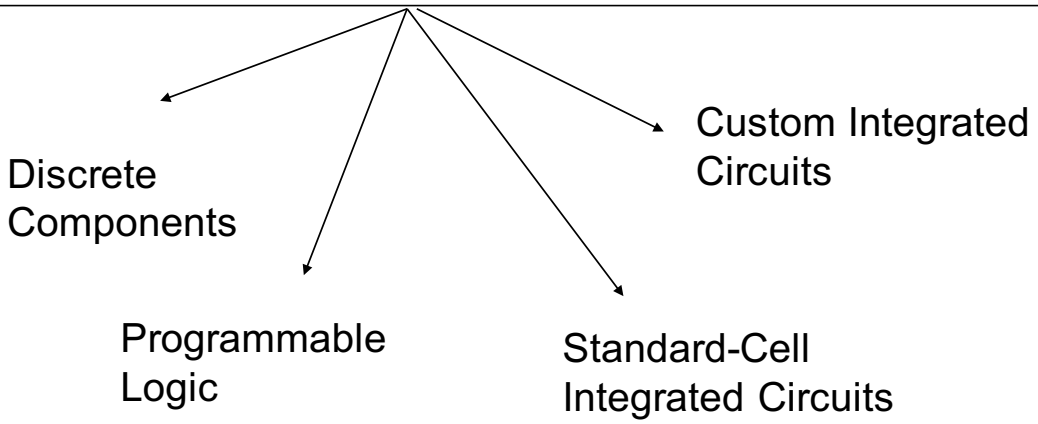
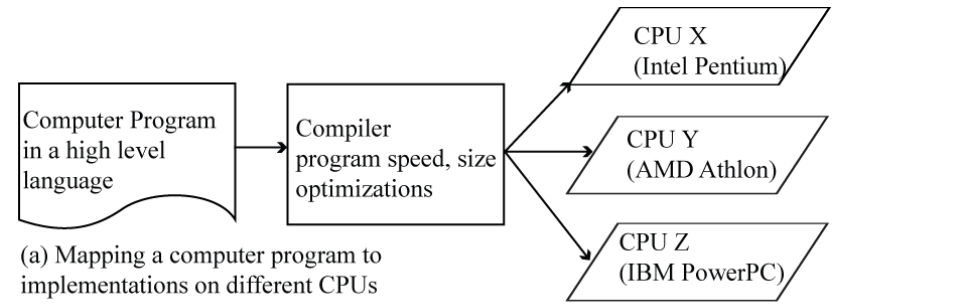
Technology Mapping

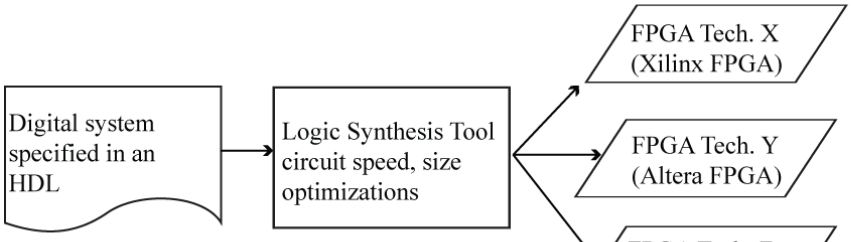
Technology mapping transforms one logic circuit model into another one. *What is the difference between this and synthesis*?



Compiler/Programmable Logic Comparison

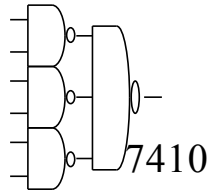


(a) Mapping a computer program to implementations on different CPUs



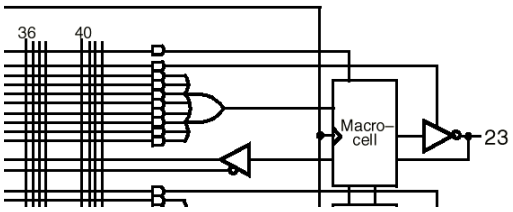
(b) Mapping a digital system to implementations using different FPGA technologies

$$F = AB + BC + AC$$

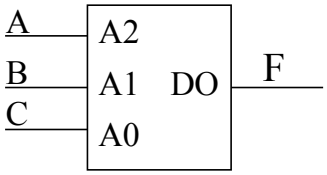


7400

22V10 PLD

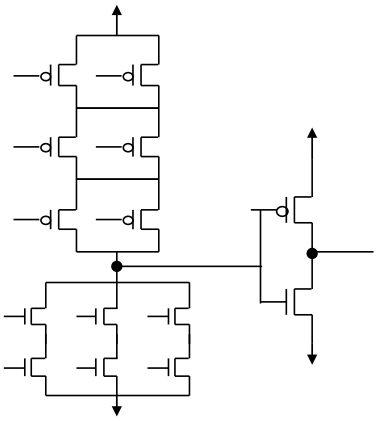


8 x 1 RAM



Lookup Table

Static CMOS Gate



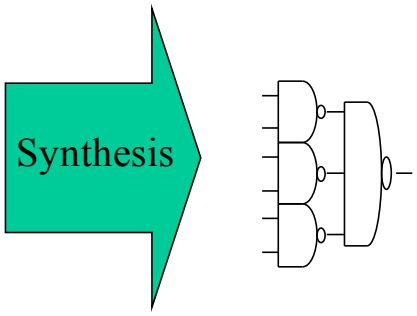
Logic Synthesis

Logic Synthesis (designer definition): convert a description of a digital system in a Hardware Description Language (HDL) to an implementation technology.

Verilog HDL description

Gates

```
// Combinational Logic Circuit
module cmb_circ(Y, A, B, C);
   input A, B, C;
   output Y;
   assign Y = (A&B)|(A&C)|(B&C);
endmodule
```

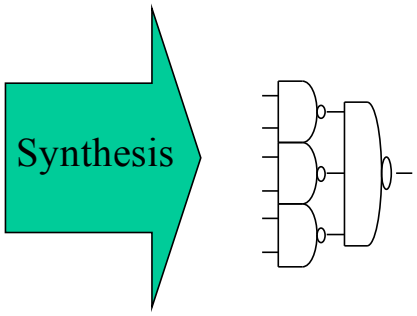


Logic Synthesis

Logic Synthesis (designer definition): convert a description of a digital system in a Hardware Description Language (HDL) to an implementation technology.

VHDL description

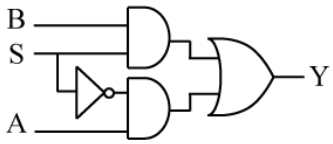
```
library ieee;
use ieee.std_logic_1164.all;

entity majority is
    port (   A, B, C :  in std_logic;
                  Y:   out std_logic
         );
end majority;

ARCHITECTURE a of majority is

 begin

    Y <= (A and B) or (A and C) or (B and C);
end a;
```



Gates

Logic Circuit Models

Schematic with gate symbols



(a) Combinational logic

Boolean Equation

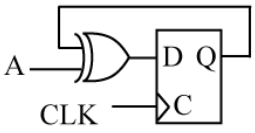
$Y = (B \wedge S) \vee (A \wedge \overline{S})$

Verilog

```
assign y = (b & s) | (a & ~s);
```

or

```
always @(a or b or s)
  if (s) y = b; else y = a;
```



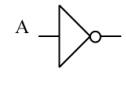
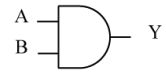
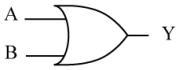
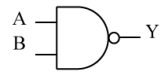
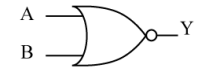
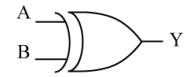
(b) Sequential logic

$Q_+ = Q \oplus A$

Must be annotated with truth table that describes Q+, Q dependence on D, CLK

```
always @(posedge clk)
  q <= q ^ a;
```

Basic Logic Gates

	Truth Table	Gate Symbol	Boolean	Verilog
NOT	A | Y 0 | 1 1 | 0		$Y = \overline{A}$	`assign y = ~a;`
AND	A B | Y 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1		$Y = A \wedge B$	`assign y = a & b;`
OR	A B | Y 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1		$Y = A \vee B$	`assign y = a | b;`
NAND	A B | Y 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0		$Y = \overline{A \wedge B}$	`assign y = ~(a & b);`
NOR	A B | Y 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0		$Y = \overline{A \vee B}$	`assign y = ~(a | b);`
XOR	A B | Y 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0		$Y = A \oplus B$	`assign y = a ^ b;`

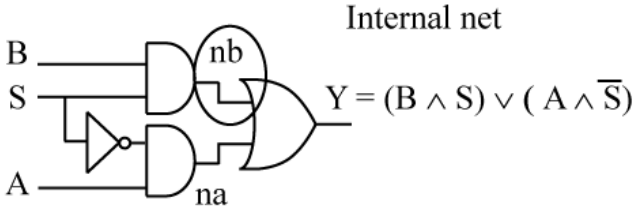
2:1 Multiplexer

Truth Table

S	B	A	Y
0	x	0	0
0	x	1	1
1	0	x	0
1	1	x	1

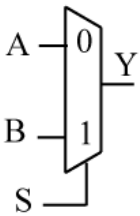
x - don't care

Gate Schematic



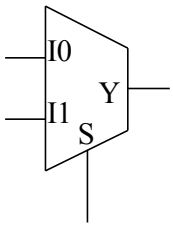
Internal net

Symbol



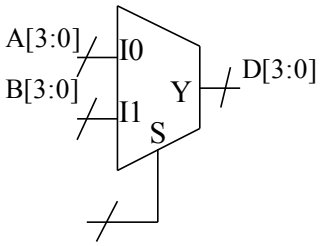
Combinational Building Blocks

1 bit Multiplexer (2:1 MUX)



if S = 0, then Y = I0
if S = 1, then Y = I1

Y = I0 S' + I1 S



Muxes are often used to select groups of bits arranged in busses.

How many wires in each bus ?

Multiplexers

Shannon Expansion Theorem

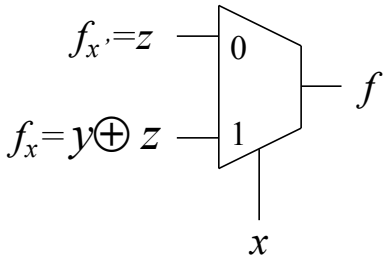
- Original Function: $f=x'y'z+x'yz+xy'z+xyz'$
- Cofactors:

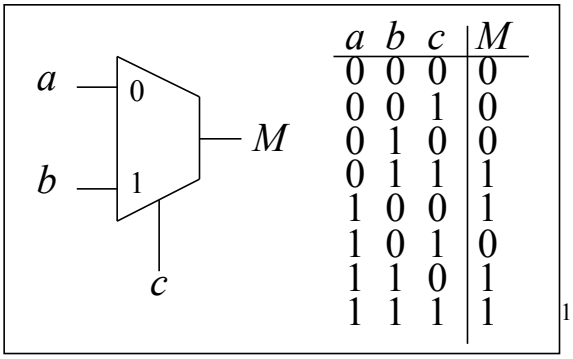
$f_{x'}=f(x=0)=y'z+yz=z$

$f_x=f(x=1)=y'z+yz'=y\oplus z$

$f=x'f_{x'}+xf_x$

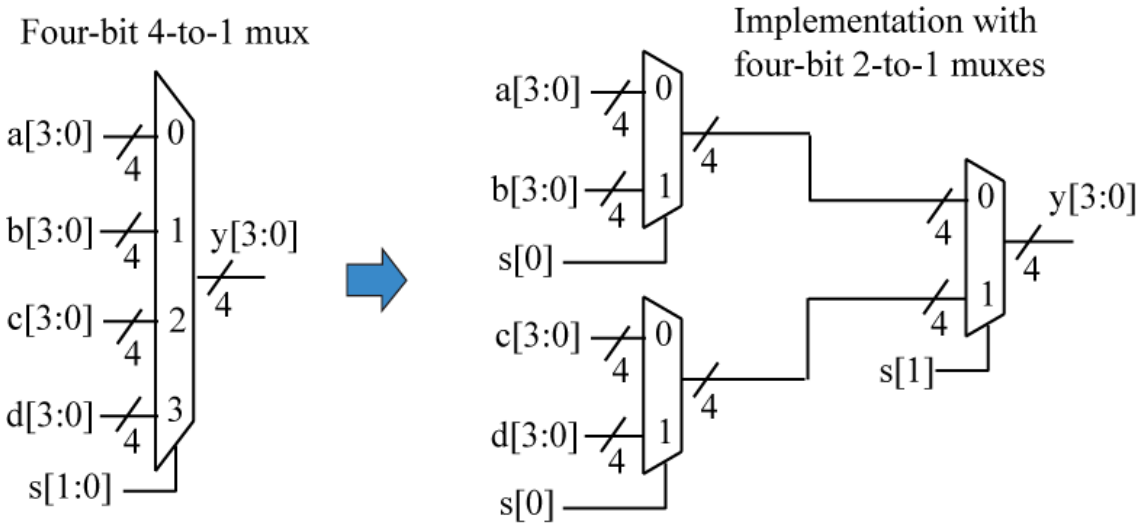
$f=x'z+x(y\oplus z)$





a	b	c	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Multiplexers



Memory Device Architecture

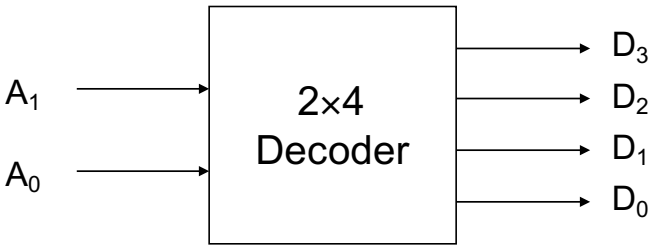
- $2^n \times m$ Device
  - *n* inputs called "address lines"
  - *m* outputs called "data lines"

- Contains 3 Main Subcircuits:
  1) Decoder (Address Decoder)
     - *1 : n ×2^n Decoder Circuit*
  2) Storage Array (Array of 1-bit Storage Cells)
     - *m•2^n : 1-bit storage cell circuits*
  3) Sense Amps (Amplifiers from Cells to Outputs)
     - *m : Single-Ended OR Differential Amplifiers*

Difference Between Memory Types (RAM, ROM, etc.)
is Primarily Due to Storage Cell Implementation

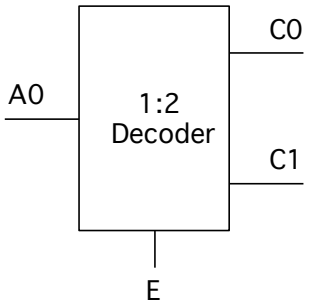
Decoder (Review)

- $n \times 2^n$ Device
  - n encoded inputs
  - 2^n decoded outputs

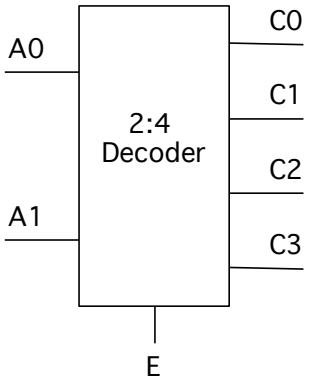


A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoders (with Enable)

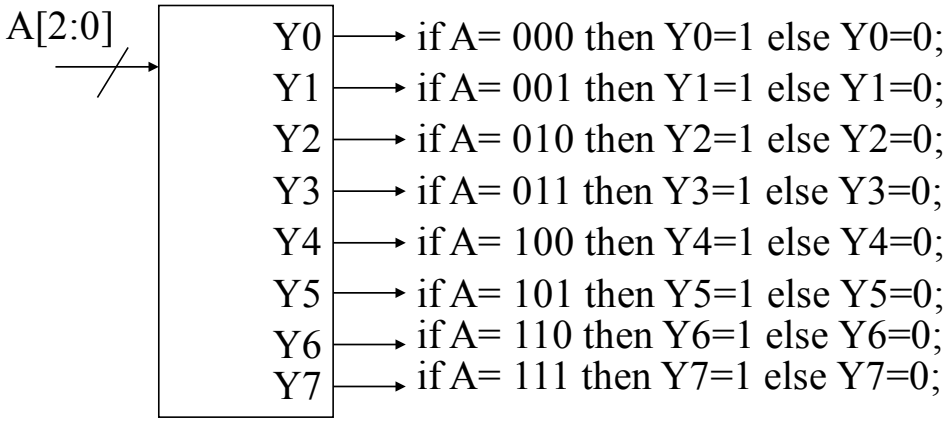


E	A0	C1	C0
1	0	0	1
1	1	1	0
0	X	0	0



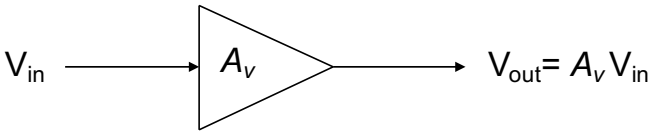
E	A1	A0	C3	C2	C1	C0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	X	X	0	0	0	0

Decoder

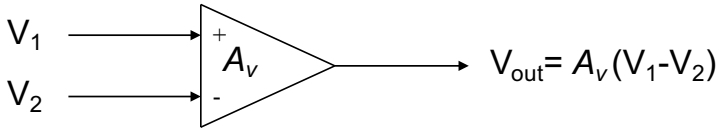


Amplifiers (Review)

- *Single-Ended Amplifier*
  - *Gain: A_v*
  - *1 input voltage, 1 output voltage referenced to common ground*



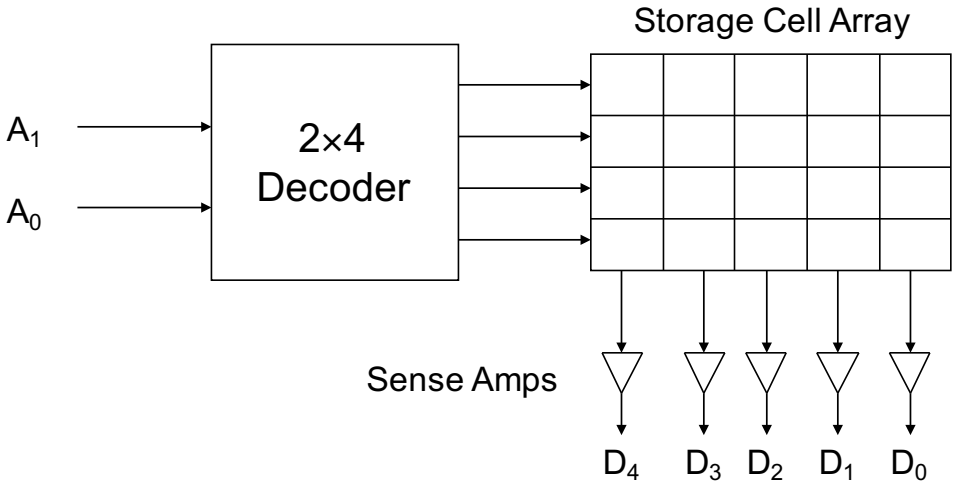
- *Differential Amplifier*
  - *Gain: A_v*
  - *2 input voltages, 1 output voltage referenced to common ground*



Buffer Generally Refers to an Amplifier with Unity Gain (A_v=1)

Semiconductor Memory Device Architecture

- $2^n \times m$ Device
  - n inputs called "address lines"
  - 2^n storage locations called "number of words"
  - m outputs called "data lines"

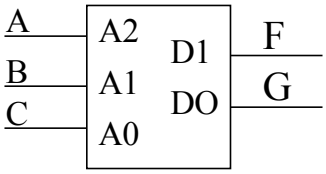


Memory example

$F(A,B,C) = A \oplus B \oplus C \qquad G = AB + AC + BC$

A B C	F	G
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

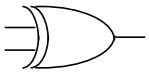
8 x 2 Memory



LookUp Table (LUT)

Recall that Exclusive OR (⊕) is

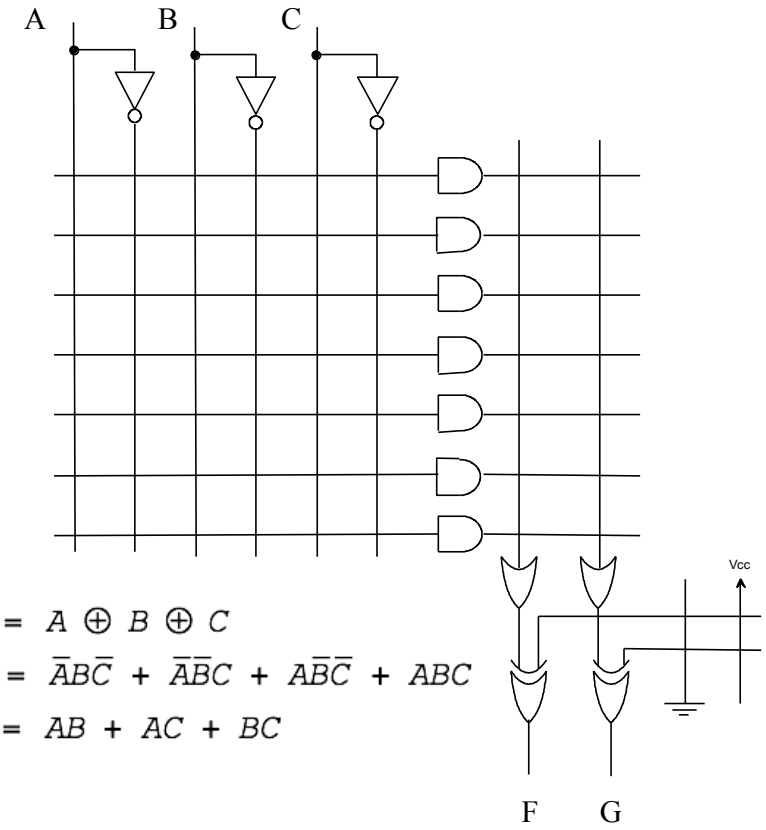
A B	Y
0 0	0
0 1	1
1 0	1
1 1	0



$Y = A \oplus B$
$= A \text{ xor } B$

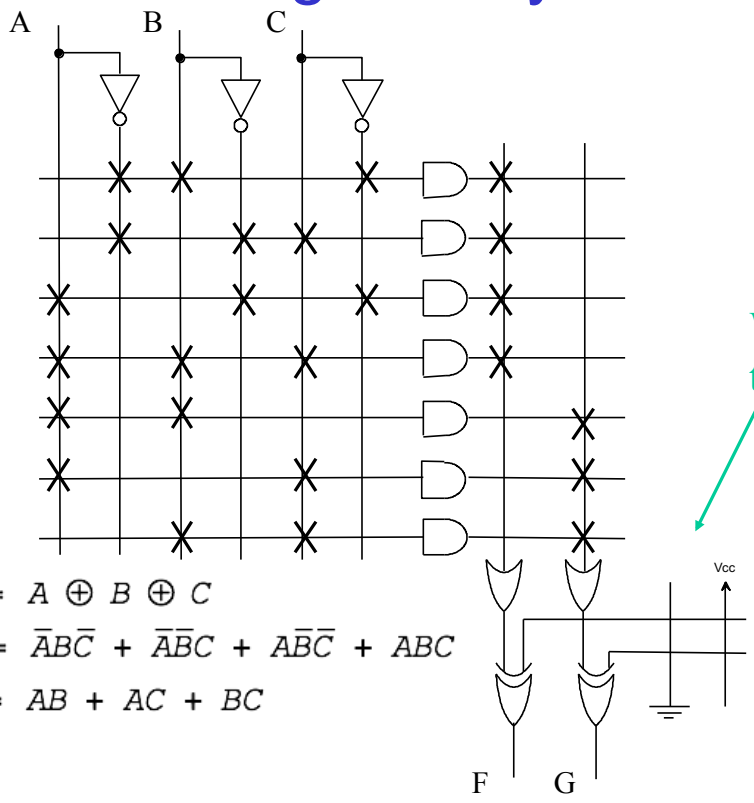
A[2:0] is 3 bit address bus, D[1:0] is 2 bit output bus.
Location 0 has "00",
Location 1 has "10",
Location 2 has "10",
etc….

Logic Arrays



$$F(A, B, C) = A \oplus B \oplus C$$
$$= \bar{A}B\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$$
$$G(A, B, C) = AB + AC + BC$$

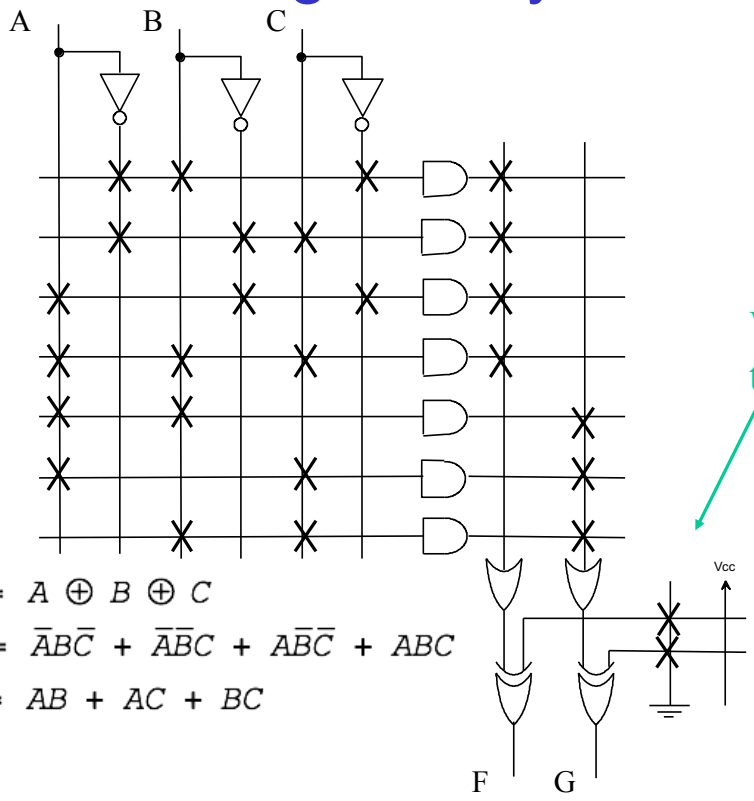
Logic Arrays



What about these?

$$F(A, B, C) = A \oplus B \oplus C$$
$$= \bar{A}B\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$$
$$G(A, B, C) = AB + AC + BC$$

Logic Arrays



What about these?

$F(A, B, C) = A \oplus B \oplus C$

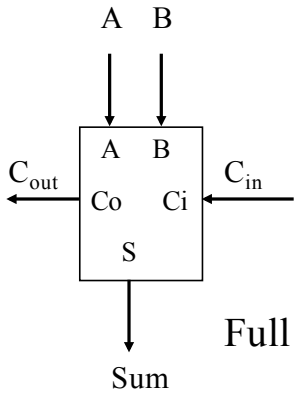
$= \bar{A}B\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$

$G(A, B, C) = AB + AC + BC$

Binary Adder

$F(A,B,C) = A \oplus B \oplus C \qquad G = AB + AC + BC$

These equations look familiar. Recall what a ***Full Adder*** is:



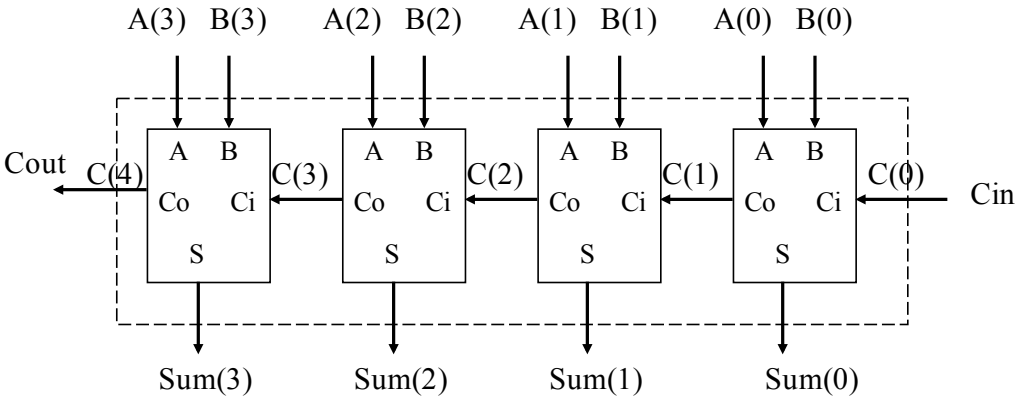
Full Adder (FA)

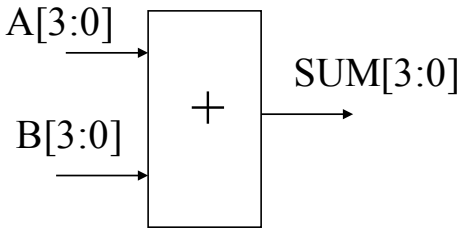
$Sum = A \oplus B \oplus C_{in}$

$Cout = AB + C_{in}A + C_{in}B$

$= AB + C_{in}(A + B)$

4 Bit Ripple Carry Adder



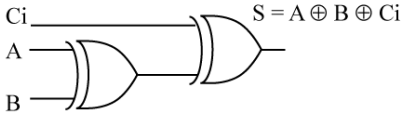


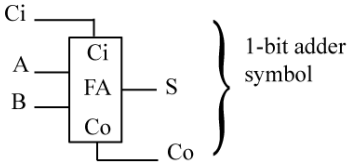
Full and Ripple Adders

A	B	Ci	S	Co
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

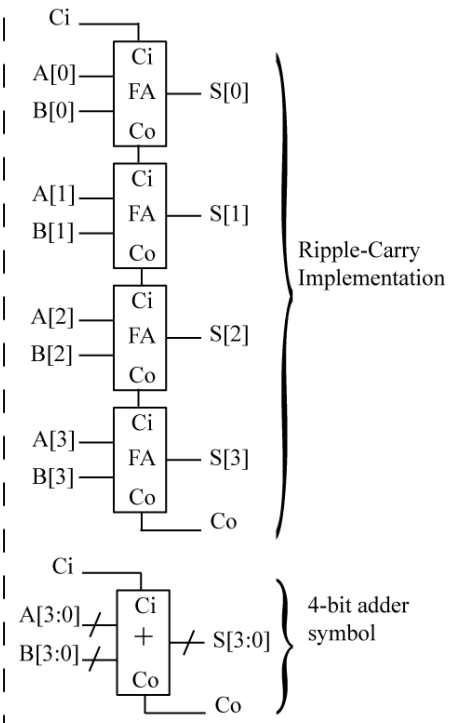
Truth Table for Full Adder
Ci - Carry In
S - Sum
Co - Carry Out

Co = Majority (A, B, Ci)
= (A ∧ B) ∨ (A ∧ Ci) ∨ (B ∧ Ci)





(a) Full Adder Truth table, Boolean equations, Symbol



(b) 4-bit adder symbol and Ripple Carry Implementation

Fixed-Point Multipliers

multiplicand r_2 r_1 r_0

multiplier x s_2 s_1 s_0

partial product s_0*r_2 s_0*r_1 s_0*r_0

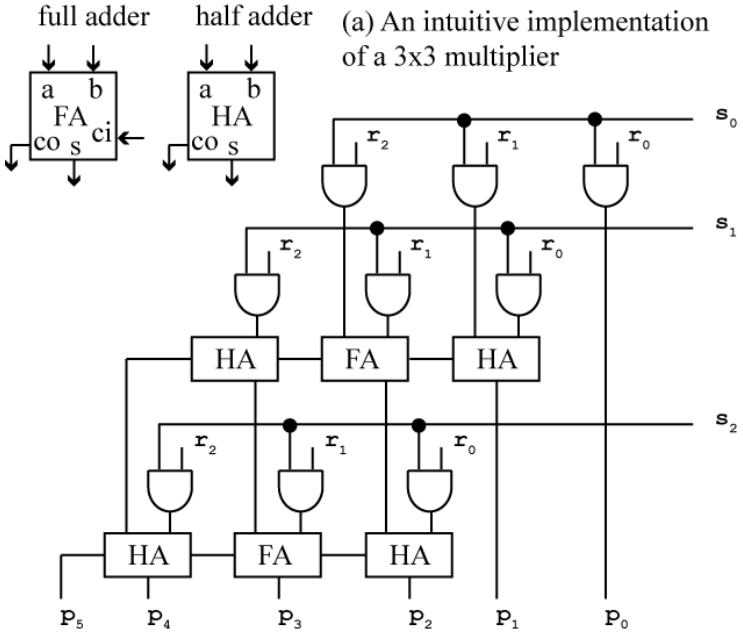
s_1*r_2 s_1*r_1 s_1*r_0

\+ s_2*r_2 s_2*r_1 s_2*r_0

p_5 p_4 p_3 p_2 p_1 p_0 product

	Binary	Decimal
	1 1 1	7
x	1 0 1	x 5
	1 1 1	35
	0 0 0	
+	1 1 1	
	1 0 0 0 1 1 = 35	

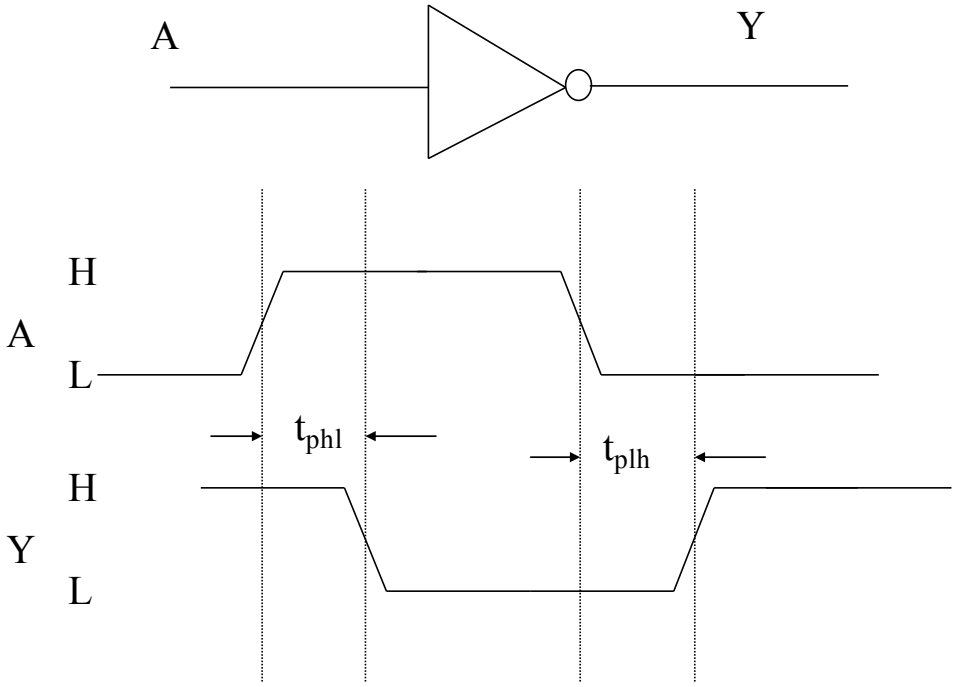
Array Multiplier Structure



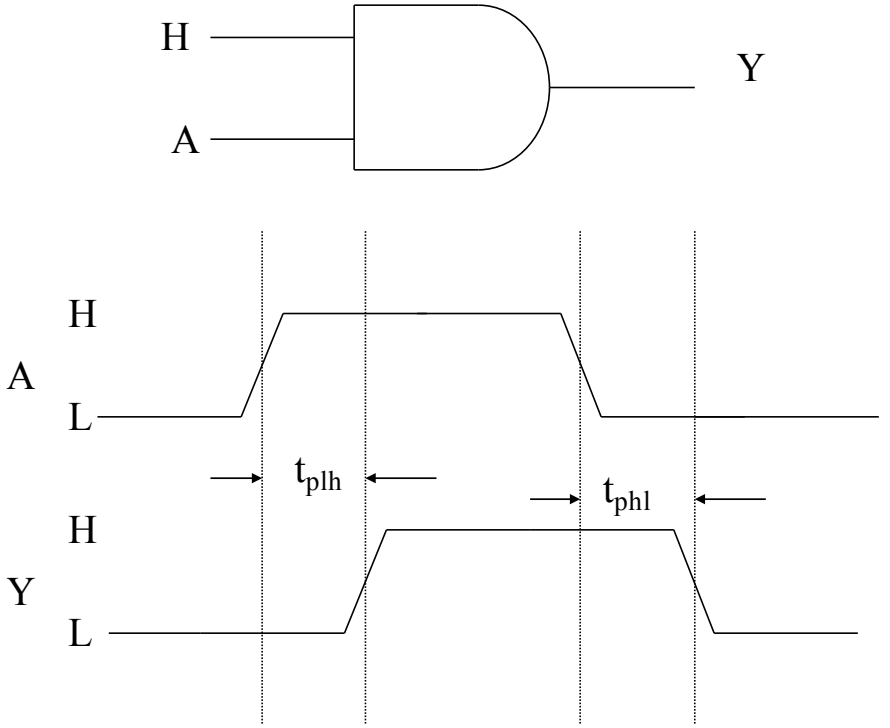
(b) A Verilog implementation of a 3x3 multiplication using the '*' operator

```
//3x3 multiplier
//using the '*' operator

module mult3x3 (a, b, y);

input [2:0] a,b;
output [5:0] y;

//do 3x3 multiply
assign y = a * b;

endmodule
```

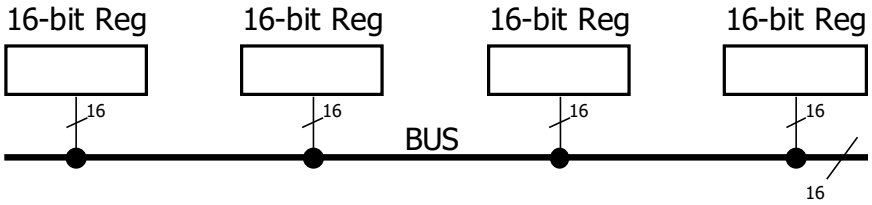
Propagation Delay



Propagation Delay (non inverting)



Busses



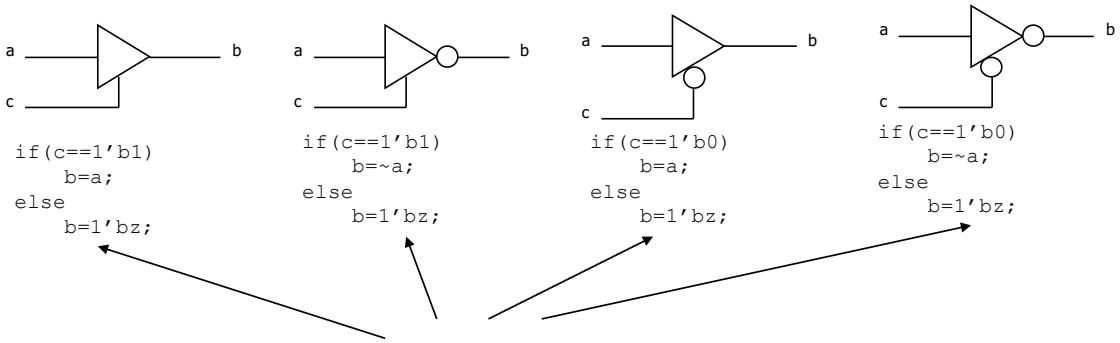
Here we require 16 wires and arbitration logic

Arbitration logic **Controls** the flow of data

Tri-State Buffer Types

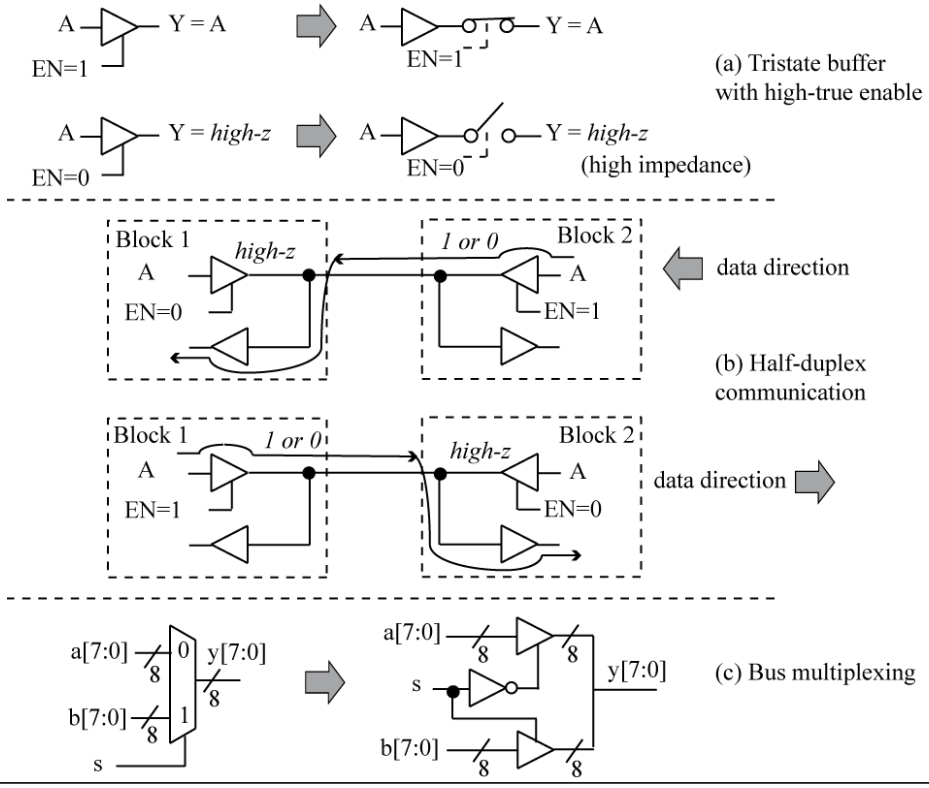
- 3 states instead of 2 (0 and 1)
- 0, 1, z; z is "high impedance" state
- "high impedance" is open circuit

Type of 3 state buffers



If-statements to explain behavior only (Verilog HDL)

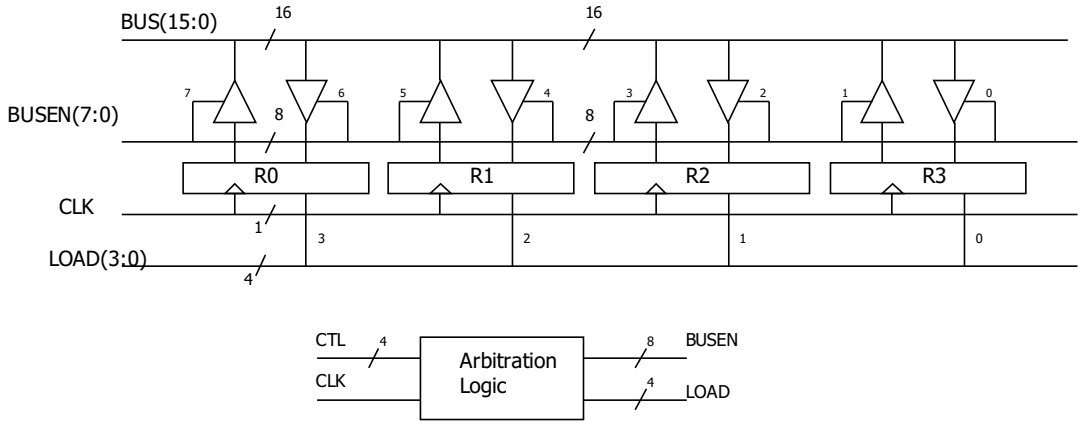
Tri-state Buffers



16-Bit Bus

Example

$R_n \leftarrow R_m$ designated by 4-Bit control word *nm*, ie 0110 means R1 ←R2



Making a Design Run Fast

- Speed is usually much more important than saving gates.
- The speed of a gate directly affects the maximum clock speed of a digital system
- Gate speed is TECHNOLOGY dependent
  - 90nm CMOS process has faster gates than 130nm CMOS process
- Implementation choice will affect Design speed
  - A Custom integrated circuit will be faster than an FPGA implementation.
- Design approaches will affect clock speed of system
  - Smart designers can make a big difference

Summary

- Need to review your Digital Logic Design notes
  - Basic Gates, Boolean algebra (algebraic minimization, up to four variable K-maps), Combinational building blocks (muxes, decoders, memories, adders)
- We will discuss Hardware Description Languages
  - Verilog is the language used in the class
- We will discuss modern implementation technologies, primarily Field Programmable Gate Arrays (FPGAs)
- We will discuss design strategies for making designs run faster, not necessarily take less gates.