

Unit 12

Adders & Arithmetic Circuits

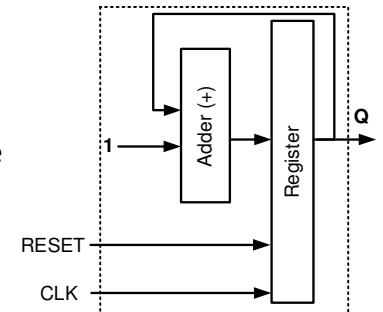
Learning Outcomes

- I understand what gates are used to design half and full adders
- I can build larger arithmetic circuits from smaller building blocks

ADDERS

Adder Intro

- Addition is one of the most common operations performed by computer systems
- We can use adders to build larger components like the _____ to the right
- Every clock cycle, the value Q (let's say 4-bits: Q[3:0]), _____ to the adder circuit which adds 1 to the value and the register captures that new value on the next clock edge
- The sequence on Q on each clock cycle would be: 0, _____...
- Could you design what's inside the adder block? How would you do it?



$$\begin{array}{r}
 0111 = \text{curr } Q \\
 + \quad 1 \\
 \hline
 1000 = \text{next } Q
 \end{array}$$

Adder Intro

- What if we had to add _____ two 4-bit numbers, X[3:0] and Y[3:0]? Do we have the techniques to build such a circuit directly?

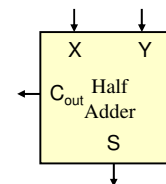
$$\begin{array}{r} 0110 = X \\ + 0111 = Y \\ \hline 1101 \end{array}$$

- _____
- _____
- _____

Adder Intro

- Idea:** Build a circuit that performs _____ column of addition and then use _____ of those circuits to perform the overall 4-bit addition
- Let's start by designing a circuit that adds 2-bits: X and Y that are in the same column of addition

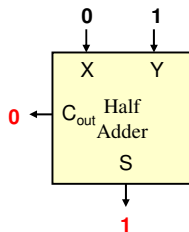
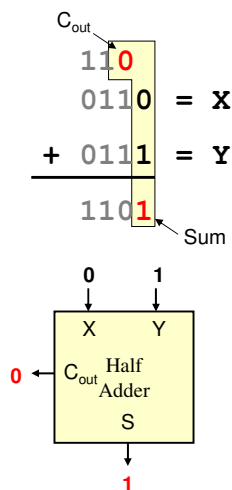
$$\begin{array}{r} 0110 = X \\ + 0111 = Y \\ \hline 1101 \end{array}$$



Addition – Half Adders

- Addition is done in columns
 - Inputs are a bit from X and bit from Y, both from the same column
 - Outputs are the Sum Bit and Carry-Out (C_{out})
- Design a Half-Adder (HA) circuit that takes in X and Y and outputs S and C_{out}
- Use the truth table to find the gate implementation

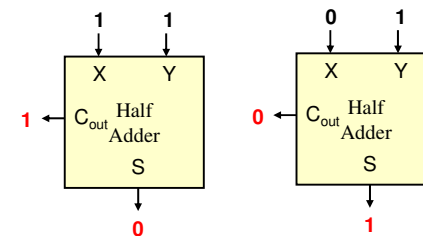
X	Y	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Problem With Half Adders

- We'd like to use one adder circuit for each column of addition
- Problem:**
 - No place for _____ of half adder to connect to the _____
- Solution**
 - Redesign adder circuit to include an _____ input for the _____

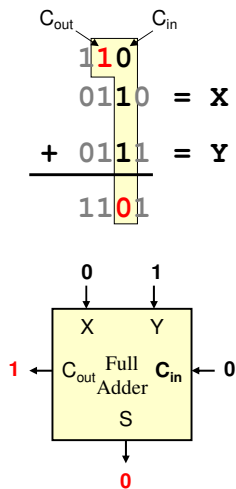
$$\begin{array}{r} 110 \\ 0110 = X \\ + 0111 = Y \\ \hline 1101 \end{array}$$



Addition – Full Adders

- Add a Carry-In input (C_{in})
- New circuit is called a Full Adder (FA)

X	Y	C_{in}	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



Addition – Full Adders

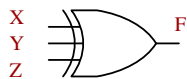
- Find the minimal 2-level implementations for C_{out} and S ...

X	Y	C_{in}	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

Recall a 2-input XOR can be written in SOP as $F = x'y + xy'$

XOR and XNOR Gates

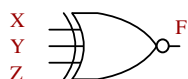
- Recall a 2-input XOR can be written in SOP as $F = x'y + xy'$
- A 2-input XNOR can be written in SOP as $F = x'y' + xy$



True if an # of inputs are true

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$F = X \text{ xor } Y \text{ xor } Z$



True if an # of inputs are true

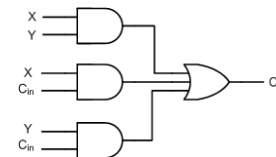
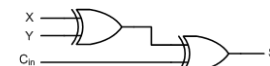
X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$F = \overline{X \text{ xor } Y \text{ xor } Z}$

A checkerboard K-map corresponds to either an XOR or XNOR

Full Adder Logic

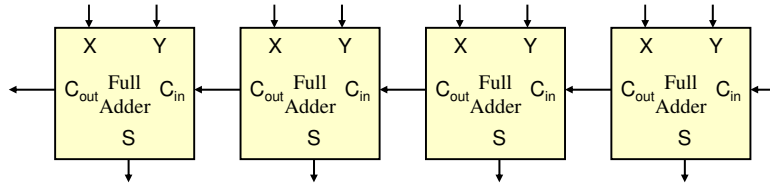
- $S =$ _____
– Recall: XOR is defined as true when ODD number of inputs are true...exactly when the sum bit should be 1
- $C_{out} =$ _____
– Carry when sum is 2 or more (i.e. when at least 2 inputs are 1)
– Circuit is just checking all combinations of 2 inputs



Addition – Full Adders (1)

- Use 1 Full Adder for each column of addition

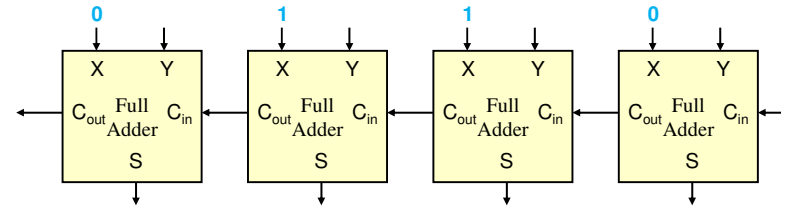
$$\begin{array}{r} 0110 \\ + 0111 \\ \hline \end{array}$$



Addition – Full Adders (2)

- Connect bits of top number to X inputs

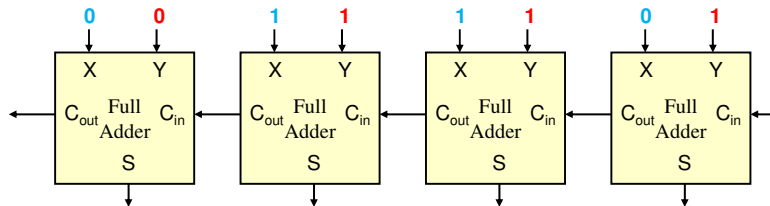
$$\begin{array}{r} 0110 \\ + 0111 \\ \hline \end{array}$$



Addition – Full Adders (3)

- Connect bits of bottom number to Y inputs

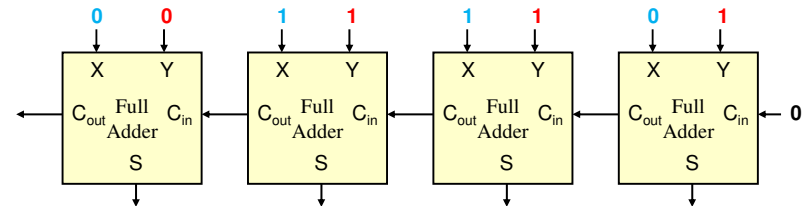
$$\begin{array}{r} 0110 = X \\ + 0111 = Y \\ \hline \end{array}$$



Addition – Full Adders (4)

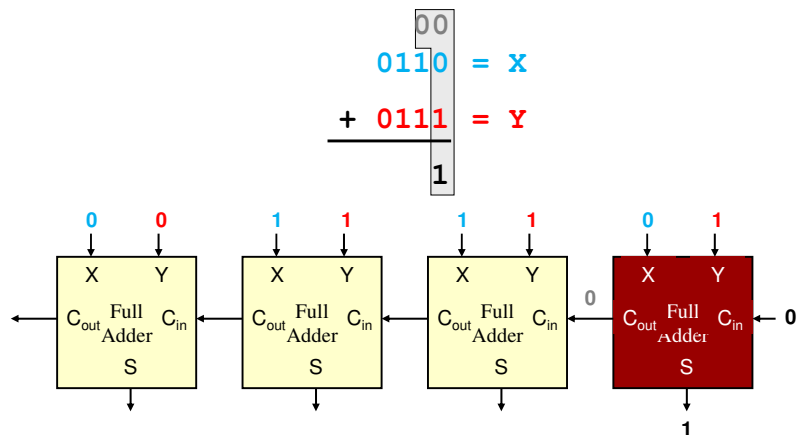
- Be sure to connect first C_{in} to 0

$$\begin{array}{r} 0110 = X \\ + 0111 = Y \\ \hline \end{array}$$



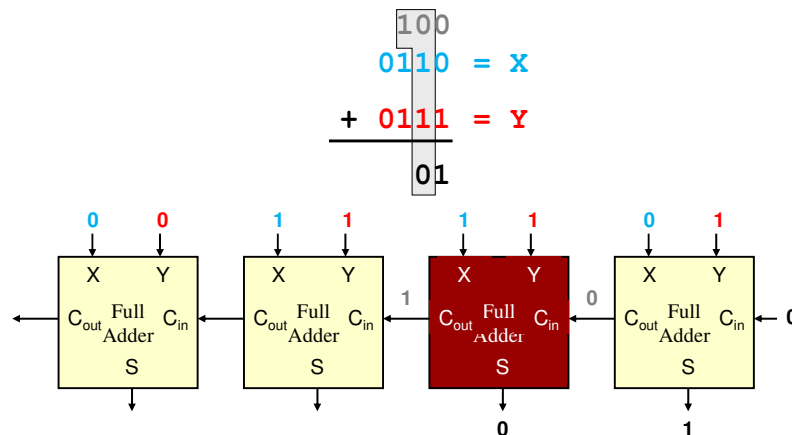
Addition – Full Adders (5)

- Use 1 Full Adder for each column of addition



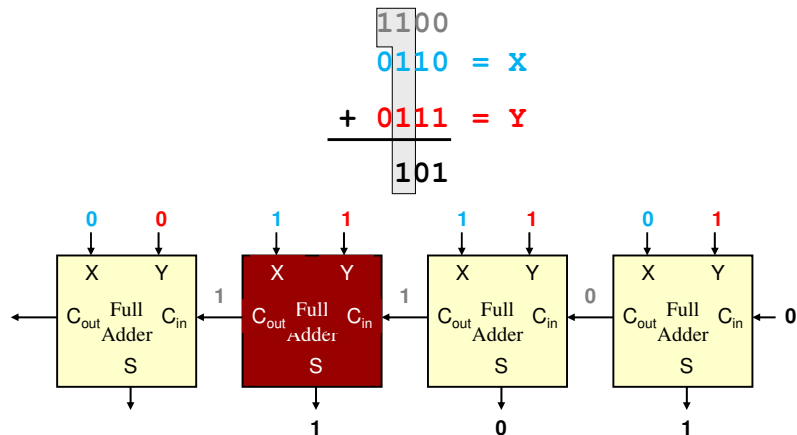
Addition – Full Adders (6)

- Use 1 Full Adder for each column of addition



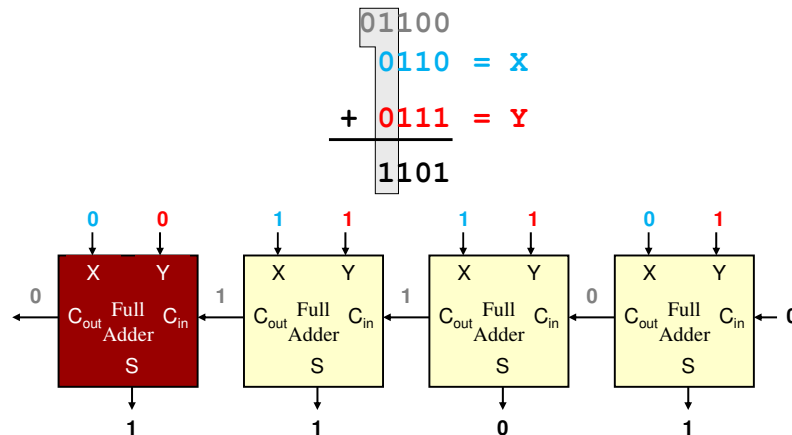
Addition – Full Adders (7)

- Use 1 Full Adder for each column of addition



Addition – Full Adders (8)

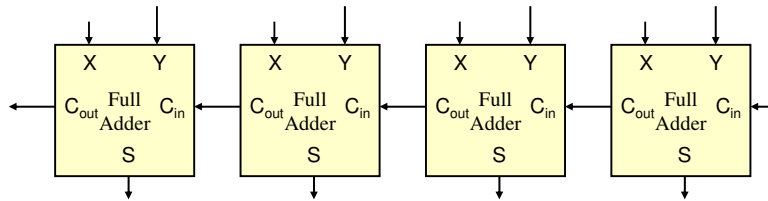
- Use 1 Full Adder for each column of addition



Performing Subtraction

- To subtract

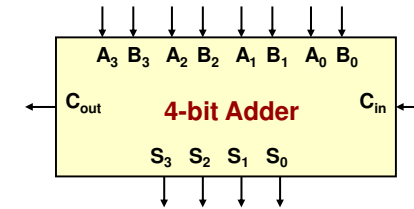
$$\begin{array}{r} 0101 = X \\ - 0011 = Y \\ \hline 0010 \end{array} \Rightarrow$$



4-bit Adders

- We can create a component to perform 4-bit addition

$$\begin{array}{r} A_3 A_2 A_1 A_0 = A \\ + B_3 B_2 B_1 B_0 = B \\ \hline S_4 S_3 S_2 S_1 S_0 = S \end{array}$$



Device vs. System Labels

- When using hierarchy (i.e. building blocks) to design a circuit be sure to show both device and system labels
 - Device Labels: Signal names used _____ the block
 - Placeholder names the designer/manufacturer of the block uses to indicate which input/output is which to the outside user (Names may vary; read the manual)
 - System labels: Signal names used _____ the block
 - _____ signals from the circuit being built and given by the designer
 - Can have the same name as the device label if such a signal name exists at the outside level

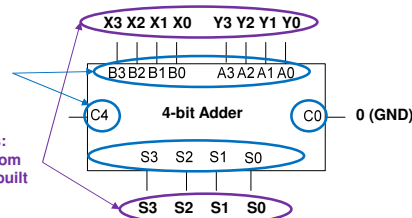
Analogy: Formal and Actual parameters

- a and b are like device labels and indicate the names used inside a block.
- x and y are like system labels and represent the actual values to be used.

```
int div(int a, int b)
{ int s = a/b;
  return s; }
int main()
{
  int x=10, y=2;
  int s = div(x,y);
}
```

Device Labels:
Indicate which input/output is which inside the block.

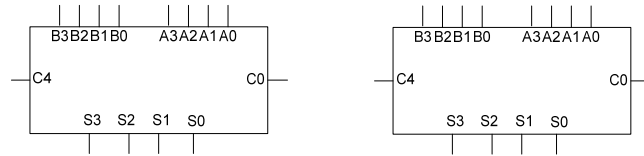
System Labels:
Actual signals from the circuit being built



EXERCISES

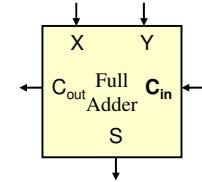
Building an 8-bit Adder

- Use (2) 4-bit adders to build an 8-bit adder to add $X=X[7:0]$ and $Y=Y[7:0]$ and produce a sum, $S=[7:0]$ and a carry-out, C_8 .
 - Label the inputs and outputs and make appropriate connections



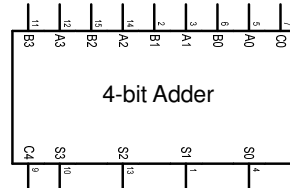
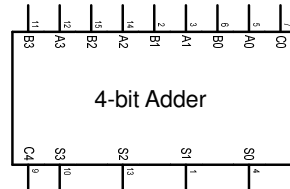
Adding Many Bits

- You know that an FA adds $X + Y + C_i$
- Use FA and/or HA components to add 4 individual bits:
 $A + B + C + D$



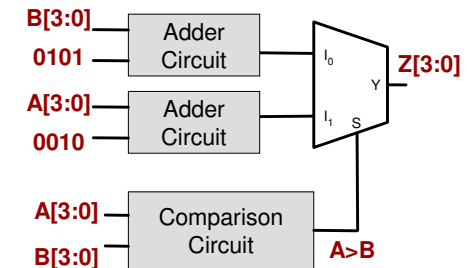
Adding 3 Numbers

- Add $X[3:0] + Y[3:0] + Z[3:0]$ to produce $F[7:0]$ using the adders shown plus any FA and HA components you need



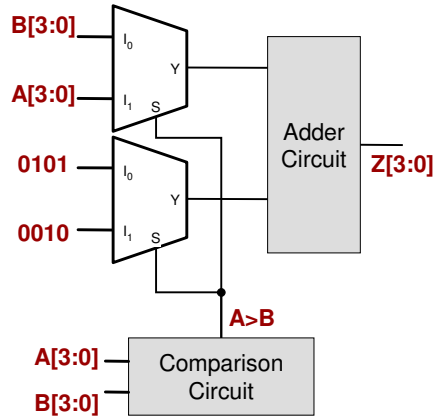
Mapping Algorithms to HW

- Wherever an if..then..else statement is used usually requires a mux
 - if($A[3:0] > B[3:0]$)
 - $Z = A+2$
 - else
 - $Z = B+5$



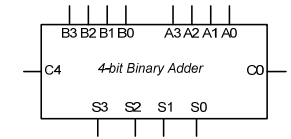
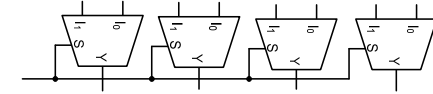
Mapping Algorithms to HW

- Wherever an if..then..else statement is used usually requires a mux
 - if(A[3:0] > B[3:0])
 - Z = A+2
 - else
 - Z = B+5



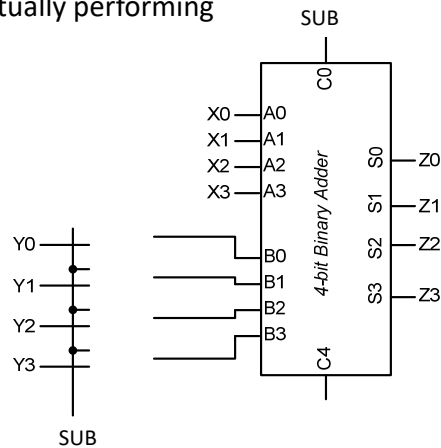
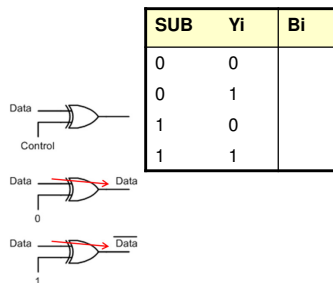
Adder / Subtractor

- If sub == 1
 - Z = X[3:0]-Y[3:0]
- Else
 - Z = X[3:0]+Y[3:0]



Adder / Subtractor

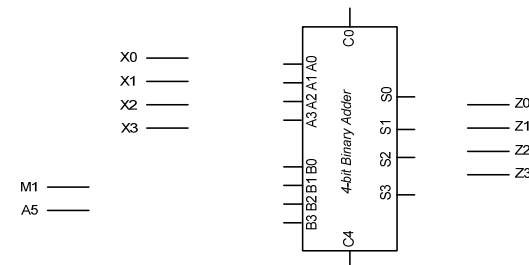
- Go back and optimize the muxes by determining what logic function they are actually performing
- If sub == 1
 - Z = X[3:0]-Y[3:0]
- Else
 - Z = X[3:0]+Y[3:0]



Another Example

- Design a circuit that takes a 4-bit binary number, X, and two control signals, A5 and M1 and produces a 4-bit result, Z, such that:
 - Z = X + 5, when A5, M1 = 1, 0
 - Z = X - 1, when A5, M1 = 0, 1
 - Z = X, when A5, M1 = 0, 0

		4-bit Adder Input			
A5	M1	B3	B2	B1	B0
0	0				
0	1				
1	0				
1	1	d	d	d	d



MULTIPLIERS

Unsigned Multiplication Review

- Same rules as decimal multiplication
- Multiply each bit of Q by M shifting as you go
- An m-bit * n-bit mult. produces an m+n bit result (i.e. n-bit * n-bit produces 2*n bit result)
- Notice each partial product is a shifted copy of M or 0 (zero)

$$\begin{array}{r} 1010 \text{ M (Multiplicand)} \\ * 1011 \text{ Q (Multiplier)} \\ \hline \end{array}$$

Unsigned Multiplication Review

- Same rules as decimal multiplication
- Multiply each bit of Q by M shifting as you go
- An m-bit * n-bit mult. produces an m+n bit result (i.e. n-bit * n-bit produces 2*n bit result)
- Notice each partial product is a shifted copy of M or 0 (zero)

$$\begin{array}{r} 1010 \text{ M (Multiplicand)} \\ * 1011 \text{ Q (Multiplier)} \\ \hline 1010 \\ 1010_ \text{ PP (Partial} \\ 0000_ \text{ Products)} \\ + 1010_ \\ \hline 01101110 \text{ P (Product)} \end{array}$$

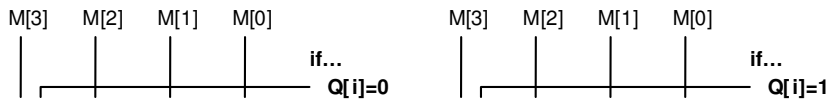
Combinational Multiplier

- Partial Product (PP_i) Generation
 - Multiply Q[i] * M
 - if Q[i]=0 => PP_i = 0
 - if Q[i]=1 => PP_i = M

$$\begin{array}{r} 1010 \text{ M (Multiplicand)} \\ * 1011 \text{ Q (Multiplier)} \\ \hline 1010 \\ 1010_ \text{ PP (Partial} \\ 0000_ \text{ Products)} \\ + 1010_ \\ \hline 01101110 \text{ P (Product)} \end{array}$$

Combinational Multiplier

- Partial Product (PP_i) Generation
 - Multiply $Q[i] * M$
 - if $Q[i]=0 \Rightarrow PP_i = \underline{\hspace{1cm}}$
 - if $Q[i]=1 \Rightarrow PP_i = \underline{\hspace{1cm}}$
 - $\underline{\hspace{1cm}}$ gates can be used to generate each partial product

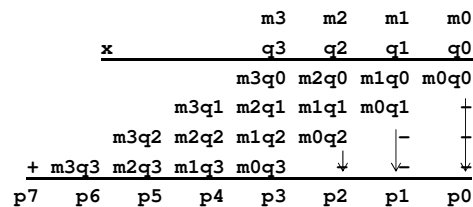
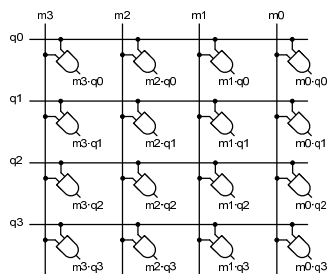


Combinational Multiplier

- Partial Products must be added together
- Combinational multipliers require long propagation delay through the adders
 - propagation delay is proportional to the number of partial products (i.e. number of bits of input) and the width of each adder

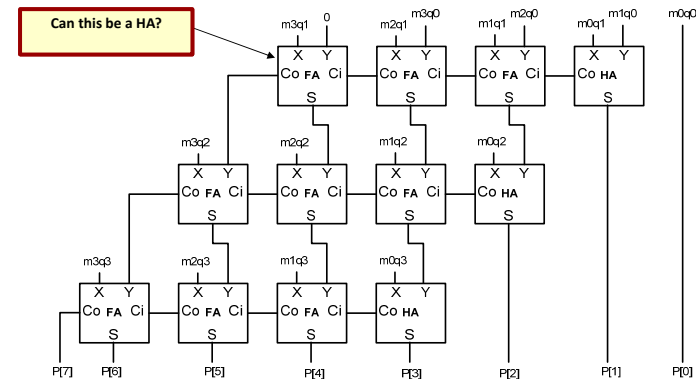
Multiplication Overview

- Combinational: Array multiplier uses an array of adders
 - Can be as simple as N-1 ripple-carry adders for an NxN multiplication



AND Gate Array produces partial product terms

Array Multiplier



- Maximum n-bit * n-bit delay is proportional to _____