Number system

Binary: 0/1
Decimal: 0~9
Hexadecimal: 0~F

Conversion

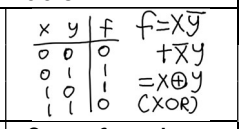
B->D: add each '1's
D->B: divide by 2
B<->H:4bit B=1bit H

Logic Gates

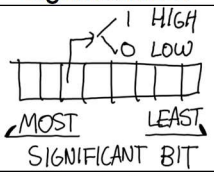
AND		$x \cdot y$	NAND		$\overline{x \cdot y}$
OR		$x + y$	NOR		$\overline{x + y}$
NOT		$\bar{x}$	XOR		$x \oplus y$

Truth Table

Show input sets and output sets in one table



Significant bit



Axioms and Rules for Boolean Algebra

$0 \cdot 0 = 0, 1 \cdot 1 = 1$
$0 \cdot 1 = 1 \cdot 0 = 0$
$x = 0, \bar{x} = 1$
$x \cdot 0 = 0$
$x \cdot 1 = x$

$x \cdot x = x$
$x + \bar{x} = 1$
$\bar{\bar{x}} = x$
$x \cdot y = y \cdot x$
$x \cdot (y \cdot z) = (x \cdot y) \cdot z$

$x \cdot (y + z) = x \cdot y + x \cdot z$
$x + x \cdot y = x$
$xy + x\bar{y} = x$
$\overline{xy} = \bar{x} + \bar{y}$
$x + (\bar{x} \cdot y) = x + y$

BoolAlg Order

Parentheses (bracket)
→ NOT
→ AND
→ OR

Sum of products

Add all **midterms** where output=1

$f = \sum \mathrm{m}_{(f=1)}$

(Canonical term)

Product of sums

Add all **maxterms** where **output=0**

$f = \prod M_{(f=0)}$

(Canonical Term)

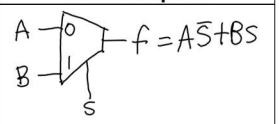
Synthesis and Cost

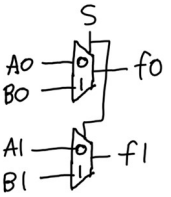
Synthesis: add all min/maxterms into minimal cost form
Cost = #Gates + #inputs

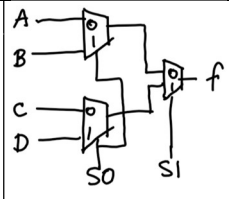
Multiplex (MUX)

Selects one signal between multiple signals
Depends on another digital signal(s) "S"

2:1 Multiplex








```
module name (a,b,t,f)
  input a, b, t;
  output f;
assign f=(~t&a) | (t&b);
endmodule
```

Hierarchy

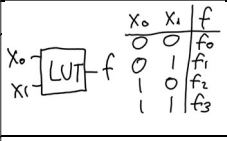
Add sub-modules into main module (similar with functions)

Common cathode

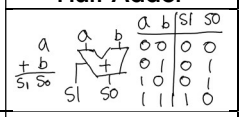
Turns on when segment value is 1 (anode -> 0)

Lookup table (LUT)

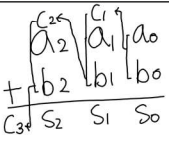
Programmable gate array, hold interconnection 'code' of the gates

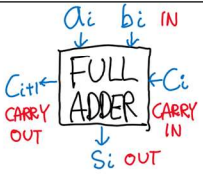


Half Adder



Full Adder



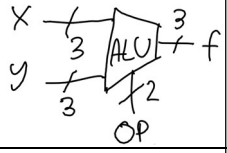


Always block

Defines behaviour using procedural statements (output reg ~~)

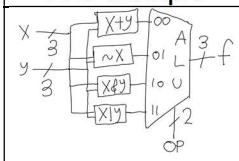
```
output reg out;
always@(*)
  begin
    (procedural stmt)
  end
```

ALU




```
reg out;
always@(*)
  begin
    case(??)
      (cases & default)
    endcase
end
```

ALU example

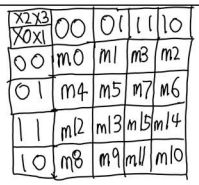


Karnaugh map

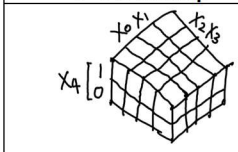
X1	X2	f
0	0	f0
0	1	f1
1	0	f2
1	1	f3

X1 \ X2	0	1
0	f0	f1
1	f2	f3

2D table! (right)



5-bit K map



Literal: variable of its complement
Implicant: 1s in SOP, 0s in POS
Prime implicant (PI): groups of 1s

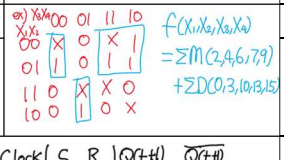
Cover: Any group of implicants that covers all 1s (in equation form)
Essential P.I.: PIs not in contact with other PIs

Construct K-map

1. Find all prime implicants
2. Identify essential prime implicants
3. Choose non-essential PIs to save cost

Don't care

Output doesn't matter for my purpose
Indicated as 'X'

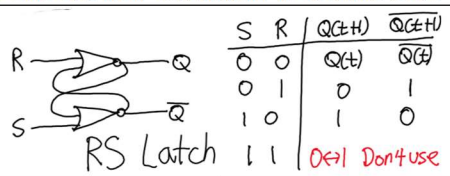


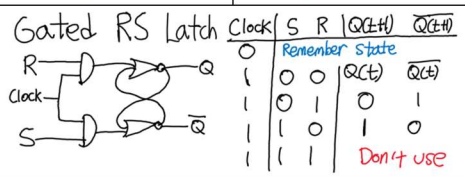
Sequential circuit

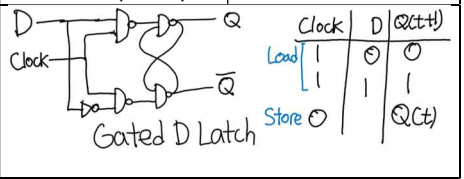
Output depends on input and past values (state)

State

Function of past values, must be remembered



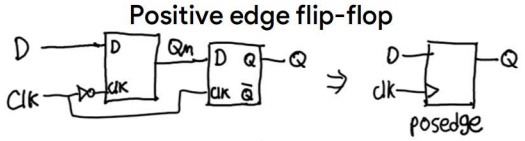




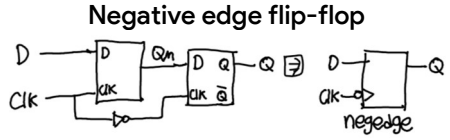
Flip Flop

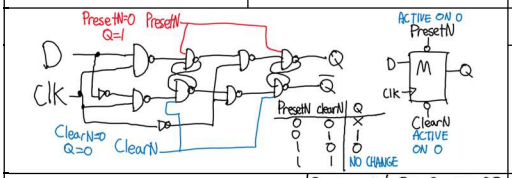
Storage element, change output at edge of clock

Positive edge flip-flop



Negative edge flip-flop





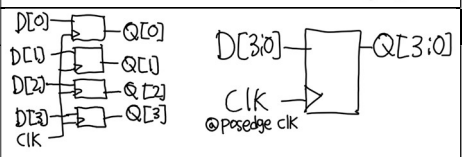
Synchronous

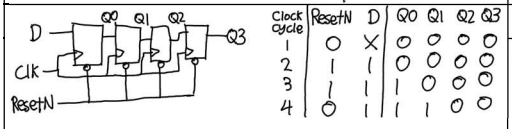
Dependent on clock

always@(posedge clk)

Asynchronous

Output changes independent on clock (resetN, presetN)





Blocking (=)

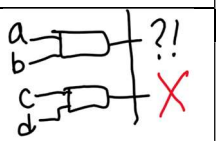
Assignments in always block are evaluated in order *(order matters)*

Non-blocking (<=)

Assignments in always block are evaluated concurrently

Don't do

Put output into two always block



Verilog tips

No initial statement (use always)
Draw schematics!

Bitwise expression

Result -> length(in)
~A, A | B, A&B, A^B

Logical

Result in 1 bit
!A, A || B, A&&B

Show number

n'b(n bits binary)
ex) 3'b101

Relations

>, <, >=, <=, ==

Reduction

&a
|a
Add multiple elements

Arithmetic

a+b, a-b, a/b, a*b, a%b

Concatenate

out={b,a}
when out[5:0]
b[1:0]
a[3:0]