Lectures 3-5 – 1/24/11 – 1/28/11

- Announcements
  - Hwk 1 posted, due Thursday 2/3 in your discussion section
- Last Week (P&P 1)
  - Some basic concepts in computer systems
- This Week's Lectures (P&P 3.1-3)
  - Boolean Logic
  - The mighty transistor
  - Circuits
- Next Week's Lectures (P&P 3.4-6)
  - Storage, sequential circuits & clocks

Boolean Algebra

NOT

a	$\overline{a}$
0	1
1	0

OR

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

AND

a	b	a*b
0	0	0
0	1	0
1	0	0
1	1	1

- Combinations:
  - NOT (A OR B) aka A NOR B
  - …
- Identities:
  - $X+0 = X$, $X+1 = 1$, $X*0 = 0$, $X*1 = X$, $X+\overline{X} = 1$, $X*\overline{X} = 0$, $X+X = X$, $X*X = X$ $\overline{\overline{X}} = X$
- Commutative, associative & distributive
  - $A + B = B + A$
  - $A + (B + C) = (A + B) + C$
  - $A * (B + C) = (A * B) + (A * C)$

DeMorgan's Laws

$A+B = \overline{\overline{A} \cdot \overline{B}}$

A	B	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$	$\overline{\overline{A} \cdot \overline{B}}$	A+B
0	0	1	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	0	1	1

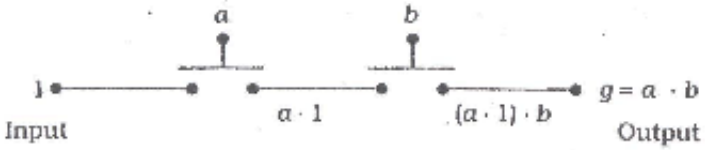
A*B =

Why do we care?

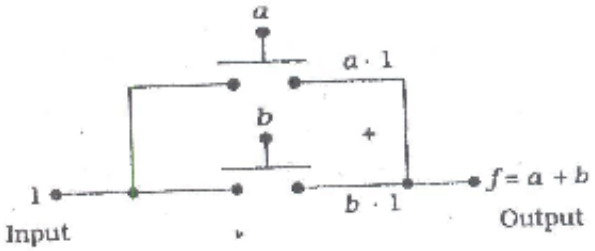
Controlled Switches 1

- Assert high controlled switch

y = x * A (path iff A = 1)
(path is underfined when A = 0)



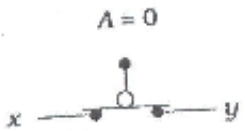
(path only when a = b = 1)



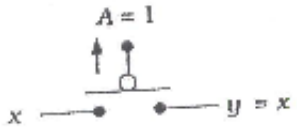
(path only when a = 1 or b = 1)

Controlled Switches 2

- Assert low controlled switch

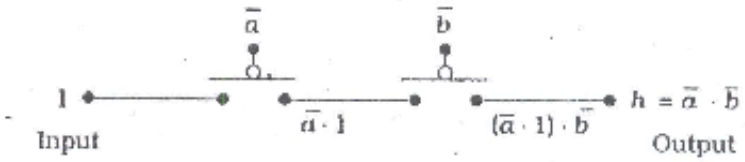


(a) Closed



(b) Open

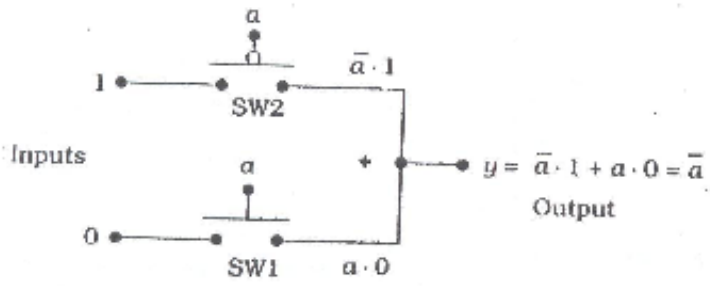
= x * $\overline{A}$ (path iff A = 0)



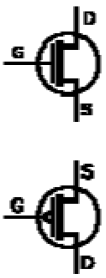
(path only when a = b = 0)

Controlled Switches 3

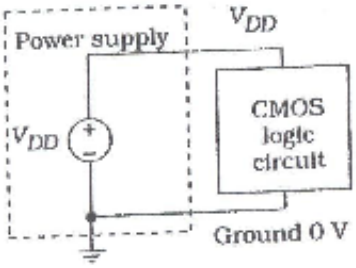
- Output is defined for some, but not all, cases for an individual assert-high or an individual assert-low controlled switch
- Combine them (using the two-stone theory) to create an implementation of the NOT Boolean function
  - Output is defined for all cases!

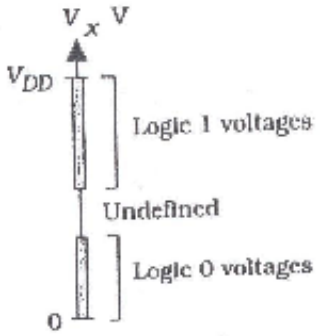


Enter The Mighty Transistor 1



- An electronic switch
  - Behaves like a controlled switch
    - n-Type is the analog of an assert high controlled switch
    - p-Type is the analog of an assert low controlled switch
  - Three terminals
    - Gate (G) controls flow of electrons between Source (S) and Drain (D)
  - Powered by voltage
    - Vdd (positive) is the analog of a "1" in a controlled switch
    - Gnd (0) is the analog of a "0" in a controlled switch

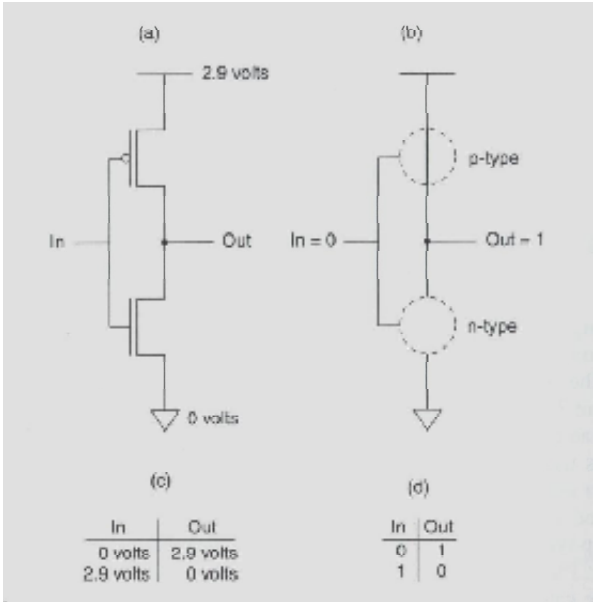


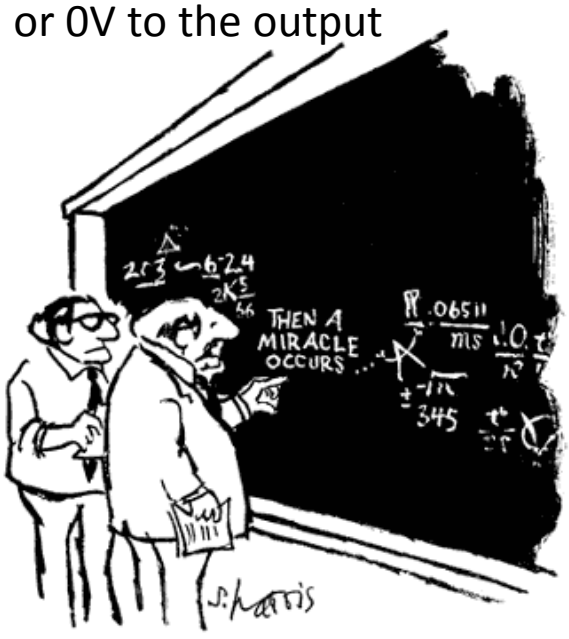


5V → 3.5V → 1.1V

Enter The Mighty Transistor 2

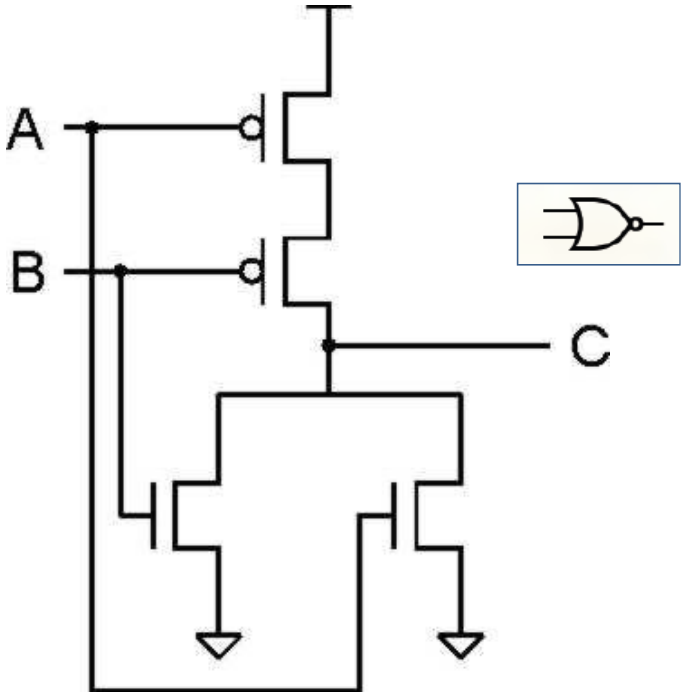
- An n-type transistor passes strong logic 0 and weak logic 1
- A p-type transistor passes strong logic 0 and weak logic 0
- We use them in pairs to divert Vdd or 0V to the output

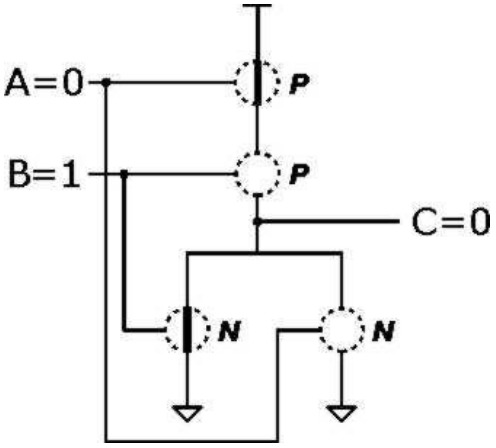




"I think you should be more explicit here in step two."

NOR Gate (NOT OR)

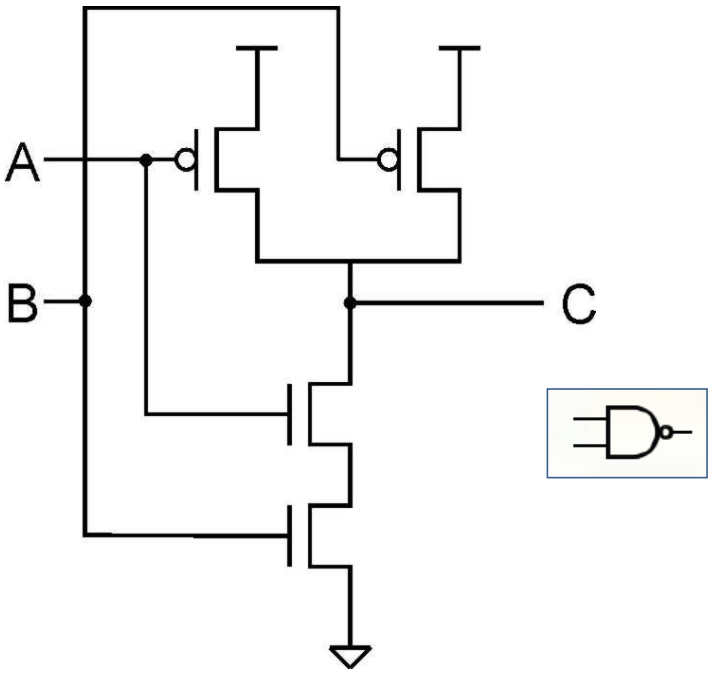


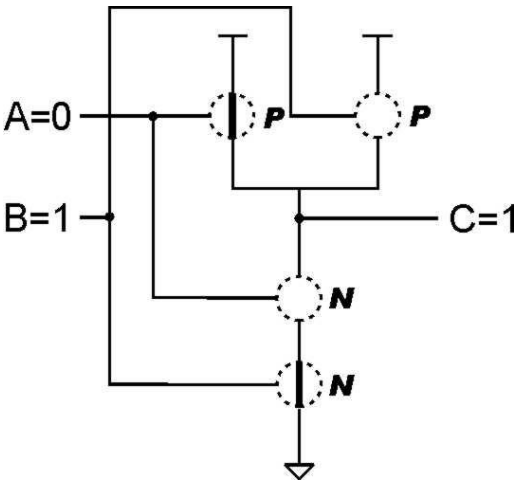


A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Note: Serial structure of p-types, parallel of n-types.

NAND Gate (NOT AND)





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

Note: Parallel structure of p-types, serial of n-types.

How About…

- AND & OR gates?
- N-input gates (N > 2)?
- Circuits that implement a given function F of K inputs $F(in_1, in_2, \ldots, in_K)$?

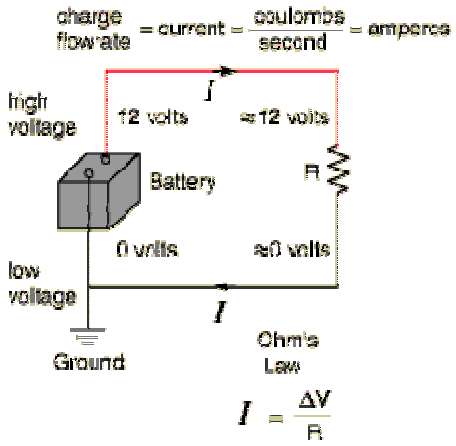
CMOS Circuits

- Complementary Metal Oxide Semiconductor
  - Technology for constructing Integrated Circuits (ICs)
  - High noise immunity
  - Low static power consumption (when not switching)
- P-type
  - Attached to + voltage (Vdd)
  - Pulls output voltage UP when input (G) is 0
- N-type
  - Attached to 0V (Gnd)
  - Pulls output voltage DOWN when input (G) is 1
- For all inputs output must be connect to Vdd or Gnd but not both!
  - Duality
  - Define one in terms of the NOT of the other
  - Parallel P-types require N-types in series (DeMorgan)

See: http://tams-www.informatik.uni-hamburg.de/applets/cmos/cmosdemo.html
for animation of various CMOS simple circuits

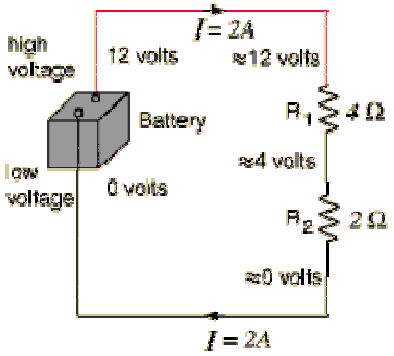
So, How Do Transistors Work Anyway?

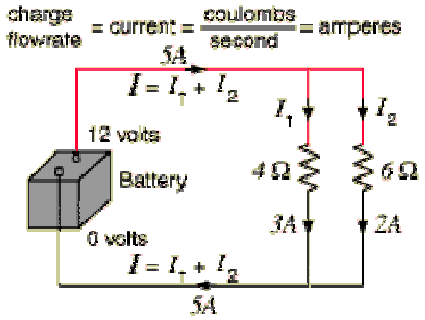
- Current (I) flows through a conductor
  - (In the form of electrons)
  - (Which flow in the opposite direction to the current)
- Voltage (V) is the "electrical pressure" pushing the electrons
- Resistance (R) is what prevents electron flow
  - Resistance value is a property of the conductor
- Ohm's Law: V = I*R



More of Ohm's Law

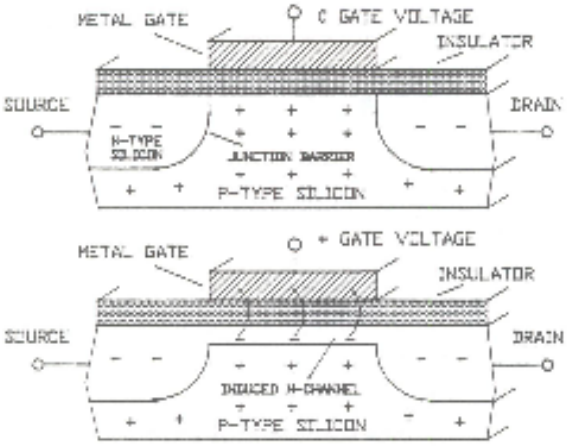
- Resistor networks are possible
  - In series: R = R1 + R2
  - In parallel: 1/R = 1/R1 + 1/R2





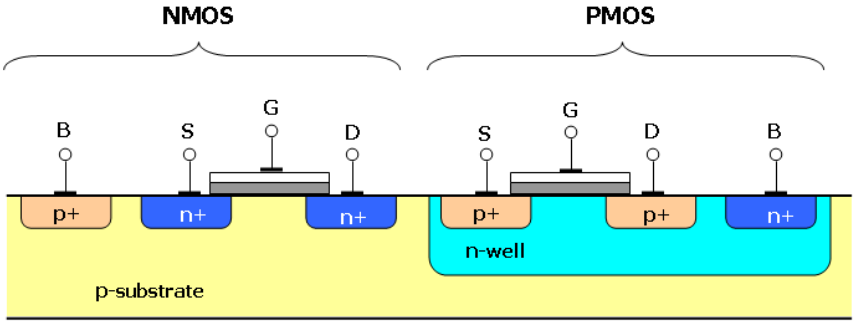
See http://hyperphysics.phy-astr.gsu.edu/hbase/electric/watcir2.html for a hydraulic analogy explaining Ohm's law

So, What Do Transistors Look Like?

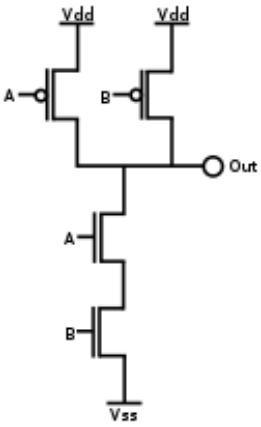


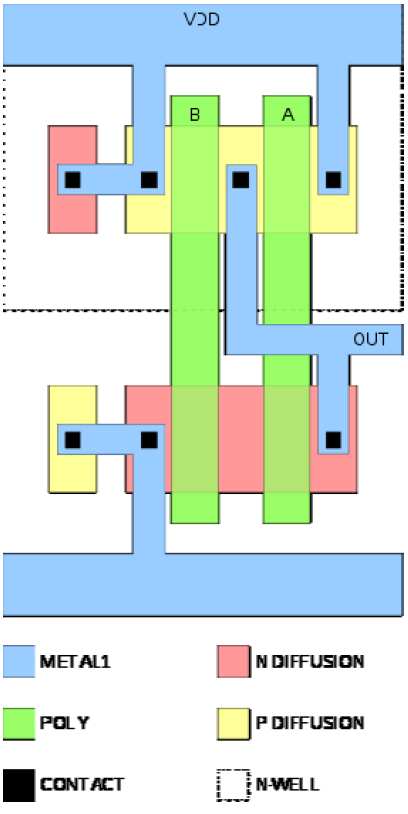
0V @G: nothing happens because the Junction Barrier prevents movement of electrons between S and D (it *resists*)

Vdd @G: the electrical field across G floods the P-type silicon under the insulator with negative charge creating an "induced" N channel for *conduction*



More Specifically, A NAND Gate





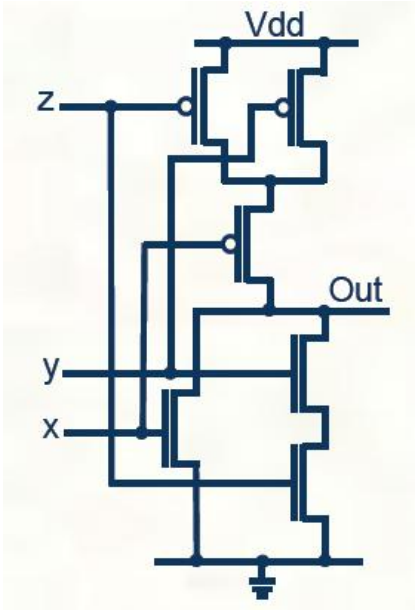
More Complex Circuits

- Want to implement functions of arbitrary complexity
- Procedure:
  - Truth table
  - Sum of products expression
  - Minimize

X	Y	Z	Out
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Two Implementations

- Expression: (X + (Y*Z))'
- Option 1: Multiple basic gates



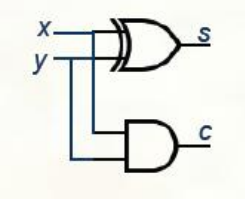
- Option 2: Custom implementation

X	Y	Z	Out
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Half Adder

- 1-bit addition: X+Y
- Two outputs: S (Sum), C ($Carry_{out}$)
  - S = 1 iff one input == 1
  - S = XY' + X'Y (X XOR Y = X ⊕ Y)
  - C = 1 iff both inputs == 1
  - C = XY
- What if we want a 2-bit adder?
  - Need a $carry_{in}$

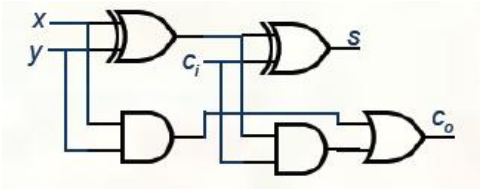
X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full Adder (1-bit)

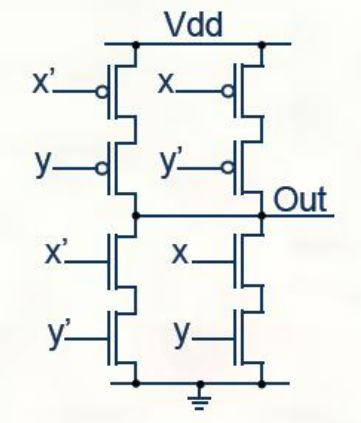
- Includes the C_i bit as an input
- S = 1 iff an odd number of inputs == 1
- $S = X'YC_i' + XY'C_i' + XYC_i + X'Y'C_i$
  $= C_i'(X'Y + XY') + C_i(XY + X'Y')$
  $= C_i'(X'Y + XY') + C_i(X'Y + XY')'$
  $= C_i \oplus (X \oplus Y)$
- C_o = 1 iff at least two of the inputs == 1
- $C_o = X'YC_i + XY'C_i + XYC_i' + XYC_i$
  $= C_i(X'Y + XY') + XY(C_i' + C_i)$
  $= C_i(X \oplus Y) + XY$

X	Y	C_i	S	C_o
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



Full Adder Realization 1

- Implement an XOR gate and use it



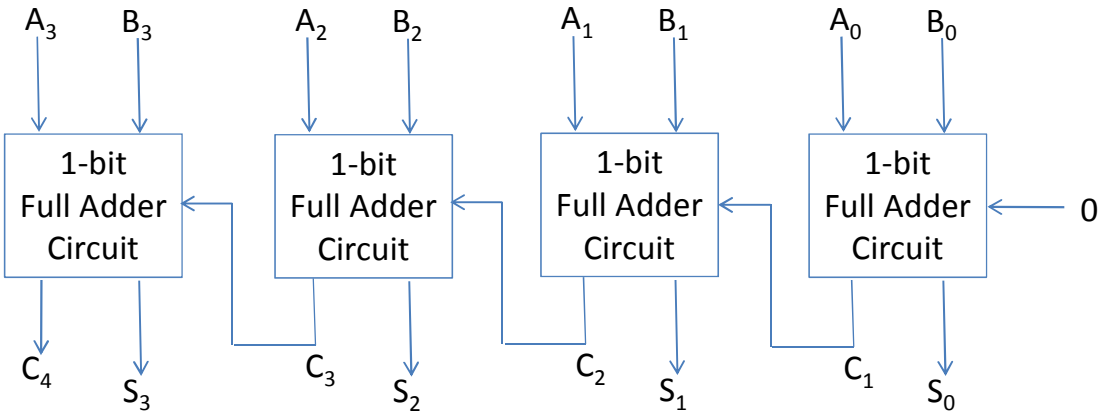
- S requires two inverters (for X' and Y') and two XOR gates
- C_o requires an XOR, one AND and one OR

Full Adder Realization 2 & 3

2. Implement a Half Adder and re-express the original equations in terms of Half Adders
3. Manipulate the original equations to use "basic gates"
   - Including some with more than two inputs

Adding Two n-Bit Numbers

- String along several 1-bit Full Adders to form a *Ripple-Carry* Adder

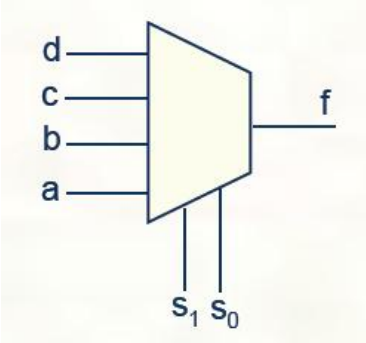


A 4-bit ripple carry adder

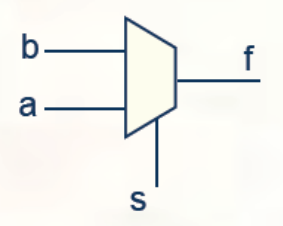
See: http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/20-arithmetic/10-adders/chapter.html

Multiplexor

- A Multiplexor (selector, mux) selects one of n inputs to be output
- Used how?
  - Select one of four registers as input to the adder (for example)



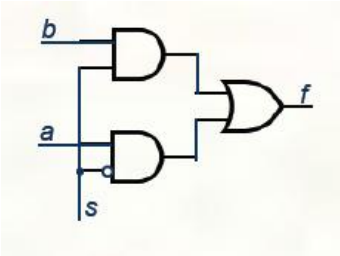
- Let's make it easy and start with a 2-input mux
  - When s == 0 f = a
  - When s == 1 f = b



2-Input Multiplexor Design

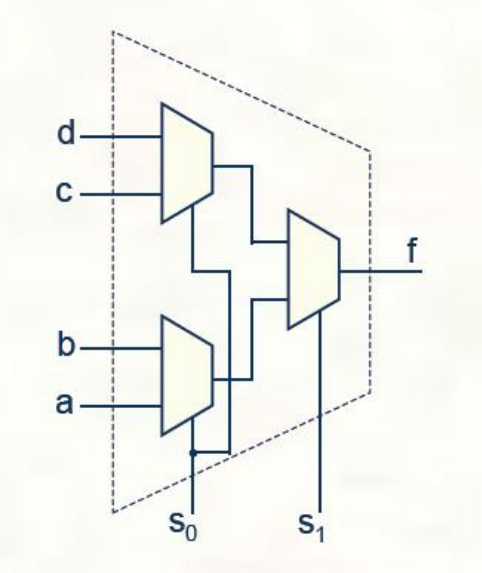
s	a	b	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

f = s'ab' + s'ab + sa'b + sab
= s'a(b' + b) + sb(a' + a)
= s'a + sb



Making Larger Muxes

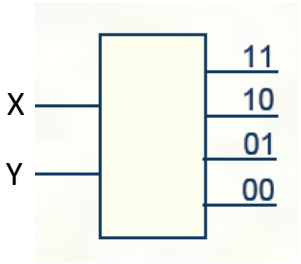
- Design them from scratch, or just glue together smaller muxes

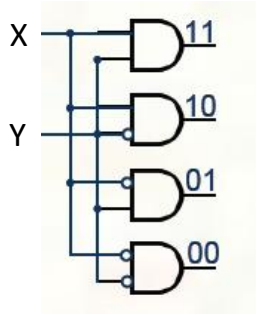


Decoder

- Decode an n-bit value to set one of 2^n output lines

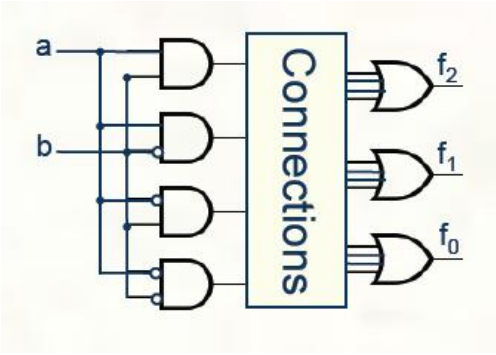
X	Y	00	01	10	11
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1





Programmable Logic Arrays (PLAs)

- Since all boolean functions can be expressed in sum of products form, we can implement a set of n input functions systematically in hardware using a PLA



- An n input function needs a PLA with:
  - 2^n n-input AND gates
  - k OR gates, k = number of output columns in the truth table
  - Connect the output of an AND gate to an OR gate if the corresponding row of the truth table produces an output of 1

Logical Completeness

- We can implement any logic function using AND, OR and NOT gates
  - {AND, OR, NOT} is therefore logically complete
- {AND, NOT} is logically complete
  - By DeMorgan's Laws we can express OR using {AND, NOT}
  - A+B = (A' * B')'
- {NAND} is logically complete