25-08-2020

1.4.2

Decimal numbers

$1396 = 10^0 \times 6 + 10^1 \times 9 + 10^2 \times 3 + 10^3 \times 1$

Binary numbers

$\underbrace{1}_{\text{Most significant bit}}011\underbrace{0}_{\text{Least significant bit}} = 0 \cdot 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 = 22$

Decimal to Binary

$263 = 2 \times 131 + 1$

$131 = 2 \times 65 + 1$

$65 = 2 \times 32 + 1$

$32 = 2 \times 16 + 0$

$16 = 2 \times 8 + 0$

$8 = 4 \times 2 + 0$

$4 = 2 \times 2 + 0$

$2 = 2 \times 1 + 0$

$1 = 2 \times 0 + 1$

$263_{10} = \boxed{100000111}_2$

Binary to decimal

$\boxed{101101}_2 =$

```
 1 × 1
 2 × 0
 4 × 1
 8 × 1
16 × 0
32 × 1
------
 1
 4
 8
32
------
45
```

26-08-2020

Signed binary numbers

2 ways

good ·· → 2's compliment

Bad ·· → Sign/magnitude numbers

System	range
unsigned	$[0, 2^n - 1]$
Sign/mag.	$[-2^{N-1}+1, 2^{N-1}-1]$
2's comp	$[-2^{n-1}, 2^{n-1}-1]$

Binary addition

```
  1   ← memory
  77
+ 16
----
  93
```

```
  1 1
 010110
 001100
-------
 100010
```

```
  1 1
  101
  011
 ----
 1000
```

careful overflow ↳ $\boxed{1}$

2's compliment

$7_{10} = 0111_2$

to get minus 7 invert all digits and add +1

$0111 \rightarrow 1000 \rightarrow 1001$

$\Rightarrow -7_{10} = 1001_2$

(works in reverse)

Sign/magnitude numbers

first digit shows if + or −

$+7_{10} = 0111_2$ $-7_{10} = \boxed{1}111_2$ ↳ means ⊖

However:

```
   1 11
  00111
+ 01111
-------
  10110 ≠ 0
```

⇒ addition does not work

2's compliment addition

```
     11
 7   0111
-7  +1001
    -----
   (1)0000 = 0
```

⇒ addition works

(1) ↓ ignored when 1 number is ⊕ and ell ⊖

	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	+8
2's comp	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	
Sign	☒	1111	1110	1101	1100	1011	1010	1001	0000 1000	0001	0010	0011	0100	0101	0110	0111	

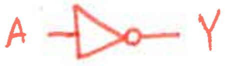
no -8

two different 0

26-08-2020

1.5 Logic gates

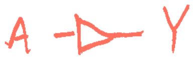
NOT



$Y = \overline{A}$

A	Y
0	1
1	0

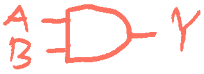
BUF



$Y = A$

A	Y
0	0
1	1

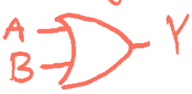
AND



$Y = A \cdot B$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

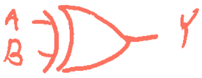
OR



$Y = A + B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

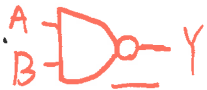
XOR



$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

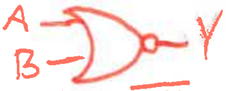
NAND



$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

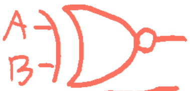
NOR



$Y = \overline{A+B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

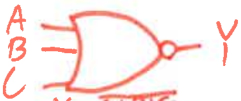
XNOR



$Y = \overline{A \oplus B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

NOR3



$Y = \overline{A+B+C}$

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

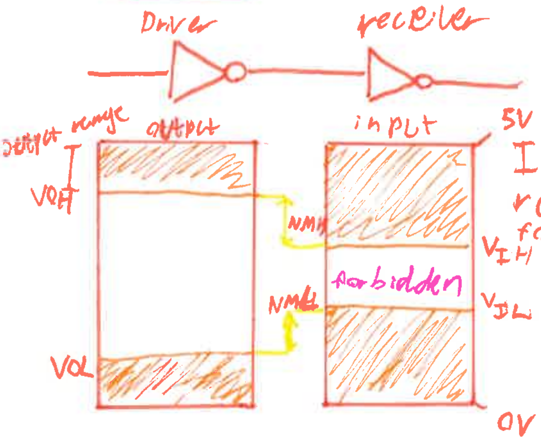
26-08-2020

1.6 Beneath the Digital Abstraction

$0 \Leftrightarrow 0V \Leftrightarrow \perp \Leftrightarrow GND$

$1 \Leftrightarrow 5V \Leftrightarrow V_{DD}$

Noise

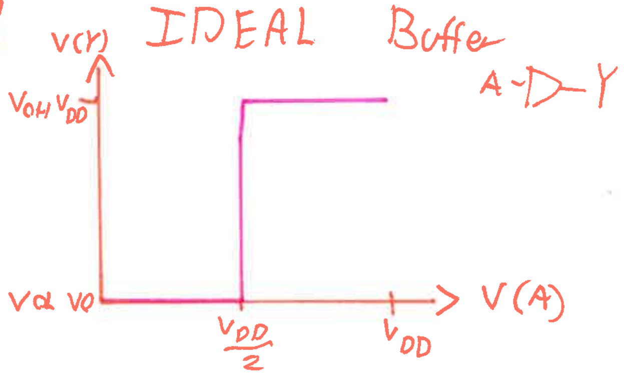


Input is a greater range to account for noise.

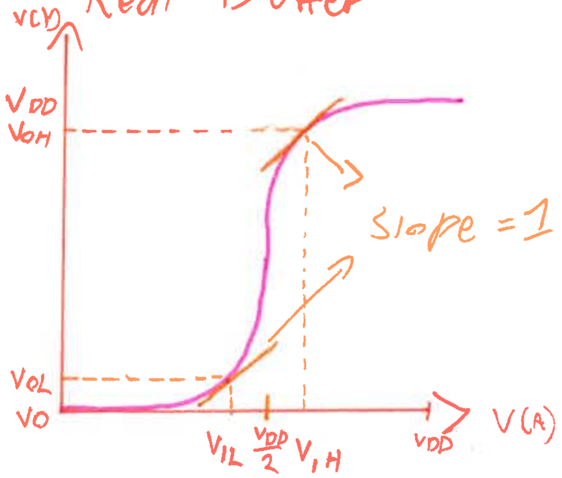
$NMH = V_{OH} - V_{IH}$

$NML = V_{IL} - V_{OL}$

IDEAL Buffer

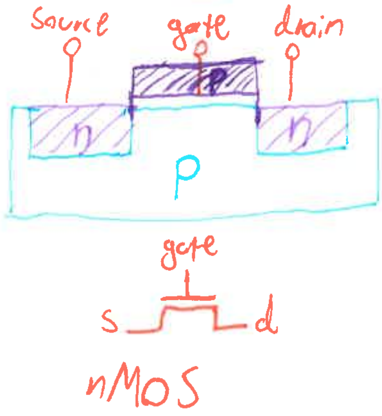


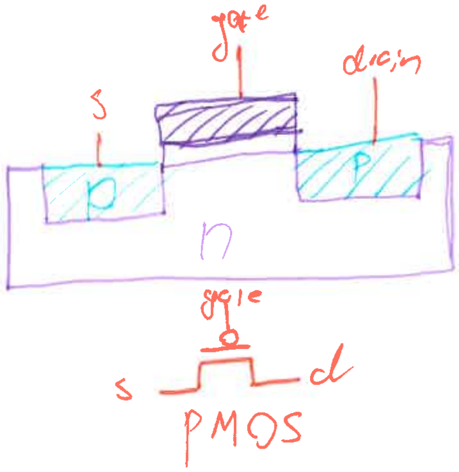
Real Buffer

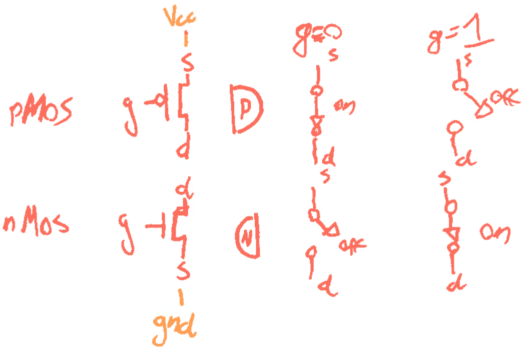


02-04-2020

Transistors 1.7.4





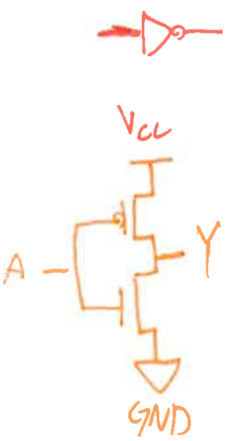


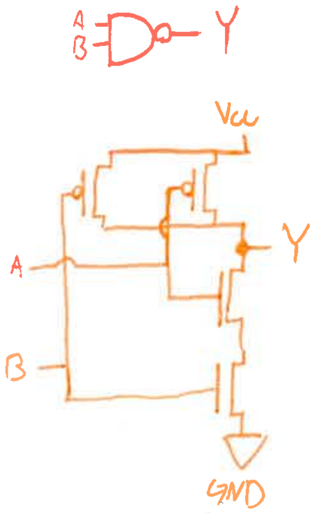
CMOS

gates built out of transistors.
is functionally complect

NOT

NAND





Power consumption

- Energy required to change capacitance (c) to V_{DD} is CV_{DD}^2
- capacitence is changed $\frac{1}{2}$ of the time for each frequency (f). (1 to 0 is free)

=> dynamic power = $\frac{1}{2}CV^2f$

=> static power = IV

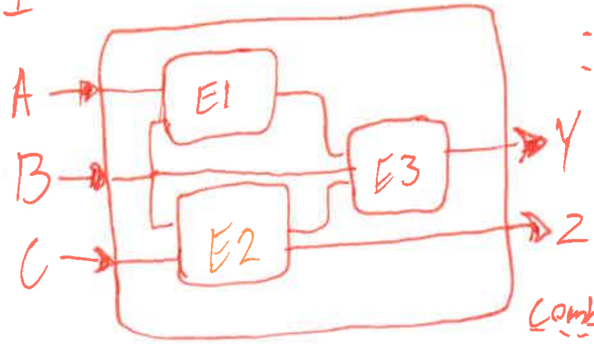
=> Total = $\frac{1}{2}CV^2f + IV$

Chapter 2

Circuits 2.1

Nodes
- inputs ABC
- outputs Y, Z
- internals n1

Circuit Elements
- E1, E2, E3
- all of which is a circuit



functional spec
timing spec

Combinational
- no memory
- output is determined by inputs

↓

Combinational rules
1 every element is also combinational
2 every node is an input or connects to exactly 1 output.
3 no circuit is cyclical

Sequential
- has memory
- output determined by previous and current values of inputs

Boolean Equations 2.2

Variable: A, B, C ...
A B C

Compliment: variable wit a bar
$\bar{A}, \bar{B}, \bar{C}$

Literal: variable or compliment
$A, \bar{A}, B, \bar{B}, C, \bar{C}$

implicant: product of literal
$AB\bar{C}$, $\bar{A}C$, BC

Minterm: product that includes all variables
ABC, $AB\bar{C}$, $\bar{A}BC$

Maxterm: sum that includes all variables
$(A+\bar{B}+C)$, $(\bar{A}+B+C)$, $(\bar{A}+\bar{B}+\bar{C})$

Boolean axioms

A1 $B=0$ or $B\neq1$

A2 $\bar{0}=1$ $\bar{1}=0$

A3 $0\cdot0=0$ $1+1=1$

A4 $1\cdot1=1$ $0+0=0$

A5 $1\cdot0=0$ $1+0=1$

Sum of products (SOP)

- all equation can be written as Sop
- each product is a minterm

A	B	Y	Minterm	
0	0	0	$\bar{A}\,\bar{B}$	m_0
0	1	1	$\bar{A}\,B$	m_1
1	0	0	$A\,\bar{B}$	m_2
1	1	1	$A\,B$	m_3

minterm of all true values.

$$\Rightarrow Y=\underbrace{\bar{A}B+AB}_{\text{can be simplified}}$$

Products of sums (pos)

- all equations can be written as pos
- each sum is a maxterm

A	B	Y	Maxterm	
0	0	0	$A+B$	M_0
0	1	1	$A+\bar{B}$	M_1
1	0	0	$\bar{A}+B$	M_2
1	1	1	$\bar{A}+\bar{B}$	M_3

Maxterm of all false values

$$\Rightarrow Y=(A+B)(\bar{A}+B)$$

Boolean Theorems

T1 $B\cdot1=B$ $B+0=B$ Identity

T2 $B\cdot0=0$ $B+1=1$ Null element

T3 $B\cdot B=B$ $B+B=B$ Idempotency

T4 $\bar{\bar{B}}=B$ Involution

T5 $B\cdot\bar{B}=0$ $B+\bar{B}=1$ Complements

T6 $B\cdot C=C\cdot B$ $B+C=C+B$ commutative

T7 $(B\cdot C)\cdot D=B(C\cdot D)$ $(B+C)+D=B+(C+D)$ assosiatk

T8 $B\cdot(C+D)=(B\cdot C)+(B\cdot D)$ $B+(C\cdot D)=(B+C)(B+D)$ Distributive

T9 $B\cdot(B+C)=B$ $B+(BC)=B$ covering

T10 $(B\cdot C)+(B\bar{C})=B$ $(B+C)(B+\bar{C})=B$ combining

T11 $(BC)+(\bar{B}D)+(CD)=(BC)+(\bar{B}D)$ $(B+C)(\bar{B}+D)(C+D)=(B+C)(\bar{B}+D$ consensus

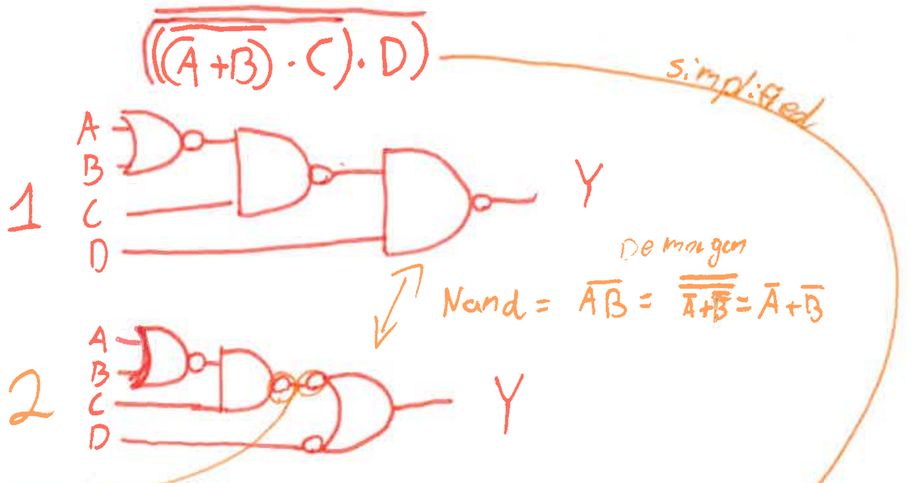
T12 DeMorgan's Law

$$\overline{B_1\cdot B_2 - B_n}=\bar{B}_1+\bar{B}_2+\bar{B}_3\cdots\bar{B}_n$$

$$\overline{B_1+B_2\cdots B_n}=\bar{B}_1\cdot\bar{B}_2\cdots\cdot\bar{B}_n$$

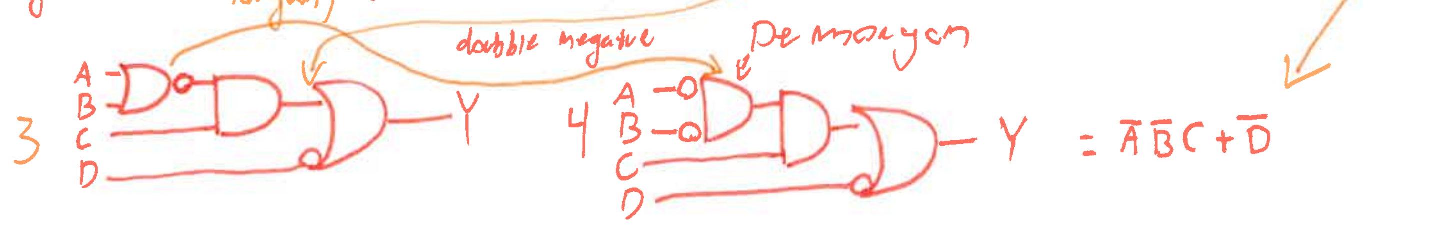
Bubble pushing 2.5.2

- Begin at outputs and work towards inputs
- Push bubbles back towards inputs
- Working backwards cancel out bubbles and change gates. (de' morgan)



De morgen

Nand = $\overline{AB} = \overline{\overline{\overline{A}+\overline{B}}} = \overline{A}+\overline{B}$



$= \overline{A}\,\overline{B}C+\overline{D}$

Dont cares (x) 2.6.1

X represents a value that is either 1 or 0 depending on your liking.

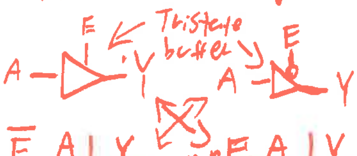
A	B	Y
0	0	0
0	1	1
1	0	X
1	1	X

=>

A	B	Y
0	0	0
X	1	1

Floating values (Z) 2.6.2

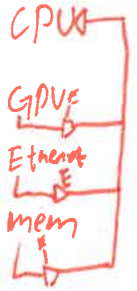
- Z represents float/high impedance.
- It may be 0 or may be 1 or inbetween.
- Not always bad
- Allows outputs to be connected.



$\overline{E}$	A	Y
0	0	0
0	1	1
1	0	Z
1	1	Z

goede fel !!

E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

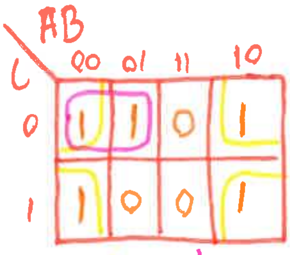


Kmaps 2.7

- Simplifying Boolean equations

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

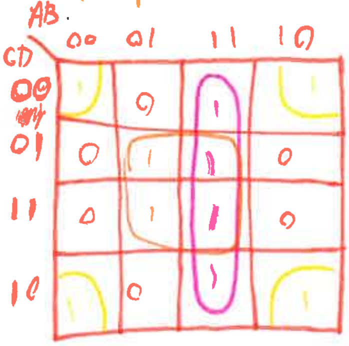
3-input



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

$Y = \overline{B} + \overline{A}\,\overline{C}$

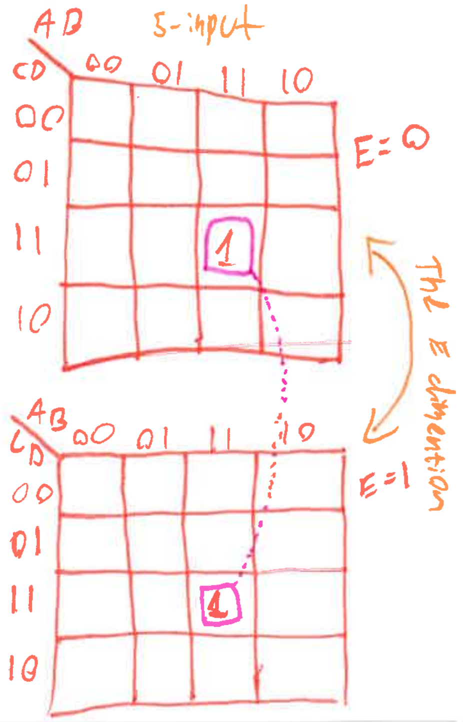
4-input



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

$Y = \overline{B}\,\overline{D} + BD + AB$

5-input



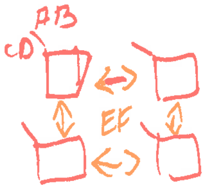
E=0

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

E=1

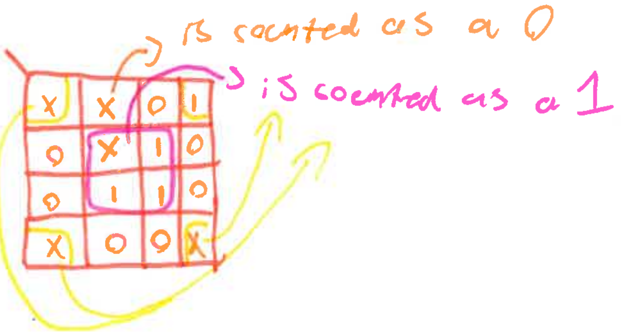
CD \ AB	00	01	11	10
00				
01				
11			1	
10				

exists for dimentions up to 6



Kmaps with don't cares 2.7.3

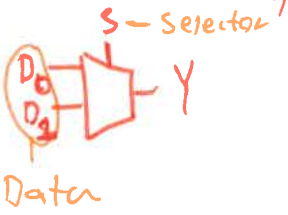
- Don't cares can be used as either 1 or 0 depending on whats needed.



X	X	0	1
0	X	1	0
0	1	1	0
X	0	0	X

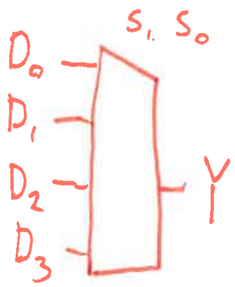
Multiplexer 2.8.1

- choose one output based on several inputs.

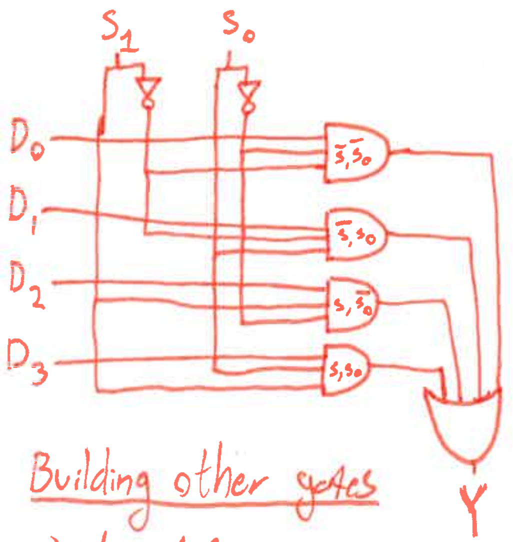


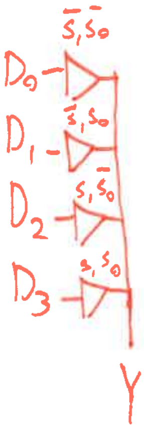
S	D_1	D_0	Y	
0	0	0	0	D_0
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	D_1
1	0	1	0	
1	1	0	1	
1	1	1	1	

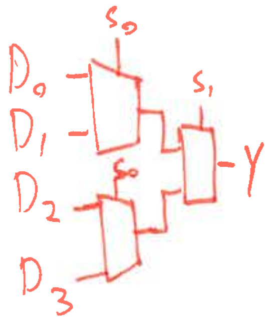
- number of data inputs is equal to $2^{|S|}$



Building a multiplexer







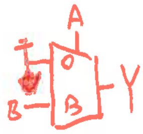
Building other gates

and gates

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

->

A	Y
0	0
1	B

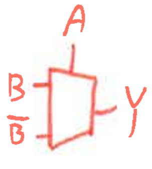


xor gates

A	B	Y
0	0	0
0	1	[illegible]
1	0	1
0	0	0

->

A	Y
0	B
1	$\overline{B}$

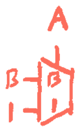


or

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

->

A	Y
0	B
1	1

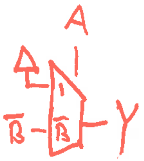


Nand

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

->

A	Y
0	1
1	$\overline{B}$



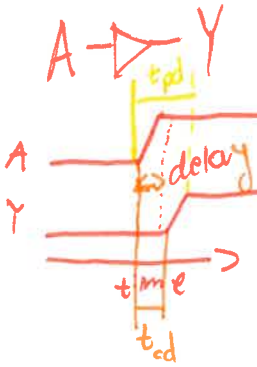
Decoder 2.8.2

- N inputs 2^n outputs
- binary to tally conversion
- one-hot because exactly one output is hot (on) at a time
- can also be used to make compact function (like a mux can)

A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0						0	1
0	0	1						0	1	0
0	1	0					0	1	0	
0	1	1				0	1	0		
1	0	0			0	1	0			
1	0	1		0	1	0				
1	1	0	0	1	0					
1	1	1	1	0						0

Timing 2.9

- Circuits are not ideal and require time to change
- rising edge =: High to Low
- falling edge =: Low to High



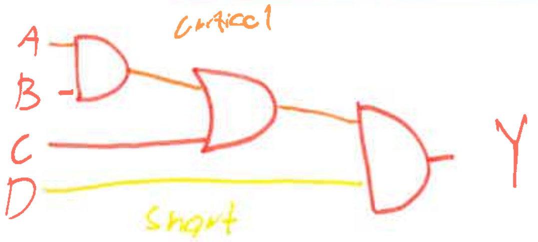
propagation delay = t_{pd}

contamination delay = t_{cd}

Depend on:

- Different rising and falling delays
- Multiple inputs have different speeds
- temperature. High is faster.

Critical and Short path



t_{pd} is related to the critical path: $t_{pd} = 2 \cdot t_{pd\_AND} + t_{pd\_OR}$

t_{cd} is related to the short path: $t_{cd} = t_{cd\_AND}$

Timing of Elements

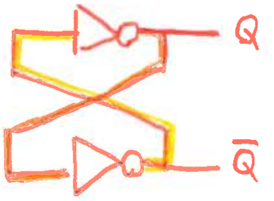
	t_{pd} (ps)
NOT	30
2 AND	60
3 AND	80
4 OR	90
tri A to Y	50
tri E to Y	35

Sequential logic E

- Output depends on current and previous values.
- previous values is called State

Bistable Circuit

- the cercuit is stable for two values.

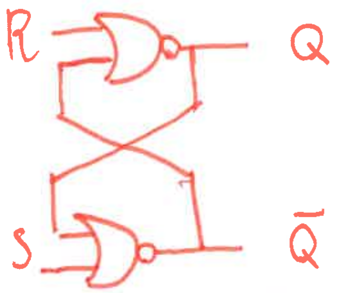


case 1 Q=1 $\overline{Q}$=0

case 2 Q=0 $\overline{Q}$=1

TWO states are stable

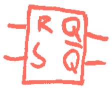
SR-Latch



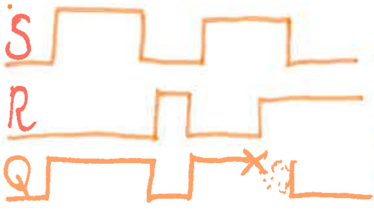
Set reset lathch

S sets Q to 1

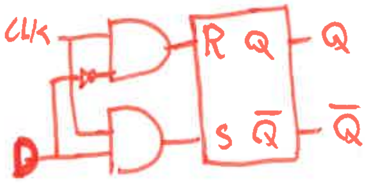
R sets Q to 0



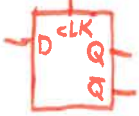
S+R = Undefined

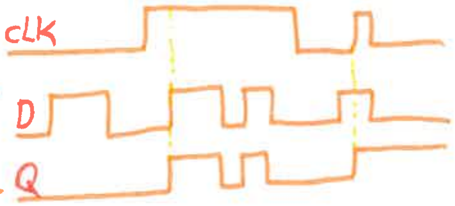


D-Latch

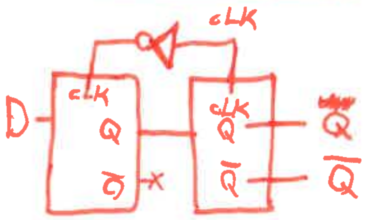


Saves every walue of D ~~whenever~~ while CLK is High. does not have undefined state.

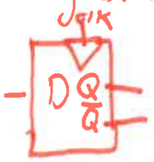


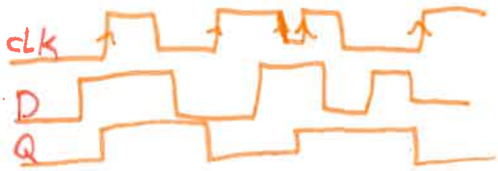


D-Flip-Flop

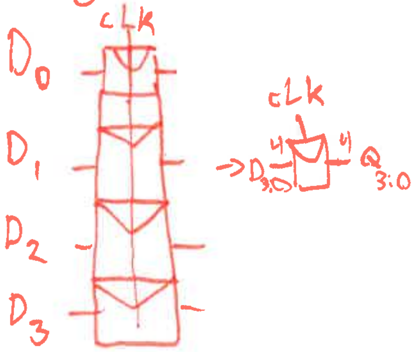


saves value of Q to the value of D every Edge (either rising or faning)

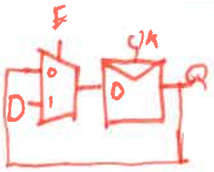




Registers



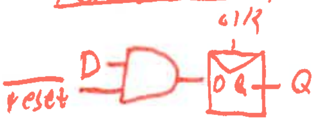
Enabled flip-flops



E = 1 => ~~retains value~~ ~~Q~~ D passes through.

E = 0 => retains value of Q

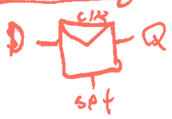
Reset flip-flops



Saynchronus -> resets with clock

Asynchronus -> reset imediatlly

Settable flip flop

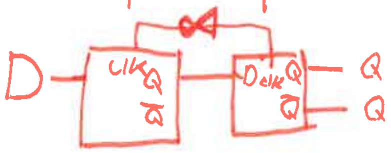


S=1 -> Q set to 1

S=0 -> normal flip flop

Flip Flops (extra Lecture)

D Flip-Flop



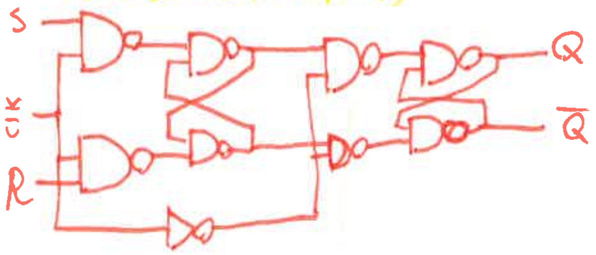
sets value of Q to value of D at rising edge

D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1

only on clk rising edge

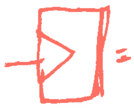
SR-Flip Flop

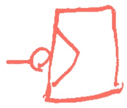
(master slave flip flop)



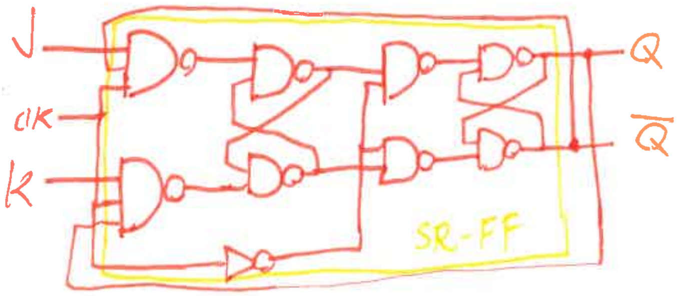
S	R	Q	$\overline{Q}^+$
0	0	Q_0	$\overline{Q}_0$
0	1	0	1
1	0	1	0
1	1	X	X

undefined


= rising edge


= falling edge

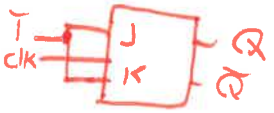
JK flip flop



J	K	Q^+
0	0	0
0	1	0
1	0	1
1	1	$\overline{Q}_0$

Like an SR-FF but 11 is now defined as a toggle of Q.

T-Flip flop

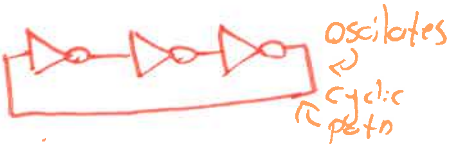


toggles every time T is on

T	Q
0	Q
1	$\overline{Q}$

Sÿnchronos logic design 3.3

- all circuts that are not combinational. are sequential
- Problematic function



- synchronos sequential ~~circuits~~

breaks cyclic paths by inserting registers.

↳ contains the state of the machine.

Rules of synchronous sequential circuits

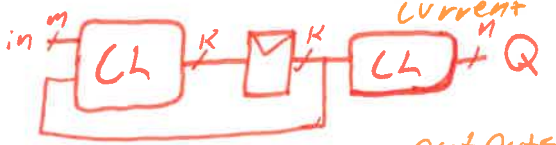
1) Every element is either a register or a combinational circuit.
2) at least one register in circuit
3) all registers share clock
4) every cyclic path has at least one register

Finite State machine (FSM) 3.4

- consists of:
  - ↳ Combinational logic:
    - ↳ computes the next state
    - ↳ computes the output
  - ↳ State registers
    - ↳ Stores current state
    - ↳ Loads next state at clock
    - ↳ 2^n states where n is the number of registers

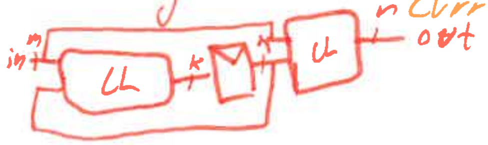
Two types

Moore FSM



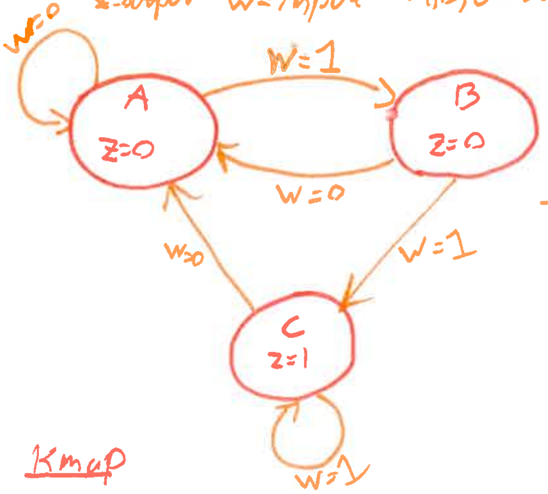
output depends on current state

Mealy FSM



outputs depend on current state and current input.

z=output w=input A,B,C=state



→

State transition Table

Present state	Next state		output
	w=0	w=1	
A	A	B	0
B	A	C	0
C	A	C	1

Encoded transition Table

present state	Next state w=0 $Y_2 Y_1$	state w=1 $Y_2 Y_1$	output z
A=00	00	01	0
B=01	00	10	0
C=11	00	10	1
X=10	XX	XX	X

$$Y_2 Y_1 = f(y_2, y_1, w)$$

$$Z = f(y_2, y_1)$$

Kmap

Y_1

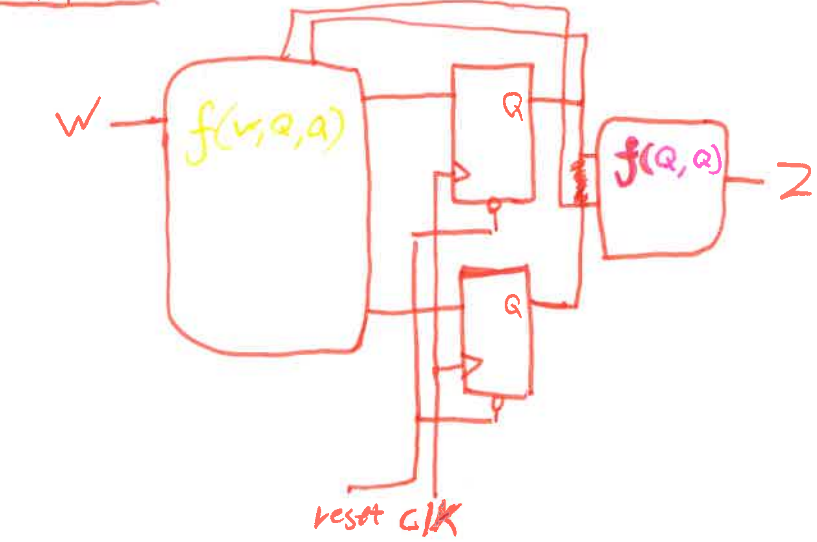
w \ $y_2 y_1$	00	01	11	10
0	0	0	X	0
1	1	0	X	0

Y_2

w \ $y_2 y_1$	00	01	11	10
0	0	0	X	0
1	0	1	X	1

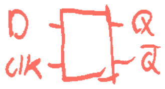
y_2 \ y_1	0	1
0	0	0
1	1	X

Schematic



Timing

The state of D must be stable at the clk pulse.



setup time

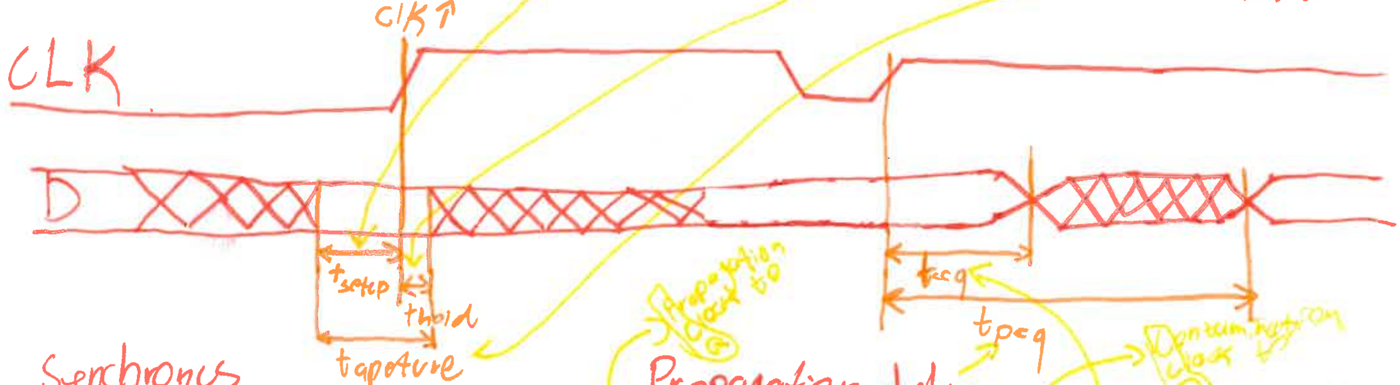
t_{setup} = time before clk edge that D must be stable

Hold time

$time_{hold}$ = time after clk edge that D must be stable.

Apeture time

$t_{setup} + t_{hold}$ the total time that D needs to be stable



Synchronous circuits must be stable in the t_a time

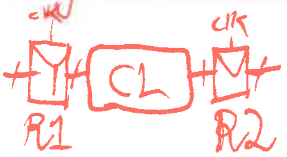
Propagation delay

t_{pcq} = time from clk edge to when Q is stable again

Contamination delay

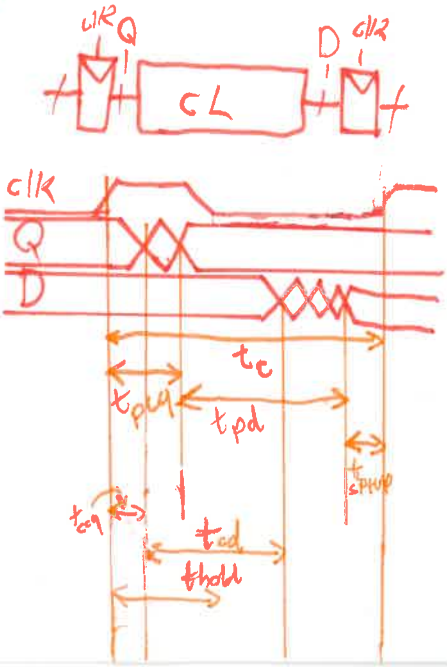
t_{ccq} = time taken for q to start changeing (become unstable) after clk edge

Dynamic Discipline



Due to the timing of CL, a max and min delay of clk is needed

Frecency $f_c = \frac{1}{T_c}$



There must be enough time to setup next clk edge

⇒ $T_c \geq + t_{pcq} + t_{pd} + t_{setup}$

⇒ $t_{pd} \leq T_c - (t_{pcq} + t_{setup})$

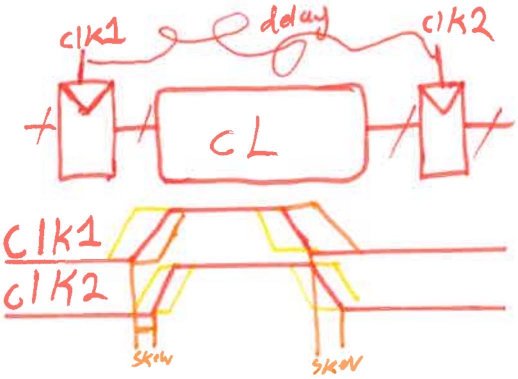
There must be enough time to hold Q before it begins changeing

⇒ $t_{hold} < t_{ccq} + t_{cd}$

⇒ $t_{cd} > t_{hold} - t_{ccq}$

Clock skew

There may be delay between each register's clock pulse(edge)



This affects the time dynamics.

$\Rightarrow T_c \geqslant t_{pcq} + t_{pd} + t_{setup} + \boxed{t_{skew}}$

$t_{pd} \leqslant T_c - (t_{pcq} + t_{setup} + t_{skew})$

$\Rightarrow t_{ccq} + t_{cd} > t_{hold} + t_{skew}$

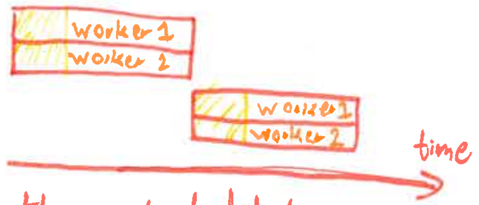
$t_{cd} > t_{hold} + t_{skew} - t_{ccq}$

Parallelism

can be used together

Spacial parallelism

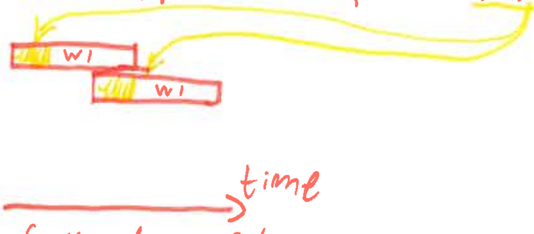
Multiple workers



throughput doubled

Temporal parallelism

Task split into parallel subtasks



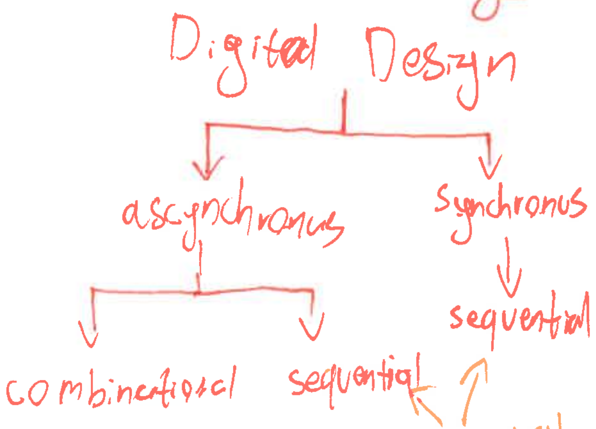
faster throughput

Token - group of inputs processed into outputs

Latency - Time taken for one token to be processed

Throughput - Number of tokens per unit time

Asynchronus Design



sequential elements are themselves ascynchronus, eg: SR-latch

Drawbacks of scynchronus Design

While synchronus circuits are easy to build and look at and analyze, they also are: A lot slower.

- 80% of paths only use 20% of the clk time

Drawbacks of asynchronus Design

- A lot harder to analyze
- Harder to manufacture
- Harder to design
- Hazards are problems when the circuit gets stuck.

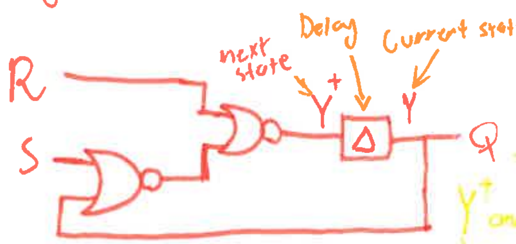
Asynchronus state Machines

Only one signal in a circuit may change its value at a time.

Golden rule listed above.

An asynchronus state machine is a state machines without flip flops and only with combination logic

Asynchronous S-R latch



Delay is only for illustrate

Y^+ and Y may be diffent But only stable when equal

Truth table

Y	S	R	Y^+
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

State table — also called excitation table

Present state Y	next state Y^+ 00	01	11	10
0	0	0	0	1
1	1	0	0	1

Circled stable states.

a state is stable Iff $Y(t)=y(t+\Delta)$ $\Leftrightarrow Y=y^+$

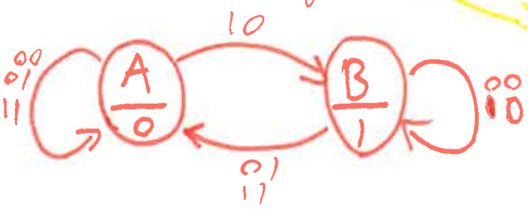
assigned state table — also known as flow table

Preset	next state 00	01	11	10	output Q
A	A	A	A	B	0
B	B	A	A	B	1

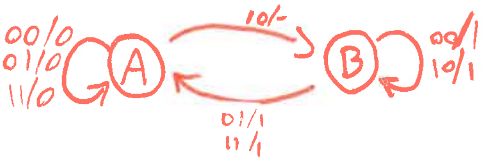
Nonstable states can also be don't cares.

The state may often transition to another state via an unstable state.

Moore:



mealy

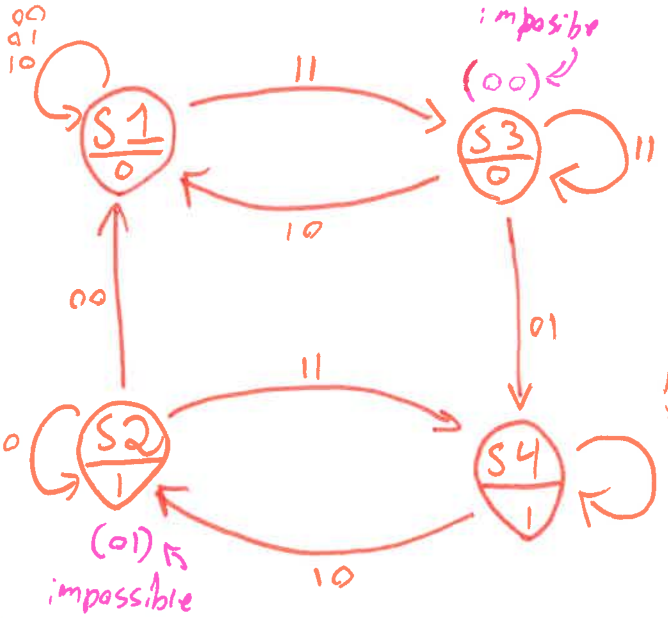


Summary of Asynchronus circuits Analysis

1) Replace feedbacks in circuits with $Y^+ \boxed{\Delta} y$ element.
2) Find expression for next state $Y^+ = A+B+y$
3) setup excitation table
4) setup flow table
5) setup/draw a state diagram.

Flow Table

Since only one input can change at a time. Some transitions are impossible.



Note: there are no S1-S4 or S3-S2 transitions since that would require two changes.

one inputchange is one step L or R from yellow =>

Present State	next state 00	01	11	10	Q
S1	S1	S1	S3	S1	0
S2	S1	~~S1~~	S4	S2	1
S4	S4	S4	S4	S2	1
S3	~~S4~~	S4	S3	S1	0

Gray code

cannot be reached from 11

Design of Asynchronus Circuits

1) Create a state diagram
2) create a flow table
3) Assign codes and create an excitation table
4) Determine expressions from K-map
5) Construct a circuit. Save for Hazards

Example: Serial parity generator (%2) mod 2

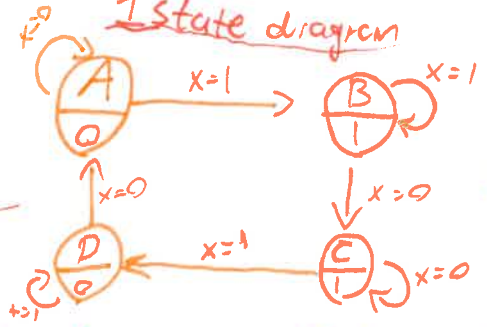
Conditions: X - input

z - output

z = 1 iff number of x pulses is odd

z = 0 iff number of x pulses is even

1 State diagram



Note that x is being read continues

2 Flow table

current state	next state x=0	x=1	z
A	A	B	0
B	C	B	1
C	C	D	1
D	A	D	0

3 Assign codes (graycode is best)

Current state	Next state x=0	x=1	Z
00	00	01	0
01	11	01	1
11	11	10	1
10	00	10	0

A = 00
B = 01
C = 11
D = 10

4 Draw K-map

Y_1^+

x \ Y_0Y_1	00	01	11	10
0	0	1	1	0
1	0	0	1	1

Y_0^+

x \ Y_0Y_1	00	01	11	10
0	0	1	1	0
1	1	1	0	0

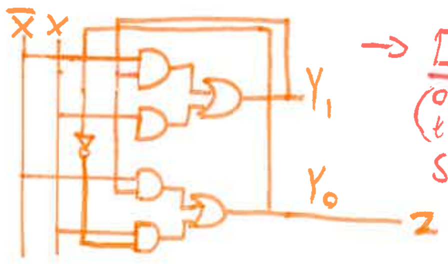
$\Rightarrow$ $Y_1^+ = \bar{x}Y_1 + xY_0$ $\quad Y_0^+ = \bar{x}y_1 + x\bar{Y_0}$

$Z = Y_0$ (trivial)

y_1 \ Y_0	0	1
0	0	1
1	0	1

$Z = Y_0$

5 Build circuit



→ Done! (or so you thought)

See hazards on next page

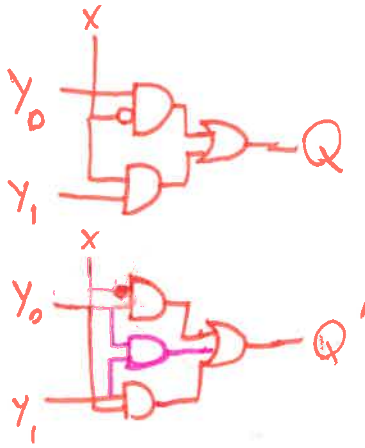
Hazards (more on that later)

Y_1^+	00	01	11	10
0	0	1	1	0
1	0	0	1	1

Y_0^+	00	01	11	10
0	0	1	1	0
1	1	1	0	0

$$Y_1^+ = \bar{x}y_0 + \boxed{y_1 y_0} + xy_1 \rightarrow$$

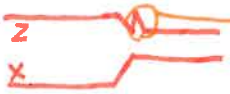
$$Y_0^+ = x\bar{y}_1 + \boxed{\bar{y}_1 y_0} + \bar{x}y_0$$



HAZARD! ← when x changes

Hazard free

In addition to the orange circles, the pink ones are needed to ensure there are no glitches in the x=0 ⟺ x=1 transition.



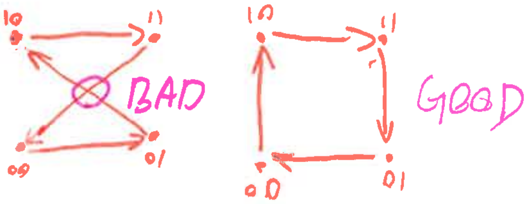
the shift from z to $\bar{z}$ is not a single edge

State Encoding

Since there cant be two changes at once the Hamming distance of two states between which there is a transition must equal 1.

The smallest number of bits that two numbes differ.
eg: 11 → 10 = 1
11 → 00 = 2
11 → 11 = 0

in short, no crosses in state diagram

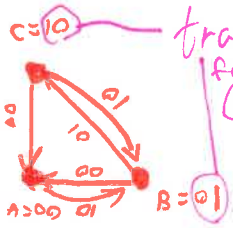


Unused states

When the number of States is not a powe of 2, There may be transitions with a Hamming distance greater than 1. then you must take use of unused states.

current state	next state x_1x_0 00	01	11	10	Q
A	A	B	–	C	0
B	A	B	B	C	1
C	A	B	C	C	1

⇒

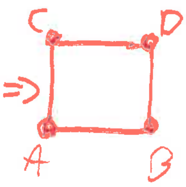


transition too few (hamming=2)

A	A	B	–	C	0
B	A	B	B	D	1
D	–	B	–	C	dd
C	A	D	C	C	1

Note: D has no stable states

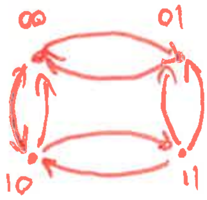
⇒



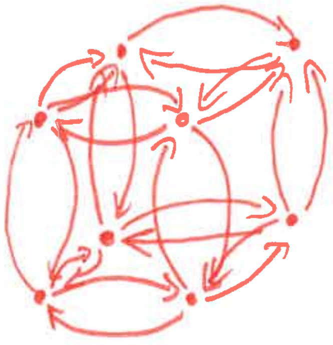
← Transition state

More dimentions can be Introduced to name more States and prevent Invalid Transitions

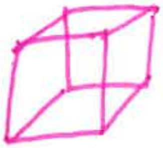
D=1
2 transitions



D=2
8 transtion



D=3
24 transitions

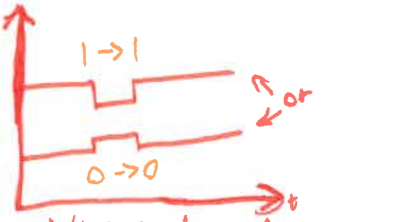


It's a cube basically

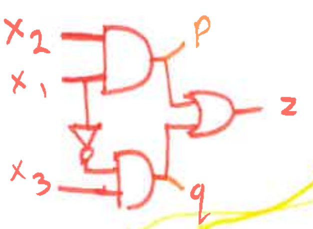
More on Hazards

Hazards can be static or dynamic. They should be avoided!

Static Hazards



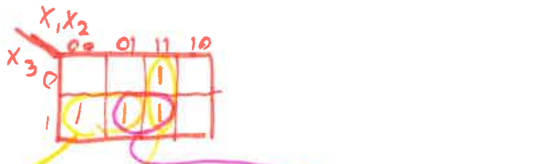
When transferring between states where the output is meant to be constant, it glitches.

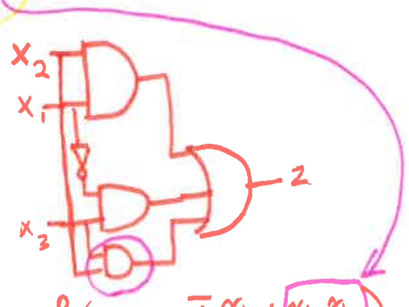


$f(x_1x_2 + \overline{x}_1x_3)$

in the 111 → 110 transition the inverter causes a delay from when p turns off to when q turns on. Causing a glitch.

How to fix →

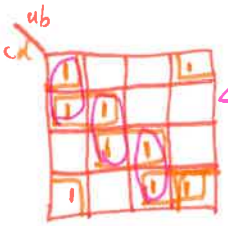




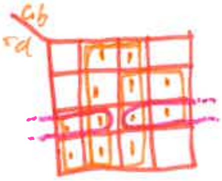
$f(x_1x_2 + \overline{x}_1x_3 + \boxed{x_3x_2})$

Hazard free ☺

Hence to avoid static Hazards, all adjacent 1s in the K-map must be covered beware of packman adjacent ones. →

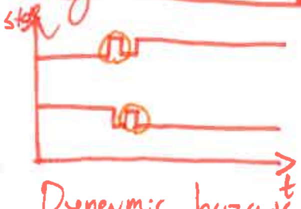


Pink circles avoid Hazards

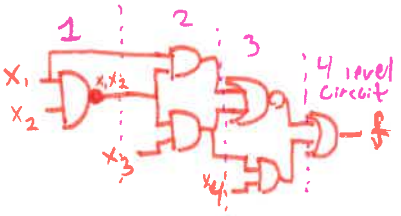


Packman circle

Dynamic Hazard

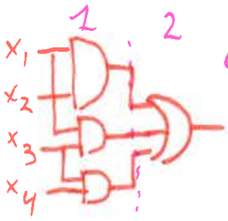


Dynamic hazards occur when switching states to a state with different output and the output changes more than once



These occur in multi-level circuits as gate transitions happen. at different times and speed.

How to fix →

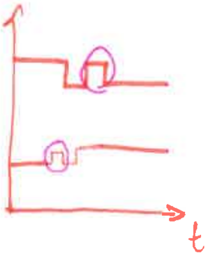


Create a 2 level Circuit with one AND and one OR level.

Glitches

There are also glitches that are NOT Hazards.

Present state $y_2 y_1$	next state $x=0$	next state $x=1$	output
0 0	0 0	0 1	1
0 1	0 0	1 1	0
1 1	0 1	1 0	1
1 0	1 1	1 0	0

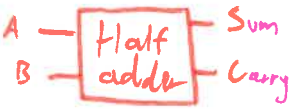


As the input x goes to 1 from 0 or vice versa, the circuit goes through unstable states with different outputs causing a glitch. This is Not a Hazard, and may be intentional.

Module 4 (chapter 5)

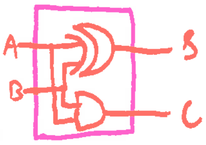
Arithmetic circuits

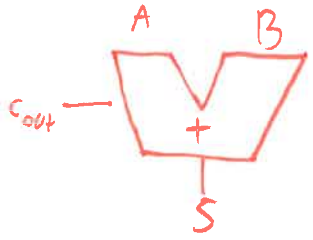
1 bit adders

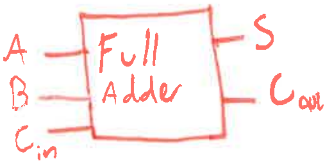


A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$\Rightarrow S = A \oplus B$
$C = AB$

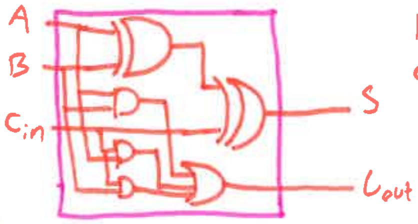


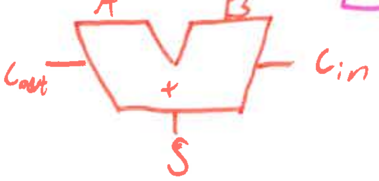




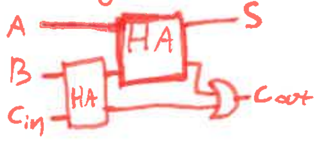
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = A \oplus B \oplus C_{in}$
$C_{out} = AB + AC_{in} + BC_{in}$





A fulladder can also be constructed from two half adders and and or gate.

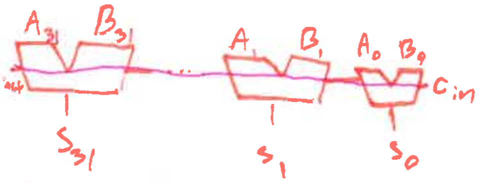


Multibit Adder

3 types:

- Ripple carry slow
- Carry lookahead fast
- Carry select inbetween

Ripple carry:



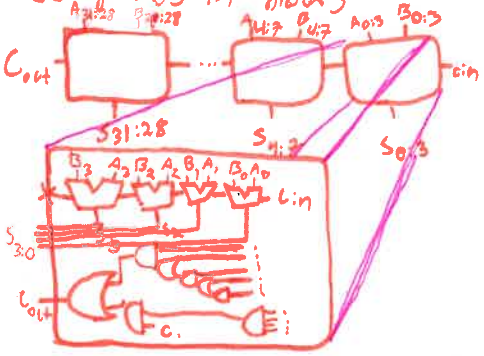
The carry needs to propagate through every single one of the 32 full addes And that takes a lot of time. Hence it is slow. The only positive is that it is easy to build and does not require a lot of gates.

$t = N \cdot t_{full\ adder}$

Number of bits

Carry lookahead:

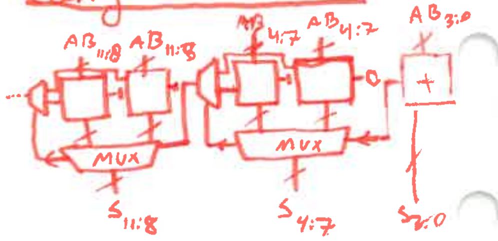
Computes in blocks



each block directly computes C_{out} so that less calculation is done sequentially.

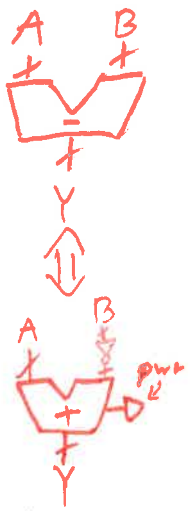
$t = t_{pg} + t_{pg\_block} + (N/k + 1)t_{AND\_OR} + kt$ [illegible]

Carry select adder:



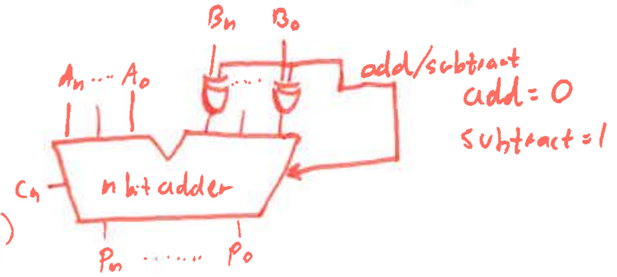
Calculates parts of the addition in paralell ahead of time for carry in Equal to both 1 and 0 and feeds C_{in} into a mux to decide betwee the two.

Subtracter

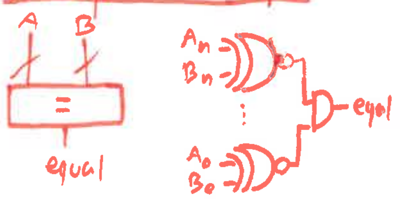


$A - B = A + (-B)$
so write B as -B
by taking twos
compliment by inverting
each bit and adding
one (done through the carry)

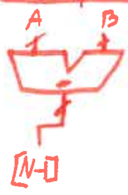
Adder and subtracter



Comparator: equal

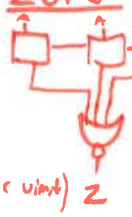


Comparator: less than



first bit of subtraction
determines whether
$A < B$

zero

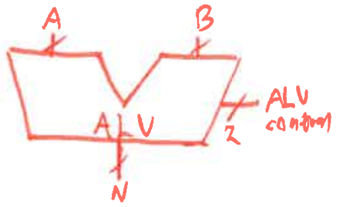


the NOR
gate of
the sum
determines
a zero

ALU (arithmetic logic unit)

Bitwise →

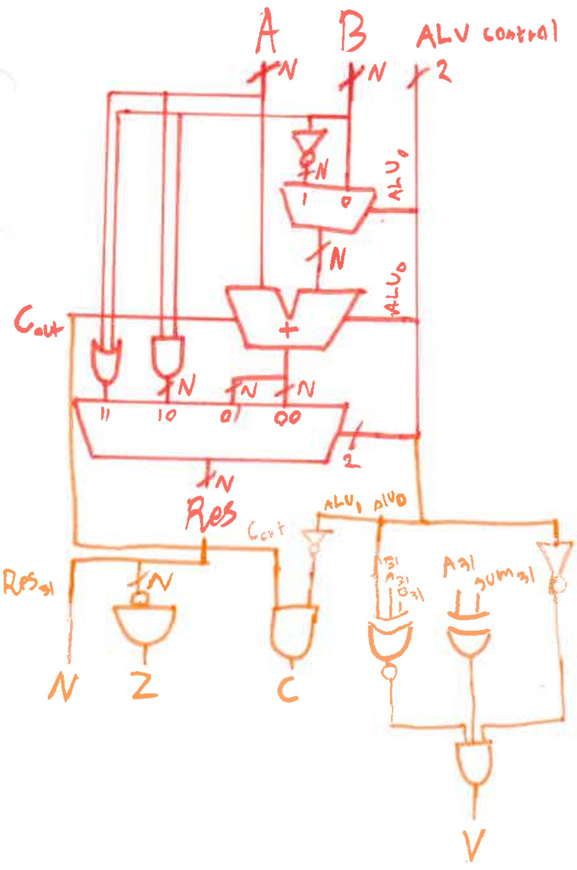
Add	00
Subtract	01
AND	10
OR	11



ALU flags:

N - Negative result
Z - Result is zero
C - Postive Carry out
V - Adder oVerflow

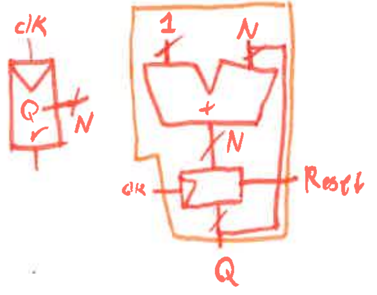
→ 0 is to similar to zero the number



Counter

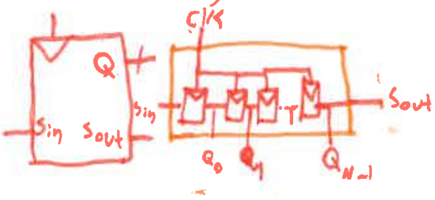
A scynchronus circuit hat increments by one each rising edge.

000 -> 001 -> 010 -> 011....



Shift Register

Shifts all bits to left or right every clk rising edge. (can be used as a serial to parallel converter)



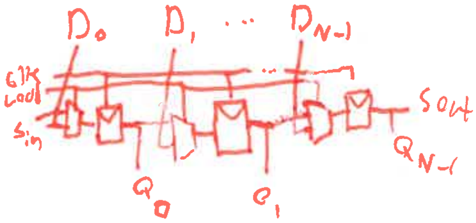
S_{in} determines what is filled

S_{out} ~~takes the Value~~ what comes out.

Shift Register w/ Parallell Load

Load = 1 acts as normal N-bit register

Load = 0 acts as a Shift register



Different Bit shift

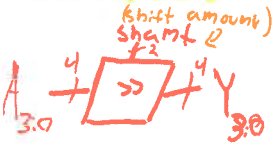
Logical shift:

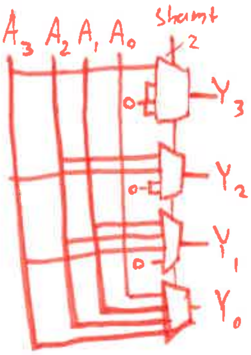
when shifting the new bits are filled with zeros.

11001 >> 2 =
00110

two new zeros

(shift amount)
shamt





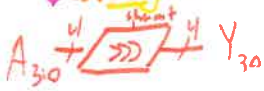
Aritmatic shift:

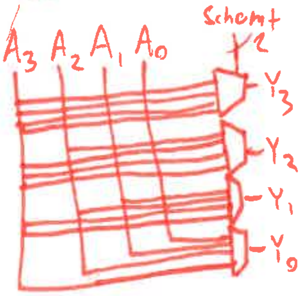
Right shift copies MSB, Left shift is the some as logical.

11001 >> 2 =
11110

MSB

11001 << 2 =
00100 zeros





Rotator:

Rotates MSB to LSB. Qute easy

11001 >> 2 =
01110

11001 << 2 =
00111

Binary multiplication

Multiplying with 2^N is equivalent to a shift by N.

$5 \cdot 2^3 = 40$

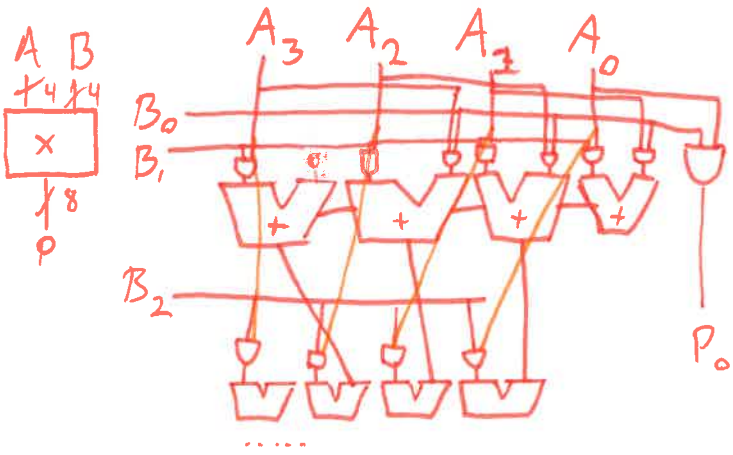
$\Leftrightarrow$

$101 \cdot 1000 = 101000$

Hence binary multiplication is the sum of multiplications of two.

```
  230          0101
x  42        x 0111
-----        ------
  460           101
 920           101
-----         101
 9660         000
             ------
             100011
```

4x4 multiplier



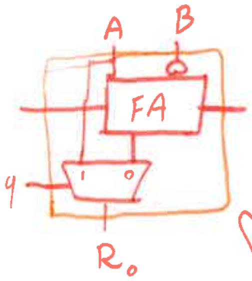
It's kinda complicated....

Binary division

Continually subtract $2^N \cdot$ denomenator from the numerator until the remainder is less than the numerator.

$101101 \div 011 = 1111$

```
      1³1²1¹1⁰  <- quotient
   ___________
11 | 1 0 1 1 0 1
   -(1 1       )
   -------------
     1 0 1 0 1
   -(1 1       )
   -------------
       1 0 0 1
    -( 1 1    )
   -------------
           1 1
        -( 1 1)
   -------------
Remainder -> 0
```



Basically it uses a lot of these

Representing fractions in binary

Fixed point:

Add an imageary comma between numbes.

$\overset{2^3}{0}\overset{2^2}{1}\overset{2^1}{0}\overset{2^0}{1}.\overset{2^{-1}}{1}\overset{2^{-2}}{1}\overset{2^{-3}}{0}\overset{2^{-4}}{1}$

$2^2 : 4$

$2^0 : 2$

$2^{-1} = .5$

$2^{-2} = .25$

$2^{-4} = .125$

$= 6.875$

This cannot represent all fractions, but can come close. Addition happens as you expect it to.

floating Point:

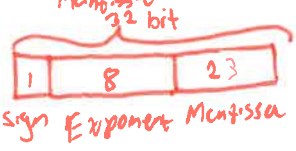
Similar to scientific notation. but in the form

1.0110×2^{127}

In general

$\pm M \times 2^{E}$

sign | Mantissa | exponent



special cases

Numbers	S	E	M
0	X	00000	00000...
∞	0	1111111	00000...
-∞	1	1111111	00000...
NaN	X	1111111	Not zero

Float to Decimal

1) Convert to binary

$'116'_{10} = 1110100_2 = m$ ↳ don't confuse M with m

2) Find exponent

$\underset{6\,5\,4\,3\,2\,1}{1110100} \Rightarrow e = 6$ ↳ don't confuse e with E

3) fill in the binaryfloat:

S = 0 if +, S = 1 if −

⇒ S = 0

E is normalized to 127

So $E = e + 127$

$\Rightarrow E = 127 + 6 = 133$

$= 10000101$

M the first 1 of m is implied so it is not inulvded And m is shifted left until m has 23 digits

$\Rightarrow M = m_{5:0} \ll 17$

$M = 11010000000000000000000$

$\Rightarrow 116 = \underbrace{0}_{S}\ \underbrace{10000101}_{E}\ \underbrace{11010000000000000000000}_{M}$

float addition:

1) extract exponent and fraction bits
2) Prepend 1 to mantissa
3) compare exponent
4) Shift mantissa of smaller exponent if needed
5) add mantissa.
6) normalize mantissa
7) Round result
8) assemble float point

Float multiplication

$a = a_{frac} \cdot 2^{a\,exp}$

$b = b_{frac} \cdot 2^{b\,exp}$

$c = ab$

$= a_{frac} \cdot b_{frac} \cdot 2^{a\,exp + b\,exp}$

Rounding

floats can only represent some of the numbes, hence rounding is needed.

there is:

Down

Up

towards zero

to nearest.