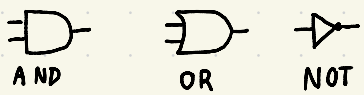
ENG 201

Basic Logic



▷ BOOLEAN ALGEBRA AXIOMS

$$\overline{(a+b+c+d+\cdots)} = \bar{a}\cdot\bar{b}\cdot\bar{c}\cdot\bar{d}\cdot\cdots$$

$$a\cdot(b+c) = (a\cdot b)+(a\cdot c)$$
$$a+(b\cdot c) = (a+b)\cdot(a+c)$$

▷ PROOF TECHNIQUES

Introduce a constant:

Ex. Prove $x+(x\cdot y)=x$

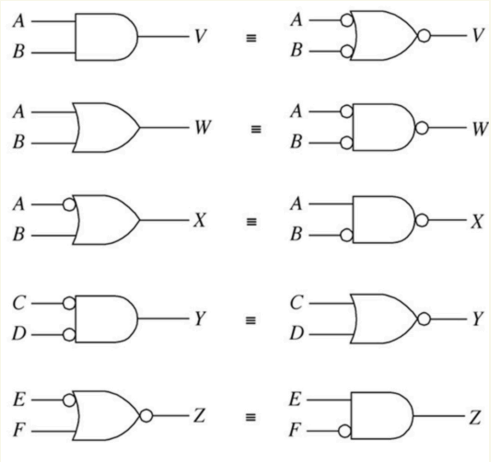
$x + x\cdot y = x\cdot 1 + x\cdot y$ identity theorem
$= x\cdot(1+y)$ distribution
$= x\cdot 1$ null element
$= x$ identity
QED

Ex. Prove $x+(y\cdot z)=(x+y)\cdot(x+z)$

$(x+y)\cdot(x+z) = x\cdot(x+z)+y\cdot(x+z)$
$= x\cdot x + x\cdot z + y\cdot x + y\cdot z$
$= x\cdot(1+z\cdot y)+y\cdot z$
$= x\cdot 1 + y\cdot z$
$= x+(y\cdot z)$

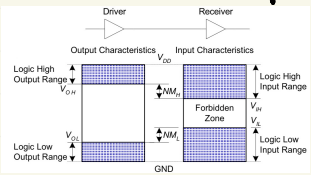
▷ BUBBLE PUSHING

- De Morgan's Law means you can "push" the bubble backwards, NAND(x,y) = OR($\bar{x}$, $\bar{y}$)



▷ NON-IDEAL CIRCUITS

- HIGH & LOW voltage ranges are wider for inputs than outputs



Power

- **Power to charge transistor gate capacitances**
  - Energy required to charge a capacitance, *C*, to V_{DD} is CV_{DD}^2
  - Circuit running at frequency *f*: transistors switch (from 1 to 0 or vice versa) at that frequency
  - Capacitor is charged *f*/2 times per second (discharging from 1 to 0 is free)
- Dynamic power consumption:

$$P_{dynamic} = \tfrac{1}{2}CV_{DD}^2 f$$

Equations

- Minterm: a product of n literals (ex. abc, $a\bar{b}\bar{c}$)
- Maxterm: sum of n literals (ex. $a+\bar{b}+\bar{c}$)
- Canonical sum of products: a sum of minterms
- Canonical product of sums: a product of maxterms

- You can make a minterm out of any row of a truth table.

0	1	1	1	→ $\bar{x}yz$

- $\bar{x}yz = \text{minterm}_{011} = m_3$ — decimal version of binary "bit flag"
- You can make a maxterm too

0	1	1	0	→ $x+\bar{y}+\bar{z}$

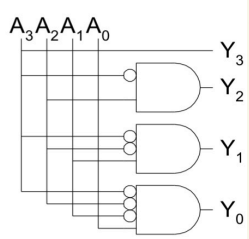
Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2 x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1 x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1 x_2 x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2 x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1 x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1 x_2 x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

Priority Circuits

- A priority circuit outputs the most significant bit.

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

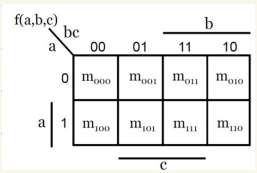
We don't care about these values

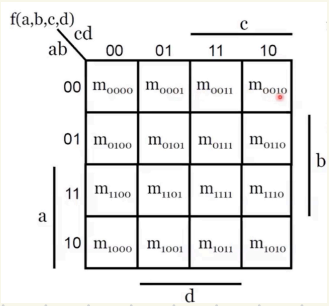


Note: X means bits we don't care about, or two outputs connected to each other.
Z refers to tristate buffers / open circuits / high impedance.

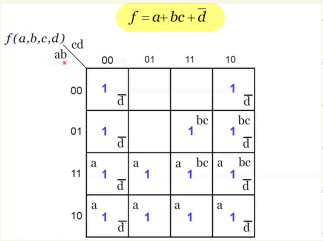
K-Maps

- K-maps are a graphical way to represent truth tables.
- You can have up to 4 variables.



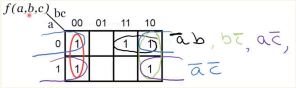


Ex.



- Adjacent cells differ by one bit

▷ SIMPLIFYING K-MAPS

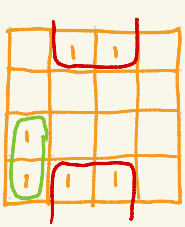


- The sum of two minterms in adjacent cells can be simplified to a single product.

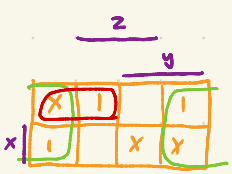
  $\bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} = \bar{b}\bar{c}(\bar{a}+a) = \bar{b}\bar{c}$

- Find groups of minterms, then sum these groups.
  - Groups must have a size of power of two
- X means "don't care" – treat as either 0 or 1

Ex.



Ex.



$= \bar{z} + \bar{x}\bar{y}$

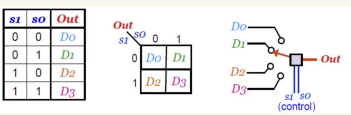
Complex Building Blocks

- Decoder: Binary 2-bit number to select one of 4 outputs

S_1	S_0	EN	D_3	D_2	D_1	D_0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0
X	X	0	0	0	0	0

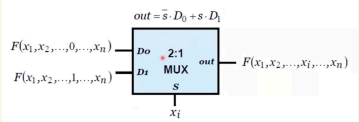
- Multiplexer: selects one signal from 4 to output. "Hardware Look-up Table"

s1	s0	Out
0	0	D0
0	1	D1
1	0	D2
1	1	D3



  - You can convert any equation to use a multiplexer by factoring out a term.

$$F(x_1,x_2,\ldots,x_i,\ldots,x_n) = \overline{x_i}\cdot F(x_1,x_2,\ldots,0,\ldots,x_n) + x_i\cdot F(x_1,x_2,\ldots,1,\ldots,x_n)$$



- Encoder: outputs a binary number depending on which input is ON

D3	D2	D1	D0	b1	b0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Adders

▷ HALF-ADDER

A	B	Co	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

▷ FULL-ADDER

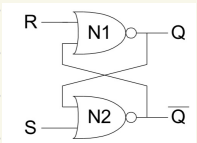
CI	A	B	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = a \oplus b \oplus c$$
$$co = ab + c(a \oplus b) = ab + c(a+b)$$

Latches

SR

- Two inputs: Set & Reset



D

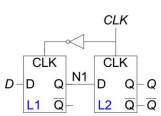
- Two inputs: clock & data
- State is set to value of D when clock is 1

- D flip-flops can be resettable or settable, where a 3rd input bypasses clock & forces it to 0 or 1, respectively
- D flip-flops can be enabled, where clock signal only passes through if a 3rd input is 1

D FLIP-FLOP

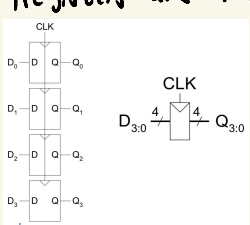
- Unlike a D latch, D flip-flop only changes on rising clock edge

- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When *CLK* = 0
  - L1 is transparent
  - L2 is opaque
  - *D* passes through to N1
- When *CLK* = 1
  - L2 is transparent
  - L1 is opaque
  - N1 passes through to *Q*
- Thus, on the edge of the clock (when *CLK* rises from 0→1)
  - *D* passes through to *Q*
  - All other times Q retains its value



REGISTERS

- Registers are a collection of flip-flops, all on the same clock signal



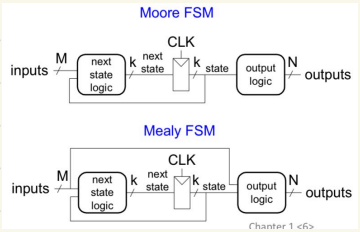
Sequential Logic

- Synchronous design with registers breaks cyclic paths
  - Registers contain the state of the system
- State changes on clock edge

▷ FINITE STATE MACHINE

- Two circuits: one computes next state, one computes output

- Moore FSM: output only dependent on state
- Mealy FSM: output depends on state & input
  - With Mealy, output can change between clock edges

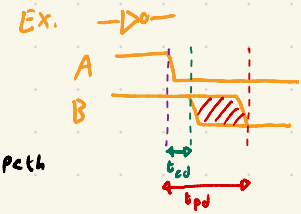


▷ TRANSITION TABLE

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X		
0	0	1	X		
0	1	X	X		
1	0	X	0		
1	0	X	1		
1	1	X	X		

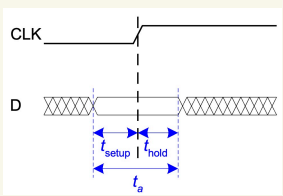
Timing

COMBINATIONAL TIMING

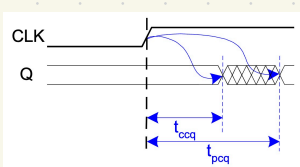


- Propagation delay: max time from when input changes to when output stabilizes (stops changing) – t_{pd}
  - To find t_{pd} of a circuit sum along longest (critical) path
- Contamination delay: min time from when input changes to when output first starts changing – t_{cd}
  - To find t_{cd} of a circuit sum along shortest path

SEQUENTIAL TIMING

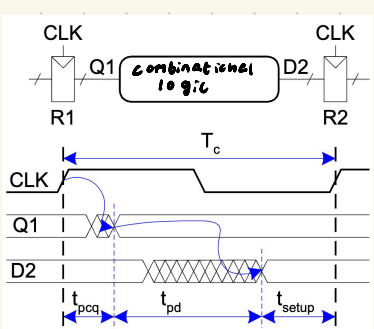


- Flip-Flops must be stable around the clock edge
  - t_{setup}: time input must be stable before clock edge
  - t_{hold}: time input must be stable after clock edge
  - $t_a = t_{setup} + t_{hold}$: aperture time

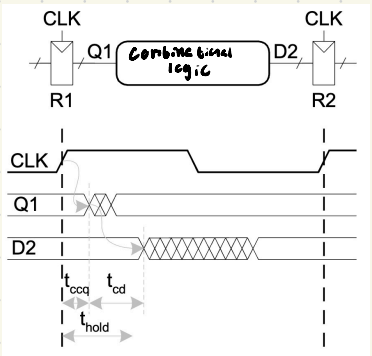


- Propagation delay: time from clock edge to when output Q stops changing
- Contamination delay: time from clock edge to when output Q might be unstable (starts changing)

- The clock's cycle time must be set so that inputs are stable during aperture



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



$t_{hold} < t_{ccq} + t_{cd}$

- Clock skew is the delay between the clock signals of any two registers

With skew:

$T_c > t_{pcq} + t_{pd} + t_{setup} + t_{skew}$

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

▷ METASTABILITY

- If an asynchronous signal (i.e. user input) is sent, the D of a flip flop might change while being sampled
- This results in a metastable state between a 1 and a 0
- You can use synchronizers to make metastable states less likely

Parallelism

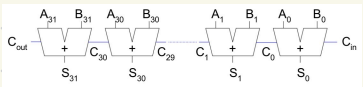
- Spatial parallelism: duplicate hardware performs multiple tasks at once
- Temporal parallelism: task broken into multiple stages

- Token: group of inputs to be processed
- Latency: time for token to pass from start to end
- Throughput: # tokens produced per unit time

Digital Building Blocks

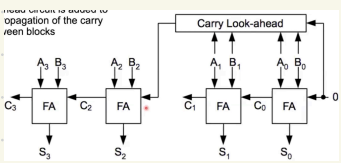
▷ RIPPLE CARRY ADDER

- Slow since carry needs time to be calculated



▷ CARRY-LOOKAHEAD ADDER

- Adders are divided into blocks
- carry is computed before addition completes so next block can start computing

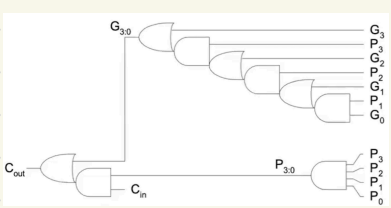


- Lookahead unit
  - Propagates previous carry if $Carry_{prev} \cdot (A + B)$
  - Generates carry if $A \cdot B$

- The carry out of column i (C_i) is:

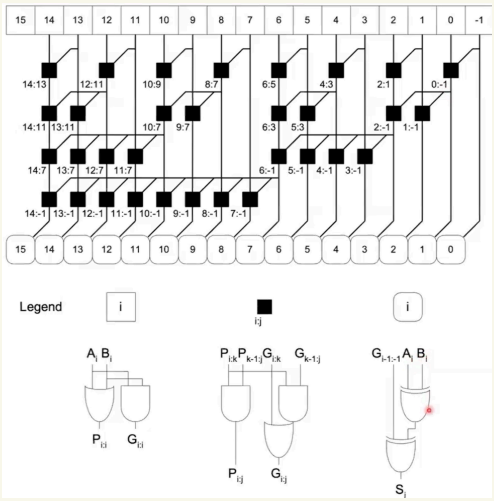
$$C_i = A_i B_i + (A_i + B_i) C_{i-1}$$

- For a 4-bit CLA:
  - $C_0 = P_3 P_2 P_1 P_0$ (if all propagates are true)
    OR $(G_3 + P_3(G_2 + P_2(G_1 + P_1(G_0))))$ (if any generate is propagated)



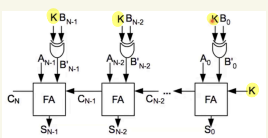
▷ PREFIX ADDER

- Prefix adder achieves Log(n) efficiency by computing P&G in a tree-like fashion



▷ SUBTRACTOR

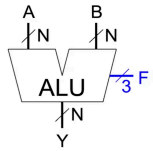
- 2's complement: to invert, flip bits & add 1 (set CI to 1)
- You can have an input K to turn adder into subtractor



- Overflow is detected as XOR of 2 most significant bits

ARITHMETIC LOGIC UNIT

- ALU combines several logical and mathematical operations inside one single unit.
- Heart of all computer systems

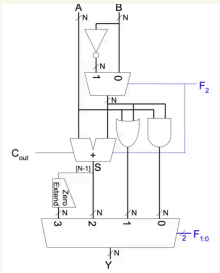


N-bit ALU with N-bit inputs and outputs

ALU gets a control signal F that specifies which function to perform.

$F_{2:0}$	Function
000	A & B
001	A | B
010	A + B
011	not used
100	A & ~B
101	A | ~B
110	A - B
111	SLT

SLT = Set if Less Than



- F selects which operation to perform

SHIFTER

- Logical shift inserts 0's
- Arithmetic SR (>>>) copies MSB
  - Ex. 11001 >>> 2 = 11110

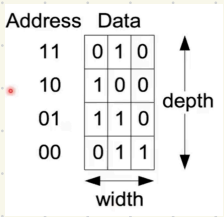
MULTIPLIER

Decimal		Binary
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
		0101
		+ 0000
9660	result	0100011

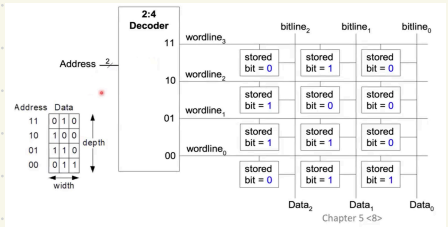
- Just like by-hand multiplication, you compute partial products, shift left, and add

Memory

Data is stored in words, and words are addressed by addresses.



Wordlines turn on/off read/write of bits.

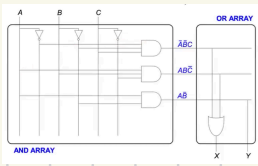


- DRAM needs to be refreshed periodically, and after each read. A simple capacitor
- SRAM is built out of an SR latch
- ROM is read-only - written by burning out fuses

LOGIC ARRAYS

- Circuits can be implemented as look-up-tables
- Logic arrays can be programmable - PLAs, FPGAs

- PLA is an AND LUT followed by an OR LUT



VON NEUMANN MODEL