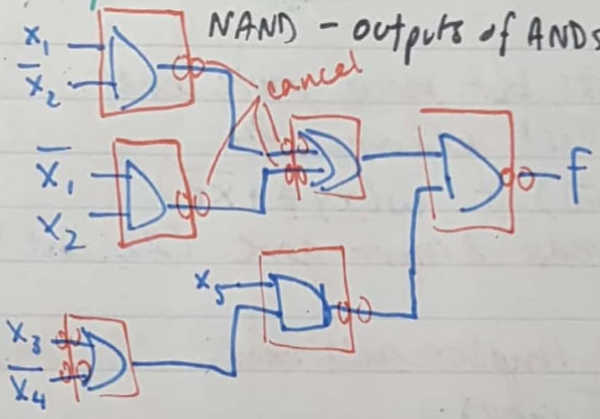
$$f = !\left[\overline{(x_1 + x_2)} + \overline{(x_3 + x_4)} + \bar{x}_5\right]$$



Multi-Leveled Circuits:

Can be done graphically. Structure doesn't change.

Example:

1) 

NAND - outputs of ANDs and inputs of ORs inverted.

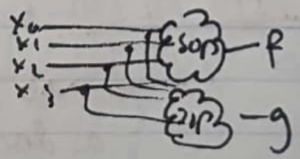
NOR - input of ANDs, outputs of ORs

Can always write 2-level SOP/POS

→ not always practical (very large gates)

→ could have multiple fcns that share a sub expr.

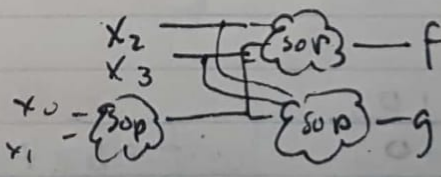
↳ $f(x_0, x_1, x_2, x_3)$ $g(x_0, x_1, x_2, x_3)$



But, if $h = (x_0, x_1)$, maybe

$f = (h, x_2, x_3)$

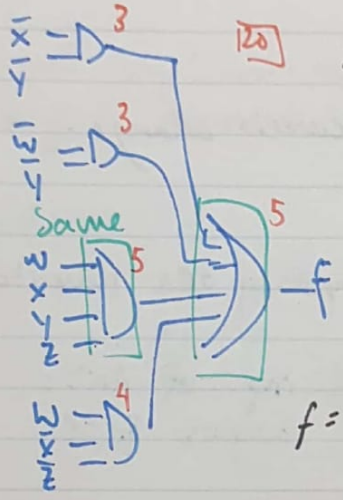
$g = (h, x_2, x_3)$



Could be less expensive.

Example:

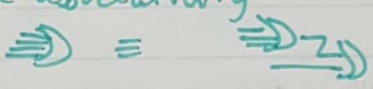
1) $f = \bar{x}\bar{y} + \bar{w}\bar{y} + wxyz + w\bar{x}\bar{z}$ min. SOP.



20

4 input gates don't exist.

Use associativity



Works, but now multi level.
Cost becomes 24

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

↳ max 3 input gate (22 cost)

Finding Optimal 2-Level Implementations: JAN 15, 2018

Karnaugh Maps: ≤5 inputs.

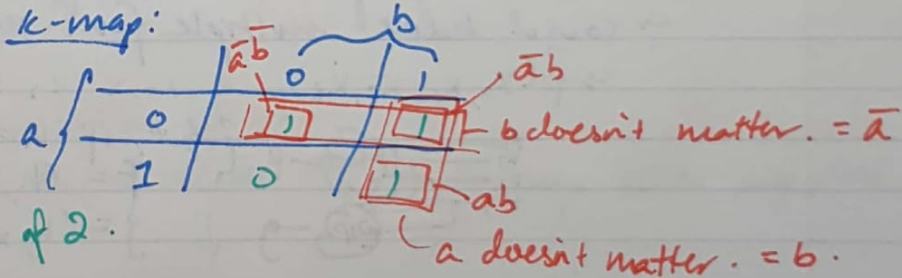
a	b	f
0	0	1
0	1	1
1	0	0
1	1	1

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

$= \bar{a}\bar{b} + \bar{a}b + \bar{a}b + ab$

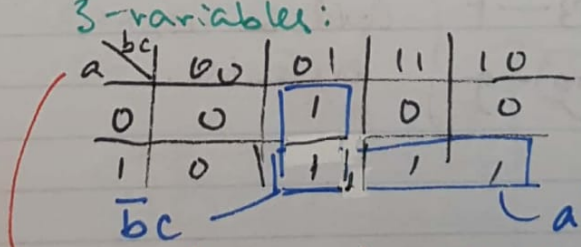
$= \bar{a} + b.$

k-map:



rectangles are always powers of 2.

3-variables:

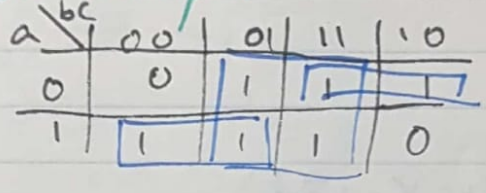


$\therefore f = \bar{b}c + ab$

↳ Ordered b/c adjacent columns only differ by 1 var.

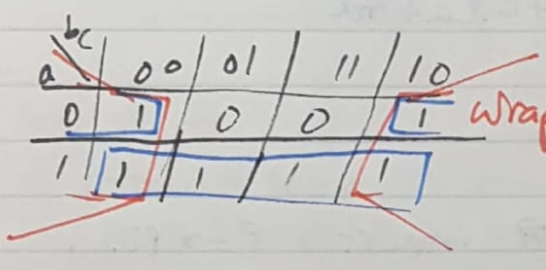
More examples:

1)



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

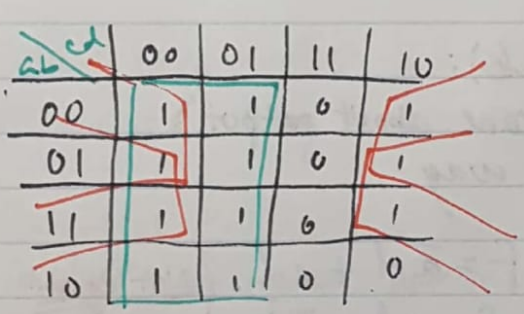
2)



wraps around

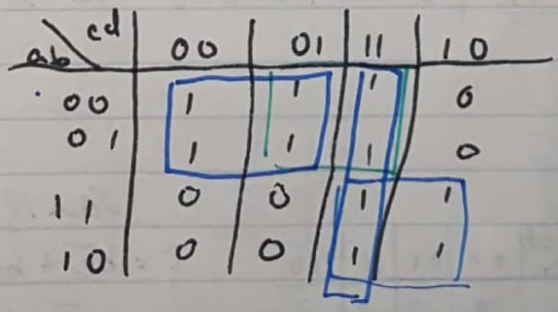
$f = a + \bar{c}$

3)



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

4)



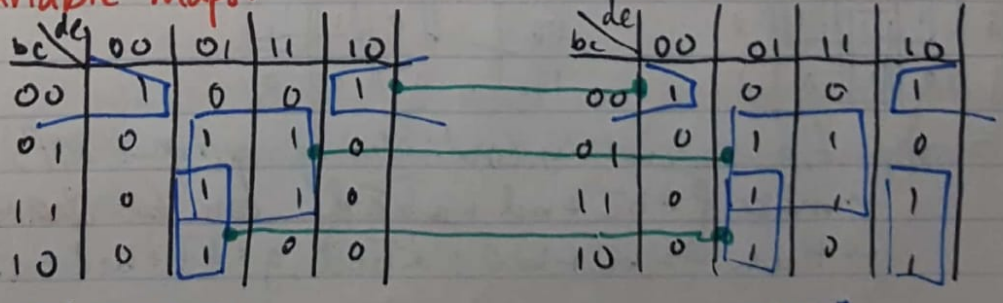
Jan 17, 2018

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

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

∴ More than one min SOP

5 variable Maps:



Join same rectangles

when a=0

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

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

a = 1

$+ ace + ab\bar{d}e + a\bar{b}\bar{c}\bar{e} + bd\bar{e}.$

Minimum POS:

ab \ 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

$a+\bar{c}+d$

$\bar{a}+c$

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

* Another way:
  Find not f and SOP, invert SOP → POS.

Function Don't Cares (Andy Take):

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

How to handle? cheapest way.

ab \ cd	00	01	11	10
00	1	1	1	x
01	0	x	1	0
11	0	0	1	0
10	0	0	1	0

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

Multiple Functions:

Jan 19, 2018

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

ab \ cd	00	01	11	10
00	0	0	0	0
01	1	0	1	1
11	1	0	1	1
10	0	1	0	0

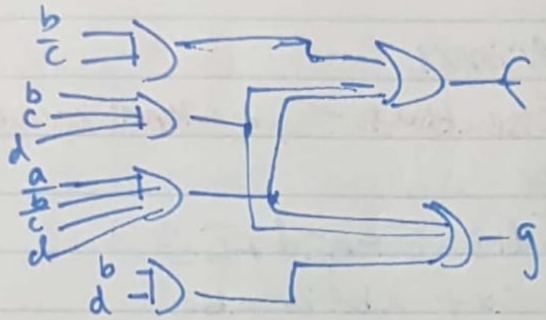
$f = b\bar{c} + bd + a\bar{c}d$

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

To imp f, g seperately (individually optimal SOP) = 29

↳ make $f = b\bar{c} + bd + a\bar{b}\bar{c}d$ can be shared.

↳ then $f = b\bar{c} + bcd + a\bar{b}\bar{c}d$, $g = b\bar{d} + bcd + a\bar{b}\bar{c}d$



cheaper than individual fcns.

Implicants:

A product term is an implicant: { ex: minterm when f=1

↳ term = 1 ⟹ f = 1.

Prime Implicants:

Removing any variable ⟹ no longer implicant

Essential Prime:

Includes minterm not inside another prime.

Example:

1)

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

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

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

$\bar{b}$ *prime

*can't make rect larger.

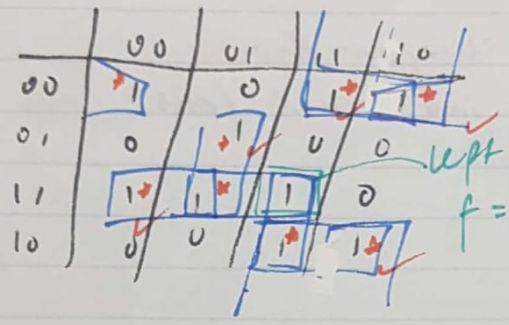
are essential, same $\bar{a}$ $\bar{b}$.

* 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)



Primes
Essentials —, minterms covered *
left over

$f = \bar{a}\bar{b}d + ab\bar{c} + b\bar{c}d + \bar{b}c$
$cs + abd$ or abc
1 of these.

Quine-McCluskey:

More algorithmic for min SOP.

1) Get all primes
2) Get essentials and other needed primes.

Example:

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

# of 1s				
0	0	0000	×	(0,2) 00-0 ✓
1	2	0010	×	(2,3) 001- ×
	3	0011	×	(2,10) -010 ×
	5	0101	×	(3,11) -011 ×
2	10	1010	×	(5,13) -101 ✓
	12	1100	×	(10,11) 101- ×
	11	1011	×	(12,13) 110- ✓
	13	1101	×	(11,15) 1-11 ✓
3	15	1111	×	(13,15) 11-1 ✓
4				

⇒ 2,3,10,11 | -01- ✓

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

Found primes, now find essentials. "covering matrix"

Columns w one ✓ one essential

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

15 left over

need 1 of these pairs

If you have Don't Cares: 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 don't care term, it's optimal to make it a 1.
   ↳ prev example: $\bar{b}c + b\bar{c} + \begin{cases} \bar{a}\bar{c}\bar{d} \\ \bar{a}\bar{b}\bar{d} \end{cases} + \begin{cases} acd \\ abd \end{cases}$

Unsigned Nums: Jan 24, 2018

Unsigned V in base r using n digits:

"positional number notation"

$(d_{n-1} d_{n-2} d_{n-3} \dots d_1 d_0)_r$

d_{n-1}: most significant; d_0: least significant

$V = d_{n-1} \cdot r^{n-1} + \dots + d_0 r^0$

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

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	
$\frac{53}{2}$	26	1	d_0
$\frac{26}{2}$	13	0	d_1
$\frac{13}{2}$	6	1	d_2
$\frac{6}{2}$	3	0	d_3
$\frac{3}{2}$	1	1	d_4
$\frac{1}{2}$	0	1	d_5

stahp!

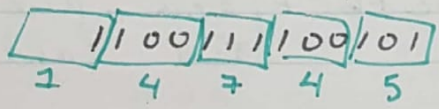
$\therefore 53 = (110101)_2$

2) $(23)_5 \rightarrow$ base-3

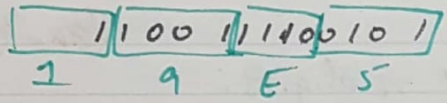
↳ Get $(23)_5$ in base 10 → then base 3.

Powers of 2 Bases:

$(1100111100101)_2 \Rightarrow$ base 8? 16?

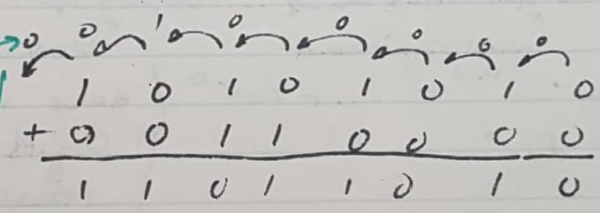


Base 8 - groups of 3 (2^3)



Base 16 - groups of 4 (2^4)

Addition:



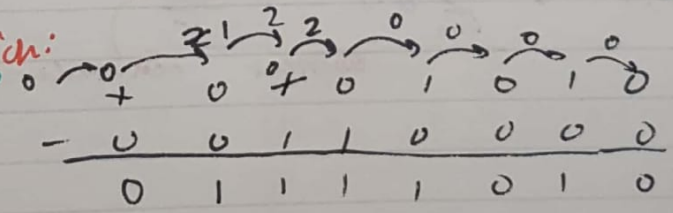
left over?
Add digits (1+0+0)

↖ Would lose data as computer doe.

→ "unsigned overflow"

→ n-bits? $[0..2^n-1]$

Subtraction:



need to borrow?
unsigned underflow

Arithmetic Circuits:

Jan 26, 2018

Half Adder:

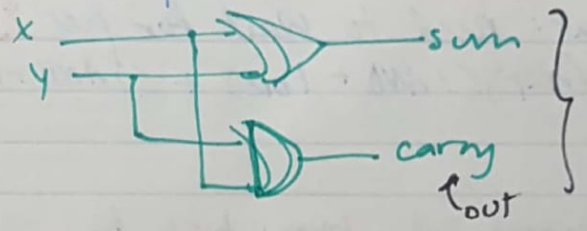
Circuit to add 2 bits x, y?

x	y	f(sum)	carry out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum: xor $x \oplus y$

carry: and xy.

↑ out



Half-adder circuit.

Full Adder:

Circuit to add 3 bits, x + y + carry in

x	y	cin	sum	cout
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

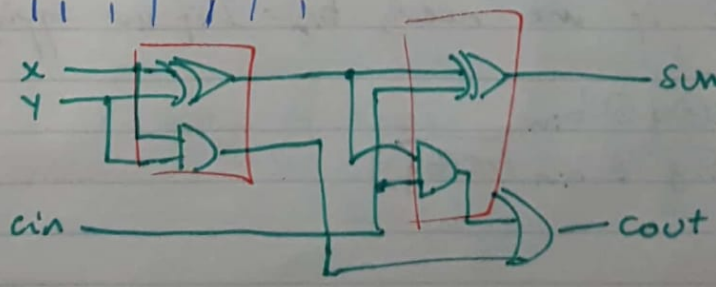
sum: $x \oplus y \oplus cin$

cout: $xy + cin(x \oplus y)$

$= g + cin\,p$

g ↑ generate

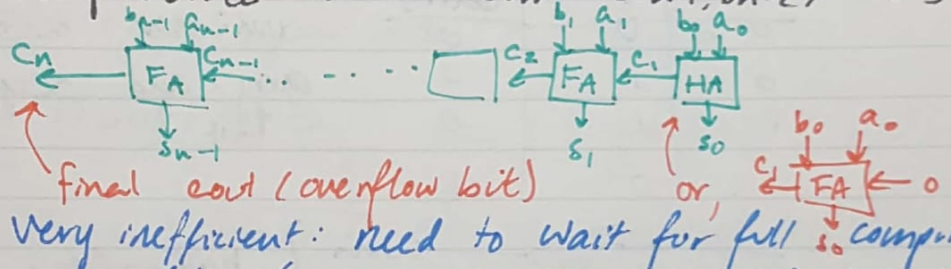
p ↑ propagate



2 half adders and an or gate = full adder

Ripple Adder:

One circuit to add 2 n-bit unsigned nums
$A(a_{n-1}, a_{n-2}, \ldots a_0)$, $B(b_{n-1}, b_{n-2}, \ldots b_0)$
to produce n-bit sum $S(s_{n-1}, s_{n-2}, \ldots s_0)$ and cout



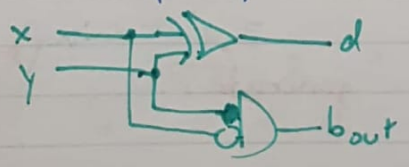
Very inefficient: need to wait for full computation.
delay: $(n-1)(1\,AND + 1\,OR) + (1\,AND + 1\,OR + 1\,XOR)$

Half Subtractors:

$x - y$. difference d, borrow bit b_{out}.

x	y	d	b_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$d = x \oplus y$
$b_{out} = \bar{x}y$

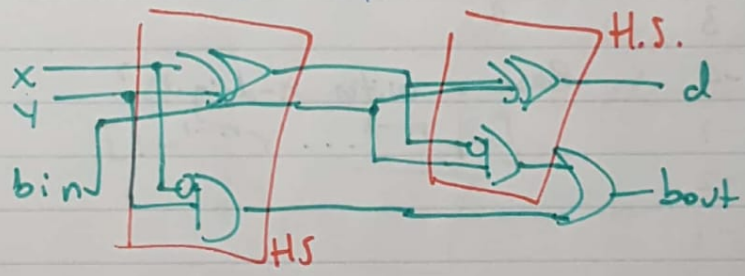


Full Subtraction:

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

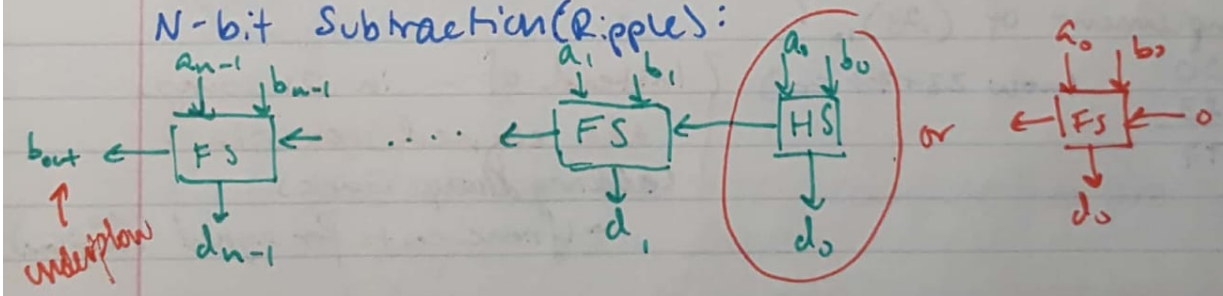
$d = x \oplus y \oplus b_{in}$
$b_{out} = \bar{x}y + b_{in}\overline{(x \oplus y)}$

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



Jan 29, 2018

N-bit Subtraction (Ripple):



Signed Numbers:

Represent signed nts in base r with n digits?

- ✗ Can use sign bit (base) → 2 reps for 0, comp. circuits
- ✓ Radix Compliments: + value V, base r, n-bits:

r's comp of V is:

max+1 $r^n - V$ 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: $(01011000)_2$ → 2^8 – this

-88: $(10101000)_2$

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?:
$[-r^{n-1} \ldots r^{n-1} - 1]$

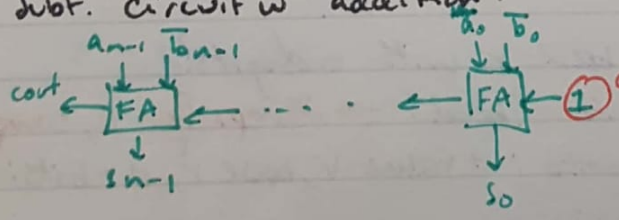
3) 2's complement of $(23)_{10}$?

$$\begin{array}{r} 100 \\ -23 \\ \hline 77 \end{array}$$

(now 23+77 = 0)

Instead of − in 2s comp
add signed version.
(adding always works)
→ ignore carry for signed (overflow)

4) Subt. circuit w addition:



add/sub flag to generalize

instead of invert, xor with control wire

$b_0 \oplus 0 = b_0$

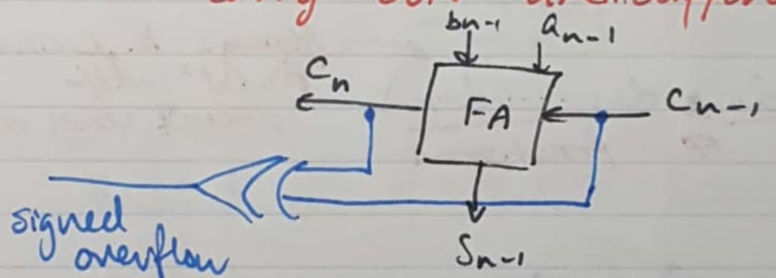
$b_0 \oplus 1 = \bar{b}_0$

Overflow Detection:

Jan 30, 2018

```
             1 1 0
 0    ↙  | 0 | 1 0 0  0 1 1 0       70
ignore + | 0 | 1 0 1  0 0 0 0     + 80
         | 1 | 0 0 1  0 1 1 0      150      Largest + is 127.
              (+)+(+) = (−). ∴ overflow
              1 1
 1    ↙  | 1 | 0 1 1 1  1 0 1 0     −70
ignore − | 1 | 0 1 1 0  0 0 0 0     −80
         | 0 | 1 1 1 0  1 0 1 0    −150      Largest − is −128.
```

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



Faster Adders:

Ripple Adders must wait + recompute for prev. adder.

↳ compute couts faster

Before: $c_1 = g_0 + c_0 p_0$ — computed at time 1. (Time 0 / Time 3)

Finishes @5 → $c_2 = g_1 + c_1 p_1$

↳ Solution? Simplify.

$c_2 = g_1 + p_1(g_0 + c_0 p_0)$ ← same

$= g_1 + g_0 p_1 + p_1 p_0 c_0$ ← Different

1 1 1 1 1 0 ← 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

~ 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 4-CLA adders for 16-bits

Fixed Point Numbers

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

$$D = (\underbrace{d_{n-1} \ldots d_0}_{\text{int}} \underbrace{.}_{\text{FP}} \underbrace{d_{-1} \ldots d_{-k}}_{\text{fractional}})$$

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

Example:

1) $(1321.312)_4 = 1\times4^3 + 3\times4^2 + 2\times4^1 + 1\times4^0 + 3\times4^{-1} + 1\times4^{-2} + 2\times4^{-3}$
$= 121.84375$

2) Go backwards.
↳ Integer is obv (repeated div)
↳ Frac: repeated mult:
$0.625 \times 2 = 1.250$ $(d_{-1} = 1)$
$0.250 \times 2 = 0.500$ $(d_{-2} = 0)$
$0.500 \times 2 = 1.000$ $(d_{-3} = 1)$

When dealing w/ fracs, 2's comp doesn't really work.
↳ Use sign bit instead.

Multiplexers: Jan 31, 2018

Inputs, Outputs, and select line.

↳ n inputs ⇒ log(n) select lines.

↳ Output is one of the inputs based on select line. (can nest)

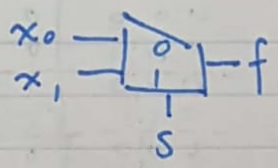
2-to-1 Multiplexer:

if statement x_0, x_1 input

s	f
0	x_0
1	x_1

$f = \bar{s}x_0 + sx_1$

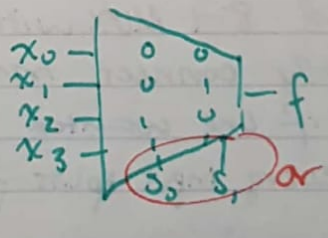
↳ Symbol:



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

$f = \bar{s}_0\bar{s}_1x_0 + \bar{s}_0s_1x_1 + s_0\bar{s}_1x_2 + s_0s_1x_3$

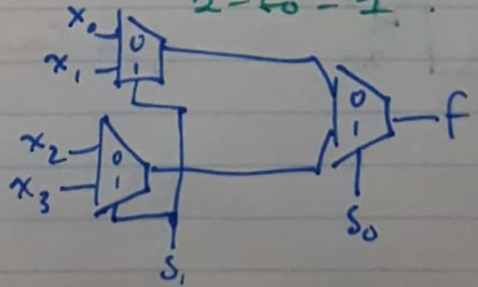


Example:

4-to-1 MUX w 2-to-1 Mux's!

$f = \bar{s}_0(\underbrace{\bar{s}_1x_0 + s_1x_1}_{\text{2-to-1}}) + s_0(\underbrace{\bar{s}_1x_2 + s_1x_3}_{\text{2-to-1}})$

2-to-1



Uses:

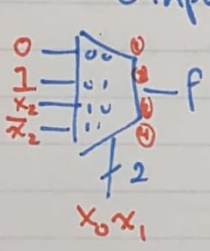
- Steer data and implement logic fcns
  - → N-input fcn can be done w/ a MUX w/ n-1 select lines.
    - → Connect n-1 inputs to select lines, then last input to MUX data.

Example:

1) $f = \bar{x}_0 x_1 + x_0 \bar{x}_1 x_2 + x_0 x_1 \bar{x}_2$ w/ 1 MUX?

3 inputs ⇒ 2 select lines.

1 MUX



x_0	x_1	x_2	f	
0	0	0	0	1
0	0	1	0	
0	1	0	1	2
0	1	1	1	
1	0	0	0	3
1	0	1	1	
1	1	0	1	4
1	1	1	0	

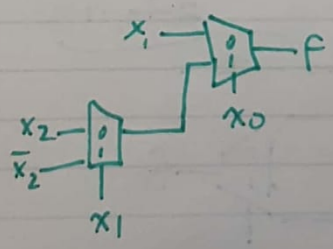
* Note: If wanted 8-1 MUX with 8 select lines:
  - → Basically connect truth table.
  - → What if one wanted smallest MUX over largest?

Shannon F'ting → → every input splits f into 2 halves ($x(\cdots) + \bar{x}(\cdots)$)

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

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

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



Doesn't matter what order of vars you factor.

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

✗ = $\bar{x}_2 (x_0 x_1) + x_2 (x_0 \bar{x}_1 + \ldots)$ whoops $(x_0 \bar{x}_2)$.

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

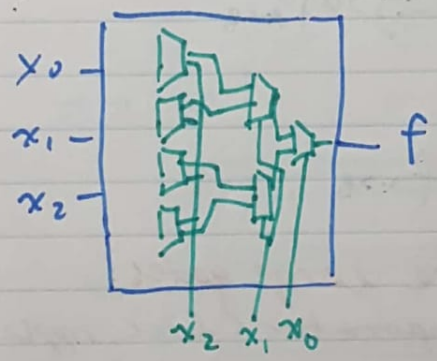
↳ Basically or with $x_2, \bar{x}_2$ in original fcn

Lookup Table:

- Built from MUXES.
- Generic programable block used in FPGAs

Example:

1) 3-LUT:



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

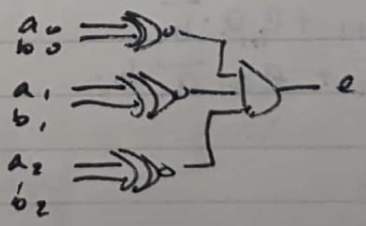
Comparators:

Feb 2, 208

Circuit to compare two unsigned numbers.

↳ Equality? $\overline{a_i \oplus b_i}$

↳ 3 bits:

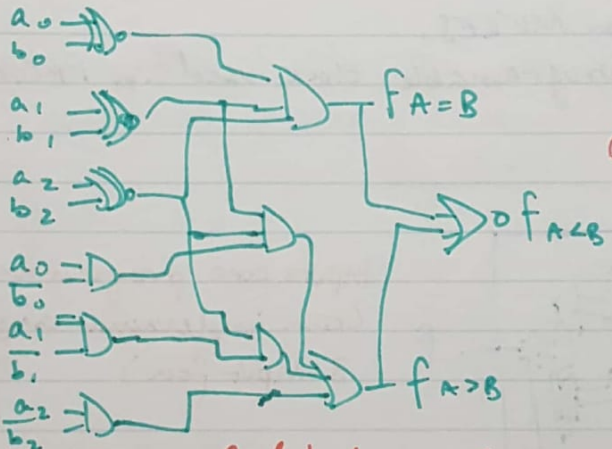


→ A>B? A<B?

a_i	b_i	$a_i > b_i$
0	0	0
0	1	0
1	0	1
1	1	0

$a_i \overline{b_i}$

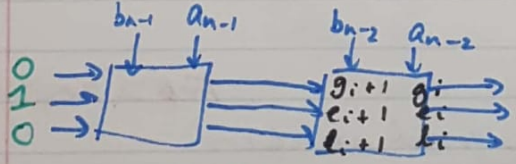
$$f = a_{n-1}\overline{b_{n-1}} + e_{n-1}a_{n-2}\overline{b_{n-2}} + e_{n-1}e_{n-2}a_{n-3}\overline{b_{n-3}} + \dots + e_{n-1}e_{n-2}e_{n-3}\dots e_1 a_0 \overline{b_0}$$

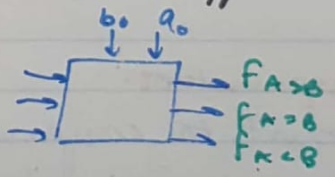


Constant depth but large gates.

What if we didn't have large gates?

→ Iterative comparator (like ripple adder)





For block i:

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

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

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

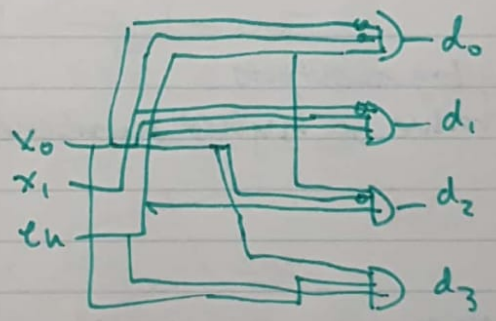
Decoders:

- 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
x	x	0	0	0	0	0

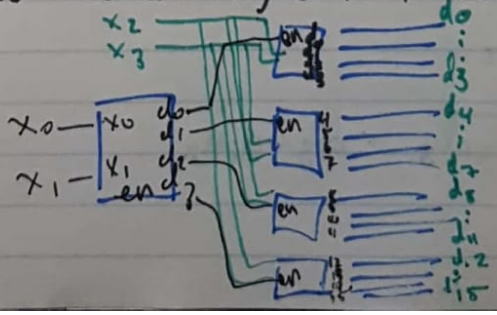


Decoder Trees:

Big decoders from small.

Example:

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



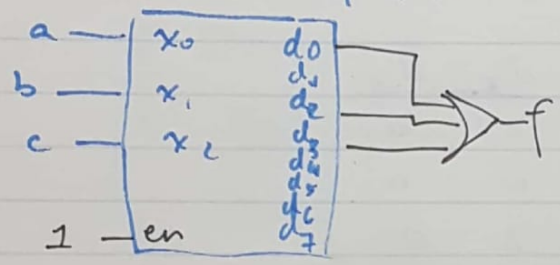
Basically, enable each decoder as needed, w/ initial decoder.

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_0 d_1 d_2 d_3	x_0 x_1	valid
0 0 0 0	0 0 Can put XX	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

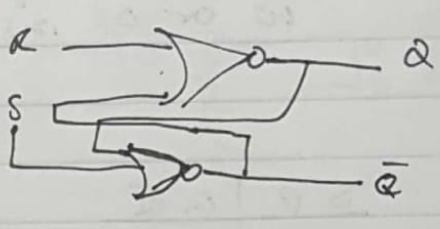
$valid = \bar{d_0}\bar{d_1}\bar{d_2}\bar{d_3}$

$x_0 = d_2 + d_3$

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

Latches:

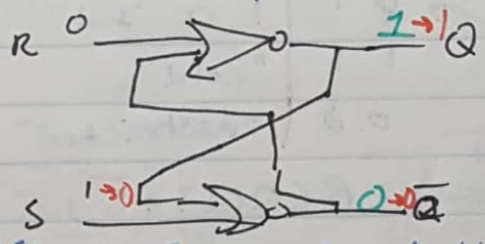
- Concept of memory
- SR Latch:
  - Can set (1) output, reset (0) output, or "hold".
  - Should have 2 complementary outputs.



NOR	f
0 0	1
0 1	0
1 0	0
1 1	0

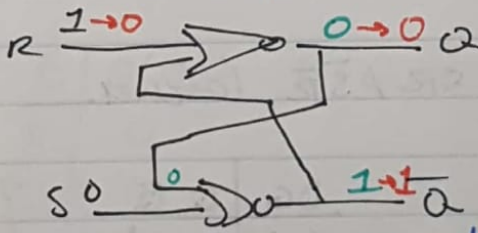
Case 1) S=1, R=0 then change S to 0.
"hold"

↓ Didn't change.



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

↓ Didn't change.



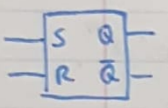
Case 3) S=1, R=1 then change both to 0.

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



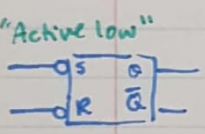
S	R	Q	Q̄	
1	0	1	0	"set"
0	1	0	1	"reset"
0	0			"hold"
1	1	0	0	"restricted"

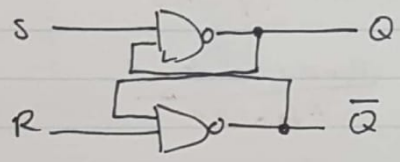
$Q(t+1) = \bar{R}(Q(t)+S)$

can safely check and change to 10 or 01 instead.

SR Latches w/ NANDs:

"Active low"





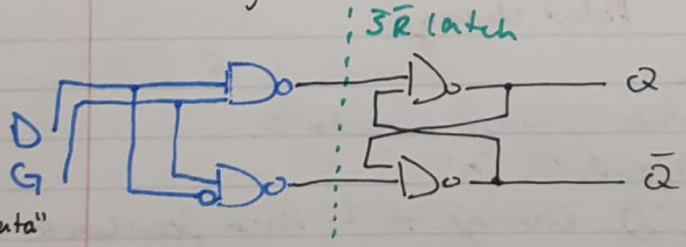
S, R are flipped

S	R	Q	Q̄	
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:

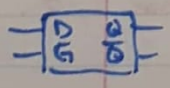
Avoiding restricted cases in SR / $\bar{S}\bar{R}$ latches.



D	G	Q	Q̄	
X	0	hold		
1	1	1	0	set
0	1	0	1	reset

No glitchy chart

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

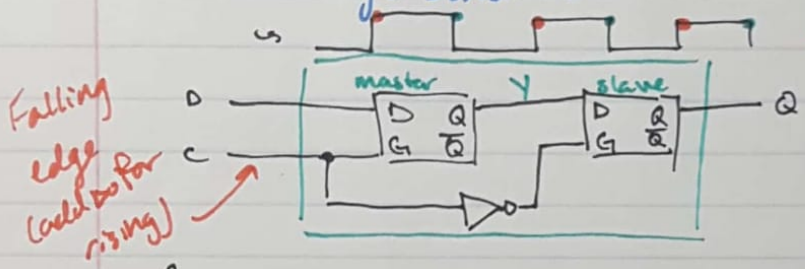


* Latches are "level sensitive". Output changes based on values of inputs
  - Change can happen when inputs change.
  - Value depends on prev. state.

Flip Flops: (DFF)

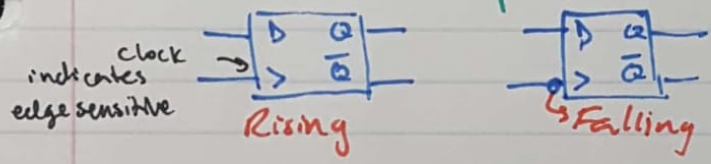
Output depends on input, but only changes when one input changes from 0→1 or 1→0

↳ "Edge sensitive"



* changes when C changes

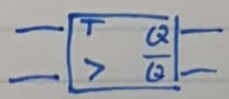
C: 1→0 { If C=1, Y=D, slave holds, Q doesn't change
↳ If C changes to a 0, Q gets value of D which was present when C changed.



D	Q(t+1)
0	0
1	1

D = Q(t+1) } Implies when C is active (the right edge)

TFF's:

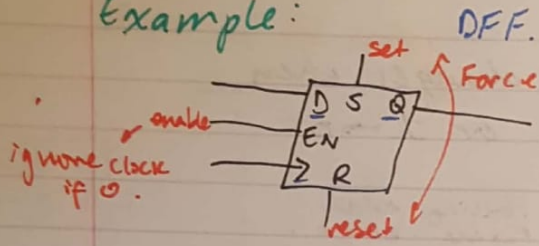


T	Q(t+1)
0	Q(t)
1	$\overline{Q(t)}$

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

"toggles"

Example:



ensures starting value is defined + consistent.

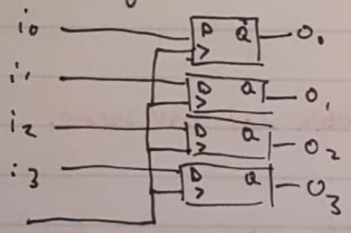
Registers:

Used to hold data for time.

↳ N-bit register: N flipflops. (Usually DFFs w same clk signal)

Example:

1) 4-bit register?



Adding a+b:
one 4 bit register to "load" each a and b, then another 4 bit register to store output.

Other functionality: Shift data, load new data, etc.

2) Design shift register. O_i doesn't change, $O_{i+1} = O_i$, $O_{i+2} = O_{i+1}$ ... etc.
Shift up: O_{i+1} becomes O_i. Shift down: O_i becomes O_{i+1}.

L(oad)	U(p)	D(own)	Action
0	0	0	hold
0	0	1	shift down
0	1	x	shift up
1	x	x	load.

Need to "direct" correct input to registers for desired action.

MIDTERM

Boolean, min, max SOP, POS, Truth table, KMap, QMcclusky, optimization, prime/ess. gates, only NAND/NORs

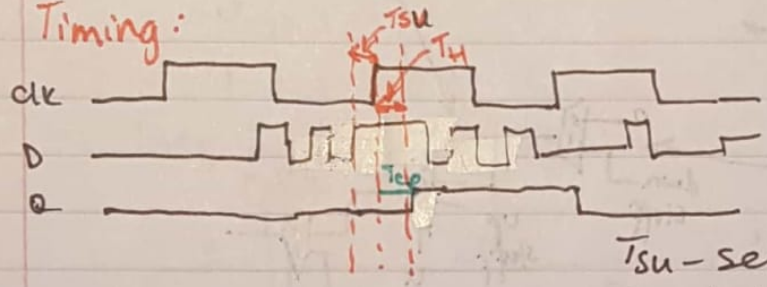
Feb 28, 2018

Logic eqn:

$$Q_j := L_{ij} + \bar{L}(UO_{j-1} + \bar{U}(D_{j+1} + \bar{D}O_j))$$

MUX (outer), MUX (middle), MUX (inner)

Timing:



Change one thing at a time, enough for Q to change.

T_{su} - set up time

T_H - hold time

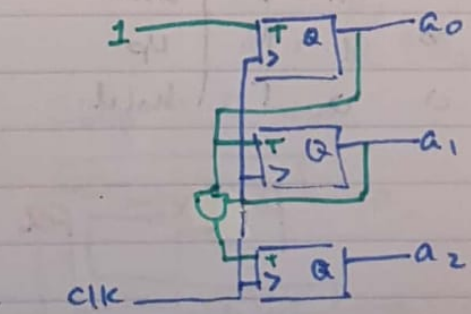
T_{co} - clock → output time

Counters:

- Flip flops to count (synchronous - same clk).

Examples:

1) Count 0 → 7 then repeat (binary).



more bits ⇒ more and gates in this pattern ↳

b/c a_0 always toggles.

now	next		
000	0	0	1
001	0	1	0
010	0	1	1
011	1	0	0
100	1	0	1
101	1	1	0
110	1	1	1
111	0	0	0

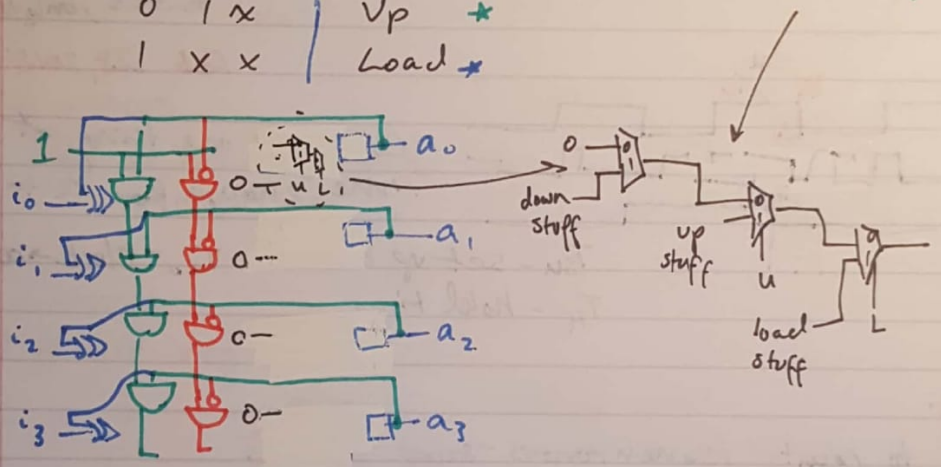
TFF when $a_0 = a_1 = 1$ | TFF when $a_0 = 1$ | TFF

To count down, toggle on 0's.

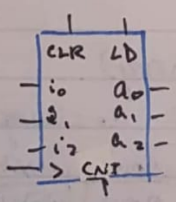
2) Count up/down, pause, load new value.

L	U	D	Action
0	0	0	Pause
0	0	1	Down
0	1	x	Up
1	x	x	Load

$t_j = L(\text{load}) + \bar{L}U(\text{up}) + \bar{L}\bar{U}D(\text{down}) + \bar{L}\bar{U}\bar{D}(\text{pause})$



Symbol:



CLR	LD	CNT	CLK	fcn
0	x	x	x	zero
1	1	x	↑	load
1	0	1	↑	up
1	0	0	↑	hold

Generic Counter Design:

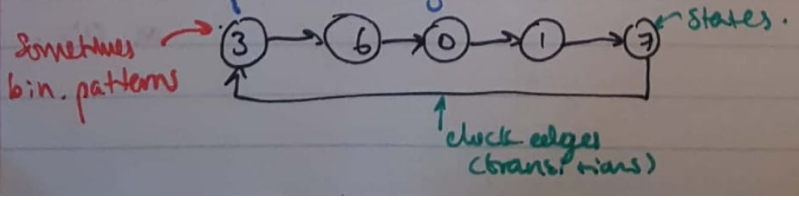
Feb 28, 2018

→ Count any sequence

Example:

1) Count 3, 6, 0, 1, 7, repeat.

Step 1: Diagram (state).



Step 2: State Table.

current	next
0 1 1	→ 1 1 0
1 1 0	→ 0 0 0
0 0 0	→ 0 0 1
0 0 1	→ 1 1 1
1 1 1	→ 0 1 1

"edges"

3 bits ⇒ 3 flip flops needed.

Step 3: Flip Flop Type:

DFF:

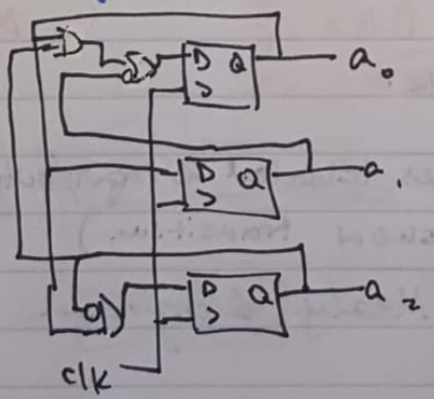
c	n	dFF:
0 1 1	1 1 0	1 1 0
1 1 0	0 0 0	0 0 0
0 0 0	0 0 1	0 0 1
0 0 1	1 1 1	1 1 1
1 1 1	0 1 1	0 1 1

} Simple inputs (just next state)

Get logic eqns for DFF inputs, filling don't cares, K-Map.

$$d_2 = \bar{a}_2 a_0$$
$$d_1 = a_0$$
$$d_0 = \bar{a}_1 + a_2 a_0$$

Step 4: Draw



Can also use TFF:

TFF inputs:

101
110
001
110
100

} when to toggle.

$t_2 = a_1 + a_0$

$t_1 = a_1 \oplus a_0$

$t_0 = \bar{a}_2(\bar{a}_0 + a_1)$

Or JKFF:

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

j_2k_2	j_1k_1	j_0k_0
1 x	x 0	x 1
x 1	x 1	0 x
0 x	0 x	1 x
1 x	1 x	x 0
x 1	x 0	x 0

$j_2 = a_0$ $\quad j_0 = \bar{a}_1$

$k_2 = 1$ $\quad k_0 \; \bar{a}_2 a_1$

$j_1 = a_0$

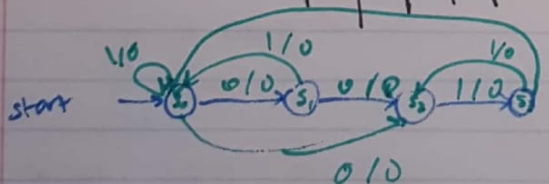
$k_1 = \bar{a}_0$

} Many eqn ... bad
Simple eqn b/c DC's ... good!

Clocked Sequential Circuits:

Mar 2, 2018

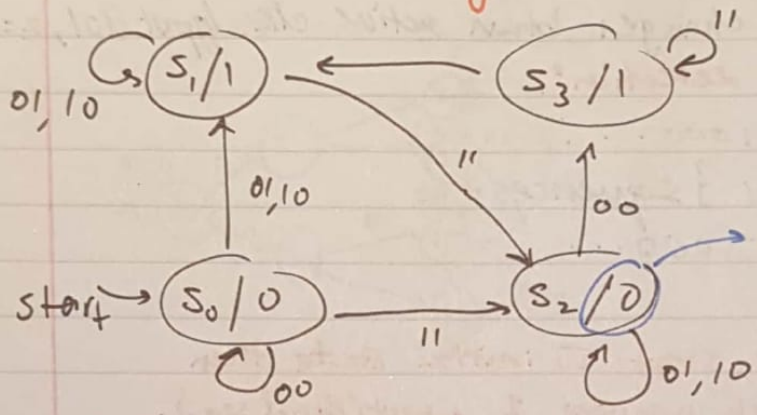
Presence of flip-flops + clock.



edges labeled w: input/output
(to show transition)

"Mealy" diagram.

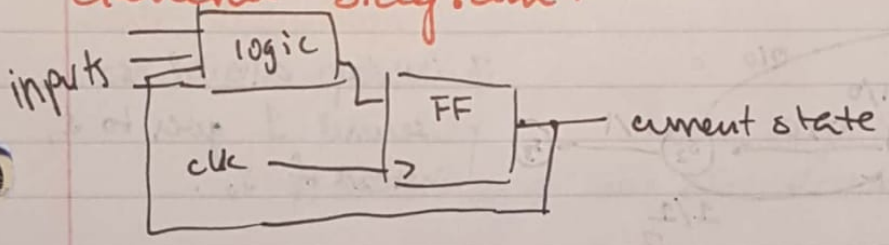
Another State Diagram:



2 inputs, labelled on edges for spec. inputs

1 output labelled in state
↳ depends only on state, not transitions. "Moore".

General Diagram:



Sequential Circuits Analysis:

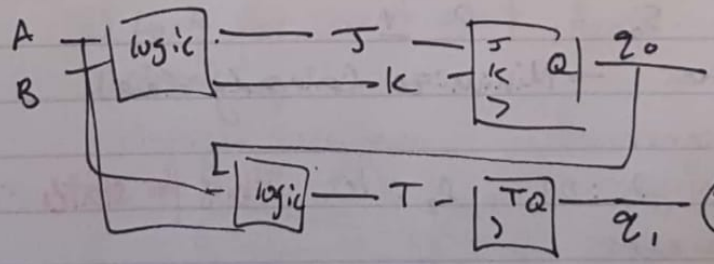
Given circuit make diagram

n flipflops ⇒ 2^n states or less

Step 1: Output eqn $z = q_2$

Step 2: Logic eqns for flipflop inputs.

$j_0 = AB$ $\quad k_0 = \overline{A}\,\overline{B}$ $\quad t_1 = A \oplus B \oplus q_0 \oplus q_1$



Determine transitions:

↳ Get t_1 in all a,b cases. Repeat j,K. Consider all current state

Example: Mar 5, 2018

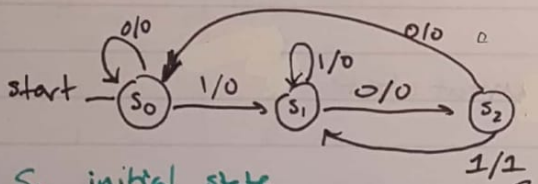
1) Input x, output z. x changes btwn active clk. Input 101, z=1. else 0.
↳ "sequence detection".

x → 0011010101000...
└──┘ } sequences.

z → 0 0000101000...

1) State Diagram

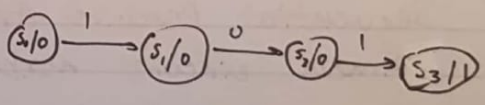
* For seq. detection, start w initial state then add states which require 1 output (found seq).



Overlap allowed, so second 1 goes to S_1 instead of S_0.

S_0 initial state
S_1 seen ....1 / seq start
S_2 seen ...10
S_3 seen ...101.

Extra state moore version:



2) Table

current	Next 0	Next 1	Output 0	Output 1
S_0	S_0	S_1	0	0
S_1	S_2	S_1	0	0
S_2	S_0	S_1	0	1

3) State reduction → Minimize (already done)

4) State Assignment

* If DC S_0 : 00 S_1 : 01 S_2 : 10 #bits for state = # flip flops.

5) FF Type + eqns

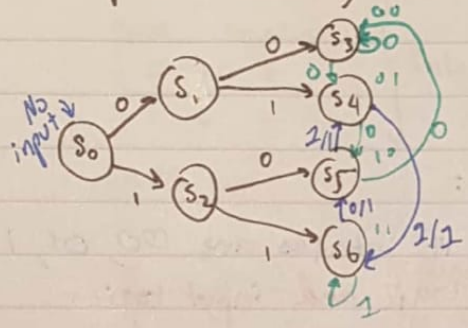
If DFF: $d_1 = \bar{x} q_0$, $d_0 = x$

If TFF: $t_1 = q_1 + \bar{x} q_0$, $t_0 = x \oplus q_0$

6) Output Logic: $z = x q_1 \bar{q}_0 = x q_1$

Mar 7, 2018

2. Mealy diagram w/ input x output z. z 1 when 2 of last 3 inputs are 1, else 0.



State Reduction:

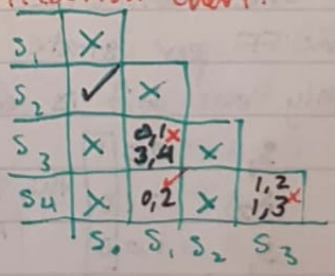
* For equivalence:
  1) For all input, same output.
  2) For all input, same edge destination/equiv destination.

Example:

1)

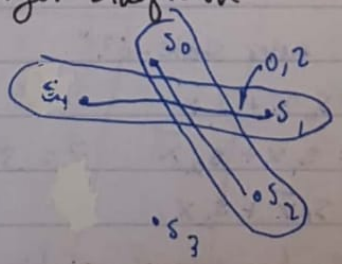
cur	next x=0	next x=1	Output x=0	Output x=1
S_0	S_3	S_2	1	1
S_1	S_0	S_4	0	0
S_2	S_3	S_0	1	1
S_3	S_1	S_3	0	0
S_4	S_2	S_1	0	0

Implication chart:



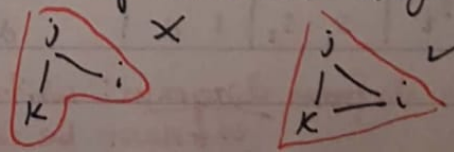
∴ 0, 2, and 1, 4 equivalent.

1) Cross out definite inequivalences based on output.
2) Look for next state equiv.
3) check equiv.
4) Merger Diagram



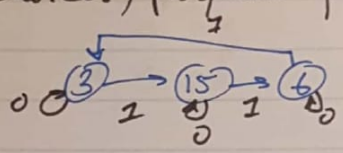
Only one possible 4-state soln since $S_4 - S_1$ depends on $S_0 - S_2$

Group COMPLETE subgraphs



State Assignment: Mar 12, 2018

Diff. choices, pros/cons for each.



Method 1: Minimize flip flops:

→ In this ex, min is 2 if states are 00, 01, 1x

+ Low FF - output logic and input logic

Method 2: Make states based on output.

→ Bin. values for 3, 15, 6.

+ Output logic - Input logic (maybe...), # of FFs, bits, uniqueness

→ If uniqueness is an issue, add extra bits, only output the "output" bit.

Method 3: One-Hot Encoding ;)

→ One FF per state. (n states ⇒ n bits).

→ Only one bit is ever 1 (like a decoder)

S_3	0 0	1
S_2	0 1	0
S_1	1 0	0

this FF output is S_3, etc.

+ Logic - #FF's

↳ IF DFFs, everything is easy.

Example:

	Cur	Next 0 / 1	Output 0 / 1
001	S_0	S_2 S_0	1 1
010	S_1	S_0 S_1	0 0
100	S_2	S_1 S_2	1 0

"when do I enter state-i?"

$d_0 = S_0 X + S_1 \bar{X}$

$d_1 = S_1 X + S_2 \bar{X}$ $\quad Z = S_0 + S_2 \bar{X}$

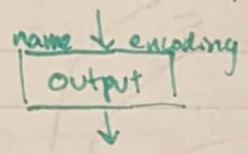
$d_2 = S_0 \bar{X} + S_2 X$

→ To do from diagram: write d_i: by arrows pointing into S_i.
must be complete. watch for implicit loops.

Algorithmic State Machines (ASM):

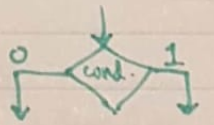
Mar 13, 2018

→ Flow chart (alternate state diagram)



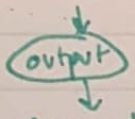
} Only show outputs if not "default" (ex: only 1). Like bubbles.

State Box



} contains variable / expression. Like edges.

Decision Box

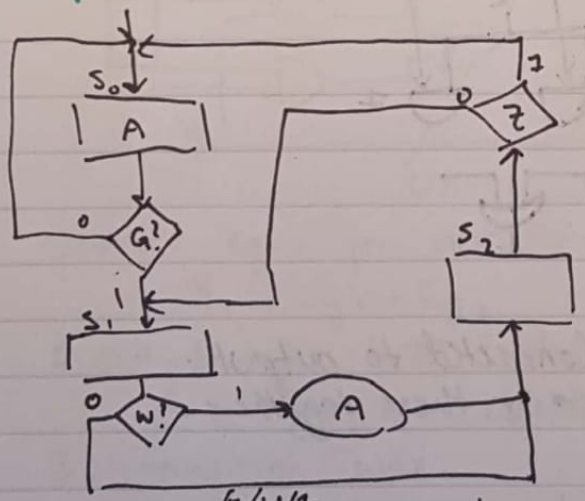


} Mealy output.

Conditional Output Box

Example:

1)



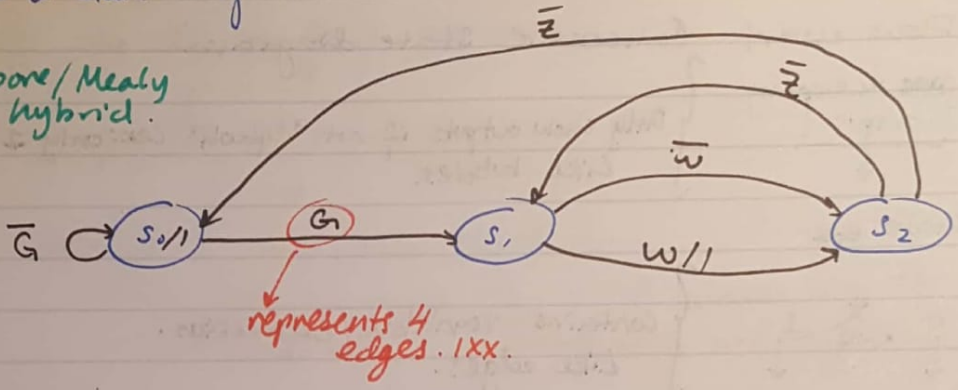
3 states (boxes)
3 inputs (◇) G, W, Z
1 output A.
↳ A = 1 when S_0
↳ A = 1 when S_1 and W = 1.
↳ A = 0 everywhere else.

Design process is the same.

cur	inputs G/W/Z	next	output (A)
S_0	0 x x	S_0	1
S_0	1 x x	S_1	1
S_1	x 0 x	S_2	0
S_1	x 1 x	S_2	1
S_2	x x 0	S_1	0
S_2	x x 1	S_0	0

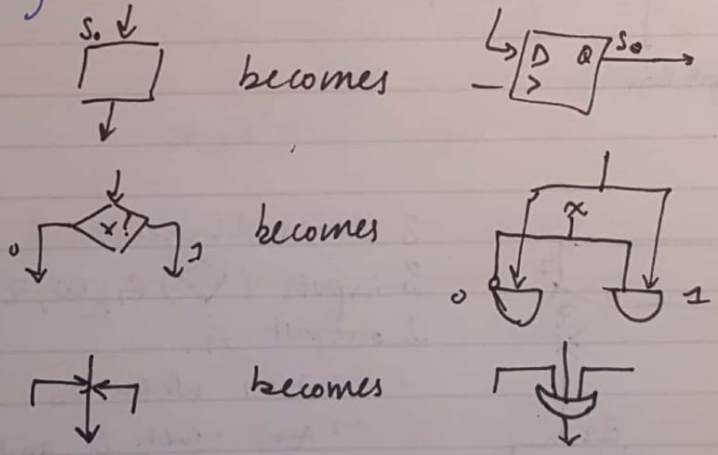
As a state diagram:

Moore/Mealy hybrid.



Converting Directly to Circuits:

Using One-Hot + DFF.

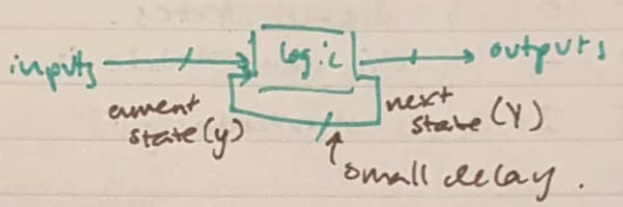


Cond. Boxes becomes Connected to output.

↳ or these together.

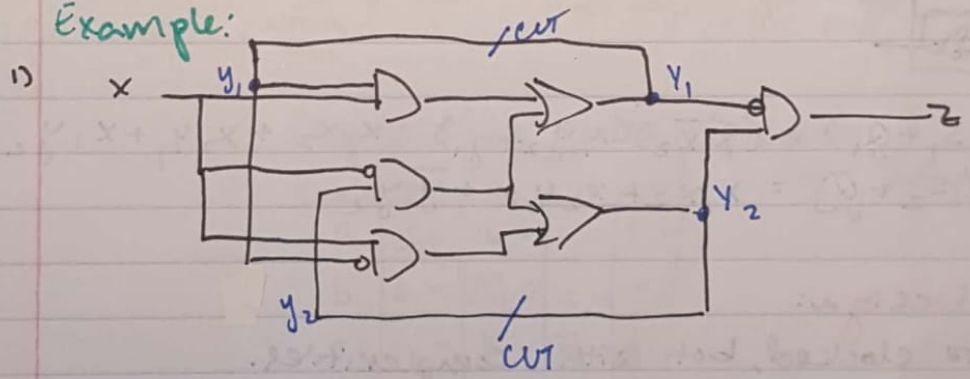
Asynchronous Circuits: Mar 14, 2018

- Concept of state w/o FF or clock
  - → Not combinational or clock. (Latches)



- Analysis:
  - → Similar to clock – derive binary transition/state/flow tables. symbols
  - → Stable states should be identified.
    - → Y and y equal and unchanging.

Example:



1) Write eqns for next states.

$$Y_1 = xy_1 + \bar{x}y_2 \qquad Y_2 = x\bar{Y}_1 + \bar{x}y_2$$

2) Output eqns (use little letters) – in terms of current state.

$$z = \bar{y}_1 y_2 \quad \text{(little y)}.$$

3) Transition Table:

steady states.

cur (Y_2Y_1)	next (Y_2Y_1) $x=0$	next (Y_2Y_1) $x=1$	output (z) $x=0$	output (z) $x=1$
0 0	[0 0]	1 0	0	0
0 1	0 0	[0 1]	0	0
1 1	[1 1]	0 1	0	0
1 0	1 1	[1 0]	1	1

4) Get flow table

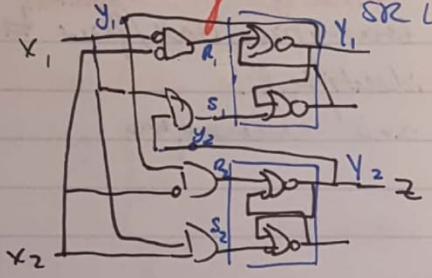
ew	next x=0/1		output x=0/1	
a	a	d	0	-
b	a	b	-	0
c	c	b	0	-
d	c	d	-	1

"undo" state assignment

Only output during stable states. B/c non stable is momentary.

Latch Analysis:

Mar 16, 2018



SR LATCHES. Sorta look like FFs (clock circuit).

use latch eqns

→ $Q = \overline{(S+Q)+R} = \bar{R}(S+Q)$

→ Find R, S eqns, sub here ↑.

$$Y_1 = \bar{R}_1(S_1 + y_1) = \overline{(\bar{x}_1\bar{x}_2)}(x_1 y_2 + y_1) = x_1x_2 + x_2y_1 + x_1y_2$$

$$Y_2 = \bar{R}_2(S_2 + y_2) = x_1x_2 + x_2y_2 + \bar{y}_1y_2$$

Asynchronous Design:

- Similar to clocked, but with complexities.
- Assumptions: (Fundamental Mode)

"primitive" flow table
1) one input change at a time
2) Circuit reaches stable state before an input changes
→ one S.S. per row. (one box per row).

steps:
1) Diagram w/ FM assumption
2) Flow Table
3) State reduction
4) State assignment RACES
5) Next state eqns HAZARDS
6) Output eqns.

Mar 19, 2018

Example:

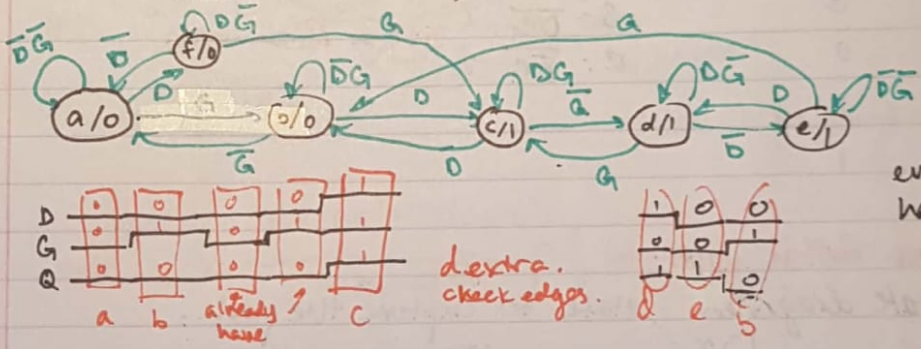
1) Inputs D, G. Output Q. G=0, Q holds. G=1, Q=D.

Step 1: Diagram (FMA)

↳ Find stable states. (initial)

↳ Change inputs one by one.

Take D=0, G=0, Q=0 as initial S.S.



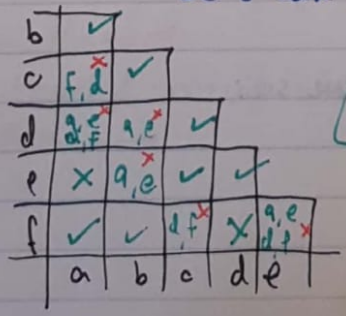
Step 2: Flow Table (primitive)

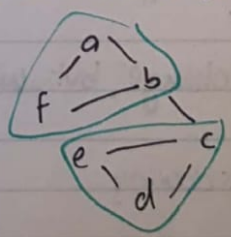
No output for unstable states.

CAN	next 00	01	11	10
a	[a],0	b,-	-,-	f,-
b	a,-	[b],0	c,-	-,-
c	-,-	b,-	[c],1	d,-
d	e,-	-,-	c,-	[d],1
e	[e],1	b,-	-,-	d,-
f	a,-	-,-	c,-	[f],0

Step 3: State Reduction

↳ DC's can become next states.





y	00	01	11	10
a	[a],0	[a],0	c,-	[a],0
c	[c],1	a,-	[c],1	[c],1

Y

$Y = DG + \bar{G}y$

output $Q = y$

Design using Latches:

→ Same steps up to state transition table.
  → Can use latches to hold next state vars.
  → Pick latch type then determine eqns.

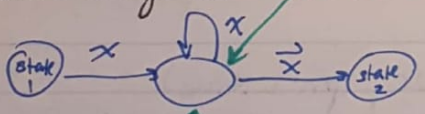
cur state		next state
S R	Q(t)	→ Q(t+1)
1 X	0	0
1 0	1	0
0 1	0	1
X 1	1	1

Get eqns for S and R from table.

$S = \overline{D}G$
$R = D\overline{G}$
} Just a $\overline{D}$-Latch.

Aside: Push Button Inputs:

In a state diagram



state to "capture" the push.
↳ Move on release

↑ no other input edges (FMA).

Hazards:

- Momentary switching transient at output
  → Due to unequal propegation delay

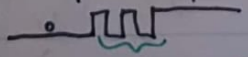
Static Hazards:

→ Static-0 and Static-1

↑ shouldn't have happened (b/c delay)

Dynamic Hazards:

→ Output should change, but we see:



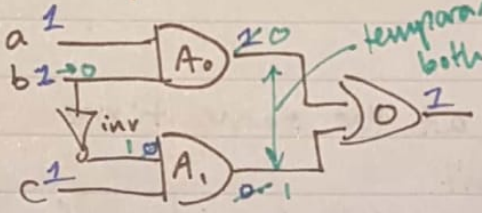
starts flickering

Concerns w Hazards:

- Clocked circuit OK, only need edge stability
- Async will change state.

Illustration of Static Hazards:

(a)

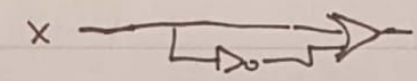


Each gate has delay 1.

Static 1 hazard

Static-1 Hazards (Basic):

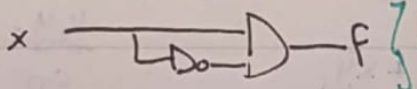
SOP



} logically 1, but can be 0 momentar

Static-0 Hazards (

POS



} logically 0, but can be 1 momentorily

Eliminating S-1 Hazards:

Same circuit in (a), $f = ab + \bar{b}c$

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

Hazard due to b "switching boxes"

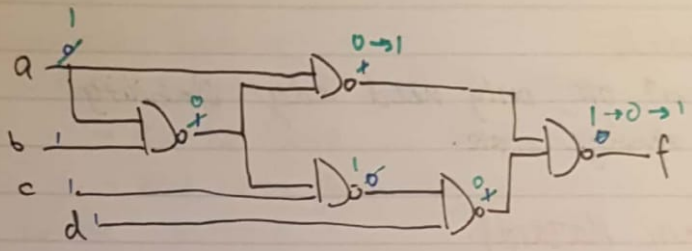
Fix: $f = ab + \bar{b}c + ac$

Adds gate independent of the variable changing
↳ encloses both boxes.

Dynamic Hazards:

Mar 23, 2018

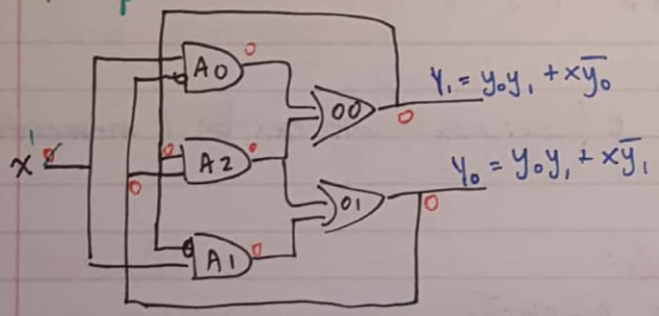
- Multi-Leveled circuits when ≥ 3 paths from an input to output
- Illustration ⟶



Races:

- ≥2 states need to change at the same time.
- Delay might mean they can't.
  → "Critical" if final stable state depends on order of change.

Example:



$y_1 y_0$	$Y_1 Y_0$ x=0	x=1
00	00	11
01	00	01
11	11	11
10	00	10

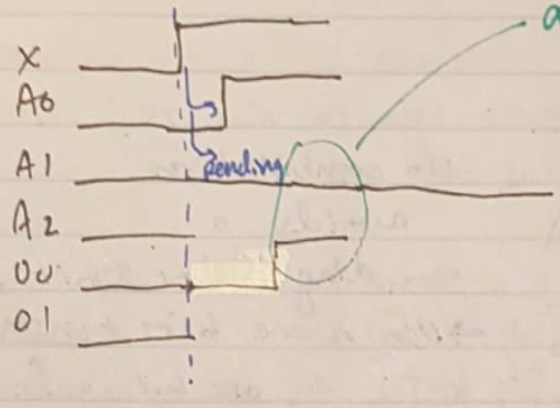
Timing Diagram:

delays ↓

x	0	1	1	1	1	1
A0	0	0	1	1	1	1
A1	0	0	1	1	1	1
A2	0	0	0	0	1	1
C0	0	0	0	1	1	1
01	0	0	0	1	1	1

Not critical.

But what if A0, 00 are very fast?



a1 doesn't change, 00 changed and then it stayed at 0.
→ stuck in wrong S.S.

Non critical Race:

$y_1 y_0$	$Y_1 Y_0$ x=0	x=1
00	[00] → 11	
01	11	[01]
11	[11]	01
10	10	11

Races?
2 vars need to change.
if same time:
$00 \rightarrow 11 \rightarrow \underline{01}$
else: $00 \rightarrow 10 \rightarrow 11 \rightarrow \underline{01}$
$00 \rightarrow \underline{01}$
} Non Critical.

More on Races:

Mar 26, 2018

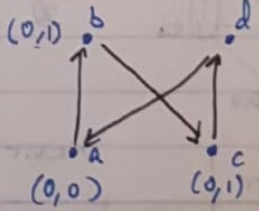
- exist b/c of State Assignment.

Avoiding Races: Transition Diagram

cur	next x=0	x=1
a	[a]	b
b	c	[b]
c	[c]	d
d	a	[d]

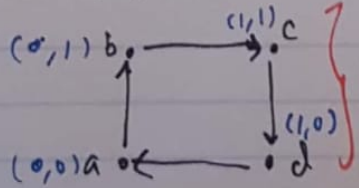
No races yet (no S. Assnt).

If we did 00, 01, 10, 11 . . .



→ : one SS to next.
Diagonals mean race.

Consider:

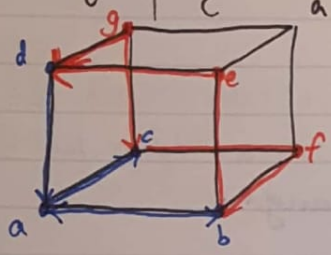


No diags ⇒ no races if coords are states.

However, not always possible:

cur	next 00	01	11	10
a	a	a	c	b
b	a	b	d	b
c	c	b	c	d
d	c	a	d	d

No combination avoids a diagonal w 2 bits, 4 states.
⇒ Use more bits + unstable states.
one bit causes next to change



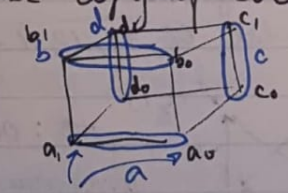
→ New table:

000	a	a	a	c	b
100	b	a	b	e	b
010	c	c	f	c	g
001	d	g	a	d	d
101	e	–	–	d	–
110	f	–	b	–	–
011	g	c	–	–	d

can get blocked, but will always work.
(for 4-state solutions)

Why does it always work?

Make copy of every state.



every state adj to every other.
2 dupes of each state.

equiv.

cur	next			
a_0	a_0	a_0	c_0	b_0
a_1	a_1	a_1	a_0	b_1
b_0	a_0	b_0	b_1	b_0
b_1	a_1	b_1	d_1	b_1
c_0	c_0	c_1	c_0	d_0
c_1	c_1	b_0	c_1	d_1
d_0	c_0	a_1	d_0	d_0
d_1	c_1	d_0	d_1	d_1

Mar 28, 2018

Avoiding Races w One Hot:

↪ 2 bits will always change.
↪ switch the one bit at first.

	cur	next			
0001	a	[a]	[a]	e ~~x~~	~~x~~ f
0010	b	f ~~x~~	[b]	g ~~x~~	[b]
0100	c	[c]	h ~~x~~	[c]	~~x~~ i
1000	d	i ~~x~~	j ~~x~~	[d]	[d]
0101	e	–	–	c	–
0011	f	a	–	–	b
1010	g	–	–	d	–
0110	h	–	b	–	–
1100	i	–	–	–	d
1001	j	–	a	–	–

– # of states
+ simplicity