



LUND  
UNIVERSITY

# EITF35: Introduction to Structured VLSI Design

Part 2.1.1: Combinational circuit

Liang Liu  
liang.liu@eit.lth.se



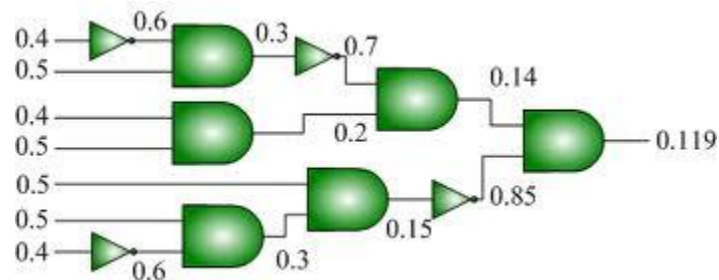
# Why Called “Combinational” Circuits?

## □ Combination

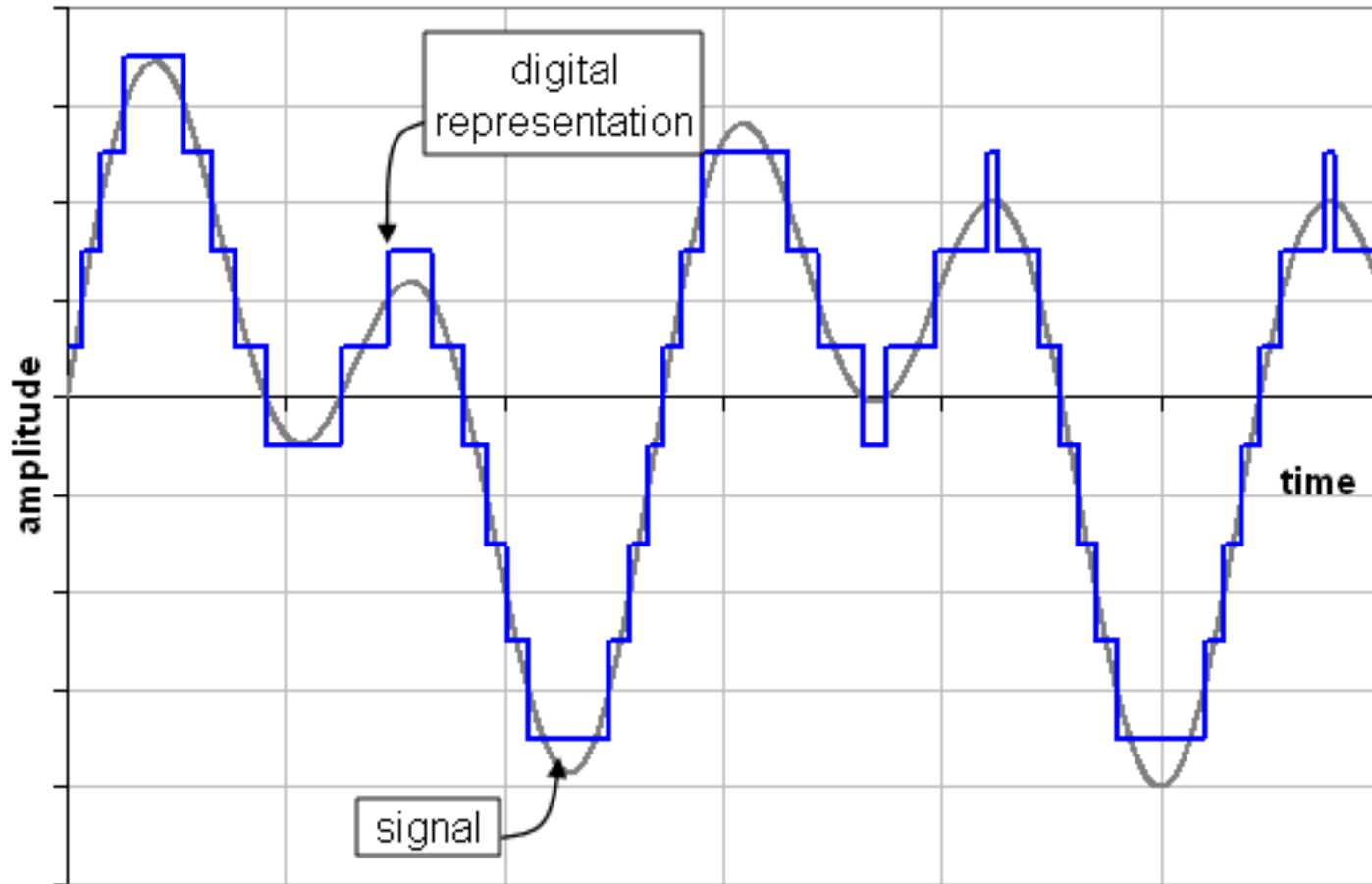
- In mathematics a combination is a way of selecting several things out of a larger group
- Select two fruits out of APPLE, PEAR, and ORANGE
- In a combination the **order** of elements is **irrelevant**

## □ Combinational Circuits

- **time-independent logic**, where the output is a pure function of the present input only.
- the **order** of inputs doesn't matter for the outputs.



# 'Digital'- quantization



# Data Representation

## □ Unsigned

- Unsigned integer:  $\sum_{i=0}^{n-1} \text{bit}_i 2^i$

## □ Signed (Two's complement)

- The result of **subtracting the number from  $2^{N-1}$**
- Inverting all bits and **adding 1**

$$\text{bit}_{n-1} (-2^{n-1}) + \sum_{i=0}^{n-2} \text{bit}_i 2^i$$

**11110100**<sub>2</sub> = **-12**<sub>10</sub>

↑                      ↑  
**Sign bit**          **2's complement**



# 8-bit Signed/Unsigned Integers

Signed integers	Signed overflow ↑	-128	1000 0000	
		-127	1000 0001	
		...	...	
			1111 1100	
			1111 1101	
			1111 1110	
			1111 1111	
			0000 0000	0
			0000 0001	1
			0000 0010	2
			0000 0011	3
			...	...
			0111 1110	126
		0111 1111	127	
		1000 0000	128	
		1000 0001	129	
		...	...	
		1111 1110	254	
		1111 1111	255	
	Signed overflow ↓			Unsigned integers
				Unsigned overflow ↓

MSB defines sign



# General Fixed-Point Representation

## □ *Qm.n* notation

- *m* bits for integer portion, *n* bits for fractional portion
- Total number of bits  $N = m + n + 1$ , for signed numbers
- Example: 16-bit number ( $N=16$ ) and Q2.13 format

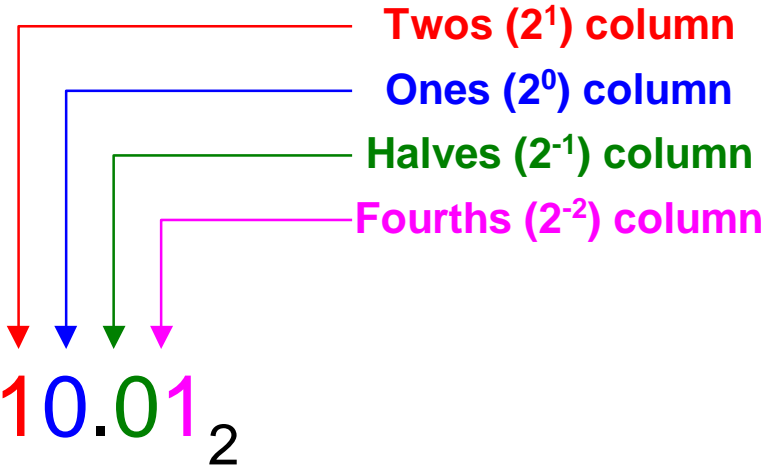


Diagram illustrating the expansion of the binary number  $10.01_2$  into a sum of products:

$$= 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

## Decimal Point in Hardware?



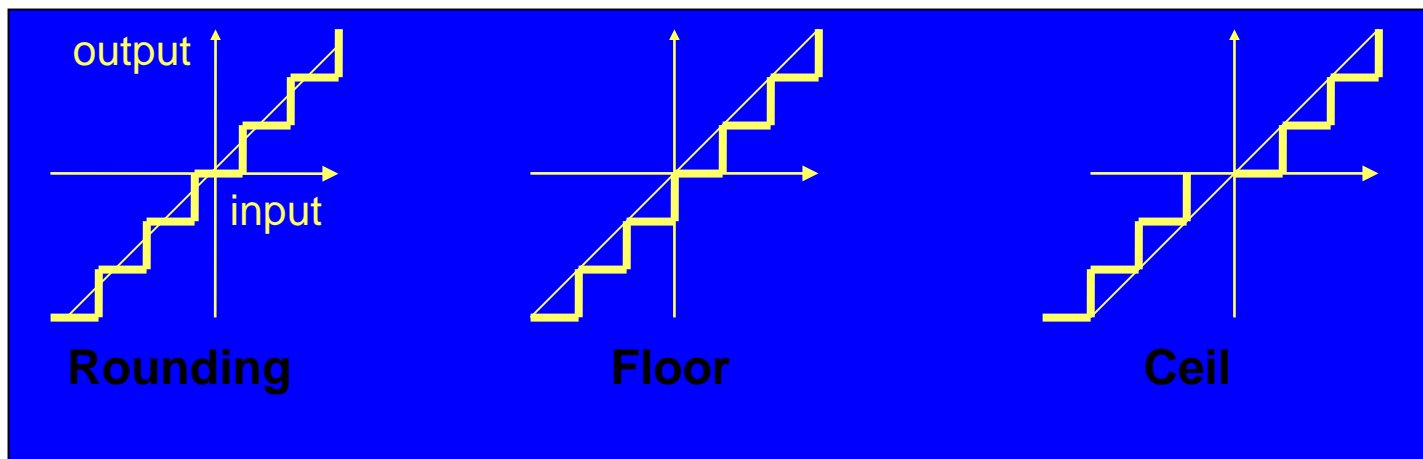
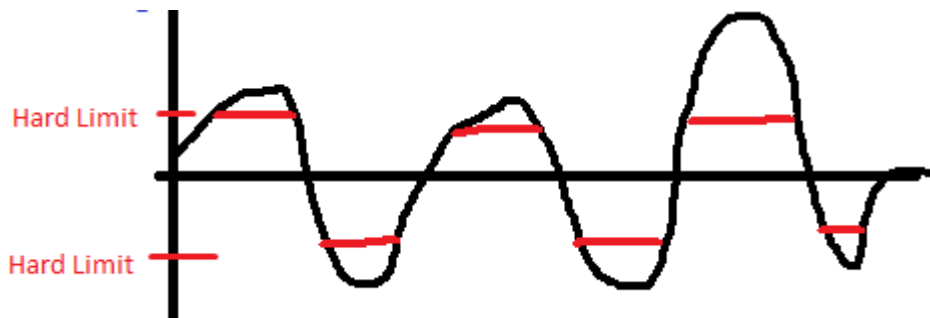
# Finite Word-Length Effect

## □ Overflow

- Saturation

## □ Quantization error

- Round
- Truncation



$$\text{round}(0.51)=1$$

$$\text{floor}(0.51)=0$$

$$\text{ceil}(0.49)=1$$

**Will learn more in DSP-Design course**



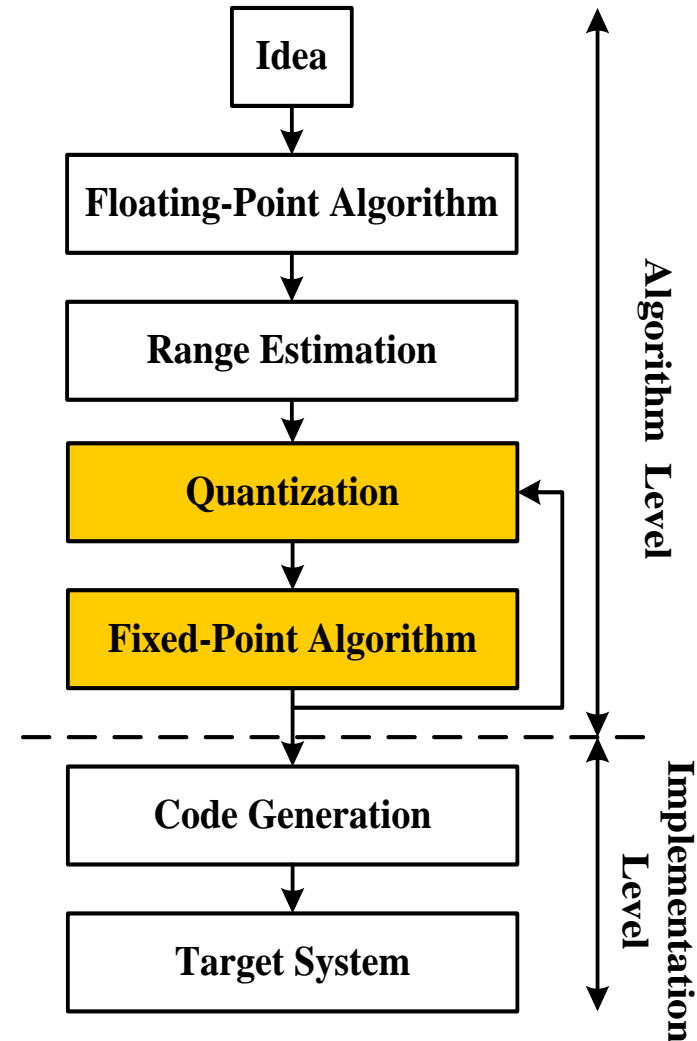
# Fixed-Point Design

## □ DSP algorithms

- Often developed in floating point
- Later mapped into **fixed point** for digital hardware realization

## □ Fixed-point digital VLSI

- Lower area
- Lower power
- Quantization error & small dynamic range

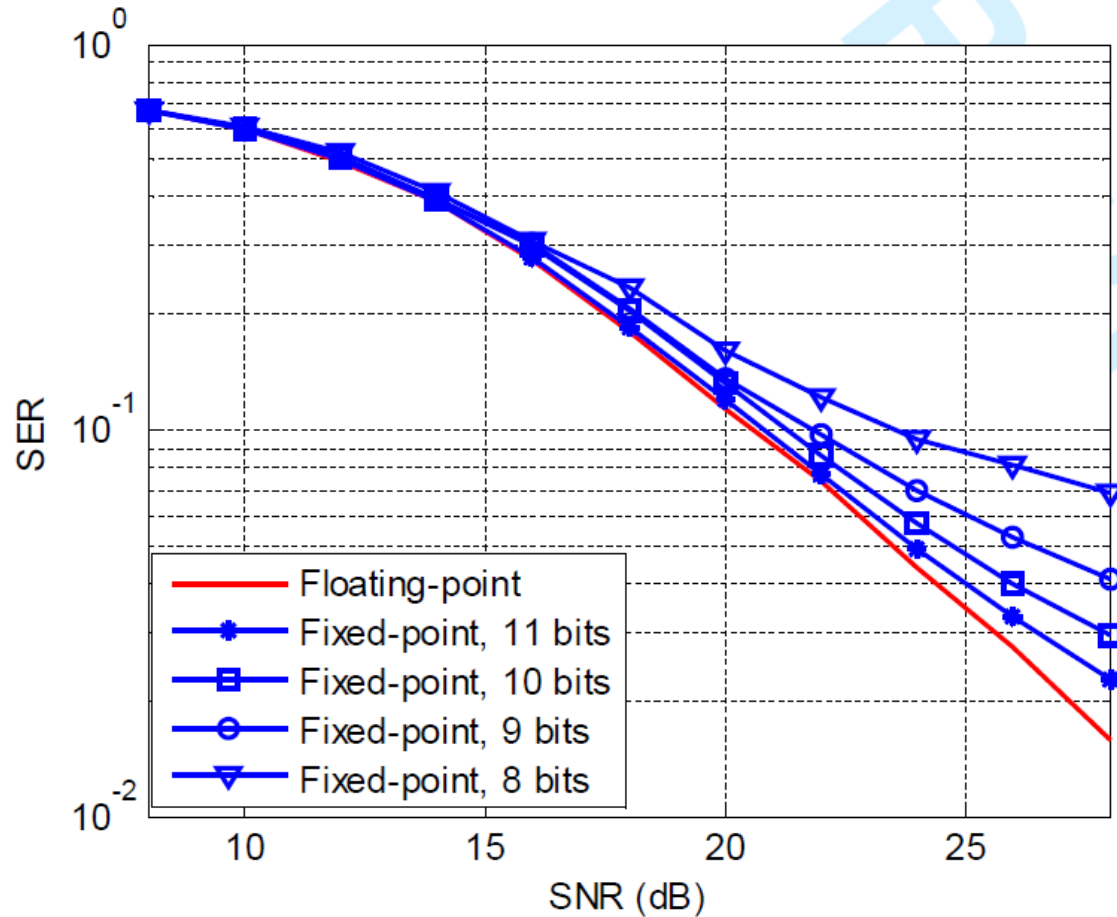




# Optimum Word-Length

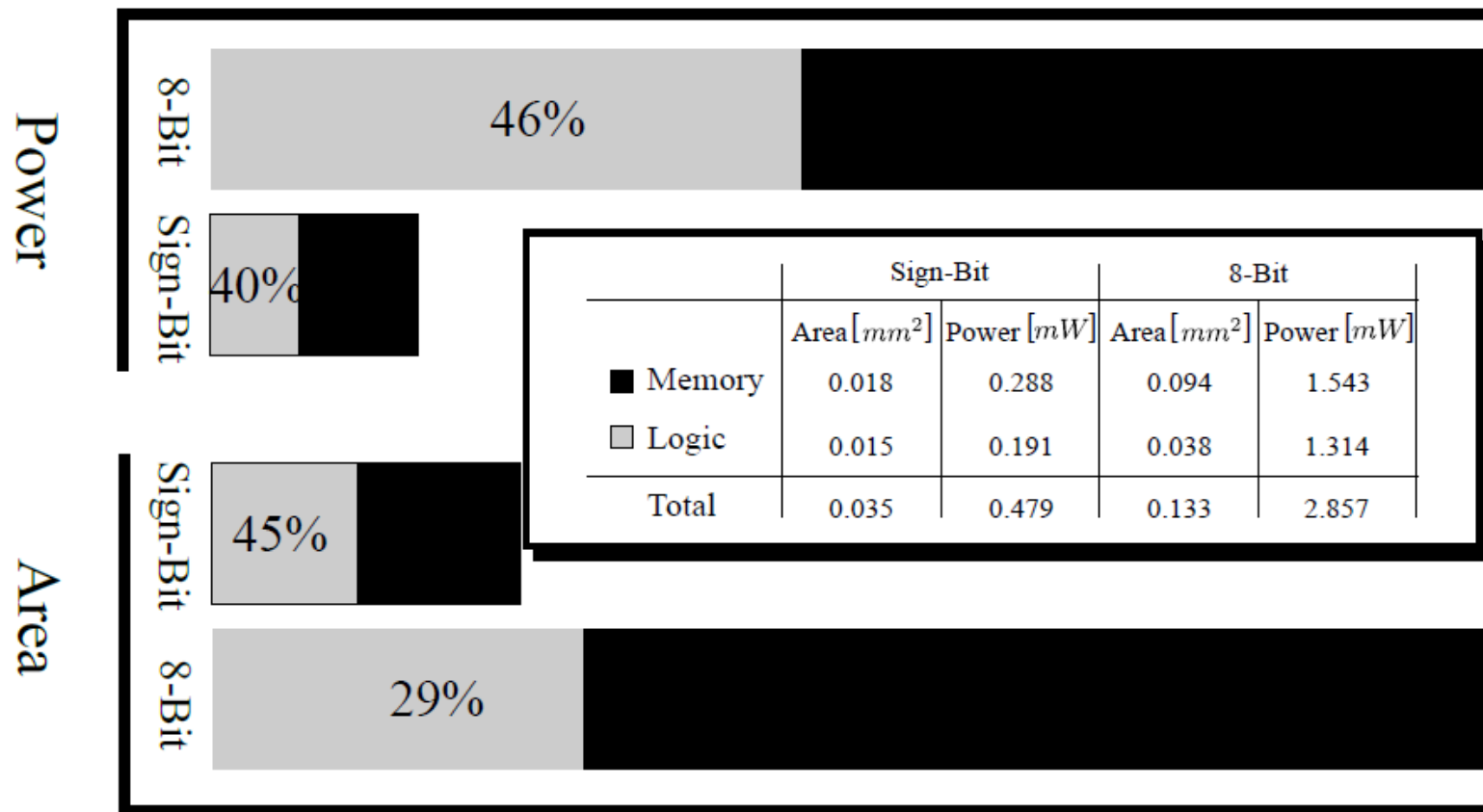
□ Range Analysis

□ Fixed-point Simulation



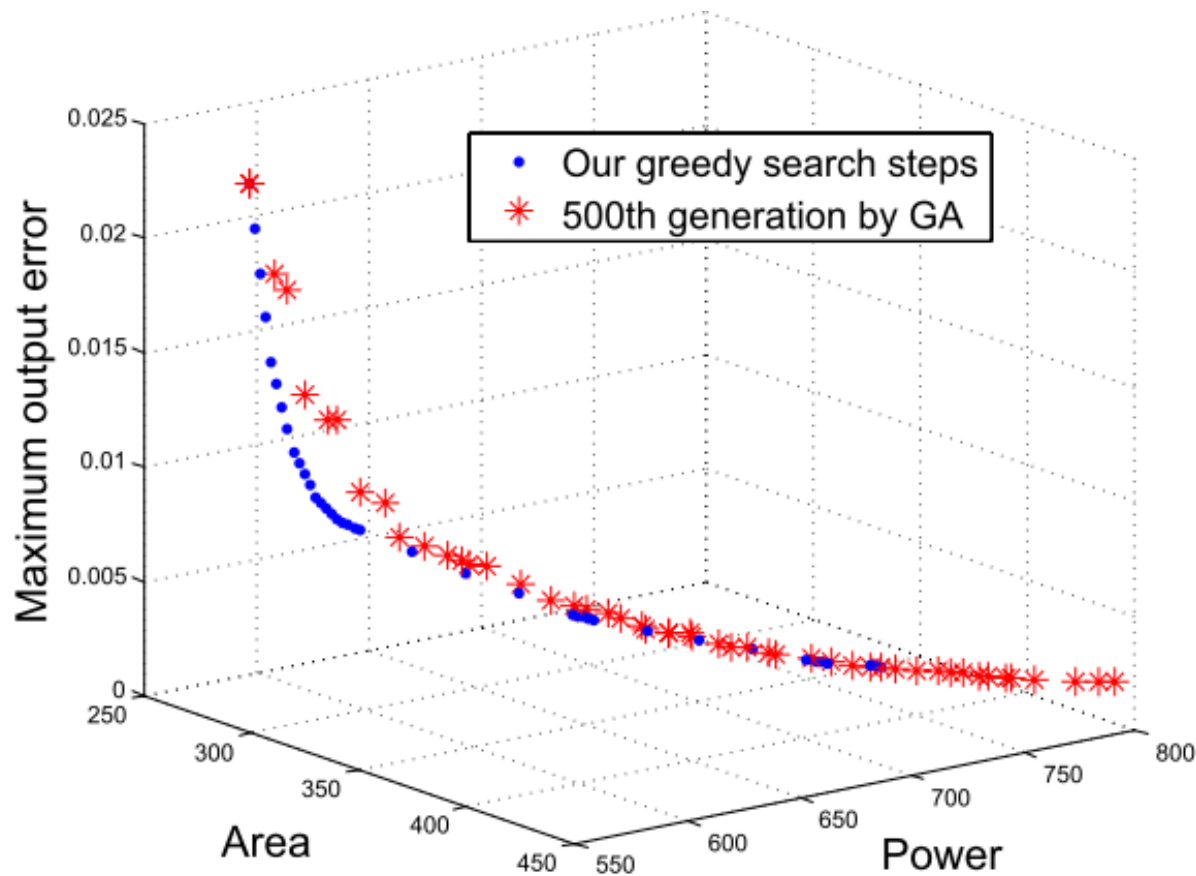
# Hardware Consumption Analysis

- Complexity analysis
- Quick prototype



# Hardware Consumption Analysis

- Complexity analysis
- Quick prototype



# Design Trade-off

Implement the best HW realization. **Best??**



**Flexibility**  
**Complexity**



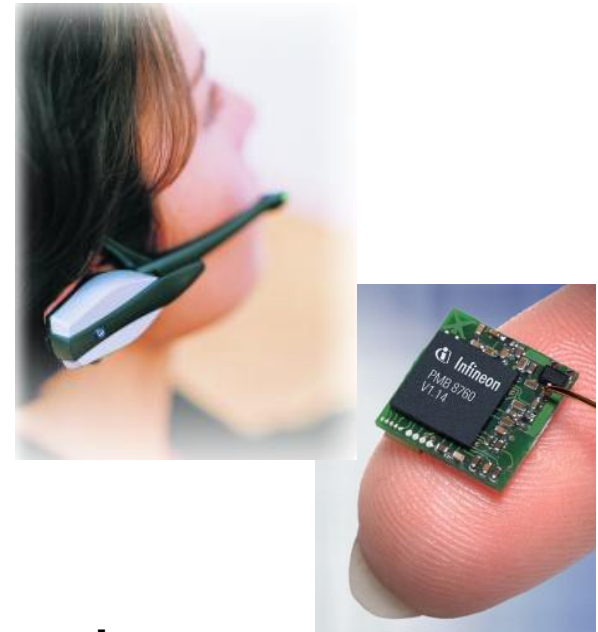
- **Processors**
- **FPGAs**



**Low power**  
**Low cost**  
**Flexibility**



- **Processors**
- **Dedicated HW**



**Lower power**  
**Lower cost**



- **Dedicated HW**
- **Processors**



# Design Trade-off

Implement the best HW realization. **Best??**

Different applications, different demands...  
Thus, "*just good enough*" is the best in engineering.

Try to find a **BALANCE** between effort and cost!

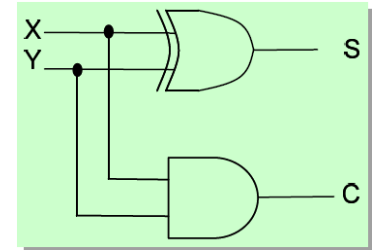
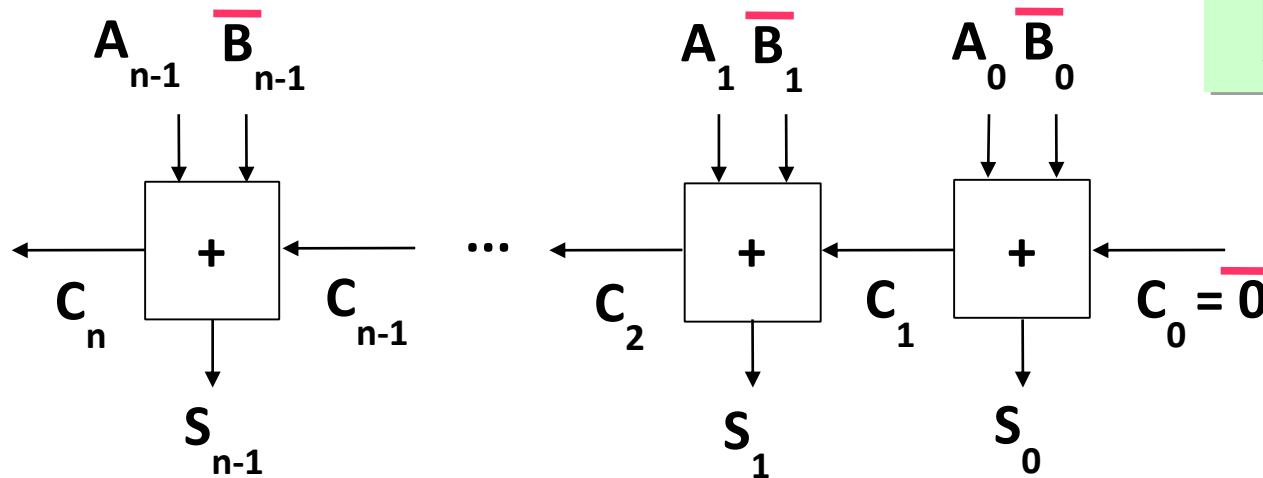


# Overview

- Fixed-Point Representation
- **Add/Subtract**
- Multiplication
- Timing & Techniques to Reduce Delay



# Add/Subtract



▣ The HW for sum/difference (S) does NOT care about signed/unsigned

## ▣ Overflow

- Unsigned overflow =  $C_n$
- Signed overflow =  $C_n \oplus C_{n-1}$
- True sign =  $S_{n-1} \oplus \text{signed overflow}$   
 $= (A_{n-1} \oplus B_{n-1} \oplus C_{n-1}) \oplus (C_n \oplus C_{n-1}) = A_{n-1} \oplus B_{n-1} \oplus C_n$



# Signed Overflow Example

## 4-Bit signed addition

6+7 = 13, outside [-8..7]

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

$C_4 = 0$   
 $C_3 = 1$

$C_n \oplus C_{n-1} = C_4 \oplus C_3 = 0 \oplus 1 = 1 \Leftrightarrow$   
Carry-outs different  $\Leftrightarrow$  Signed overflow

$S_{n-1} \oplus \text{signed overflow} =$   
 $A_{n-1} \oplus B_{n-1} \oplus C_n = A_3 \oplus B_3 \oplus C_4 = 0 \oplus 0 \oplus 0 = 0 \Leftrightarrow$  True sign = Positive/zero

## Overflow Check in Hardware?

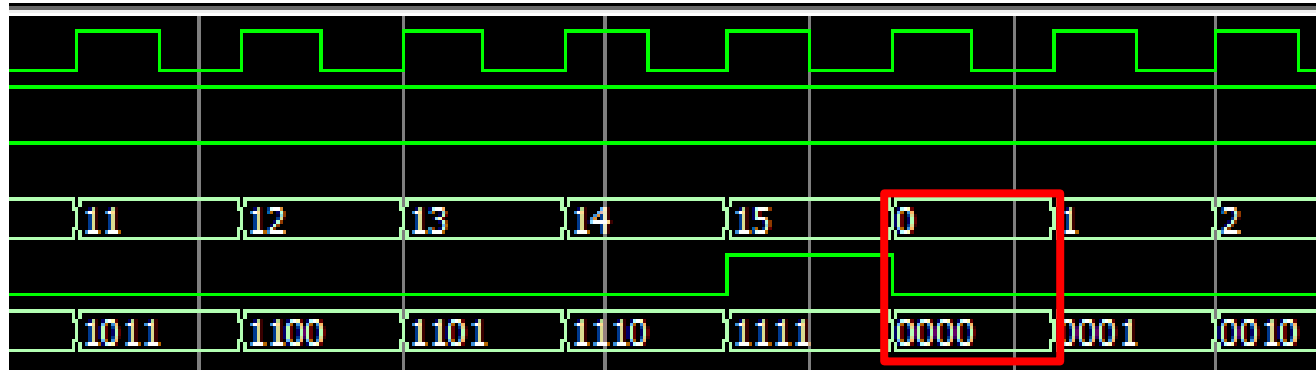




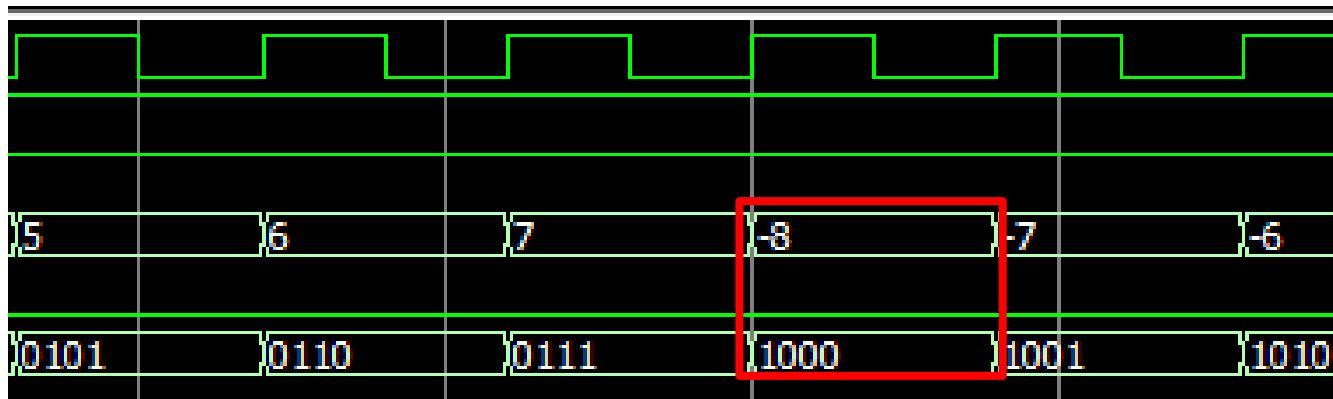
# Overflow in Hardware

## Hardware does not take care of the overflow for you

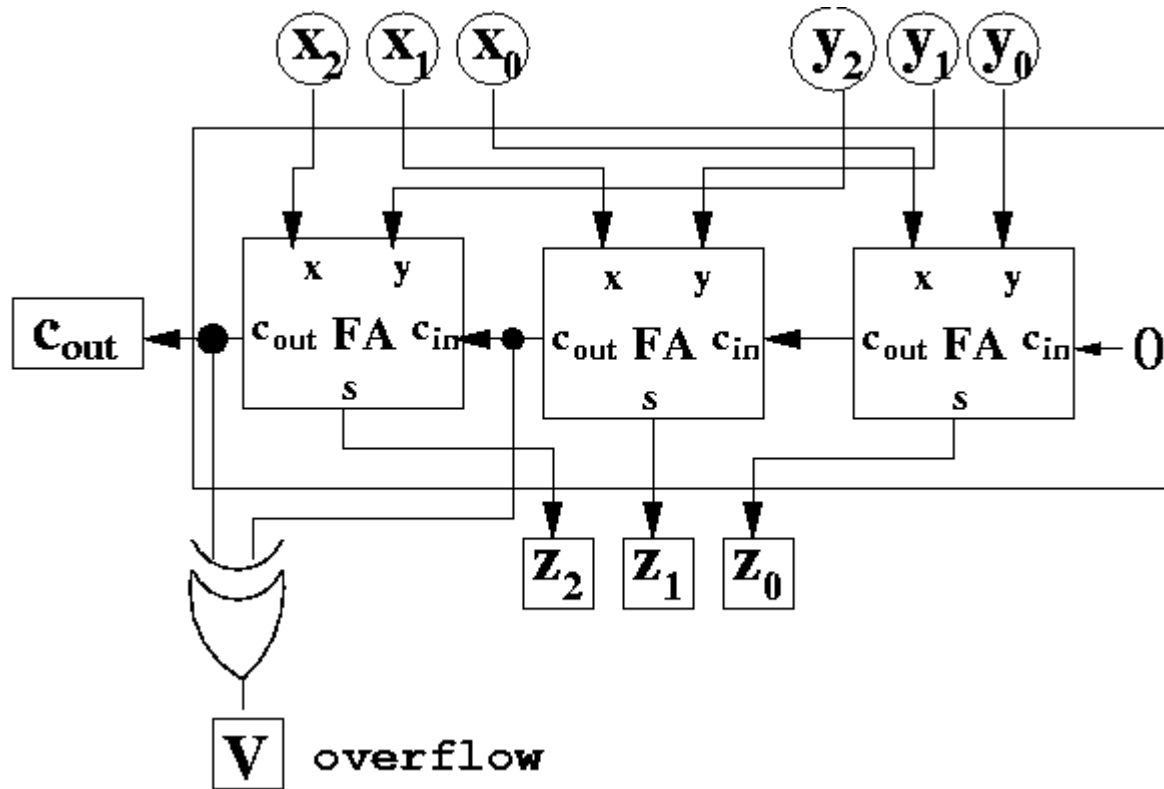
- Unsigned



- Signed



# Overflow in Hardware



***Saturation or wrap-around or 1 more bit***



# Two's Complement Signed Extension

□ To add two numbers, we should represent them with the **same number of bits**: **0100**+**1100**

- If we just pad with **zeroes** on the left:

4-bit

**0100** (4)

**1100** (-4)

8-bit

**00000100** (still 4)

**00001100** (12, not -4)

- Instead, replicate the MS bit -- **the sign bit**:

4-bit

**0100** (4)

**1100** (-4)

8-bit

**00000100** (still 4)

**11111100** (still -4)



# Decimal Mark in Hardware

- Matlab aligns the decimal mark automatically

$$1.32+100.2343= 101.5543$$

- Hardware **does NOT**

- Decimal mark is just a concept

$$01.100+001.01=?$$

$$10001$$

- You need to align the decimal mark manually

$$001.100+001.010=010.110$$



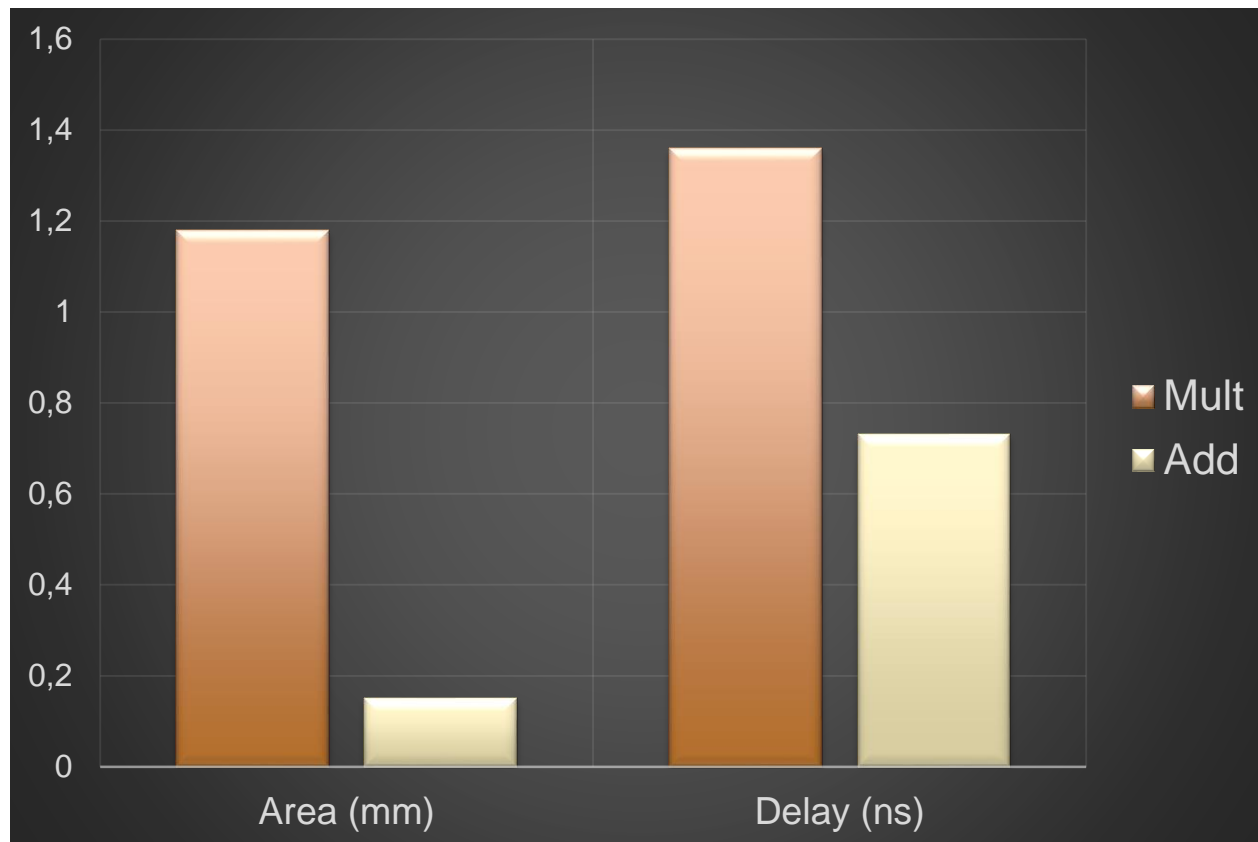
# Overview

□ Fixed-Point Representation

□ Add/Subtract

□ **Multiplication**

□ Timing & Techniques to Reduce Delay



# Array Multiplier (unsigned)

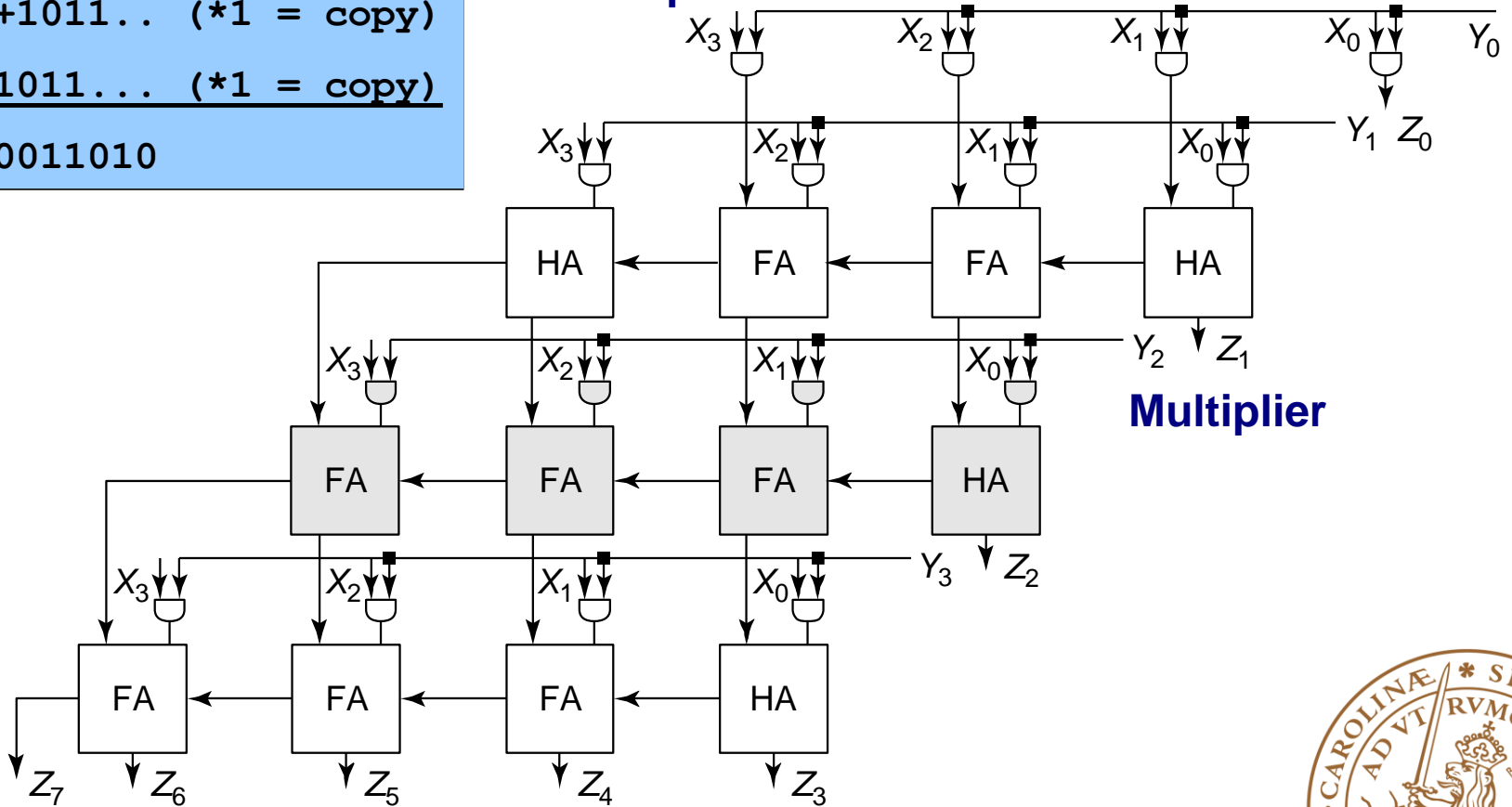
```

1011 * 1110
-----
0000 (*0 = zero)
+1011. (*1 = copy)
+1011.. (*1 = copy)
+1011... (*1 = copy)
-----
10011010
    
```

## Direct Mapping

- Horizontal** : partial product using AND
- Vertical** : shift-add of partial product

Multiplicand



# Don't Forget ... Signed Multiplication

$$\begin{array}{r} 1011 \quad -5x \\ 0011 \quad +3 \\ \hline \end{array}$$

?

$$\begin{array}{r} \hline 11110001 \quad -15 \end{array}$$









# Signed Multiplication

## □ Or directly perform signed multiplication:

- Multiplier & Multiplicand: positive or negative
- Sign extend the partial products when adding up
- Subtract instead of adding last partial product

Ex:

- 5	1011	multiplicand
x - 3	x 1101	multiplier
15	00000	partial product
	11011	shifted multiplicand
	111011	partial product
	00000	shifted multiplicand
	1111011	partial product
	11011	shifted multiplicand
	11100111	partial product
	00101	shifted and negated multiplicand
	00001111	product

Added bit using sign extension → (points to the leading 1s in the partial products)

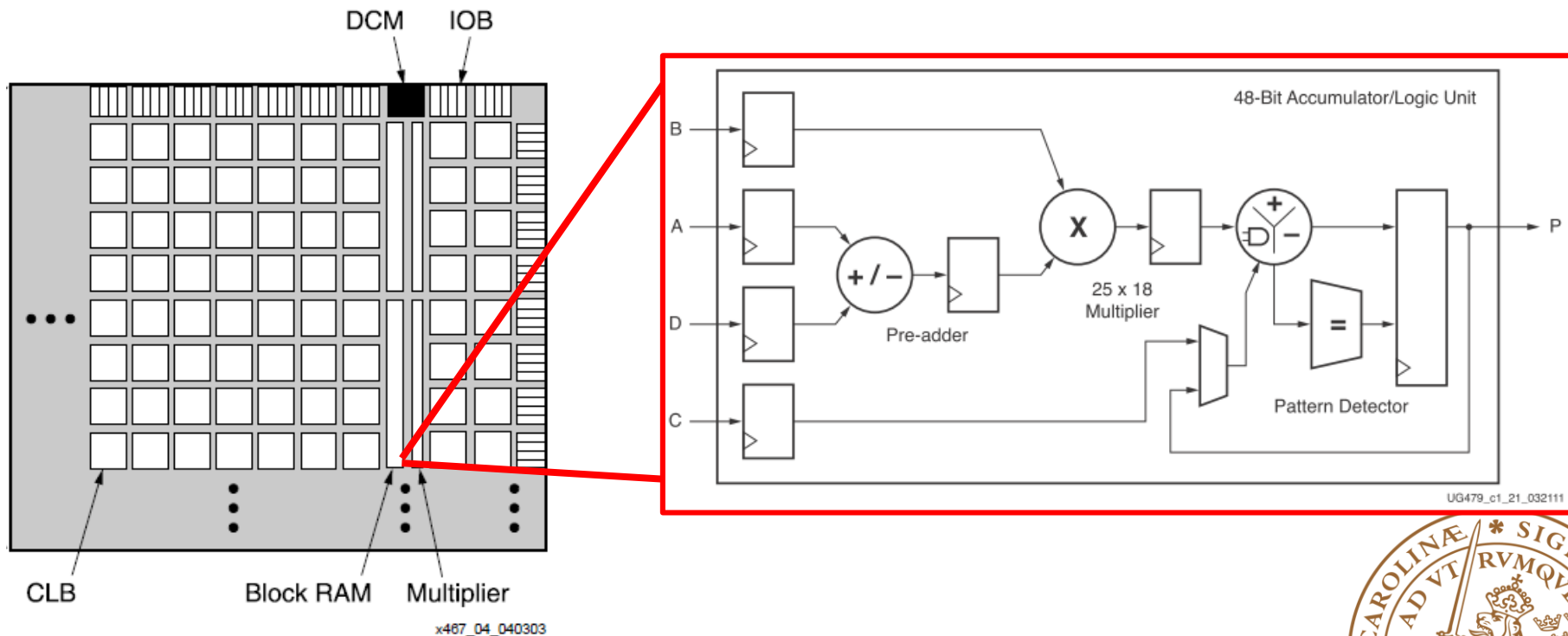


# Multiplier in Xilinx FPGA

## Embedded DSP48E1

- $25 \times 18$  embedded multipliers (**two's-complement multiplier**)
- Using Embedded Multipliers in Artix-7 FPGAs

[http://www.xilinx.com/support/documentation/user\\_guides/ug479\\_7Series\\_DSP48E1.pdf](http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf)



# Multiplier in Xilinx FPGA

## □ DSP48E1

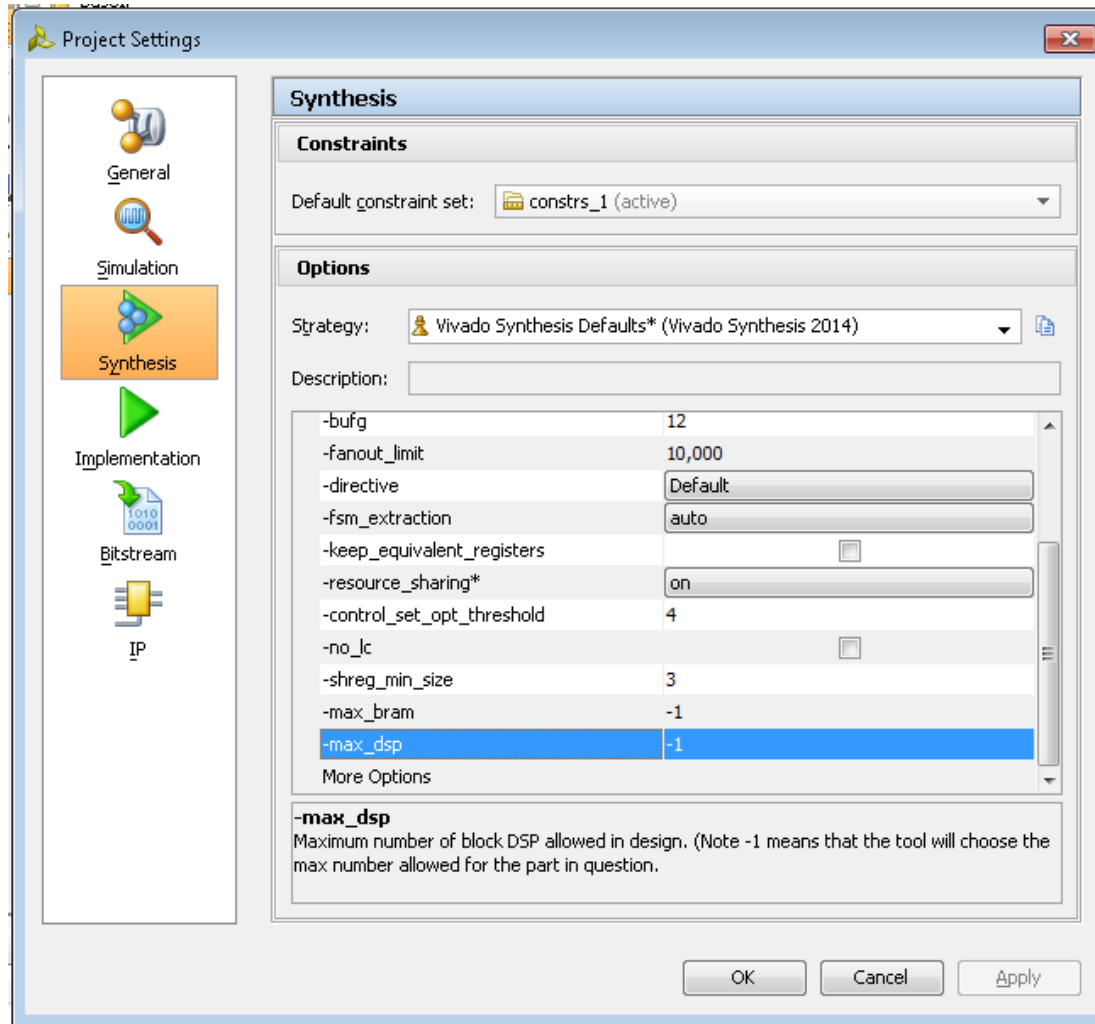
- $25 \times 18$  two's-complement multiplier
- 48-bit accumulator
- Single-instruction-multiple-data (SIMD) arithmetic unit: Dual 24-bit or quad 12-bit add/subtract/accumulate
- Optional pipelining and dedicated buses for cascading

## □ Use suggestions from Xilinx

- Use signed values in HDL source (setting MSB 0 for unsigned)
- Pipeline for performance and lower power, both in the DSP48E1 slice and fabric
- Use the configurable logic block (CLB) carry logic to implement small multipliers, adders, and counters



# Multiplier in Xilinx FPGA



# Multiplier in Xilinx FPGA

## USE\_DSP48 Verilog Example

```
(* use_dsp48 = "yes" *) module test(clk, in1, in2, out1);
```

## USE\_DSP48 VHDL Example

```
attribute use_dsp48 : string;  
attribute use_dsp48 of P_reg : signal is "no"
```

```
architecture archi of use_dsp48_example is  
    signal s : std_logic_vector (7 downto 0);  
    attribute use_dsp48 : string;  
    attribute use_dsp48 of s : signal is "yes";  
begin  
    process (clk)  
    begin  
        if clk'event and clk = '1' then  
            s <= s + a;  
        end if;  
    end process;  
end archi;
```

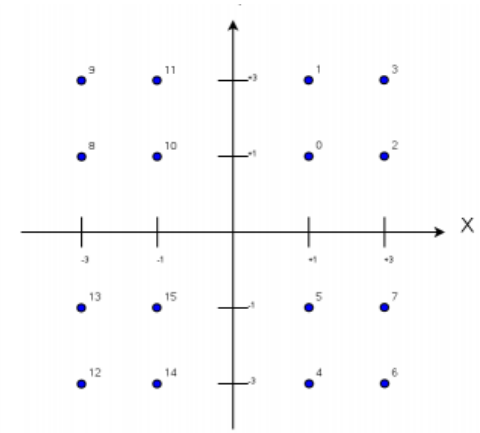


# Constant Multiplication

## Examples:

- **Twiddle factor** in FFTs
- **Constellation points** in wireless communication

$$y_j = \sum_{k=0}^{n-1} e^{-\frac{2\pi i}{n}jk} x_k$$



Software may be not smart enough to optimize

Designer should optimize that multiplications with a small constant is accomplished by **shifts & adds**

Some numerical examples:

\*2 (\*10<sub>2</sub>): multiplicand << 1

\*3 (\*11<sub>2</sub>): multiplicand << 1 + multiplicand

\*5 (\*101<sub>2</sub>): multiplicand << 2 + multiplicand

\*255 (\*11111111<sub>2</sub>): ?

**multiplicand << 8 – multiplicand**



# Overview

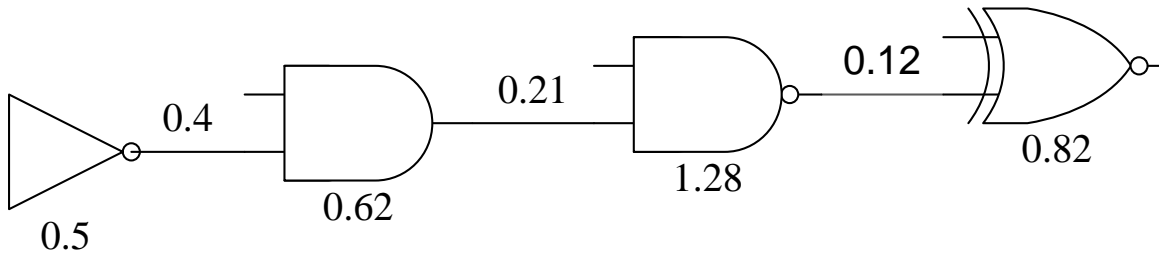
- Fixed-Point Representation
- Add/Subtract
- Multiplication
- **Timing & Techniques to Reduce Delay**



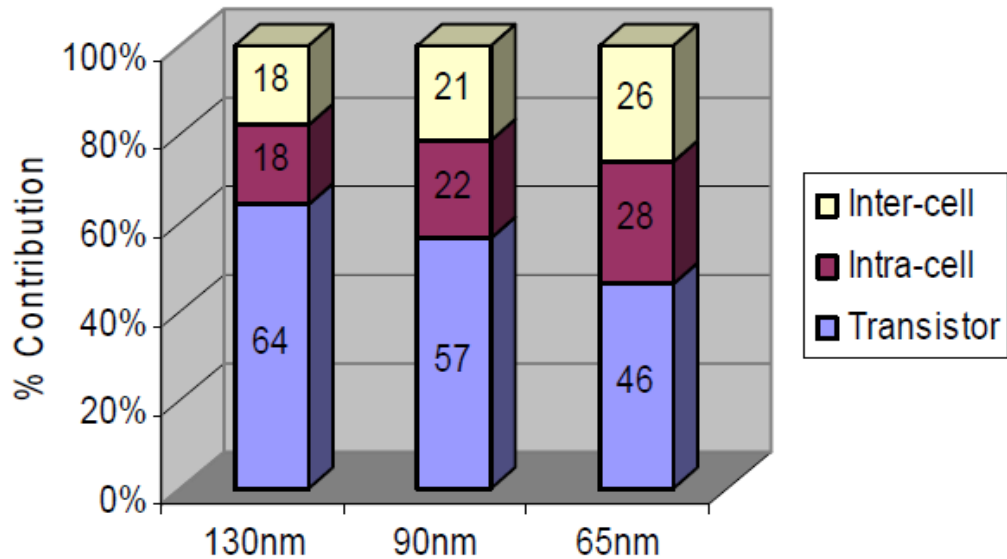


# Combinational Circuit Timing

□ Path delay = cell delay + net delay

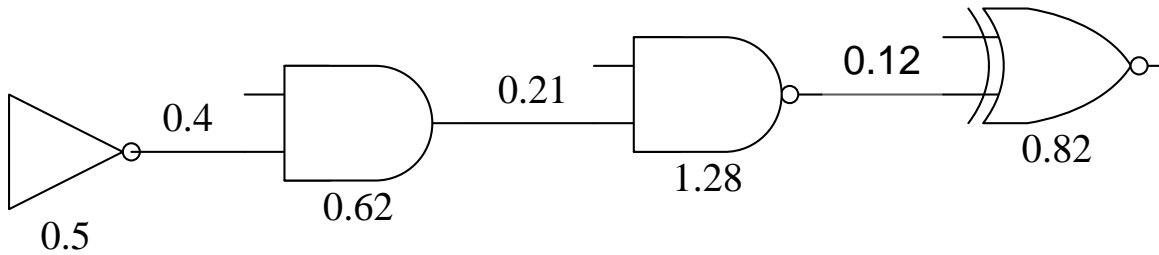


$$\text{Path Delay} = 0.5 + 0.4 + 0.62 + 0.21 + 1.28 + 0.12 + 0.82 = 3.95 \text{ ns}$$

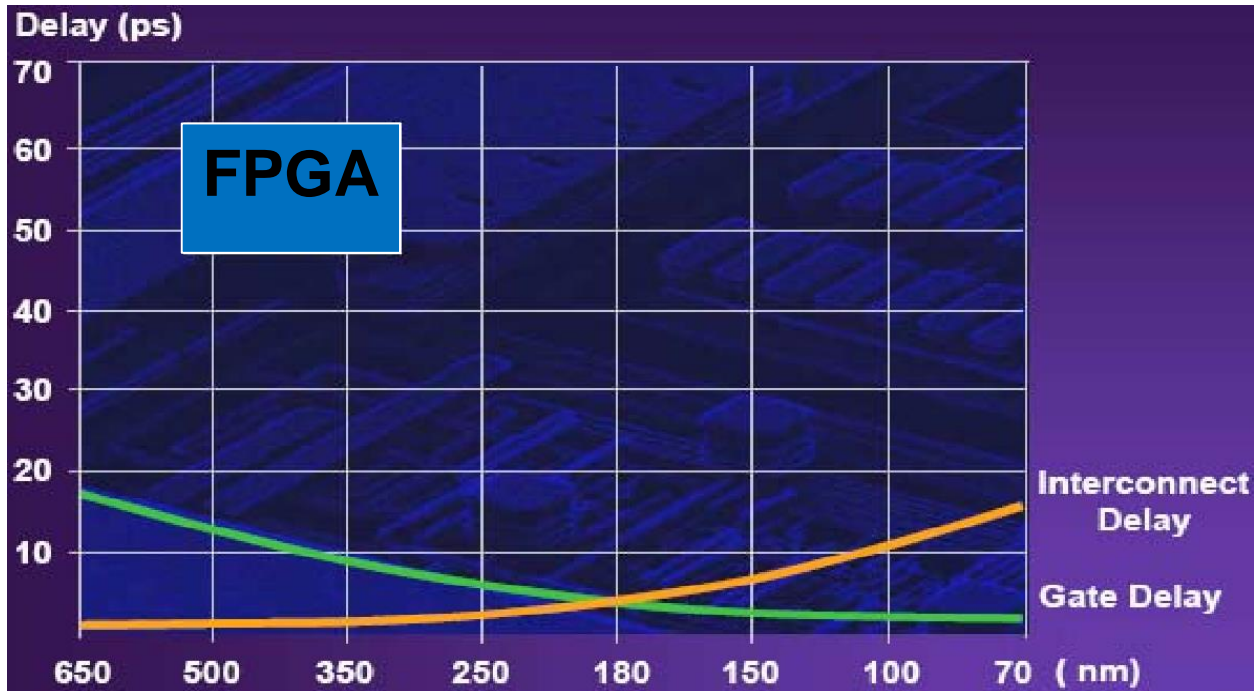


# Combinational Circuit Timing

□ Path delay = cell delay + net delay

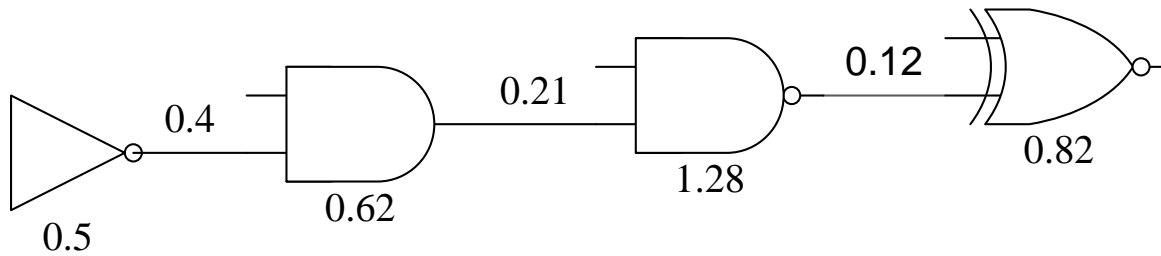


$$\text{Path Delay} = 0.5 + 0.4 + 0.62 + 0.21 + 1.28 + 0.12 + 0.82 = 3.95 \text{ ns}$$



# Combinational Circuit Timing

□ Path delay = cell delay + net delay



$$\text{Path Delay} = 0.5 + 0.4 + 0.62 + 0.21 + 1.28 + 0.12 + 0.82 = 3.95 \text{ ns}$$

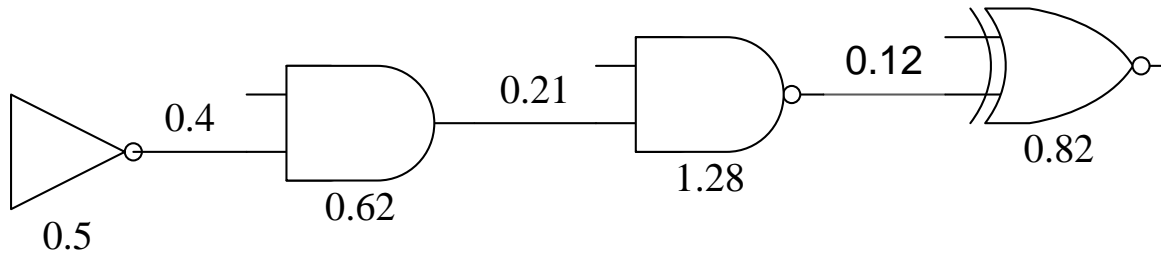
□ How to reduce processing delay

- Reduce cell delay? Standard-cell library (Digital-IC)
- Reduce net delay? Place & Route (Floor Plan)



# Combinational Circuit Timing

□ Path delay = cell delay + net delay



$$\text{Path Delay} = 0.5 + 0.4 + 0.62 + 0.21 + 1.28 + 0.12 + 0.82 = 3.95 \text{ ns}$$

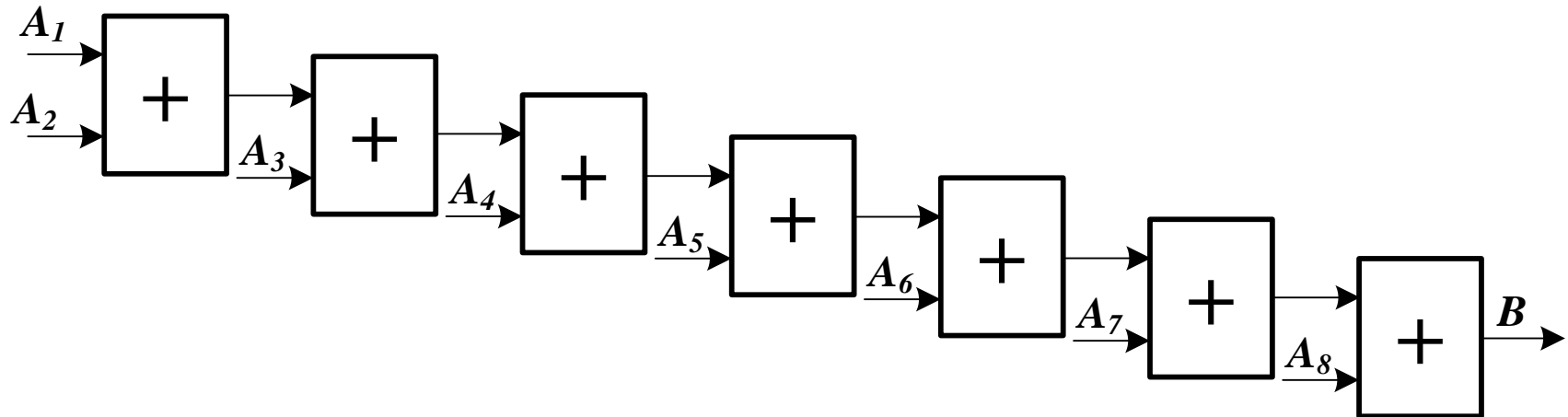
□ How to reduce processing delay

- Reduce cell delay? Standard-cell library (Digital-IC)
- Reduce net delay? Place & Route
- Or we can change the architecture



# Example1: Higher-Level Adder Chain

□ Calculate:  $B = A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8$

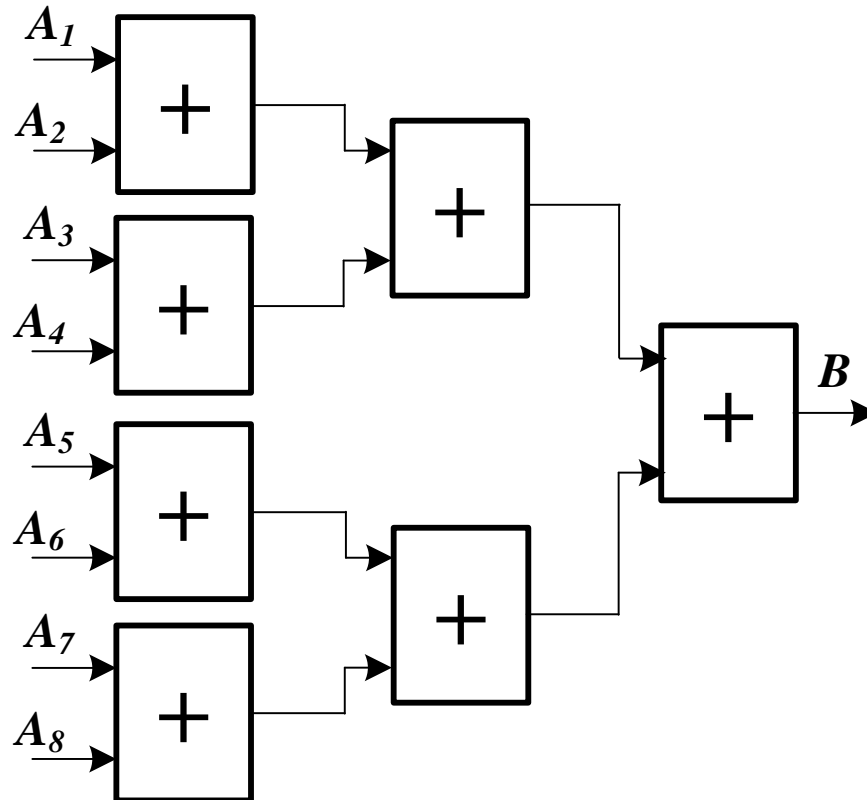


Cascaded-Chain



# Higher-Level

$$B = [(A_1 + A_2) + (A_3 + A_4)] + [(A_5 + A_6) + (A_7 + A_8)]$$



Tree



# Cascade vs. Tree

## □ Comparison of n-input adder

- Cascading chain:
  - Area:  $(n-1)$  full adder
  - Delay:  $(n-1)$
  - Flexibility: easy to modify (scale)
- Tree:
  - Area:  $(n-1)$  full adder
  - Delay:  $\log_2 n$
  - Flexibility: not so easy to modify



# Thanks!

