EITF35: Introduction to Structured VLSI Design

Part 2.1.1: Combinational circuit

Liang Liu
liang.liu@eit.lth.se

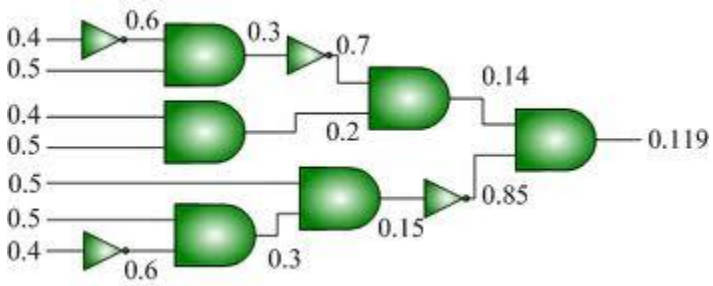
Why Called “Combinational” Circuits?

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.

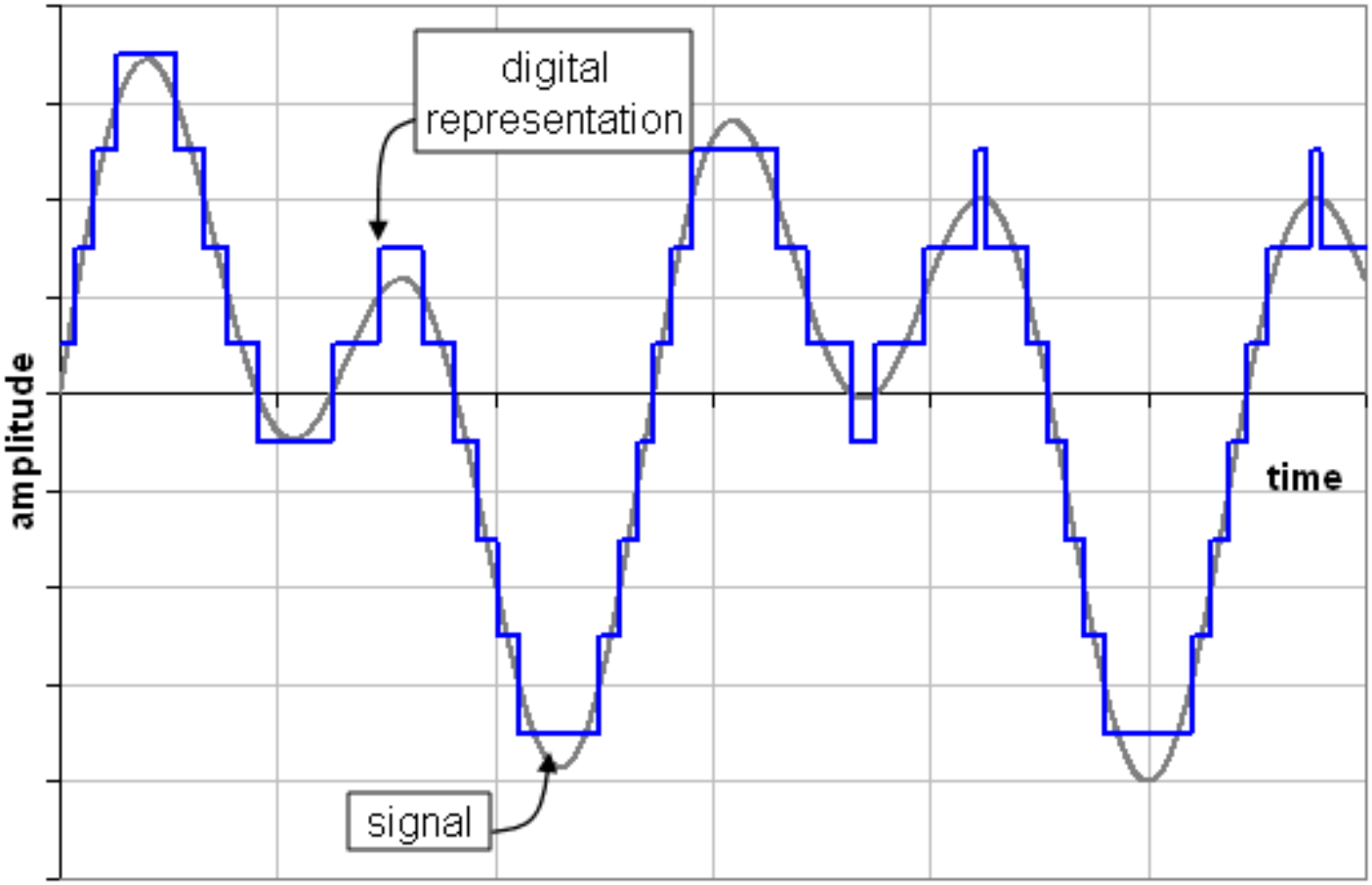


Two basic components

- Operands (Data type)

- Operations

‘Digital’- quantization



What does it mean?

00100000010010000110100100100001

What does it mean?

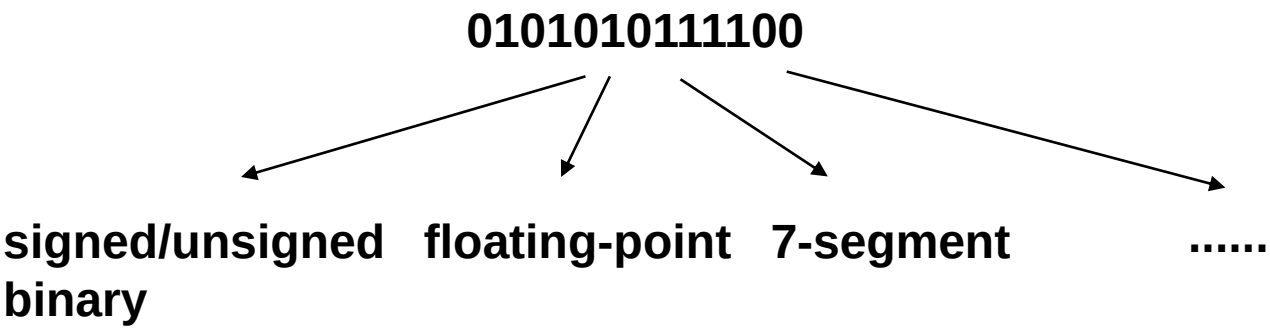
00100000010010000110100100100001

001000	00010	01000	0110100100100001
instruction	source	target	immediate
ADDI	2	8	26913

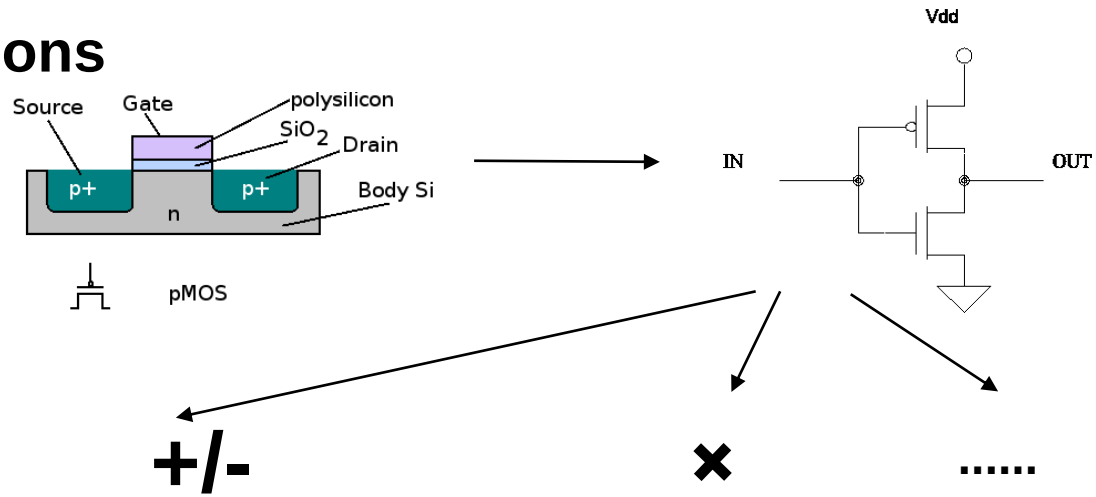
ADDI R8,R2,26913

Two basic components

- **Operands (Data type)**



- **Operations**



- **Check what is in the library!**

Data Representation

Unsigned

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

Signed (Two's complement)

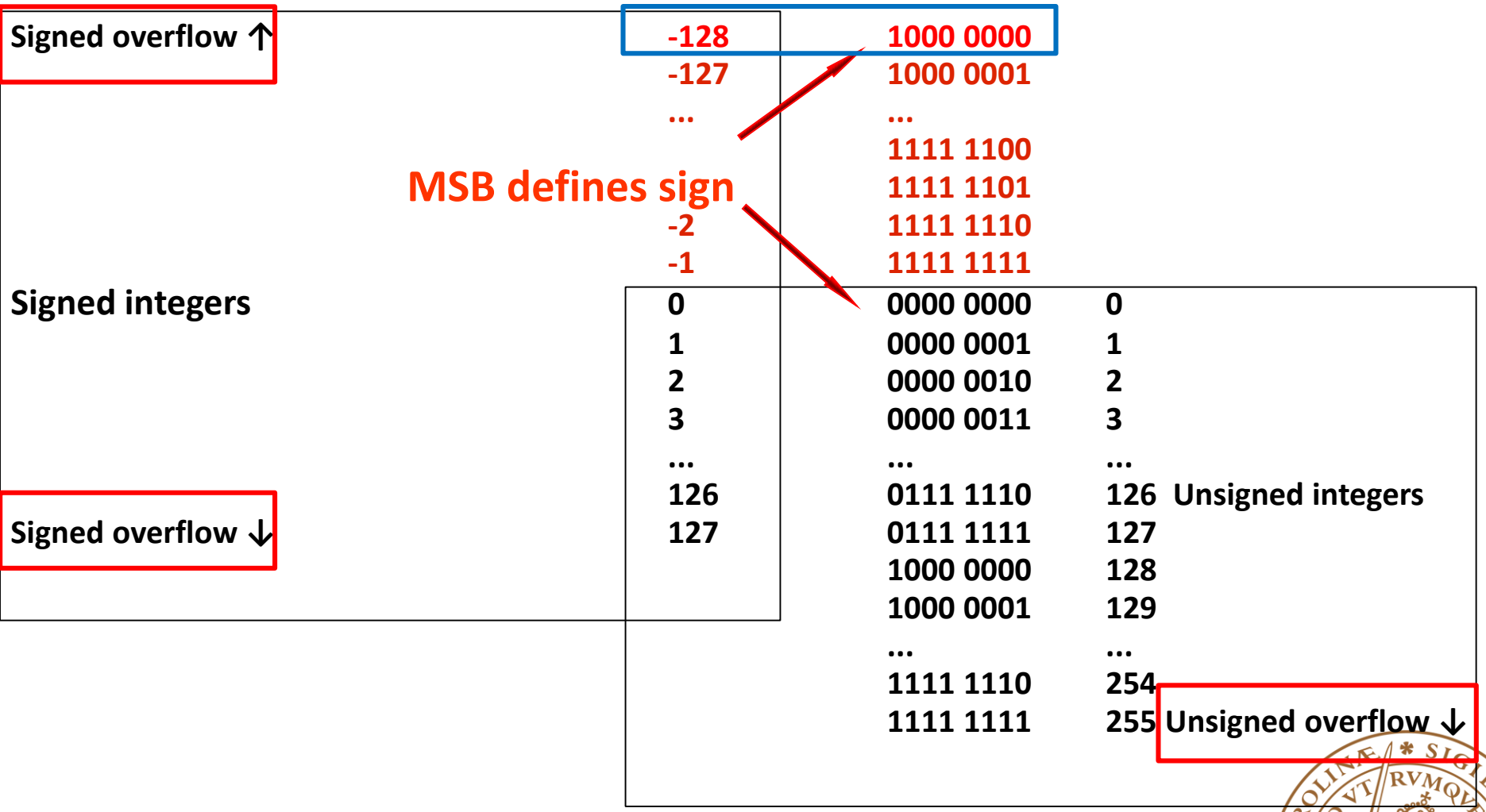
- The result of **subtracting the number from 2^{N-1}**
- Inverting all bits and **adding 1**

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

$11110100_2 = -12_{10}$

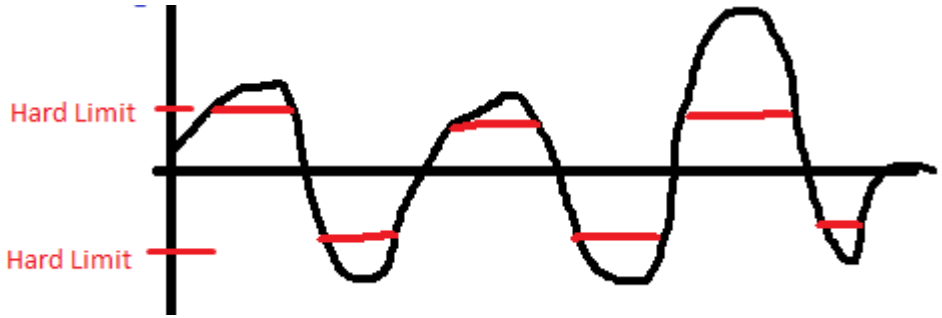
Sign bit **2's complement**

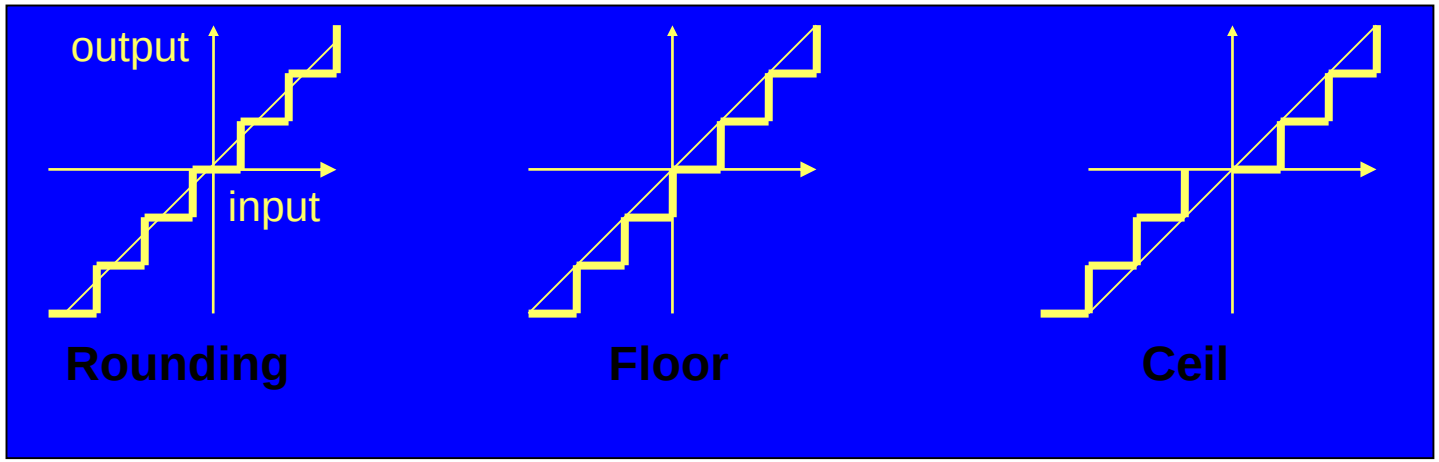
8-bit Signed/Unsigned Integers



Finite Word-Length Effect

- **Overflow**
  - Saturation
- **Quantization error**
  - Round
  - Truncation



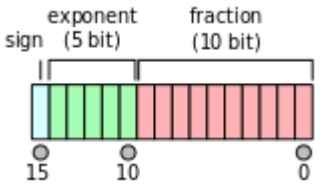


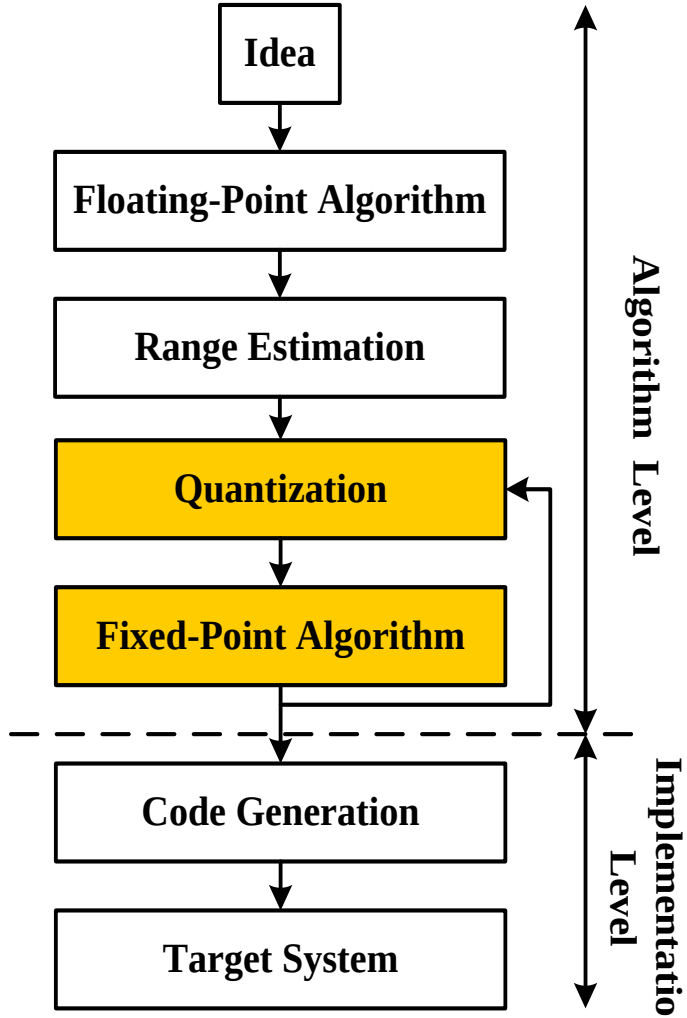
round(0.51)=1 *floor(0.51)=0* *ceil(0.49)=1*

Will learn more in DSP-Design course

Fixed-Point Design

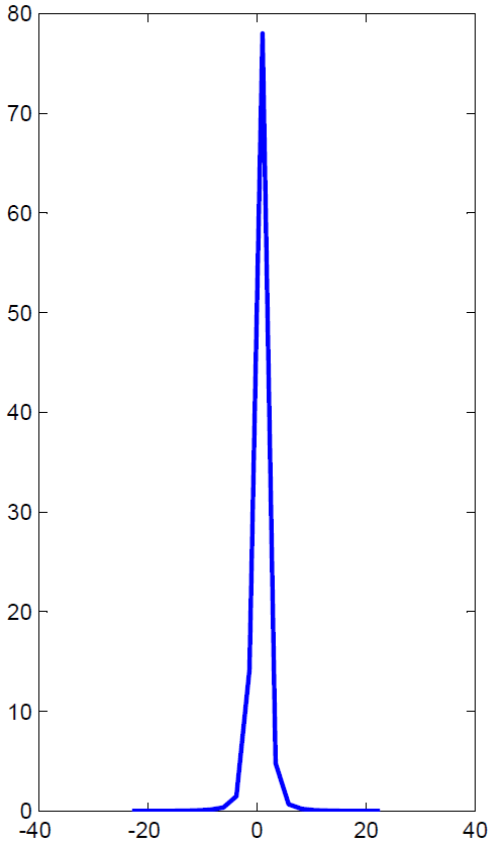
- **DSP algorithms**
  - Often developed in floating point
  - Later mapped into **fixed point** for digital hardware realization
- **Fixed-point digital VLSI**
  - Lower area
  - Lower power
  - Quantization error & small dynamic range

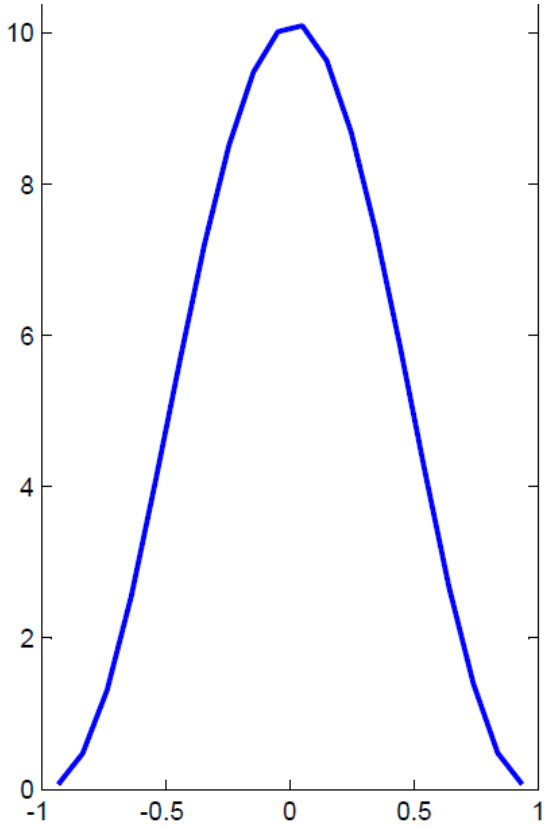




“Optimum” Word-Length

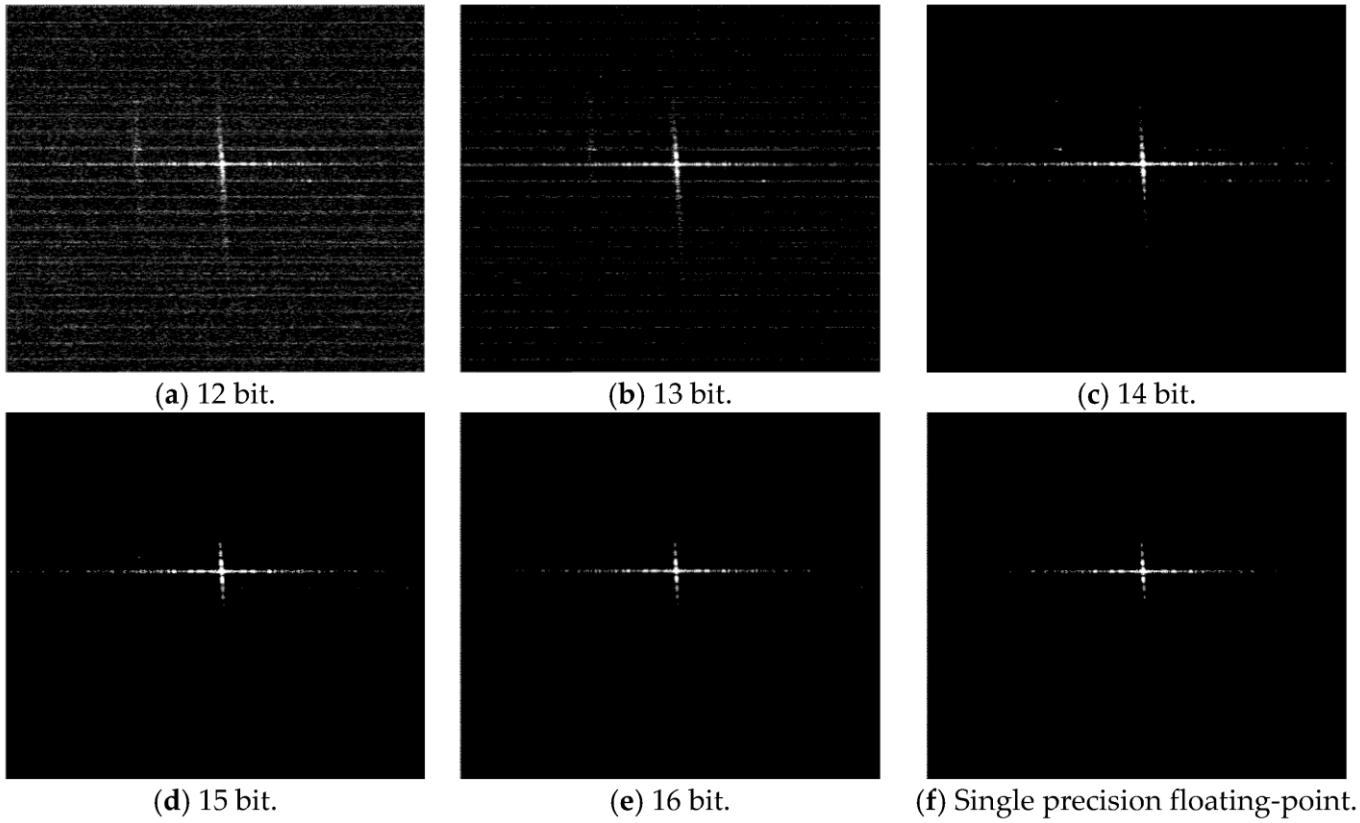
- **Range Analysis**





“Optimum” Word-Length

- Range Analysis
- Fixed-point Simulation

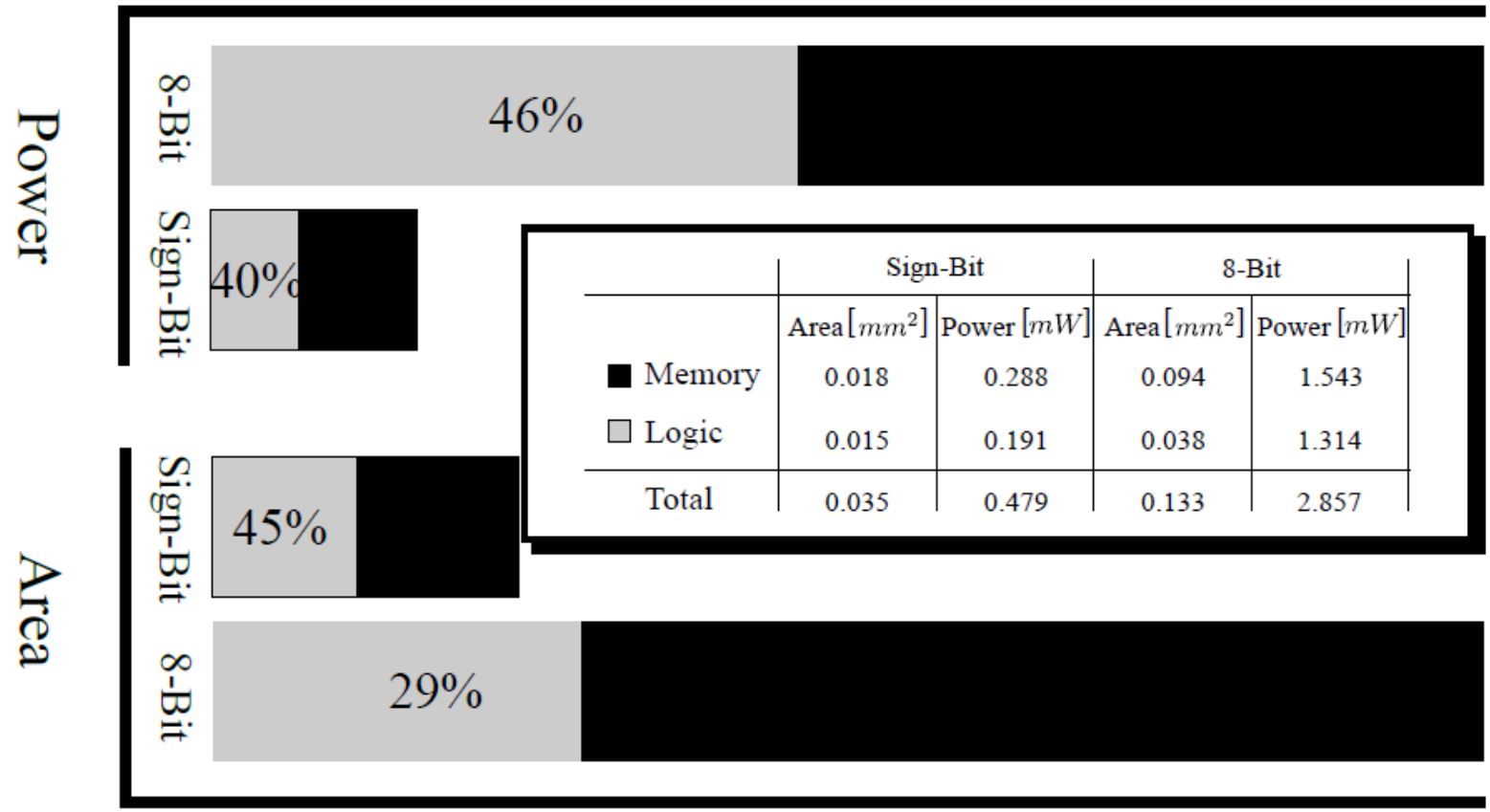


(a) 12 bit. (b) 13 bit. (c) 14 bit.

(d) 15 bit. (e) 16 bit. (f) Single precision floating-point.

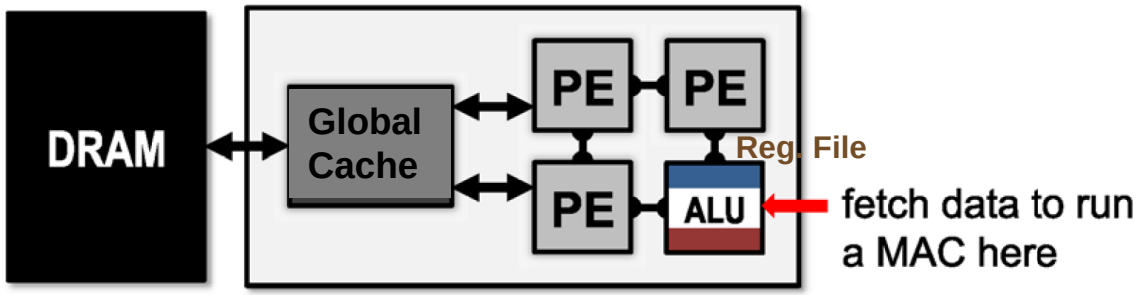
Hardware Consumption Analysis

- Complexity analysis
- Quick prototype

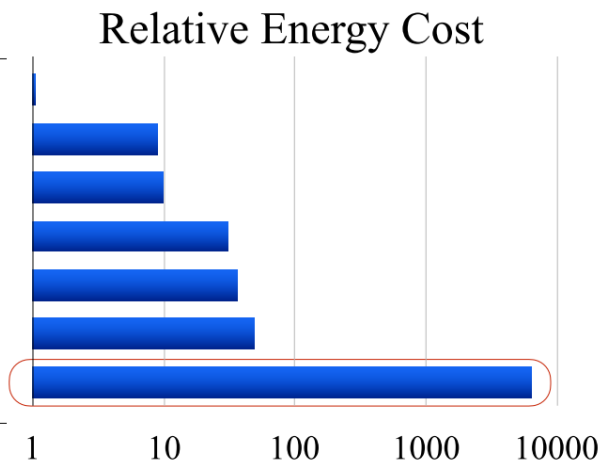


	Sign-Bit		8-Bit	
	Area $[mm^2]$	Power $[mW]$	Area $[mm^2]$	Power $[mW]$
Memory	0.018	0.288	0.094	1.543
Logic	0.015	0.191	0.038	1.314
Total	0.035	0.479	0.133	2.857

Where is the cost



Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	**640**



Source: Han Song, "Efficient Methods and Hardware for Deep Learning" & V. Sze et.al. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey"

Design Trade-off

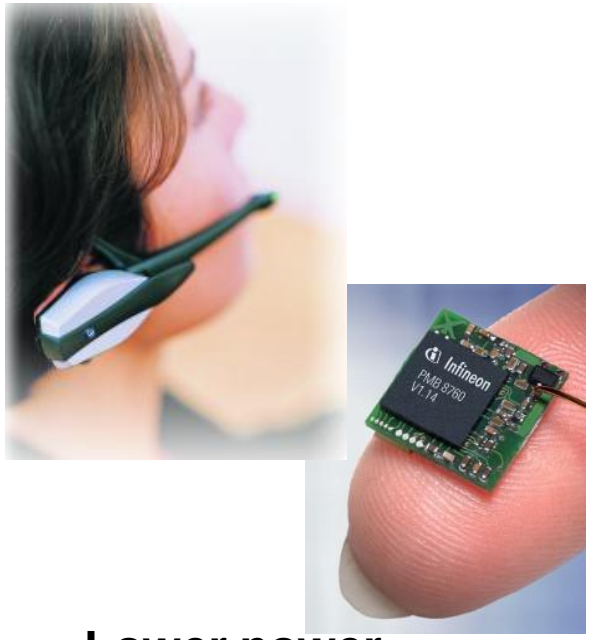
Implement the best HW realization. Best??

Flexibilty
Complexity

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

Low power
Low cost
Flexibilty

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



Lower power
Lower cost

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

Design Trade-off

Implement the best HW realization. Best??

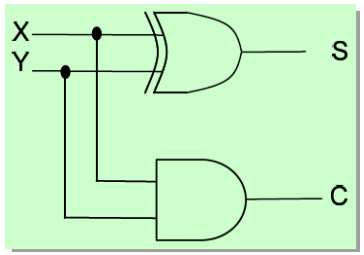
Different applications, different demands...
Thus, ”*just good enough*” is the best in engineering.

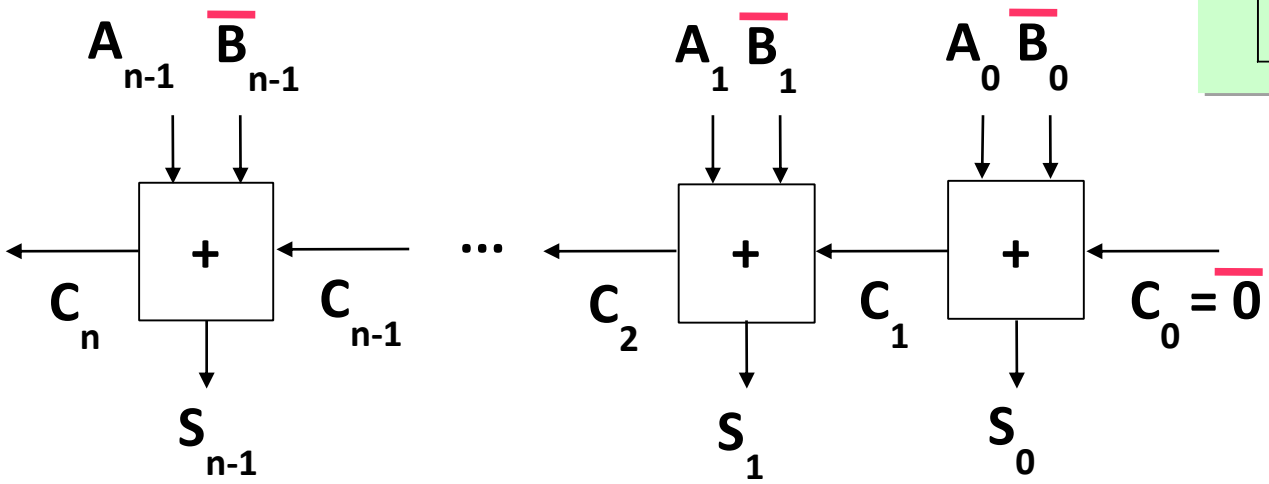
Try to find a BALANCE between effort and cost!

Overview

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

Add/Subtract (Binary)





- **The HW for sum/difference (S) does NOT care about signed/unsigned**
- **Overflow**
  - Unsigned overflow = C_n
  - Signed overflow = $C_n \oplus C_{n-1}$

Signed Overflow Example

- **4-Bit signed addition**

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

$C_4=0$

```
  0110
+0111
 1101
```

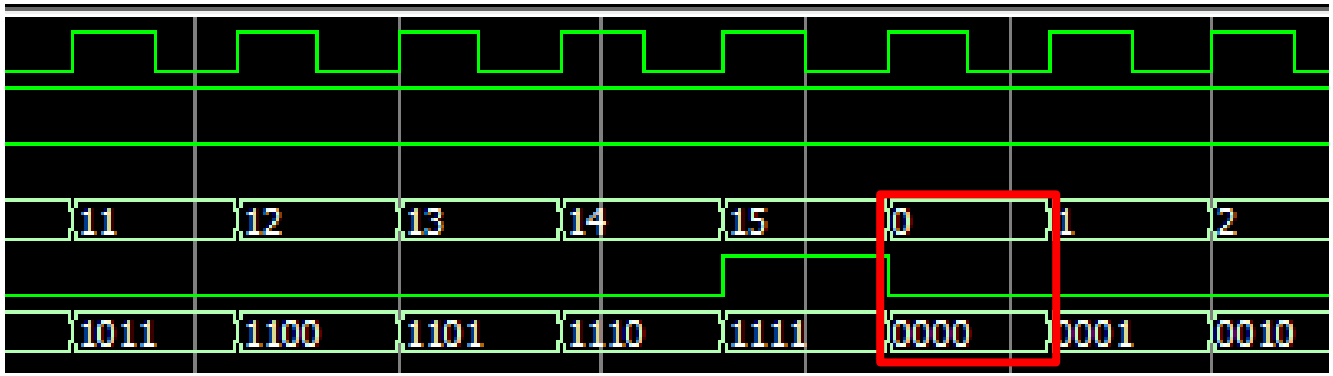
$C_3 = 1$

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

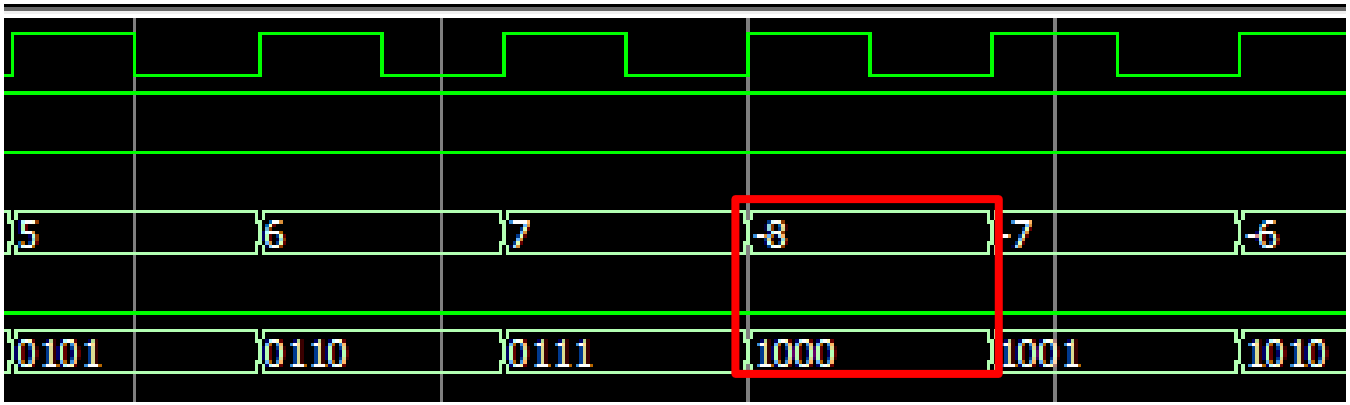
Overflow Check in Hardware?

Overflow in Hardware

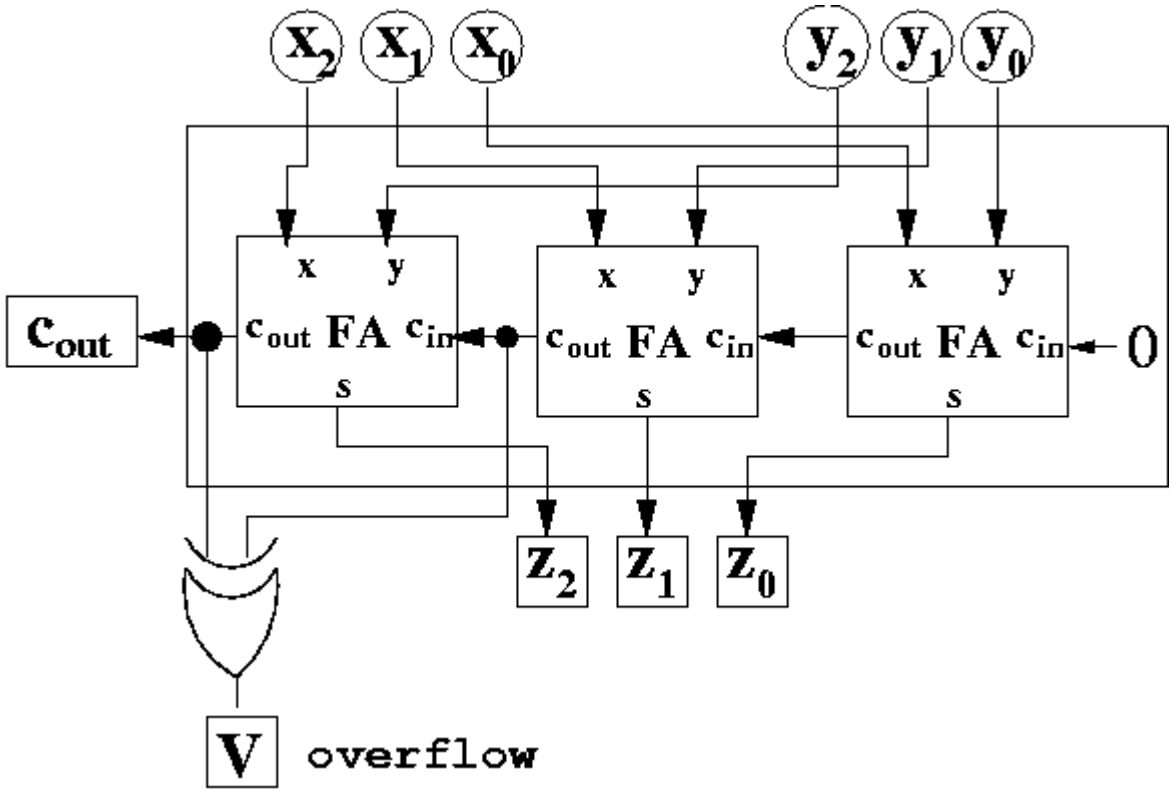
- **Hardware does not take care of the overflow for you**
  - Unsigned



  - 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+11100**
  - If we just pad with zeroes on the left:

4-bit		8-bit	
0100	(4)	00000100	(still 4)
1100	(-4)	00001100	(12, not -4)

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

4-bit		8-bit	
0100	(4)	00000100	(still 4)
1100	(-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 virtual concept

    01.100+001.01=?

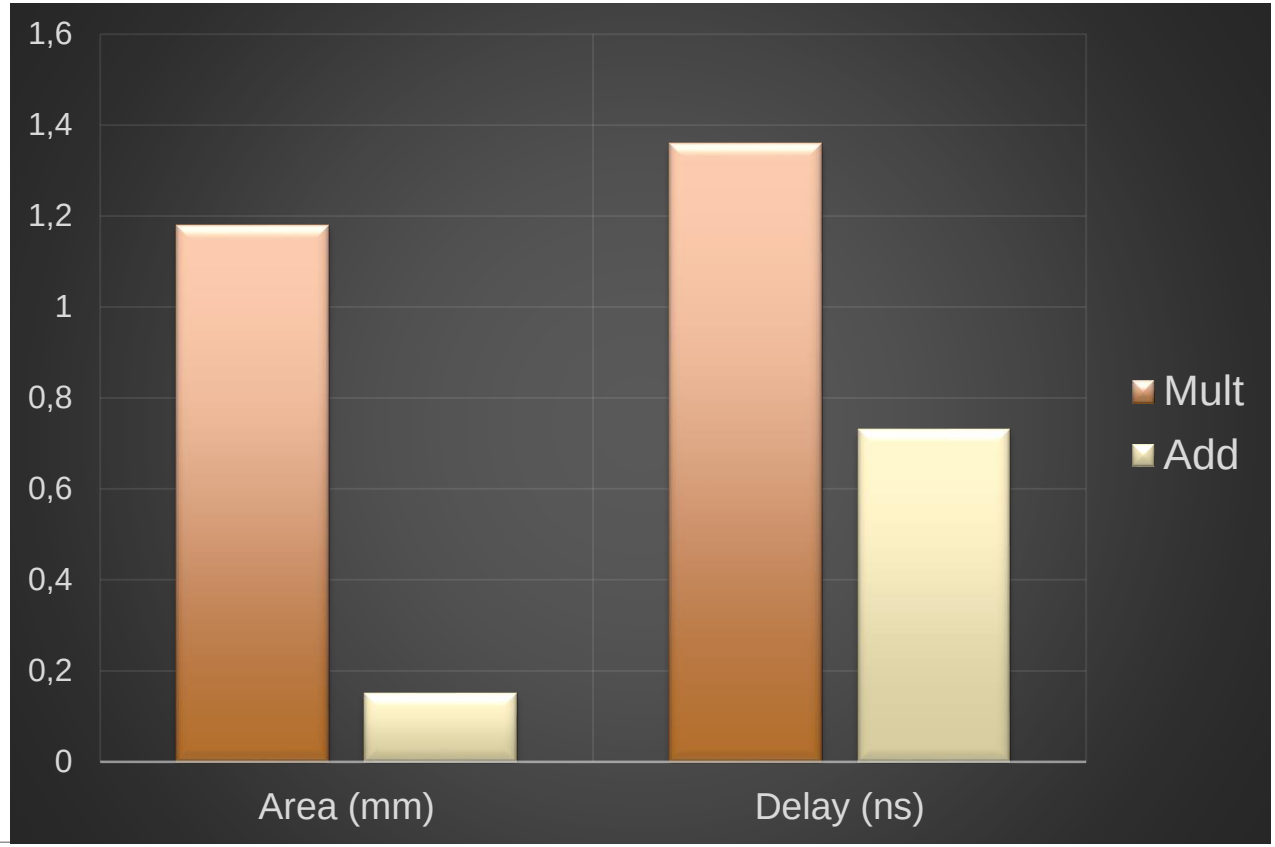
    10001

  - You need to align the decimal mark manually

    001.100+001.010=010.110

Overview

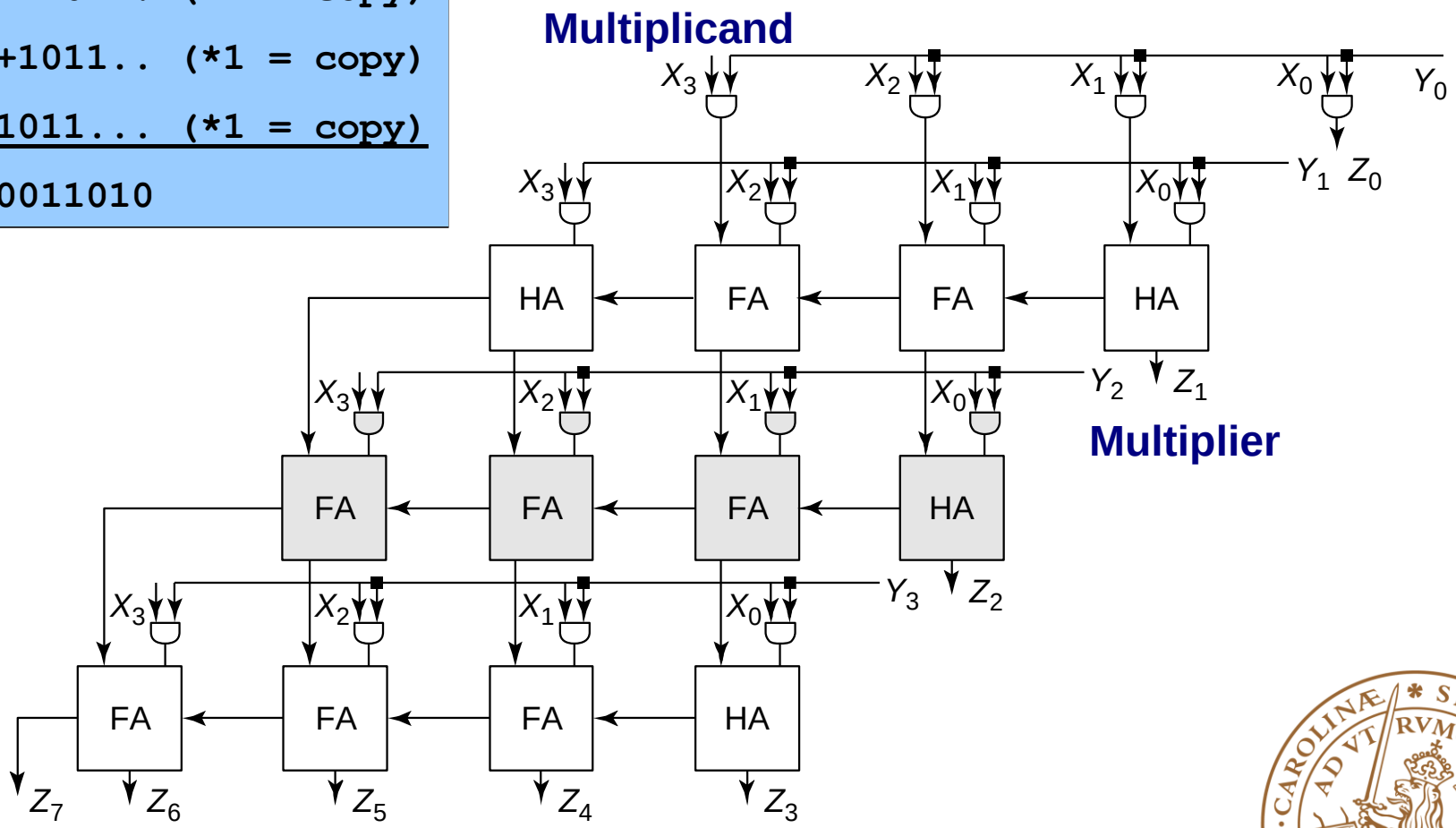
- Fixed-Point Representation
- Add/Subtract
- **Multiplication**
- Timing & Techniques to Reduce Delay



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



Don't Forget ... Signed Multiplication

1 0 1 1	-5x
0 0 1 1	+3

?

1 1 1 1 0 0 0 1	-15

Signed Multiplication

- **Either transform to multiply of non-negative integers:**
  - Record signs and negate any negative factors.
  - Perform unsigned multiplication.
  - Negate product if signs above differ.

abs(-5)=5
abs(3)=3

```
  0 1 0 1     +5x
  0 0 1 1     +3
  ---------
  0 1 0 1
0 1 0 1
0 0 0 0
0 0 0 0
0 0 0 1 1 1 1   +15
```

-1*1*15=-15

Signed Multiplication

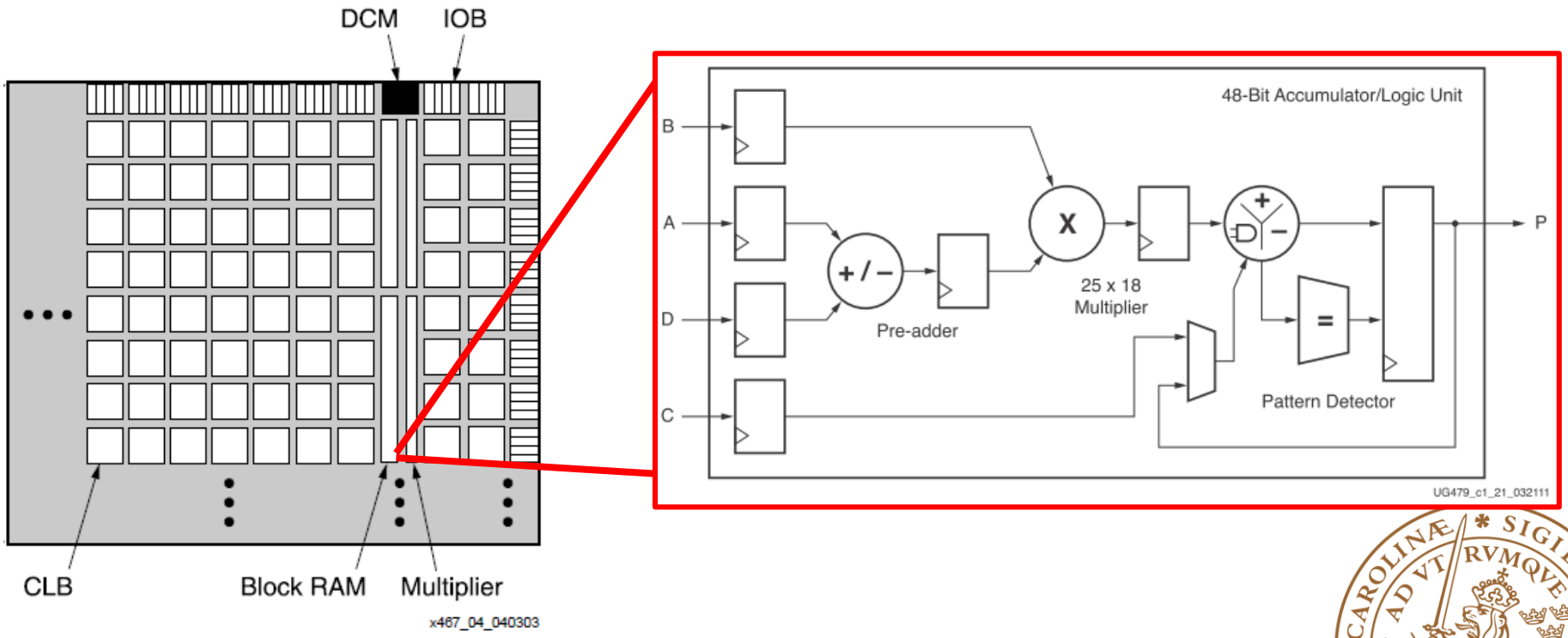
- Or directly perform signed multiplication:
  - Multiplier: positive
  - Multiplicand: positive or negative
  - Sign extend the partial products when adding up

```
          1 0 1 1    -5x
        0 0 1 1      +3
      ------------
1 1 1 1 1 0 1 1
1 1 1 1 0 1 1
0 0 0 0 0 0
0 0 0 0 0
-------------------
1 1 1 1 0 0 0 1    -15
```

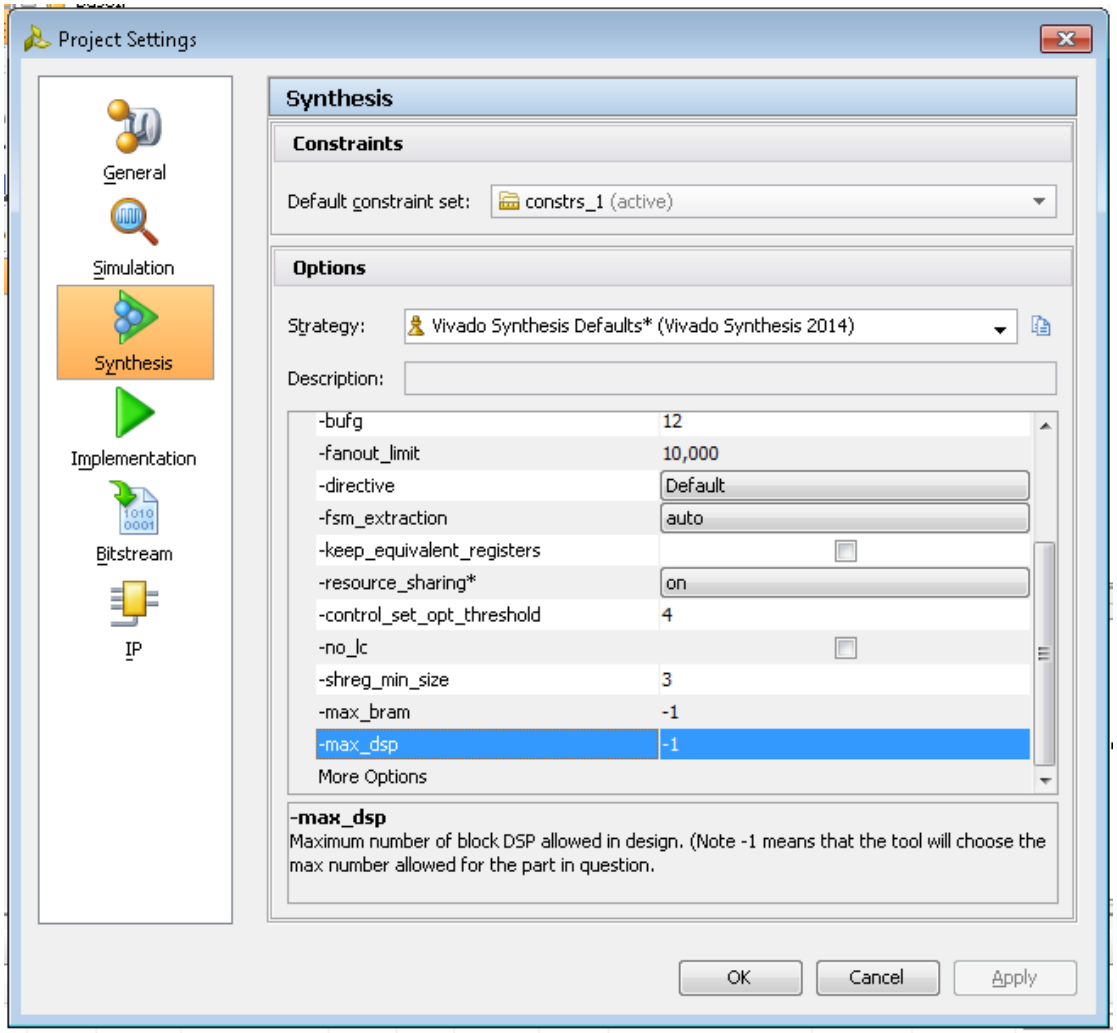
Multiplier in Xilinx FPGA

- **Embedded DSP48E1**
  - 25×18 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



Multiplier in Xilinx FPGA



Multiplier in Xilinx FPGA

USE_DSP48 Verilog Example

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

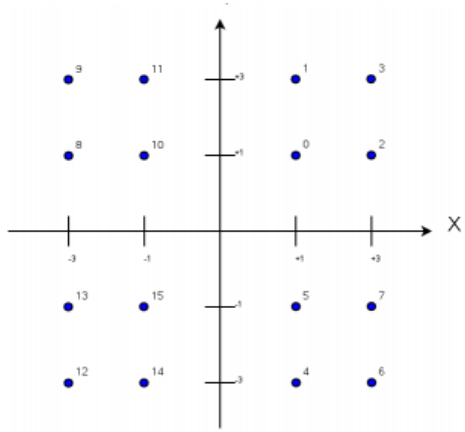
USE_DSP48 VHDL Example

```
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_2): multiplicand << 1

3 (11_2): multiplicand << 1 + multiplicand

5 (101_2): multiplicand << 2 + multiplicand

255 (11111111_2): ?

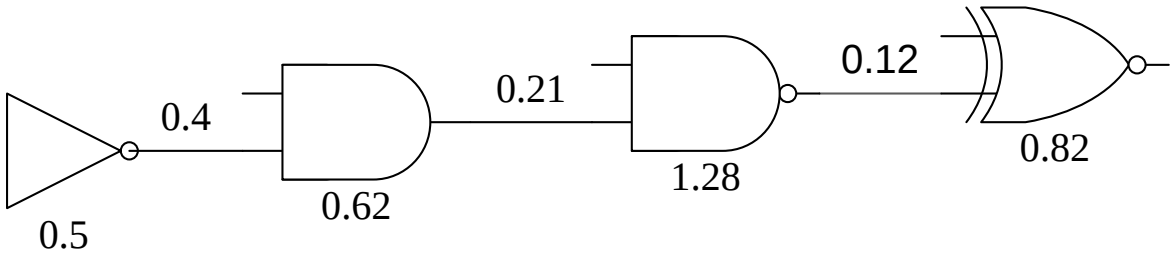
multiplicand << 8 – multiplicand

Overview

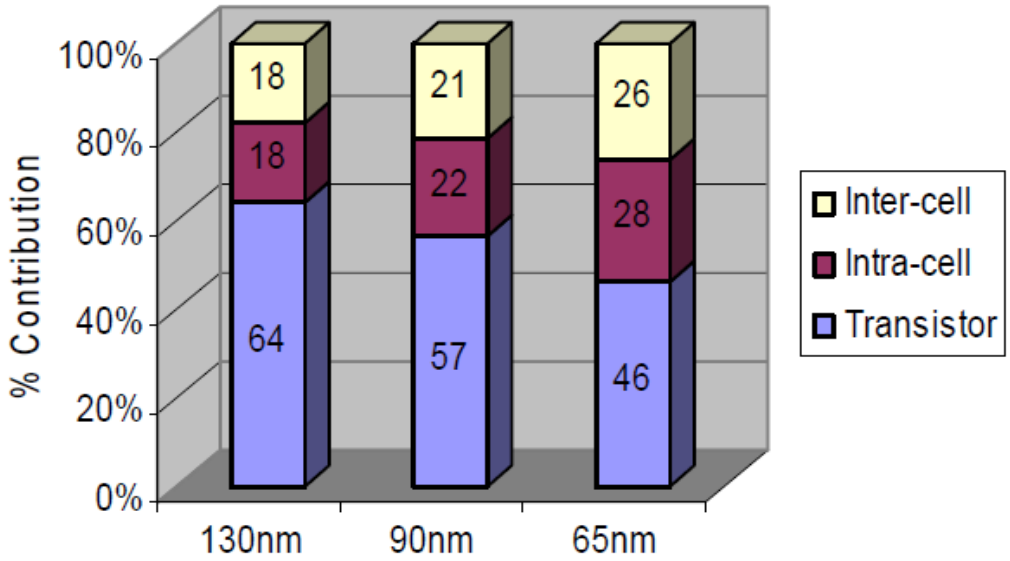
- Fixed-Point Representation
- Add/Subtract
- Multiplication
- **Timing & Techniques to Reduce Delay**

Combinational Circuit Timing

- **Path delay = cell delay + net delay**

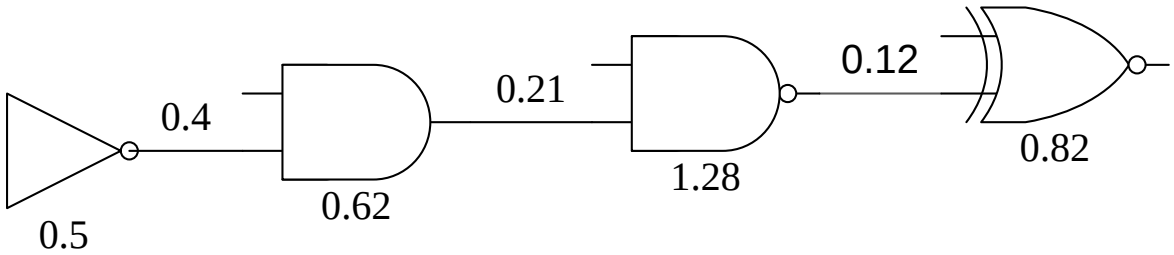


Path Delay = 0.5+0.4+0.62+0.21+1.28+0.12+0.82=3.95 ns

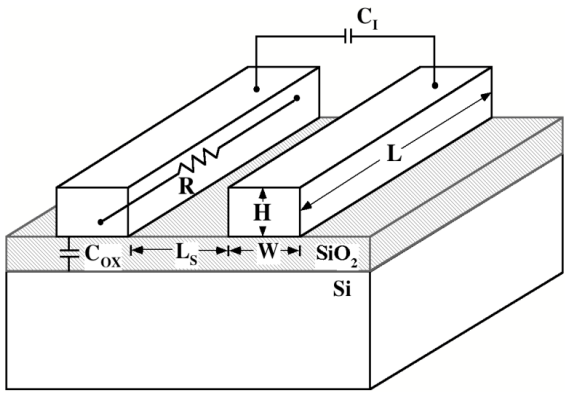


Combinational Circuit Timing

- **Path delay = cell delay + net delay**



Path Delay = 0.5+0.4+0.62+0.21+1.28+0.12+0.82=3.95 ns



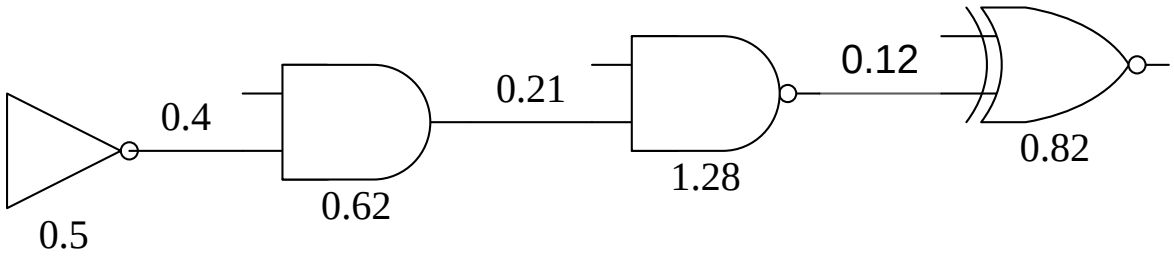
$$R = \rho \frac{L}{WH}$$

$$C_{ILD} = K_{ox}\varepsilon_o \frac{WL}{X_{ox}}$$

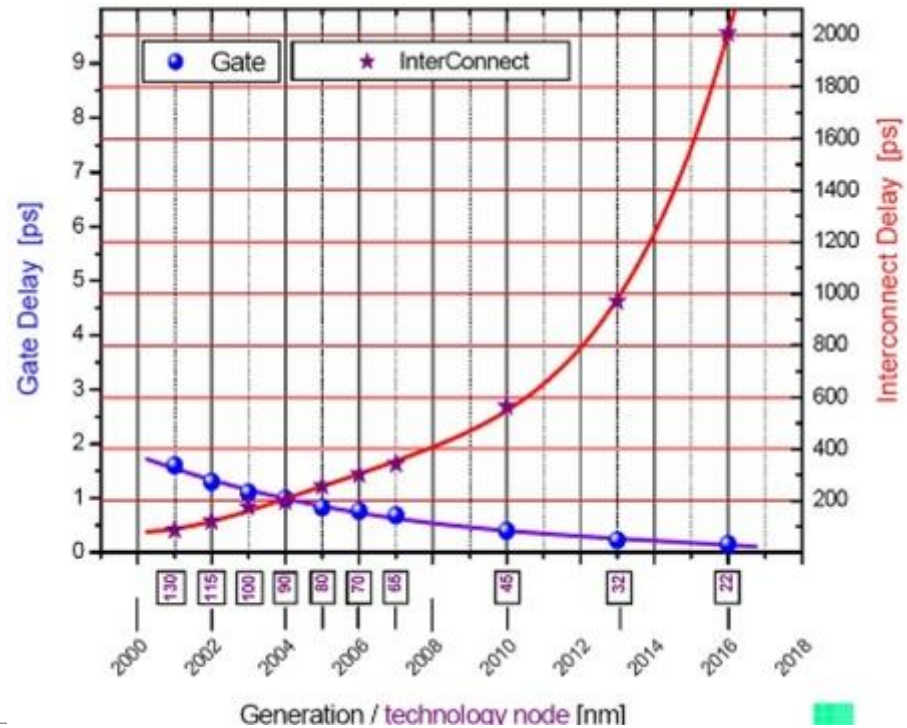
$$C_{IMD} = K_{ox}\varepsilon_o \frac{HL}{L_S}$$

Combinational Circuit Timing

- **Path delay = cell delay + net delay**

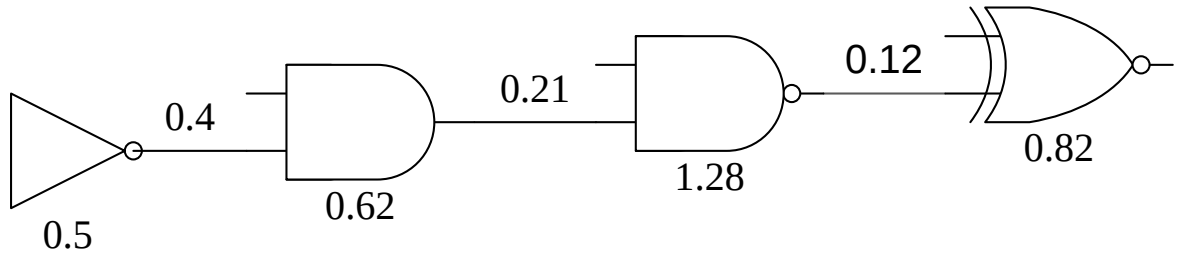


Path Delay = 0.5+0.4+0.62+0.21+1.28+0.12+0.82=3.95 ns

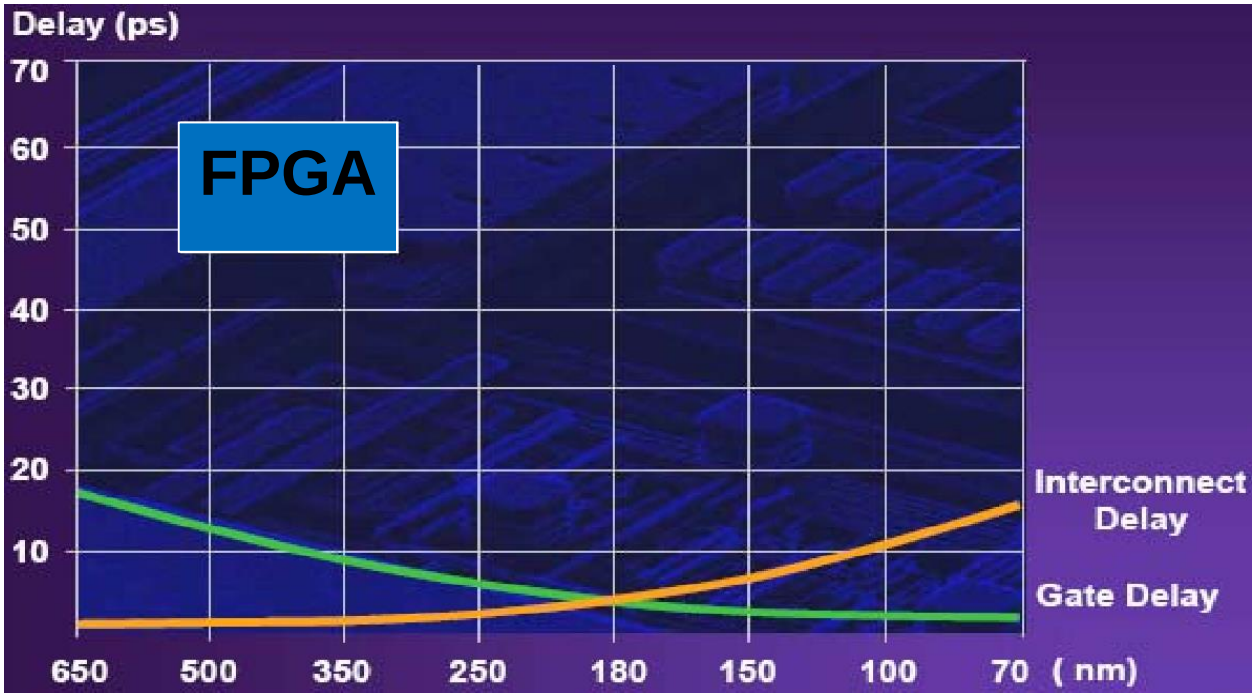


Combinational Circuit Timing

- **Path delay = cell delay + net delay**

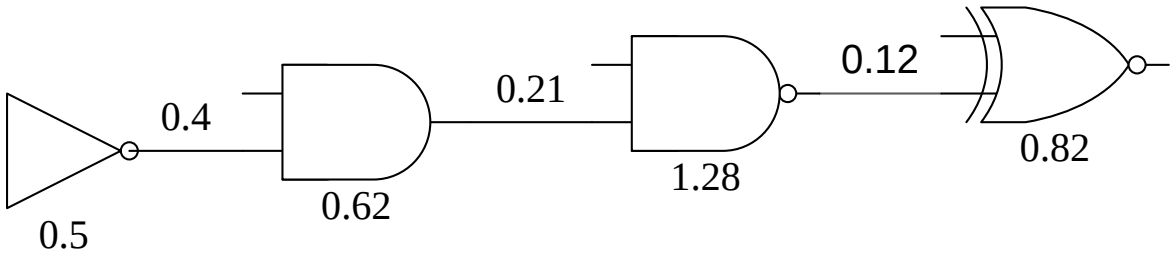


Path Delay = 0.5+0.4+0.62+0.21+1.28+0.12+0.82=3.95 ns



Combinational Circuit Timing

- **Path delay = cell delay + net delay**

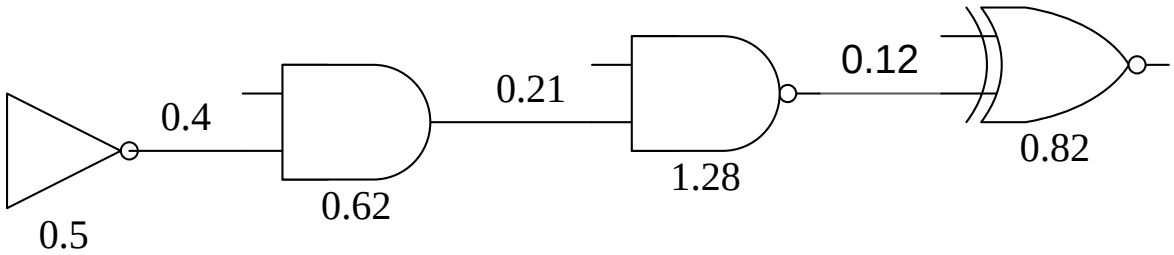


Path Delay = 0.5+0.4+0.62+0.21+1.28+0.12+0.82=3.95 ns

- **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**

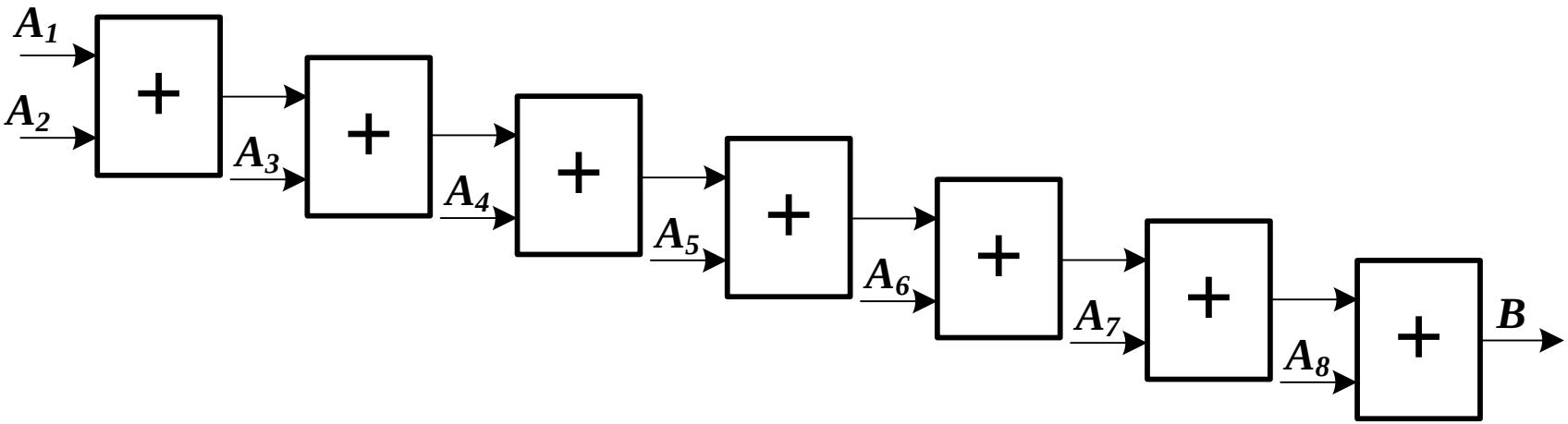


Path Delay = 0.5+0.4+0.62+0.21+1.28+0.12+0.82=3.95 ns

- **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

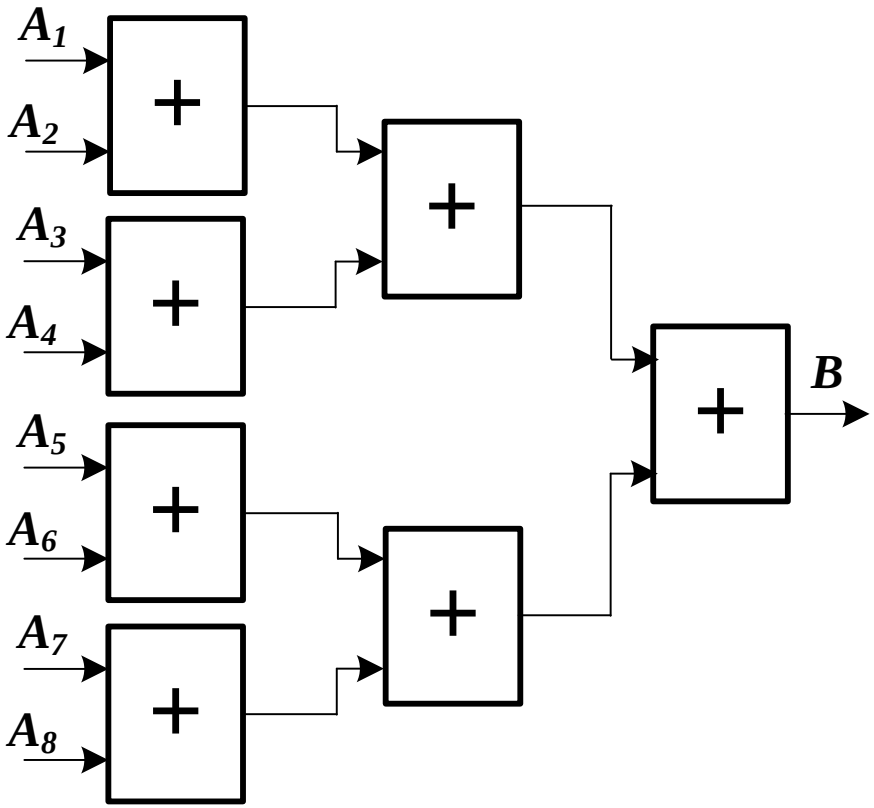
- **Calculate:** $B = A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8$



Cascaded-Chain

Higher-Level

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



Tree

Thanks!