Number Systems

Tuesday, January 4, 2022 11:10 AM

Key Bases: Decimal (10), Binary (2), Hex (16), Octal (8)

Binary → Hex: Groups 4 bits

Binary → Octal: Groups 3 bits

CMOS, nMOS, pMOS

Tuesday, January 4, 2022 11:57 AM

Basic Terminology

Voltage: Electric potential difference

Current: Flow of charged particles $\frac{\partial Q}{\partial t}$, always from (+) → (-)

Ohm's Law: $V = IR$, in general $V = IZ$

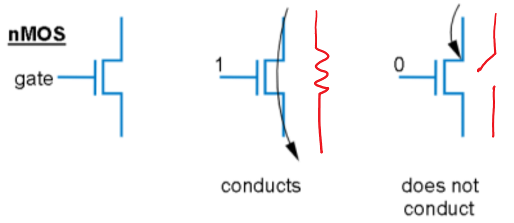
CMOS Switch

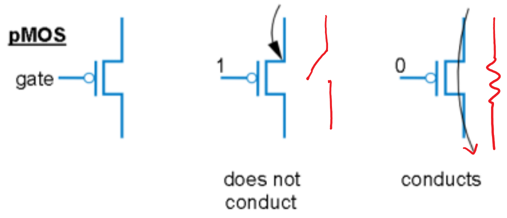
Complimentary Metal Oxide Semiconductor:

↳ Negative: conducts when gate is 1

↳ Positive: conducts when gate is 0

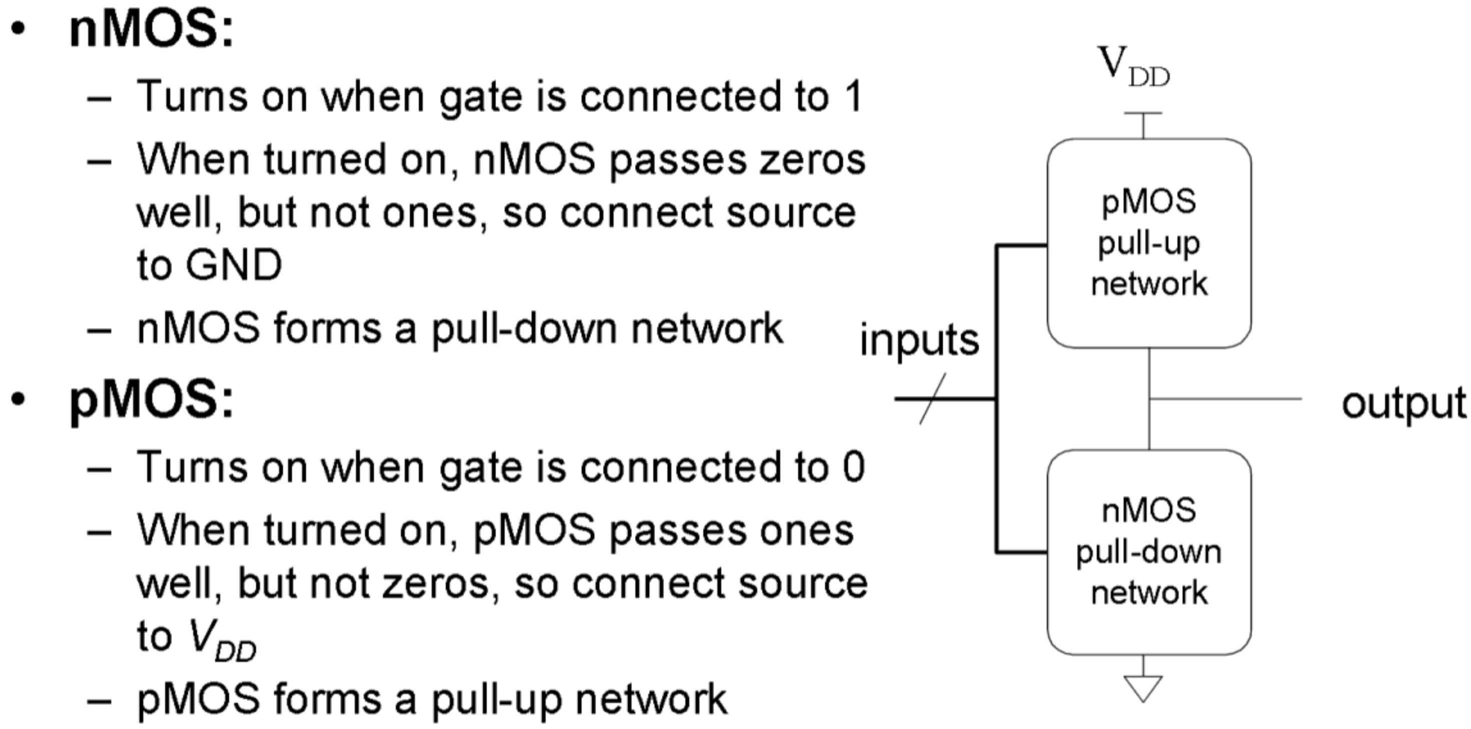
When closed, switch acts as a resistor





nMOS, pMOS: Properties Summarized

Thursday, January 6, 2022 12:10 PM



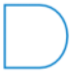
- **nMOS:**
  - Turns on when gate is connected to 1
  - When turned on, nMOS passes zeros well, but not ones, so connect source to GND
  - nMOS forms a pull-down network
- **pMOS:**
  - Turns on when gate is connected to 0
  - When turned on, pMOS passes ones well, but not zeros, so connect source to V_{DD}
  - pMOS forms a pull-up network

Logic Gates, Boolean Algebra Axioms and Theorems

Thursday, January 6, 2022 11:02 AM

Common Logic Gates

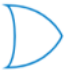
AND – 6 CMOS



$a \bullet b$

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

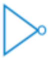
OR – 6 CMOS



$a+b$

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

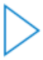
NOT – 2 CMOS



a’

a	NOT
0	1
1	0

BUFFER – 4 CMOS



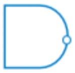
a

a	BUF
0	0
1	1

NOT
NOR } basic CMOS gates
NAND

– **Derived operators**:

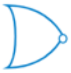
NAND – 4 CMOS



$(a \bullet b)$’

a	b	NAND
0	0	1
0	1	1
1	0	1
1	1	0

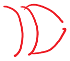
NOR – 4 CMOS



$(a+b)$’

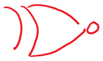
a	b	NOR
0	0	1
0	1	0
1	0	0
1	1	0

XOR – 8 CMOS



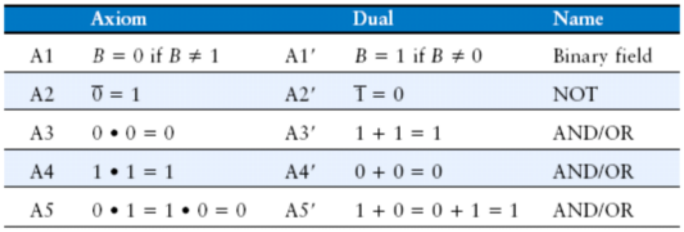
a	b	XOR
0	0	0
0	1	1
1	0	1
1	1	0

XNOR – 8 CMOS

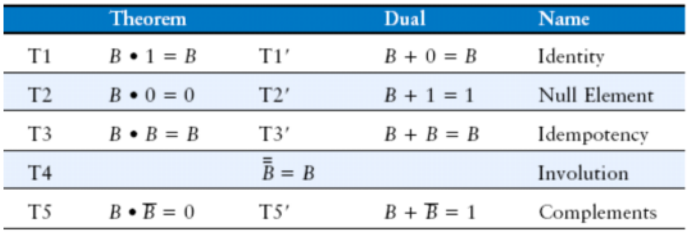


a	b	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

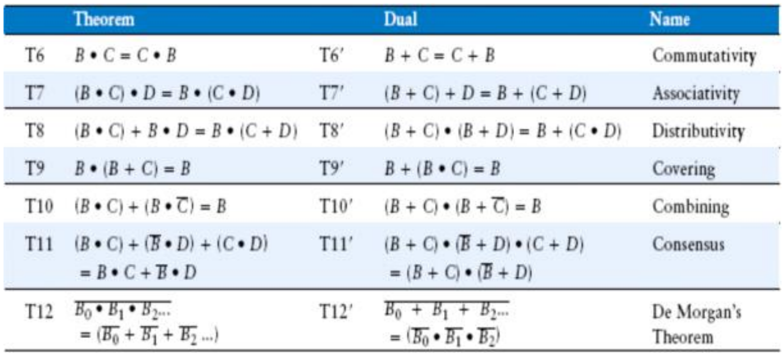
Axioms and Theorems



	Axiom		Dual	Name
A1	$B = 0$ if $B \neq 1$	A1′	$B = 1$ if $B \neq 0$	Binary field
A2	$\overline{0} = 1$	A2′	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3′	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4′	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5′	$1 + 0 = 0 + 1 = 1$	AND/OR



	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

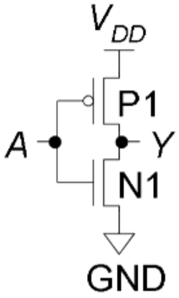


	Theorem		Dual	Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + B \bullet D = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D)$ $= B \bullet C + \overline{B} \bullet D$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D)$ $= (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots}$ $= (\overline{B_0} + \overline{B_1} + \overline{B_2} \ldots)$	T12′	$\overline{B_0 + B_1 + B_2 \ldots}$ $= (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2})$	De Morgan’s Theorem

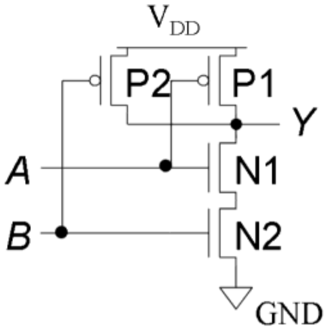
Logic Gates From CMOS, Delay

Thursday, January 6, 2022 12:11 PM

NOT Gate:



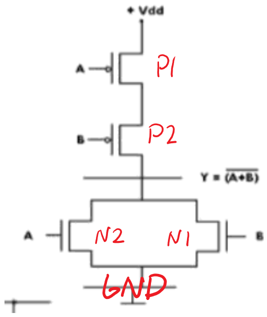
NAND Gate:



PMOS in Parallel

NMOS in Series

NOR Gate:

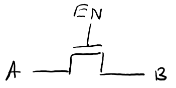


PMOS in Series

NMOS in Parallel

Transmission Gate:

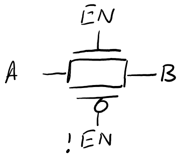
nMOS:



A	EN	Y
0	0	Z
1	0	Z
0	1	0
1	1	<1

passes 1 poorly →

cMOS:



A	EN	Y
0	0	Z
1	0	Z
0	1	0
1	1	1

Reason: Control when a signal is connected.

Delay: Delay is linear to resistance × capacitence. Going through a transistor has resistance and input into a transistor gate has capacitence.

For multiple stages, delay is additive

Bubble Pushing

Thursday, January 6, 2022 12:00 PM

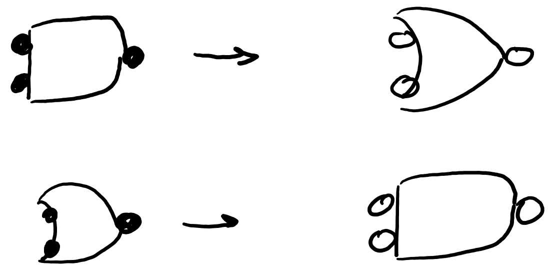
1) Convert AND ↔ OR

2) Convert ● ↔ ○

3) Combine ○ until simplified.

Uses DeMorgan's Laws to simplify circuits.

Ex



Notations, Canonical Form

Tuesday, January 11, 2022 12:07 PM

Complement: variable with bar or '

Litteral : variable or its complement

Implicant: product of litterals (AND)

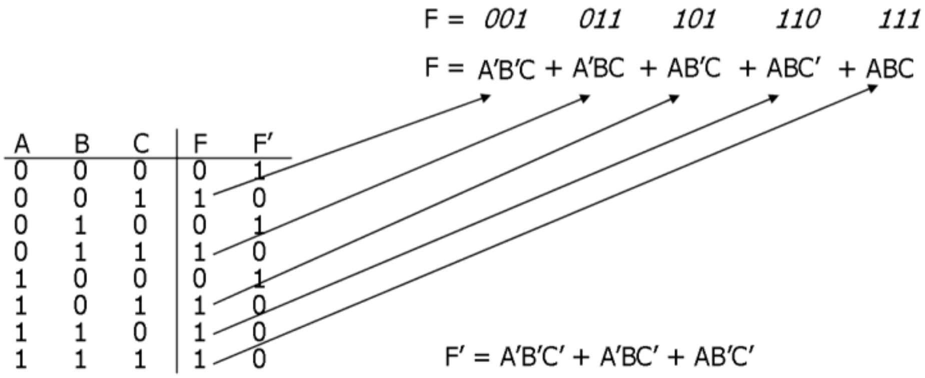
Implicate : sum of litterals (OR)

Minterm : product that includes all input variables

Maxterm : sum that includes all input variables

Canonical form: a sum of either minterms or maxterms, not a minimal form

Sum of Products Canonical form: sum of minterms: Σm



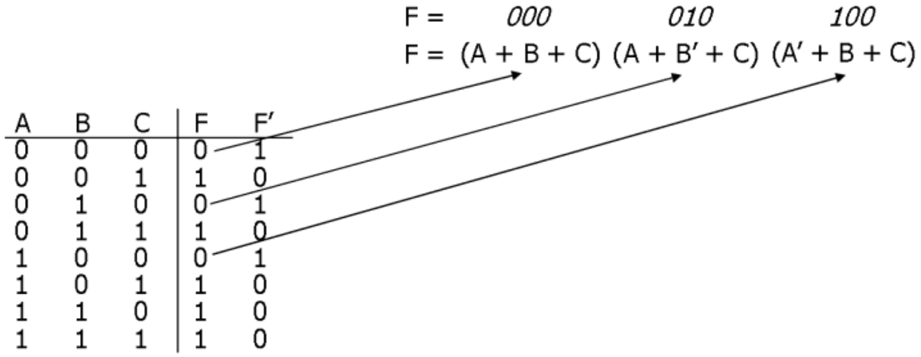
F = *001* *011* *101* *110* *111*

F = A'B'C + A'BC + AB'C + ABC' + ABC

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
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	0

F' = A'B'C' + A'BC' + AB'C'

Product of Sums Canonical form: product of maxterms: ΠM



F = *000* *010* *100*

F = (A + B + C) (A + B' + C) (A' + B + C)

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
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	0

Converting: if $F = \Sigma m$ then $F = \Pi M$ where M are the maxterms that are not used to define F in Σm

Terms for 2-Level Simplification, Kmap Simplification

Tuesday, January 18, 2022 11:26 AM

- Implicant : Any element of ON-set or DC-set , Implicate: Any element of OFF-set or DC-set
- Prime: largest group of implicants/implicates that are powers of
- Essential Prime: a Prime that alone covers an element of ON-set

<u>Def</u> Algorithm:

1) Find Prime Implicants
2) Filter only the Essential Prime Implicants to minimize size and terms
3) Create minterms or maxterms according to the Prime Implicants

<u>Note</u> Don't cares can form primes but not essential primes.

Specificaly: If a don't care can be used to form a larger group with another implicate, then it is included in a prime.

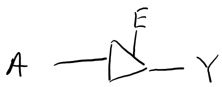
A don't care by itself will not form a prime.

ALU Elements

Tuesday, January 18, 2022 11:58 AM

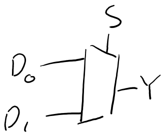
Basic Elements

- Tristate Buffer:



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

- 2:1 Multiplexer



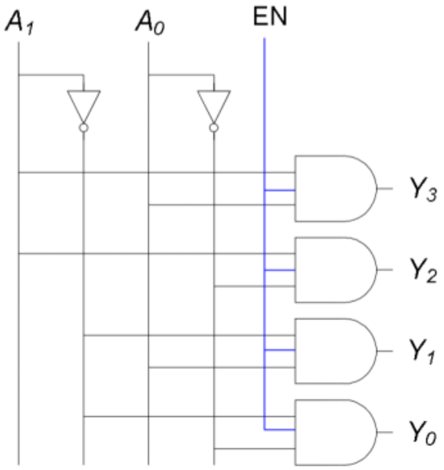
selects between 2 inputs

- N:1 Mux : selects between N inputs, $N = 2^k$

Idea We can use muxes as general purpose logic

- Demultiplexers: given an input x and selector inputs. Connects x to specific output based on selector. If EN is 0, then outputs are 0

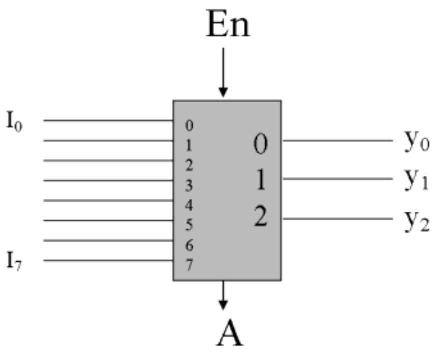
- Decoder: given inputs x, turns corresponding output 1 if x = output number used for addressing memory / peripherals, all other outputs are 0



can be used to implement general logic element with the same number of inputs

Each output implements a minterm, use additional gates to implement logic

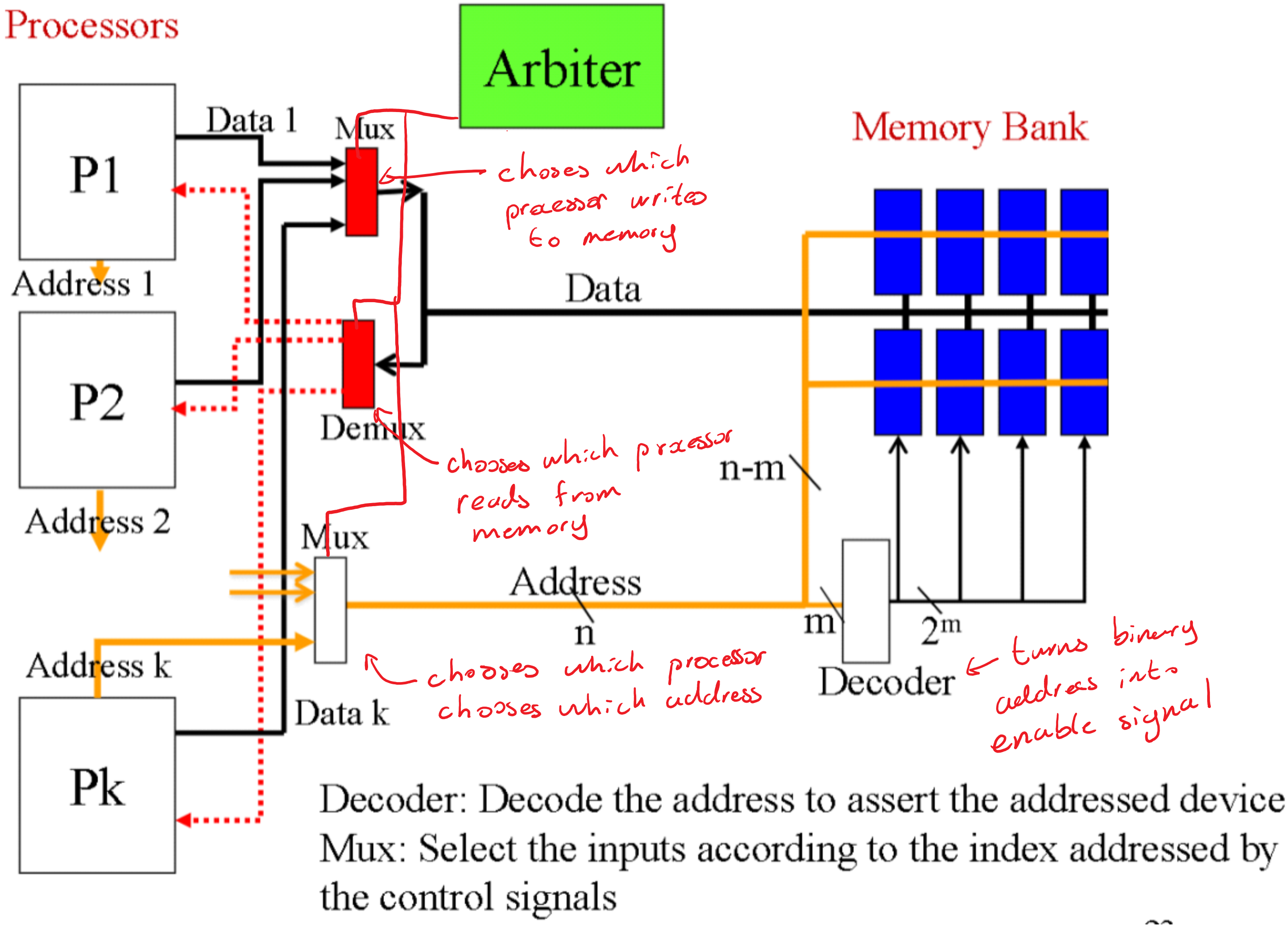
- Encoder: takes inputs and sets corresponding output to 1, else 0:



3 outputs

Basic Processor Design

Saturday, January 22, 2022 2:22 PM



choses which processor writes to memory

chooses which processor reads from memory

chooses which processor chooses which address

turns binary address into enable signal

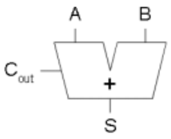
Decoder: Decode the address to assert the addressed device

Mux: Select the inputs according to the index addressed by the control signals

Adders (Ripple Carry, Carry Lookahead)

Saturday, January 22, 2022 2:25 PM

Half Adder

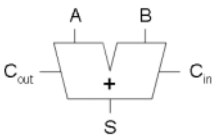


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C_{out} = AB$

Full Adder



C_{in}	A	B	C_{out}	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

$S = A \oplus B \oplus C_{in}$

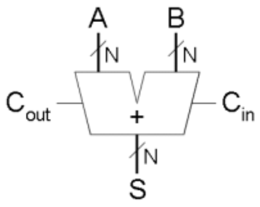
$C_{out} = AB + AC_{in} + BC_{in}$

Types of multi-bit adders

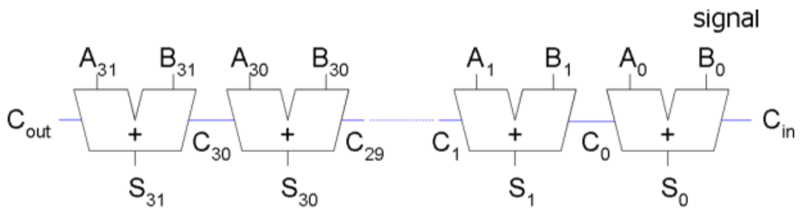
- Ripple-carry (slow)
- Carry-lookahead (faster)

Def N - number of bits to be added

Symbol



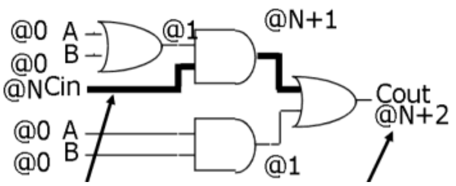
- Ripple Carry Adder: chains multiple FA together in a chain:



Disadvantage: SLOW!

Delay: $t_{ripple} = N \cdot t_{FA}$

where N is the number of bits and t_{FA} is the gate delay of Cout

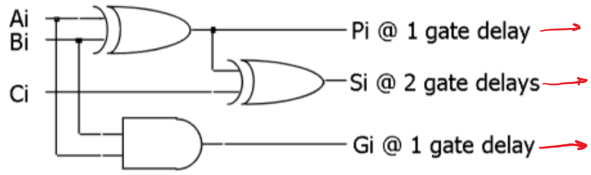


In this FA, the Cout gate delay is 2 gate delays,

So $t_{FA} = 2$ (gate delays)

and $t_{ripple} = N \cdot 2$(gate delays)

- Carry Lookahead Adder: Uses FA with propogate and generate inputs

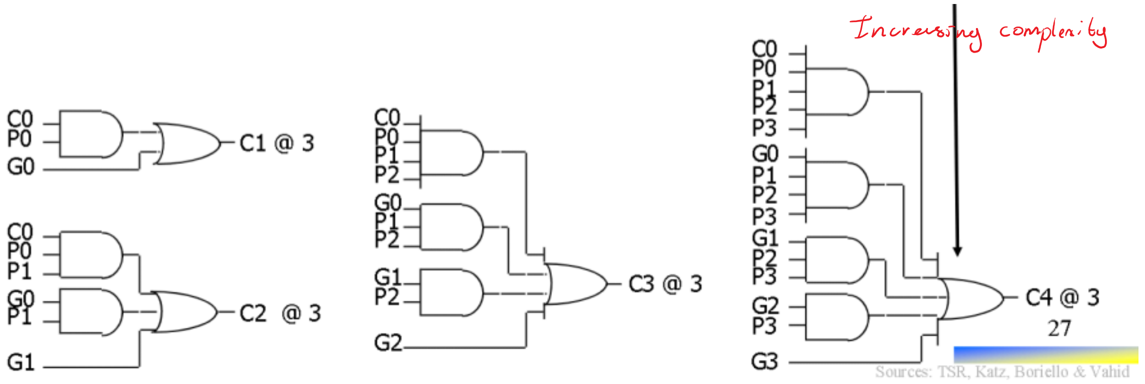


→ propogates previous carry term

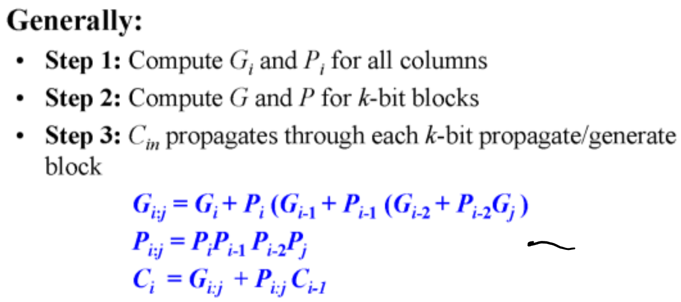
→ sum, takes 2 gate delays like before

→ generates carry for future FAs

Increasing complexity



Sources: TSR, Katz, Boriello & Vahid



Generally:

- **Step 1:** Compute G_i and P_i for all columns
- **Step 2:** Compute G and P for k-bit blocks
- **Step 3:** C_{in} propagates through each k-bit propagate/generate block

$$G_{i:j} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2}G_j)$$
$$P_{i:j} = P_iP_{i-1} P_{i-2}P_j$$
$$C_i = G_{i:j} + P_{i:j} C_{i-1}$$

Disadvantage: increasing complexity as N increases

Idea: We can combine ripple carry and carry lookahead adders to create a fast and small adder.

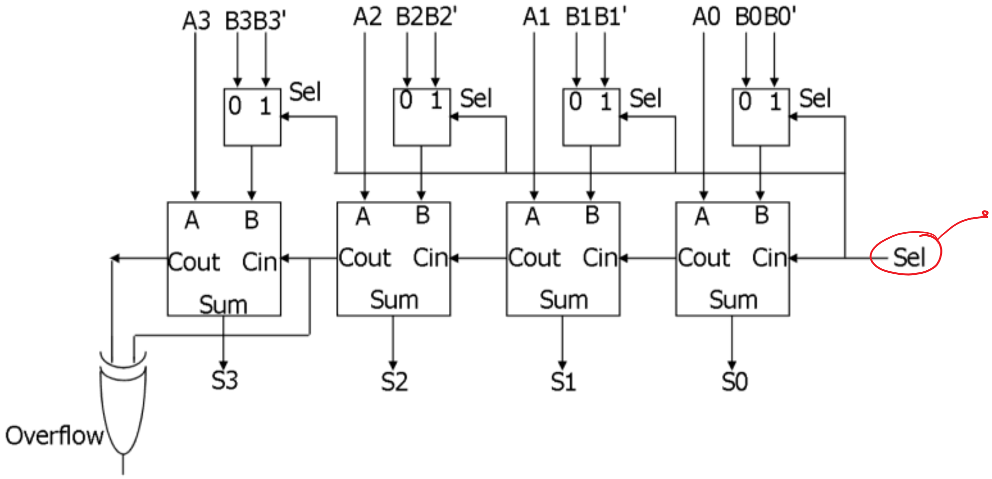
Subtractor (2's Complement)

Saturday, January 22, 2022 2:48 PM

Idea: We can implement subtraction using 2's complement.

if we can perform A + B,

then we can perform A - B with A + $\overline{B}$ + 1



when sel = 1, then we are performing subtraction

Detecting overflow: 1) If the two input sign bits are the same but result sign bit is different

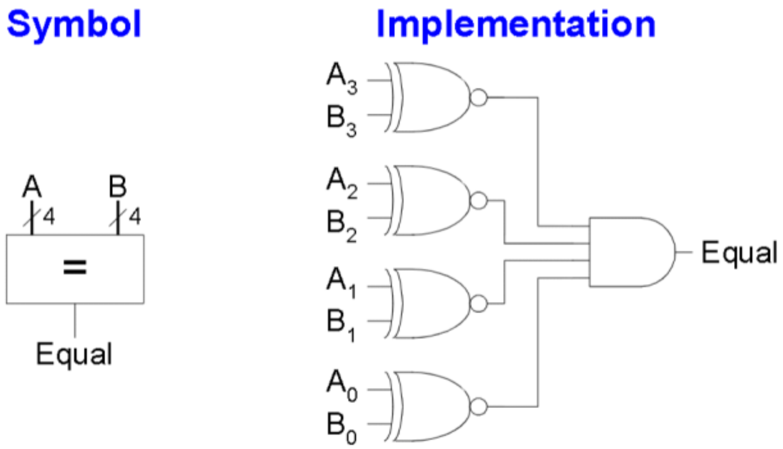
2) Difference between cin of sign bit and cout of sign bit

↳ simpler logic using 1 xor gate.

Comparators (Equal, Less Than, Greater Than)

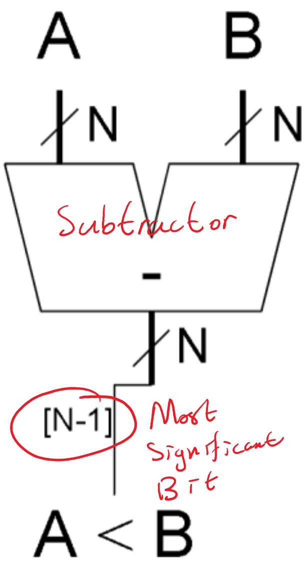
Saturday, January 22, 2022 3:01 PM

Equality Comparator:



Equal is only true when $A = B$

Less Than / Greater Than:



When output is 1, then $A < B$

when output is 0, then $A \geq B$

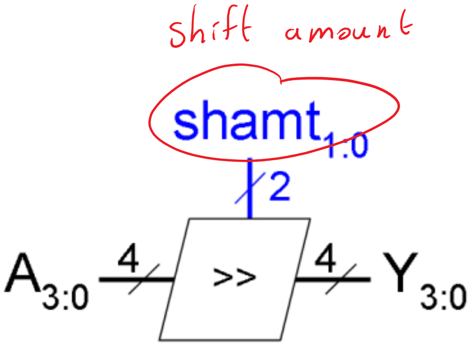
Shifters (Logical, Arithmetic, Rotator)

Saturday, January 22, 2022 3:05 PM

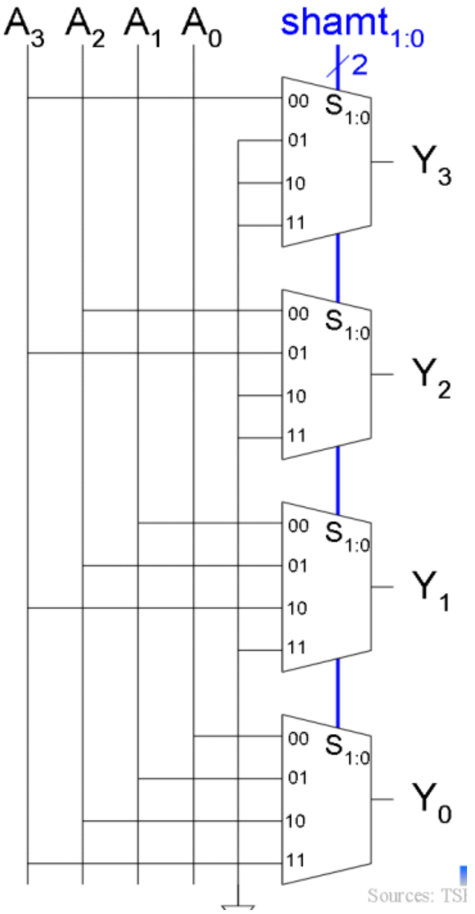
Logical Shifter: Fill empty spaces with 0

Arithmetic Shifter: On right shift, fill empty spaces with old MSB

Rotator: Wrap bits shifted off the end to new empty space


shift amount

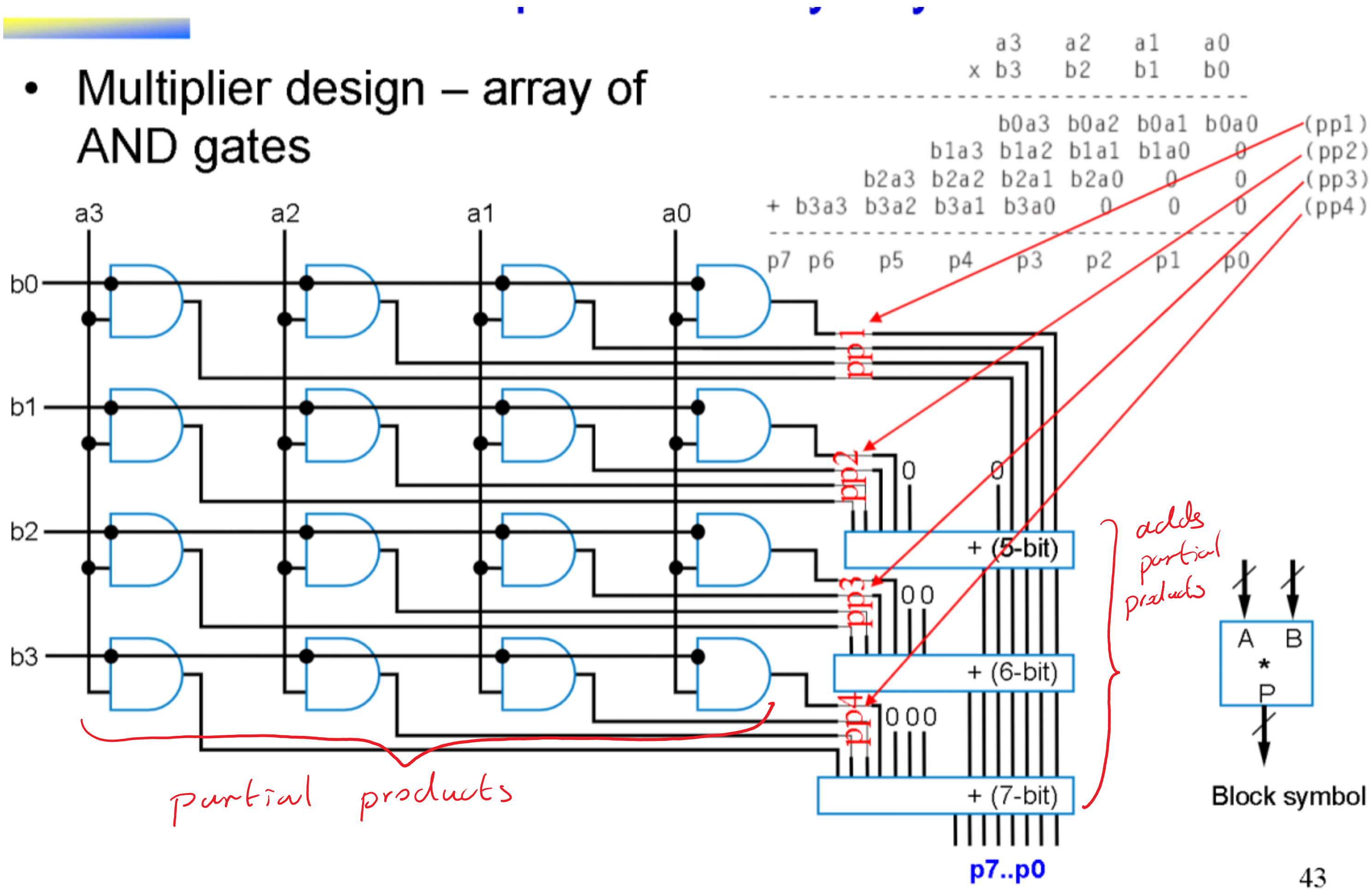
since we pad with 0, this is a Logical shifter.



Sources: TSI

Multipliers

Saturday, January 22, 2022 3:10 PM



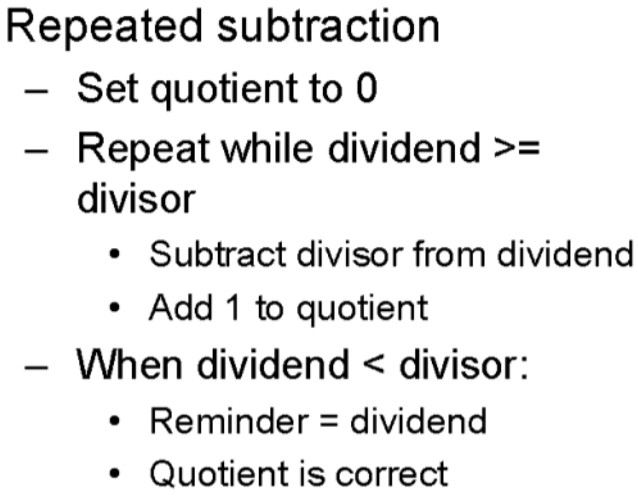
- Multiplier design – array of AND gates

```
                    a3   a2   a1   a0
                  x b3   b2   b1   b0
 -------------------------------------
                  b0a3 b0a2 b0a1 b0a0  (pp1)
             b1a3 b1a2 b1a1 b1a0    0  (pp2)
        b2a3 b2a2 b2a1 b2a0    0    0  (pp3)
 + b3a3 b3a2 b3a1 b3a0    0    0    0  (pp4)
 -------------------------------------
 p7 p6   p5   p4   p3   p2   p1   p0
```

43

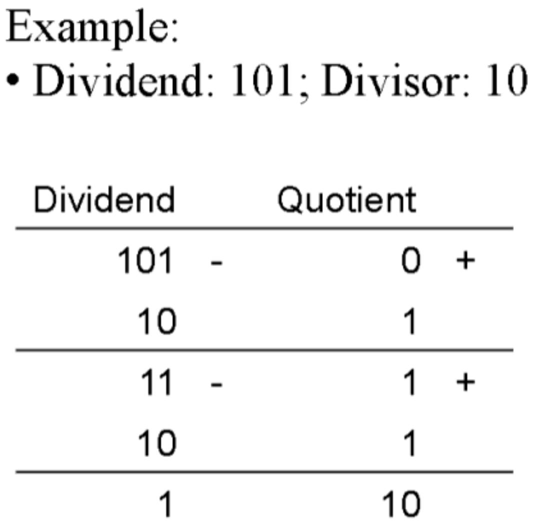
Dividers

Saturday, January 22, 2022 3:11 PM



Repeated subtraction

- Set quotient to 0
- Repeat while dividend >= divisor
  - Subtract divisor from dividend
  - Add 1 to quotient
- When dividend < divisor:
  - Reminder = dividend
  - Quotient is correct



Example:

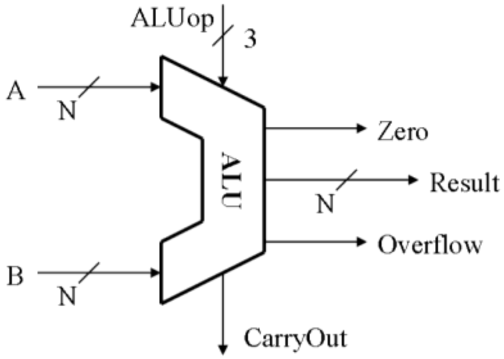
- Dividend: 101; Divisor: 10

Dividend		Quotient	
101	-	0	+
10		1	
11	-	1	+
10		1	
1		10	

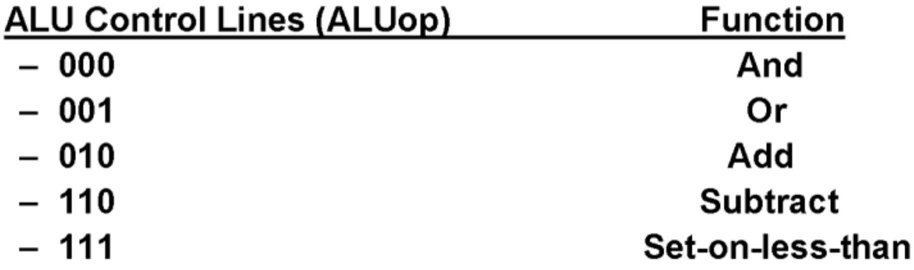
ALU Design

Saturday, January 22, 2022 3:12 PM

Block symbol:

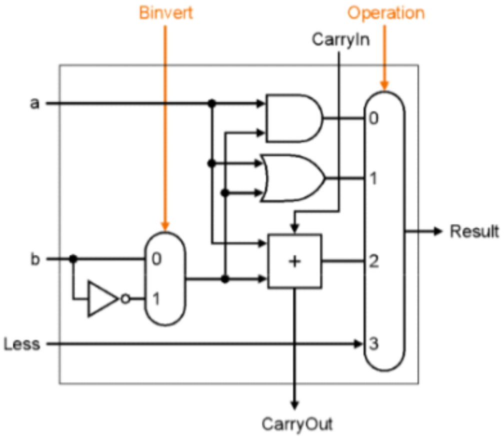


ALUop:

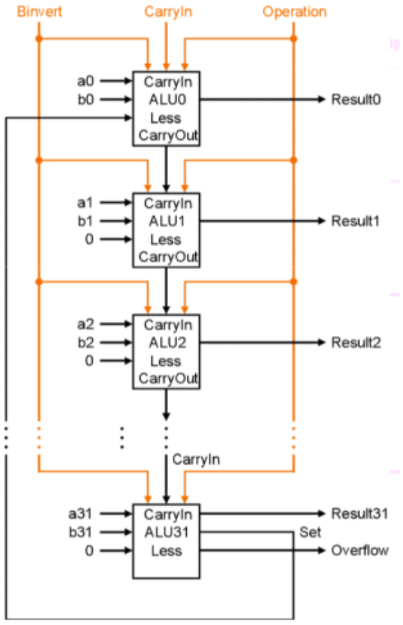


ALU Control Lines (ALUop)	Function
– 000	And
– 001	Or
– 010	Add
– 110	Subtract
– 111	Set-on-less-than

Implementation (1-bit):



Idea: We can chain together multiple 1-bit ALUs together to create larger ALUs capable of larger number operations.



Sequential Logic Components

Friday, February 4, 2022 6:00 PM

Idea: We want to create circuits to store data so that we can run a sequence of tasks.

Note: In computers: Registers → Cache → Memory → Disks

Def: often we want to use a clock cycle to control sequential logic:

A synchronous circuit is a sequential circuit with a clock

Period: time between clock pulse starts

Cycle: time interval in periods

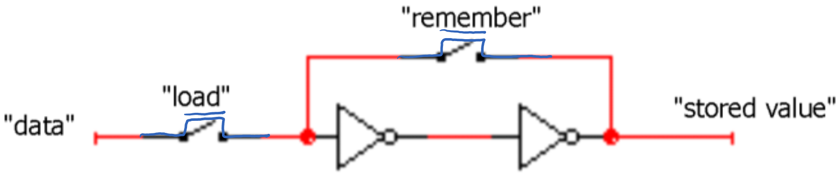
Duty Cycle: time spent in % that the clock is high

Frequency: 1/period

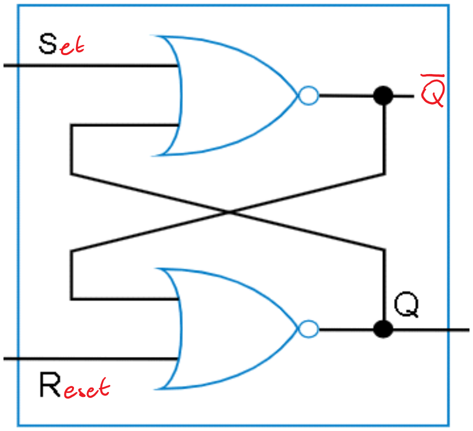
SRAM, SR Latch, D Latch

Friday, February 4, 2022 6:27 PM

SRAM: Simplest memory element, stores a value and recalls stored value



SR Latch: Set a value and hold until reset is activated

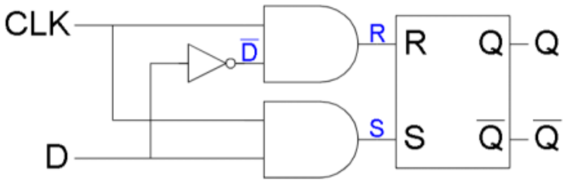


S	R	Q(t)	Q(t+∆)	
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	X	not allowed
1	1	1	X	

			S	
	0	0	X	1
Q(t)	1	0	X	1
		R		

characteristic equation
$Q(t+\Delta) = S + R' Q(t)$

D Latch: Solves the illegal state in SR Latches, but must be refreshed



CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$	
0	X	X	0	0	Q_{prev}	$\overline{Q}_{prev}$	← hold/memory
1	0	1	0	1	0	1	← reset
1	1	0	1	0	1	0	← set

The inverter and AND gates ensures R and S are never both 1

When clock goes high, it stores the value of D as Q

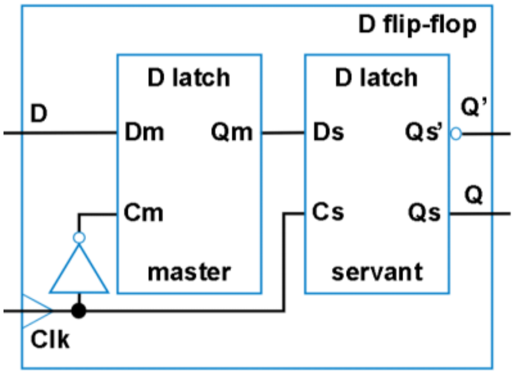
When clock goes low, it retains the previous value of Q

Summary! When clock = 1, D passes through Q (transparent)
When clock = 0, Q holds its previous value (opaque) (at the falling edge)

D Flip-Flop

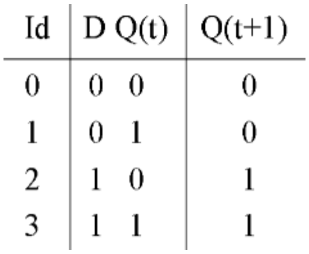
Friday, February 4, 2022 7:06 PM

D Flip-Flop:



Master latch loads when Clk = 0

Servant latch loads when Clk = 1



Id	D	Q(t)	Q(t+1)
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	1

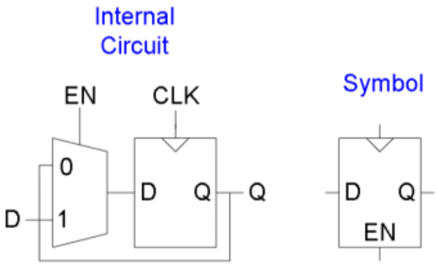
Thus for D Flip-Flop:

$Q(t+1) = D(t)$

The value of Q changes to D at every pos/neg clock edge. Otherwise it holds the value.

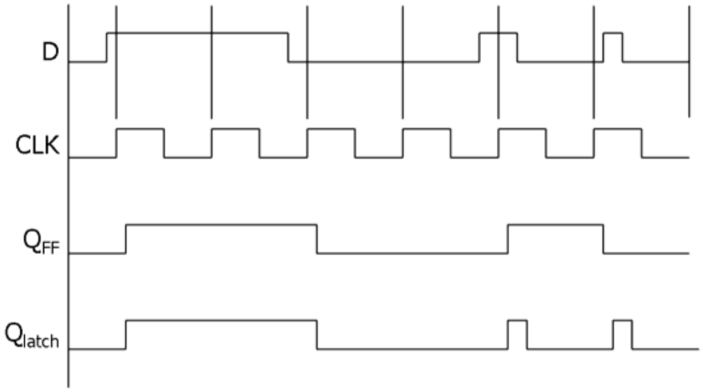
Adding an Enable:

- ***EN* = 1:** *D* passes through to *Q* on the clock edge
- ***EN* = 0:** the flip-flop retains its previous state



D Latch vs D Flip-Flip

Friday, February 4, 2022 7:42 PM



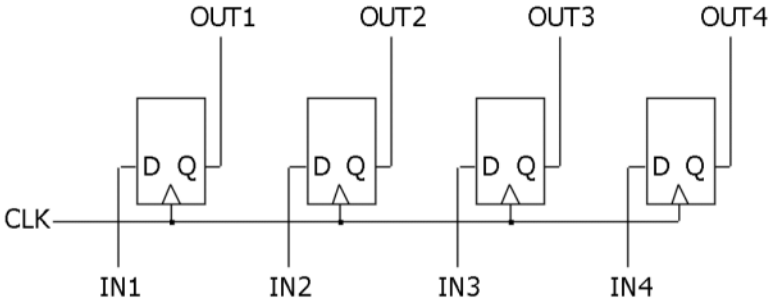
Flip Flop: takes D at ↑ and holds value until next ↑

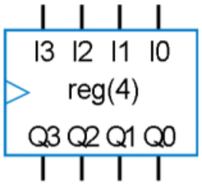
Latch: Copies D when CLK = 1, holds previous value when CLK = 0.

Basic Register, Shift Register, Counter

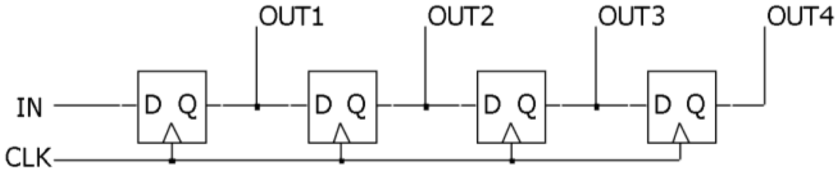
Friday, February 4, 2022 7:53 PM

Basic Register (4-bit): Holds values for one clock cycle

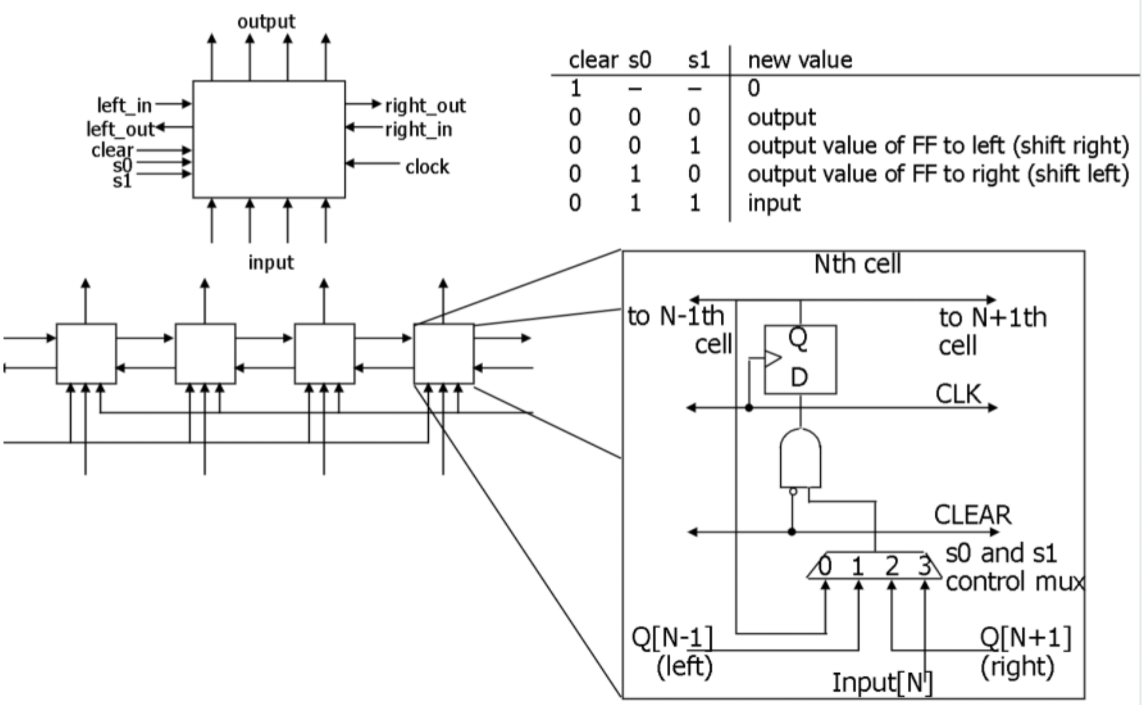




Shift Register: Holds and shifts values of consecutive samples

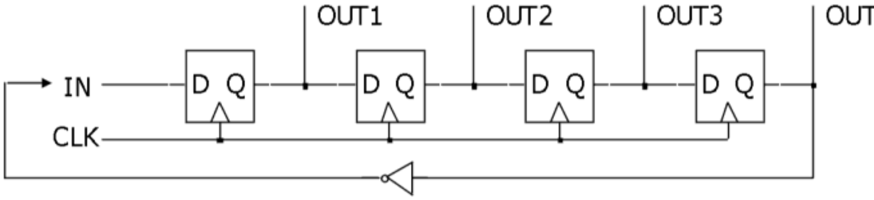


Universal Shift Register: Allows shifting in both directions



clear	s0	s1	new value
1	–	–	0
0	0	0	output
0	0	1	output value of FF to left (shift right)
0	1	0	output value of FF to right (shift left)
0	1	1	input

Counter: Sequences through a fixed set of patterns



Shifts right and inverts rightout as left in.

Finite State Machine

Friday, February 4, 2022 8:08 PM

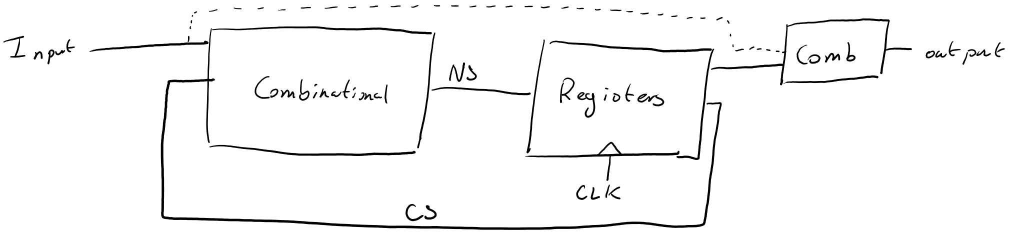
Idea: In combinational logic, we had no feedback.
In sequential logic, we have feedback and memory, which we can make finite state machines.

Def An FSM consists of:

- set of states
- set of inputs and outputs
- initial state
- set of transitions, only one can be true at a time

Notation: State Diagram: Graph where nodes are states and directed edges are transitions
State Table: Table showing each current state and next states
State Assignments: Binary representation of each state
Excitation Table: State Table but with state Assignments

Creating A Circuit: Excitation Table shows the combinational logic:



where NS is the next state
CS is the current state
Registers are D Flip-Flop(s)

Finding Combinational: Use k-maps, where each output / next state are outputs and inputs / current state are inputs.

Mealy Machine vs Moore Machine

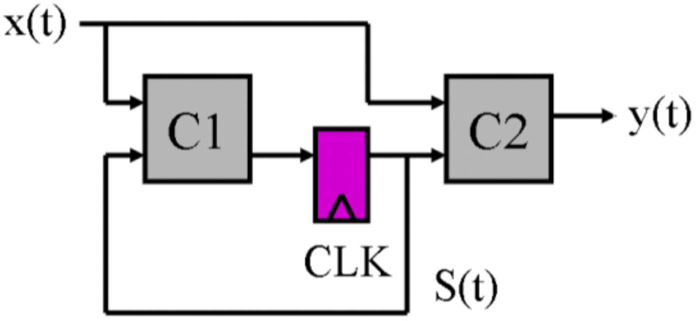
Friday, February 4, 2022 8:34 PM

Mealy Machine: $y_i(t) = f_i(X(t), S(t))$

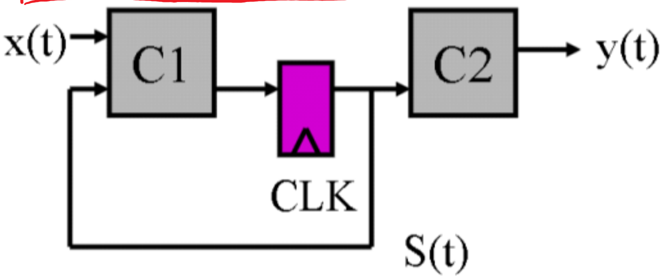
Moore Machine: $y_i(t) = f_i(S(t))$

Always:

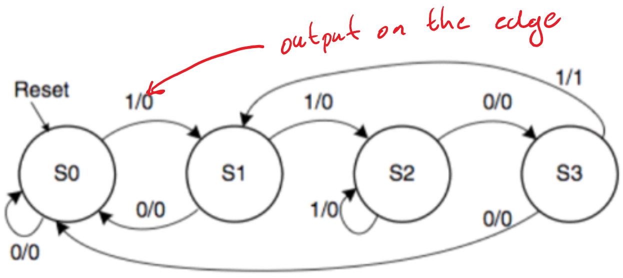
$$s_i(t+1) = g_i(X(t), S(t))$$

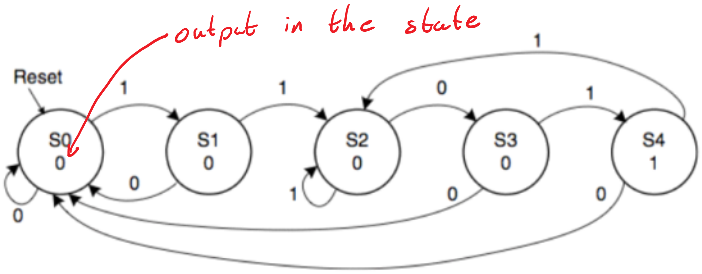


Mealy Machine



Moore Machine





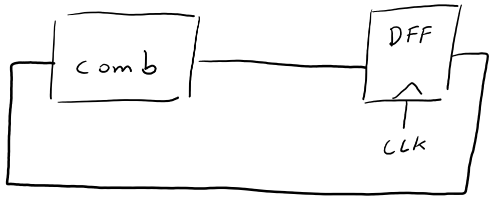
In Mealy Machine: Output is function of inputs and state

In Moore Machine: Output is function of state only

FSM Timing

Sunday, February 13, 2022 1:30 PM

Def Given the template FSM :



there are timing constraints on the FSM

Def : For combinational logic :

Min delay of a gate, also called contamination delay: t_{cd}

Minimum time from when an input changes until the output *starts* to change

Max delay of a gate, also called propagation delay: t_{pd}

Maximum time from when an input changes until the output *is* guaranteed to reach its final value (i.e., stop changing)

Def : For sequential logic :

On inputs : Setup time: t_{setup}

Time *before* the clock edge that data must be stable (i.e. not change)

Hold time: t_{hold}

Time *after* the clock edge that data must be stable

Aperture time: t_a

Time around clock edge that data must be stable ($t_a = t_{\text{setup}} + t_{\text{hold}}$)

On outputs : Min delay of FF, also called contamination delay or min CLK to Q delay: t_{ccq}

Time after clock edge that Q might be unstable (i.e., starts changing)

Max delay of FF, also called propagation delay or maximum CLK to Q delay: t_{pcq}

Time after clock edge that the output Q is guaranteed to be stable (i.e. stops changing)

Def : For clock signal :

The clock doesn't arrive at all registers at the same time

Skew: difference between the two clock edges

Perform the **worst case analysis**

Idea: when picking a clock speed

$$T_{clock} \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

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

High Level State Machine

Sunday, February 13, 2022 2:35 PM

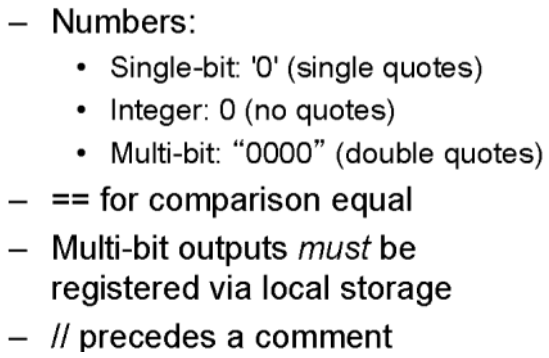
Idea Some behaviors may be too complex to describe using FSMs

Solution: Use a high level description for a theoretical FSM called High Level State Machine

Def A High Level State Machine extends FSM with:

- multiple bit input/outputs
- local storage
- arithmetic operations

Conventions:



- Numbers:
  - Single-bit: '0' (single quotes)
  - Integer: 0 (no quotes)
  - Multi-bit: “0000” (double quotes)
- == for comparison equal
- Multi-bit outputs *must* be registered via local storage
- // precedes a comment

RTL Design

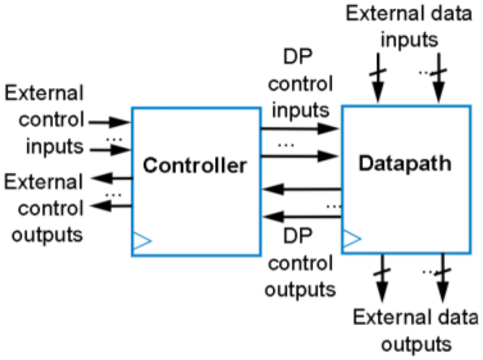
Sunday, February 13, 2022 2:52 PM

RTL design process :

Capture the behavior

Convert it to a circuit

- High-level architecture (datapath and control path)
- Datapath capable of HLSM's data operations
- Design controller to control the datapath



Steps : 1) Draw the HLSM

2) Determine what data will be stored and how it will be modified.

↳ Design the Datapath (Sequential Circuit)

3) Determine how to control the Datapath, what states have what control signals

↳ Design the Controller (FSM)

Def : A Data dominant design : extensive datapath, simple controller

A Controll dominant design : simple datapath, extensive controller

Converting C to HLSM

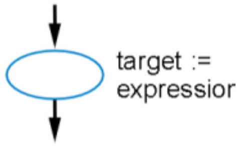
Saturday, February 26, 2022 3:58 PM

Idea: Convert each C construct to equivalent states and transitions.

Assignment statement

- Becomes one state with assignment

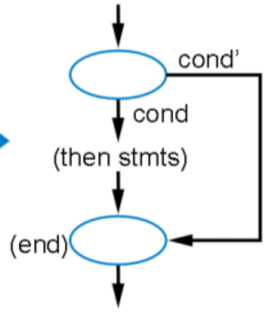
```
target = expression;
```



If-then statement

- Becomes state with condition check, transitioning to “then” statements if condition true, otherwise to ending state
  - “then” statements would also be converted to states

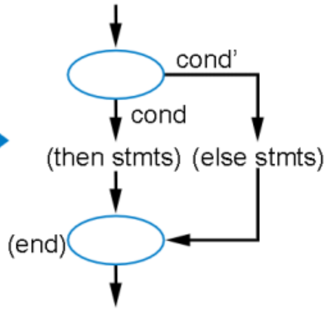
```
if (cond) {
  // then stmts
}
```



If-then-else

- Becomes state with condition check, transitioning to “then” statements if condition true, or to “else” statements if condition false

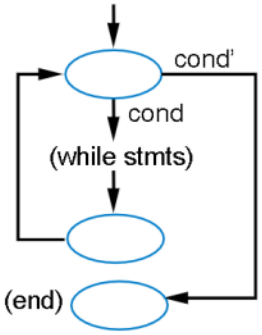
```
if (cond) {
  // then stmts
}
else {
  // else stmts
}
```



While loop statement

- Becomes state with condition check, transitioning to while loop’s statements if true, then transitioning back to condition check

```
while (cond) {
  // while stmts
}
```

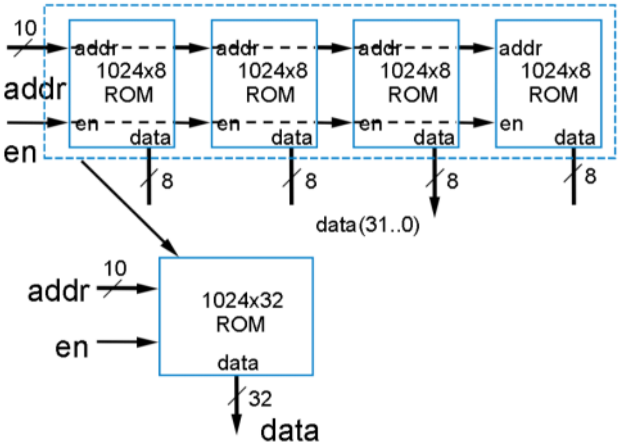


Memory, RAM vs ROM

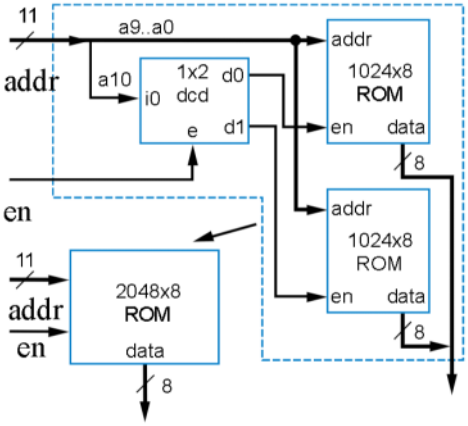
Saturday, February 26, 2022 4:23 PM

Idea: Stores a large number of bits
has r/w to select read or write
has enable to read/write when addressed
can have multiport: multiple accesses to different addresses simultaneously

Combining: Wider words: connect memory in parallel:



Combining: More memory: connect memory in parallel with decoder:



Def Traditionally: ROM is read-only, bits stored without power
RAM is dynamic, bits lost without power

Lately, distinctions blurred: Advanced ROMs can be written to (EEPROM)
Advanced RAMs can hold bits without power (NVRAM)

RAM, Types of RAM

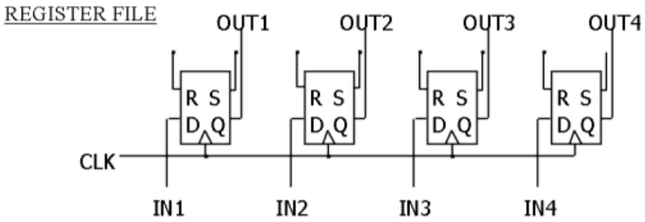
Saturday, February 26, 2022 4:44 PM

Types of RAM:

Memory is Volatile

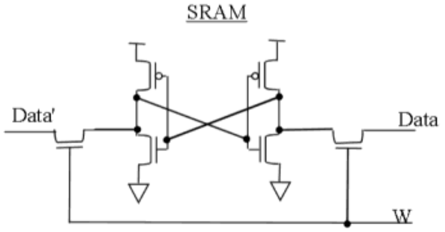
Register file
- Fastest
- But biggest size

Used in registers



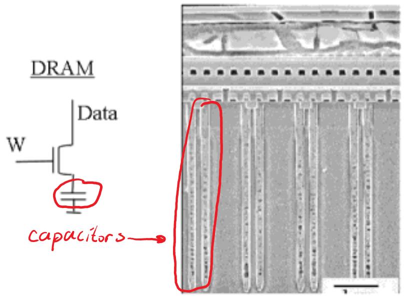
SRAM
- Fast (e.g. 10ns)
- More compact than register file

Used in cache



DRAM
- Slowest (e.g. 20ns)
  - And refreshing takes time
- But very compact
- Different technology for large caps.

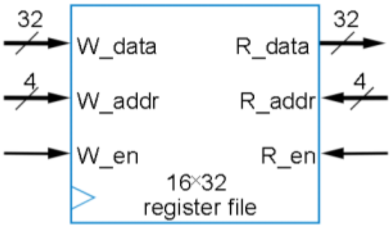
Used in off-chip RAM



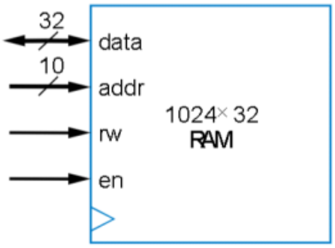
Def Block symbol and characteristics of RAM:

RAM – Readable and writable memory

- Logically the same as register file
  - RAM just one port; register file two or more
- RAM vs. register file
  - RAM is larger
  - RAM stores bits using a bit storage vs. FFs
  - RAM implemented on a chip in a square – keeps longest wires (hence delay) short



Register file



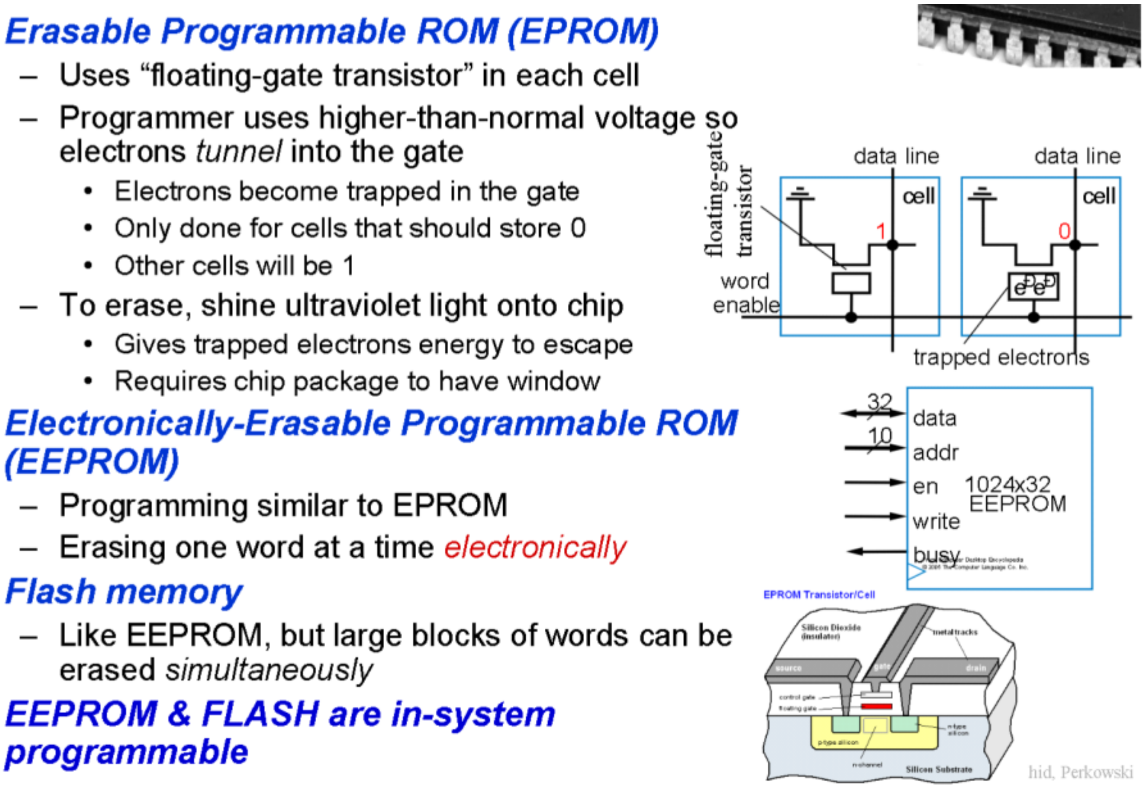
RAM block symbol

ROM, Traditional ROMs

Saturday, March 5, 2022 12:40 PM

Idea: We want to store data without having to refresh the bits
We can use ROMs

Def: Traditional ROMs use quantum tunneling to trap data in a floating gate. Use a strong voltage to free electron.



Erasable Programmable ROM (EPROM)

- Uses "floating-gate transistor" in each cell
- Programmer uses higher-than-normal voltage so electrons *tunnel* into the gate
  - Electrons become trapped in the gate
  - Only done for cells that should store 0
  - Other cells will be 1
- To erase, shine ultraviolet light onto chip
  - Gives trapped electrons energy to escape
  - Requires chip package to have window

Electronically-Erasable Programmable ROM (EEPROM)

- Programming similar to EPROM
- Erasing one word at a time *electronically*

Flash memory

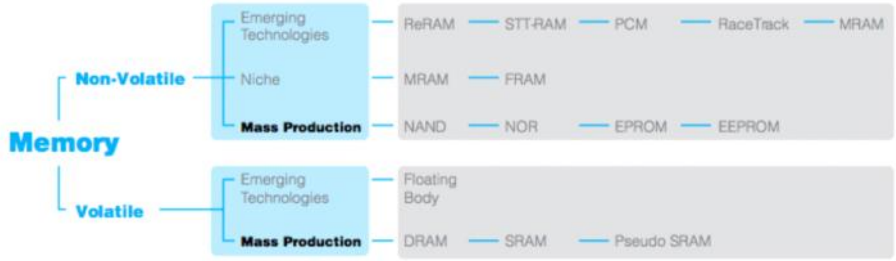
- Like EEPROM, but large blocks of words can be erased *simultaneously*

EEPROM & FLASH are in-system programmable

hid, Perkowski

Modern/New Memory Types

Saturday, March 5, 2022 12:46 PM



FeRAM: stores data in ferroelectric material.
No refresh, similar performance to DRAM but destructive reads

STT-RAM: Stores data in tunneled electrons. Value determined by electron spin.
High endurance, fast reads but high write energy

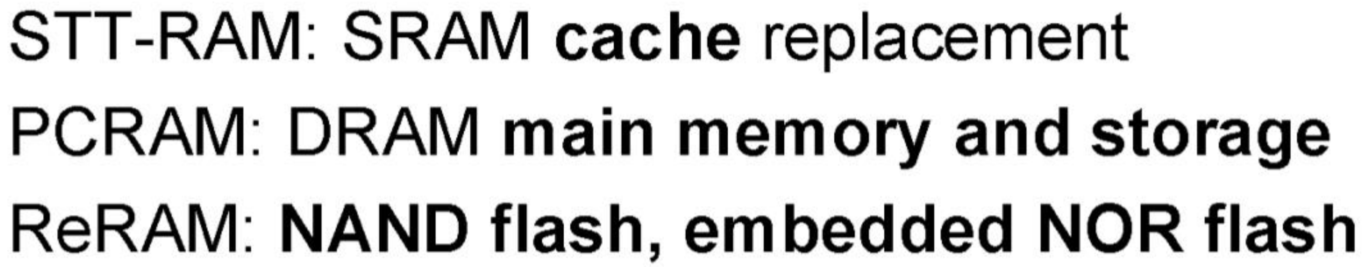
PCM: Stores data by changing state of material using a heater.
Good scalability but slow writes, low endurance.

ReRAM: Stores data by changing the resistance of a semiconductor.
Fast read/write, high density but limited endurance

Comparison of New vs Traditional Memories

Saturday, March 5, 2022 12:58 PM

Comparison of Traditional vs New memories:

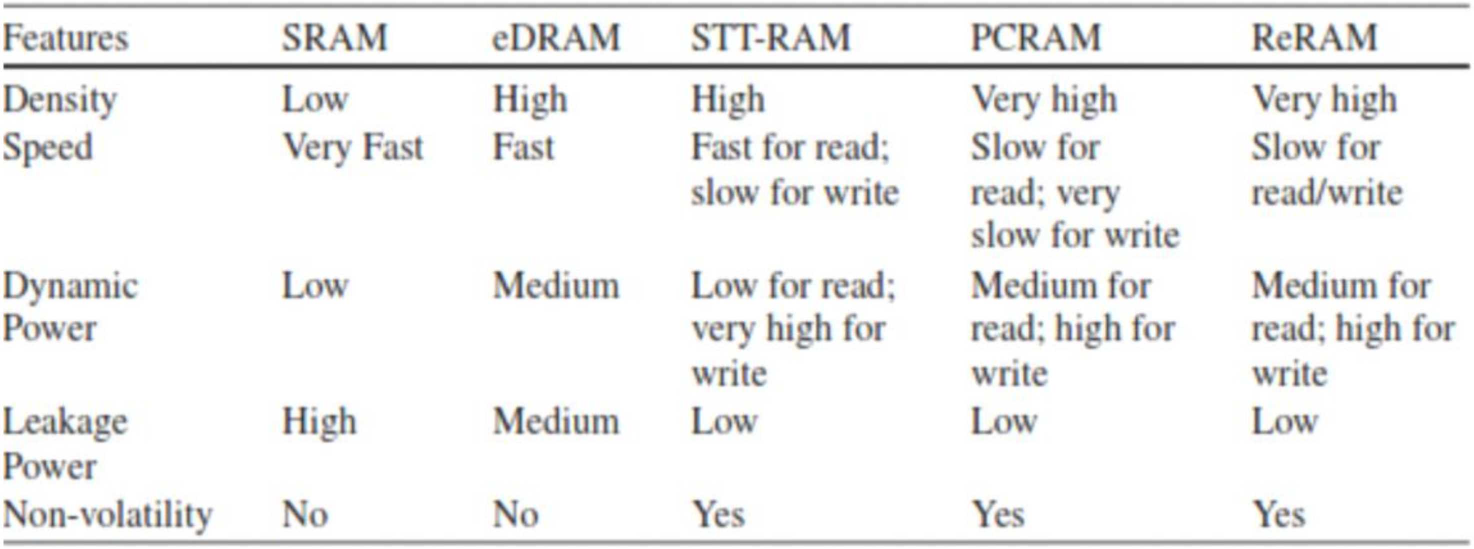


STT-RAM: SRAM **cache** replacement

PCRAM: DRAM **main memory and storage**

ReRAM: **NAND flash, embedded NOR flash**

↳ also for near-memory compute



Features	SRAM	eDRAM	STT-RAM	PCRAM	ReRAM
Density	Low	High	High	Very high	Very high
Speed	Very Fast	Fast	Fast for read; slow for write	Slow for read; very slow for write	Slow for read/write
Dynamic Power	Low	Medium	Low for read; very high for write	Medium for read; high for write	Medium for read; high for write
Leakage Power	High	Medium	Low	Low	Low
Non-volatility	No	No	Yes	Yes	Yes