Lecture

6

ECE216B

DSP Arithmetic

Prof. Dejan Marković

ee216b@gmail.com

Agenda

- Number systems
- Quantization effects
- Data dependencies
- Implications on power
- Adders and multipliers

Number Systems: Algebraic

Algebraic Number
e.g. $a = \pi + b$

- **High-level abstraction**
- **Infinite precision**
- **Often easier to understand**
- **Good for theory/algorithm development**
- **Hard to implement**

C. Shi, Floating-point to Fixed-point Conversion, Ph.D. Thesis, UC Berkeley, 2004.

Number Systems: Floating Point

- Widely used in CPUs
- Floating precision
- Good for algorithm study and validation

$$Value = (-1)^{Sign} \times 1.Mantissa \times 2^{(Exponent - Bias)}$$

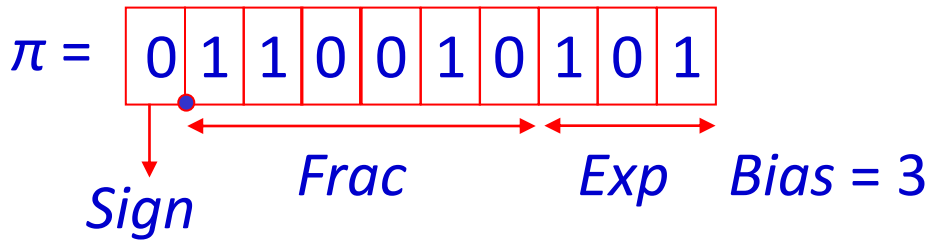
IEEE 754 standard	Sign	Exponent	Mantissa	Bias
Single precision [31:0]	1 [31]	8 [30:23]	23 [22:0]	127
Double precision [63:0]	1 [63]	11 [62:52]	52 [51:00]	1023

J.L. Hennesy and D.A. Paterson, Computer Architecture: A Quantitative Approach, (2nd Ed), Morgan Kaufmann, 1996.

Example of Floating-Point Representation

$$\textit{Value} = (-1)^{\textit{Sign}} \times \text{Fraction} \times 2^{(\textit{Exponent} - \textit{Bias})}$$

A non-IEEE-standard floating point



- **Calculate π**

$$\pi = (-1)^0 \times (1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 0\times2^{-4} + 1\times2^{-5} + 0\times2^{-6}) \times 2^{(1\times2^2 + 0\times2^1 + 1\times2^0 - 3)} = 3.125$$

- **Very few bits are used in this representation**
  $\Rightarrow$ low accuracy (actual value π = 3.141592654…)

Floating-Point Standard: IEEE 754

Properties

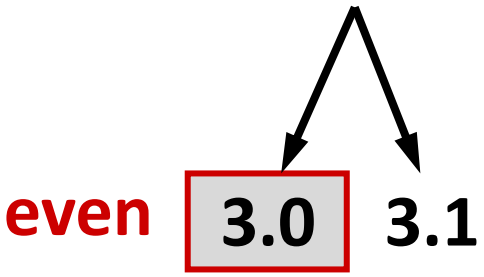
1. Half-way rounding
2. Special values
3. Denormals
4. Rounding modes

IEEE 754: Half-Way Rounding

1 **Rounding a "half-way" result to the nearest float (picks even)**

Example:

6.1 × 0.5 = 3.05 (base 10, 2 digits)



(base 10, 1 digit)

4 • **Rounding modes:**

Nearest	Toward 0	Toward –∞	Toward ∞

default

2 IEEE 754: Special Values

NaN, ∞, −∞

Examples:

- **sqrt(−0.5) = NaN, *f*(NaN) = NaN**
  - *verify this in MATLAB*
- **1/0 = ∞, 1/∞ = 0**
- **arctan(*x*) → *π*/2 as *x* →∞ ⇒ arctan(∞) = *π*/2**

3 IEEE 754: Denormals

Uses denormals to represent the result < 1.0 × base^{Emin}

***Emin* = min exponent** ⟷ Flush-to-0

Use significand < 1.0 and *Emin*
("gradual underflow")

Example:

base 10, 4 significant digits, $x = 1.234 \times 10^{Emin}$

denormals:

$x/10 \rightarrow 0.123 \times 10^{Emin}$

$x/1{,}000 \rightarrow 0.001 \times 10^{Emin}$

$x/10{,}000 \rightarrow 0$

$x = y \Leftrightarrow x - y = 0$

flush-to-0:

$x = 1.256 \times 10^{Emin}$, $y = 1.234 \times 10^{Emin}$

$x - y = 0.022 \times 10^{Emin} = 0$ (although $x \neq y$)

denormal (exact computation)

Representation of Floating-Point Numbers

IEEE 754 standard	Sign	Exponent	Mantissa	Bias
Single precision [31:0]	1 [31]	8 [30:23]	23 [22:0]	127

Single precision: 32 bits

1.*Mantissa* (for number conversion)

Example:

1 10000001 0100…0

sign exponent Mantissa

$129 - 127$ $0.01_2 = 0.25$ (1.Mantissa = 1.25)

$-1.25 \times 2^2 = -5$

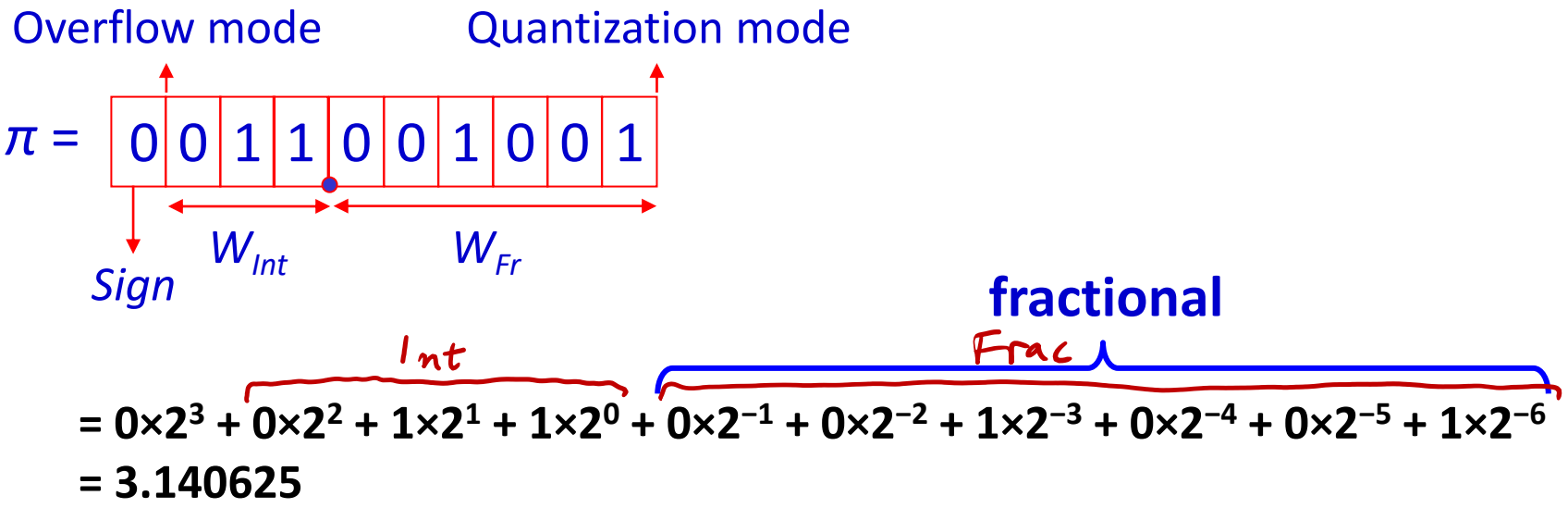
Special (Reserved) Cases

- **Notation**
  - Significand = everything other than exponent
    - Sign & mantissa
      - 1.Mantissa (default for number convsersion)

Exponent	Mantissa	Value
0	0	0
0	Nonzero	Denormal
255	0	± Infinity
255	Nonzero	NaN

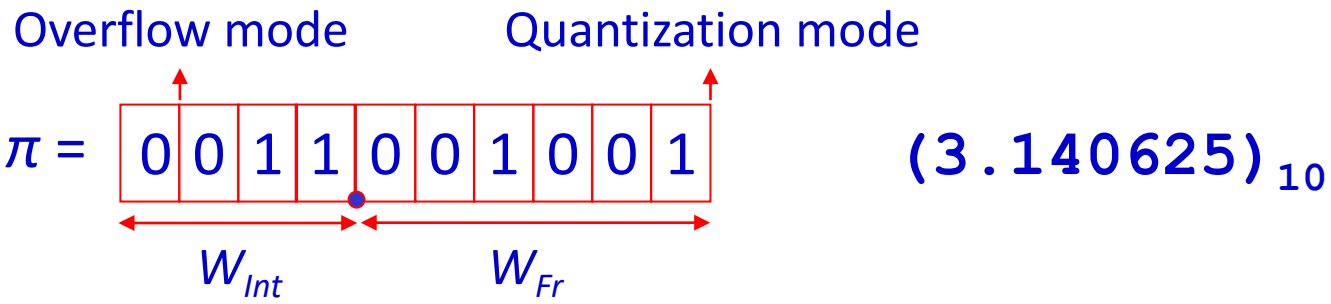
- **Exponent ranges from –126 to +127**
  - With 0 and 255 reserved as above

Fixed Point: 2's Complement Representation



- ***W_{Int}* and *W_{Fr}* suitable for predictable dynamic range**
  - o-mode (saturation, wrap-around)
  - q-mode (trunc, roundoff)
- **Economic for implementation**

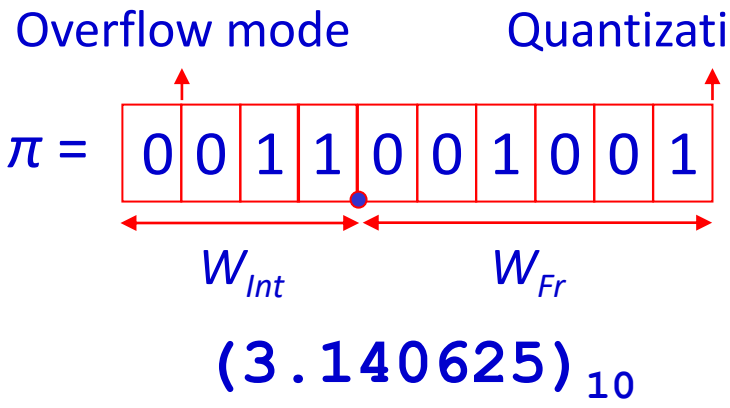
Fixed Point: Unsigned Magnitude



$(3.140625)_{10}$

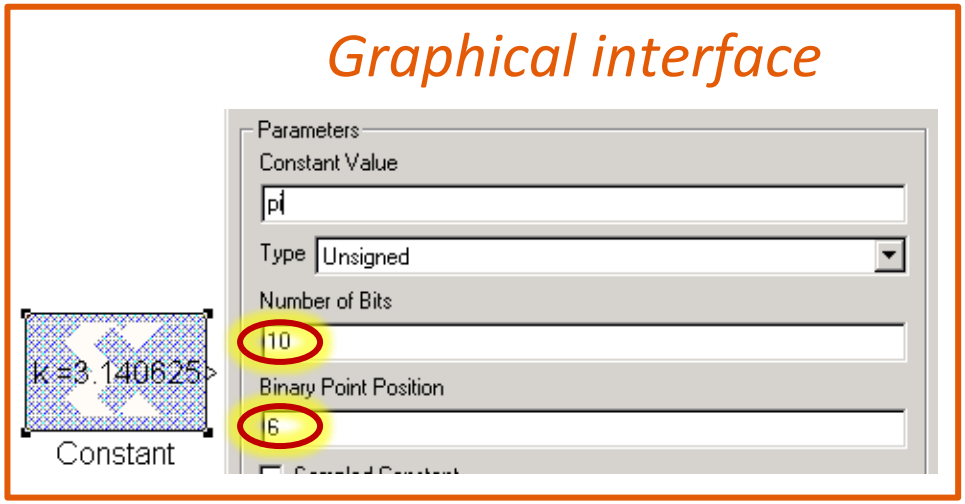
- **Useful built-in MATLAB functions:**
  - fix, round, ceil, floor, dec2bin, bin2dec, etc.
- **Converting representations in MATLAB**
  - `dec2bin(round(pi*2^6), 10)`
  - `bin2dec(above)*2^-6`
- **How to specify hardware descriptions?**

Fixed Point: Example Hardware Descriptions



Unsigned Magnitude format

- **In SysGen/Simulink:**
  - (10, 6) = (total, frac) number of bits



- **In PyGears**[1]**:**
  - Ufixp[4, 10]
    - W_{int}
    - W_{total}

Code based

```
pi = Ufixp[4, 10](math.pi)
```

[1] https://www.pygears.org

Fixed-Point Representations

- **Sign magnitude**

- **2's complement**
  - $x + (-x) = 2^n$ (complement each bit, add 1)
  - Most widely used (signed arithmetic easy to do)

- **1's complement**
  - $x + (-x) = 2^n - 1$ (complement each bit)

- **Biased:** add bias, encode as ordinary unsigned number
  - k + bias ≥ 0, bias = 2^{n-1} (typically)

Example: Fixed-Point Representations

Assume: $n = 4$ bits, $k = 3$, $-k = ?$

- Sign magnitude: $k = 0011_2 \rightarrow -k = 1011_2$

- 2's complement: $k + 1011 = 2^n$ $\quad -k = 1100$ $\quad + \quad 1$ $\quad 1101_2$

  $0011 + 1101 = 10000$

  Procedure:
  - Bit-wise inversion
  - Add "1"

- 1's complement: $-k = 1100_2$ $\quad k + (-k) = 2^n - 1$

- Biased: $k + \text{bias} = 1011_2$ $\quad 2^{n-1} = 8 = 1000_2$ $\quad -k + \text{bias} = 0101_2 = 5 \geq 0$

2's Complement Arithmetic

- **Most widely used representation, simple arithmetic**

- **Example: 5 + – 2**

$$\begin{array}{rl} 0101_2 & (5) \\ +1110_2 & (-2) \\ \hline 10011 = +3 & \end{array} \qquad \begin{array}{rl} 0010 & (-2) \\ 1101 & \\ + \quad 1 & \\ \hline 1110_2 & \end{array}$$

Discard the sign bit *(if there is no overflow)*

- **Overflow occurs when Carry into MSB ≠ Carry out of MSB**

MSB

$$\begin{array}{rll} & 0\,1\,0\,1_2 & (5) \\ + & 1\,1\,1\,0_2 & (-2) \\ \hline & 1\,0\,0\,1\,1 & (3) \end{array}$$

Carry: 1 1 0 0

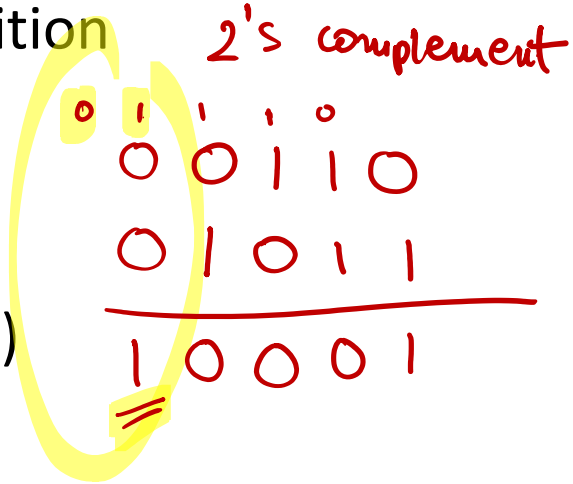
Carry out of MSB = Carry into MSB → No overflow!

Overflow

- **Example:** unsigned 4-bit addition

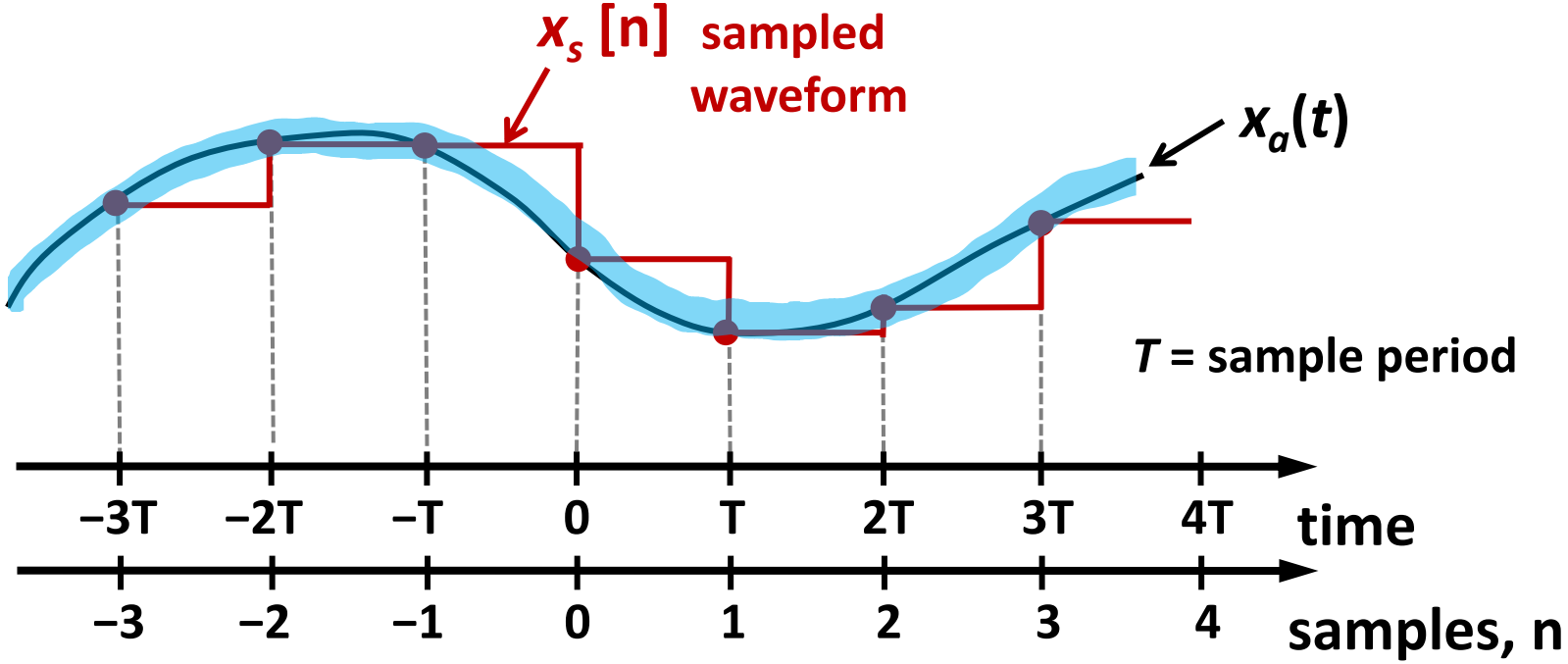
$$6 = 0110_2$$
$$+11 = 1011_2$$
$$= 17 = 10001_2 \text{ (5 bits!)}$$

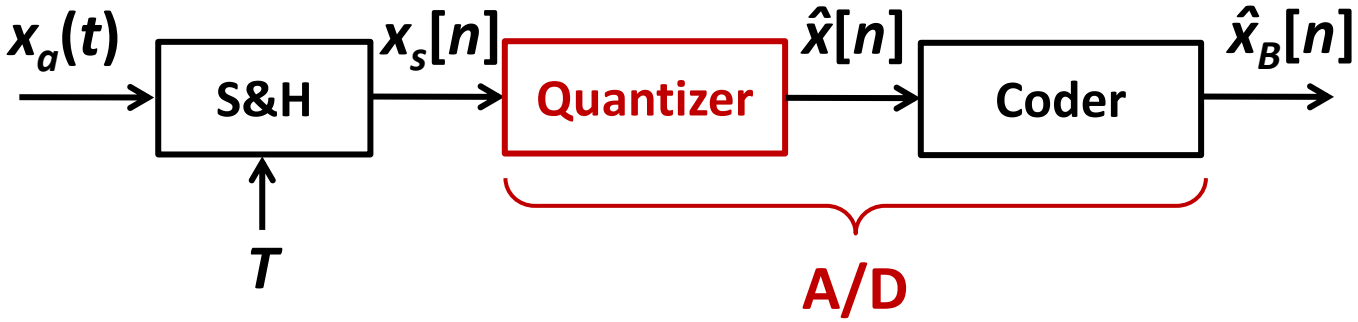
extra bit



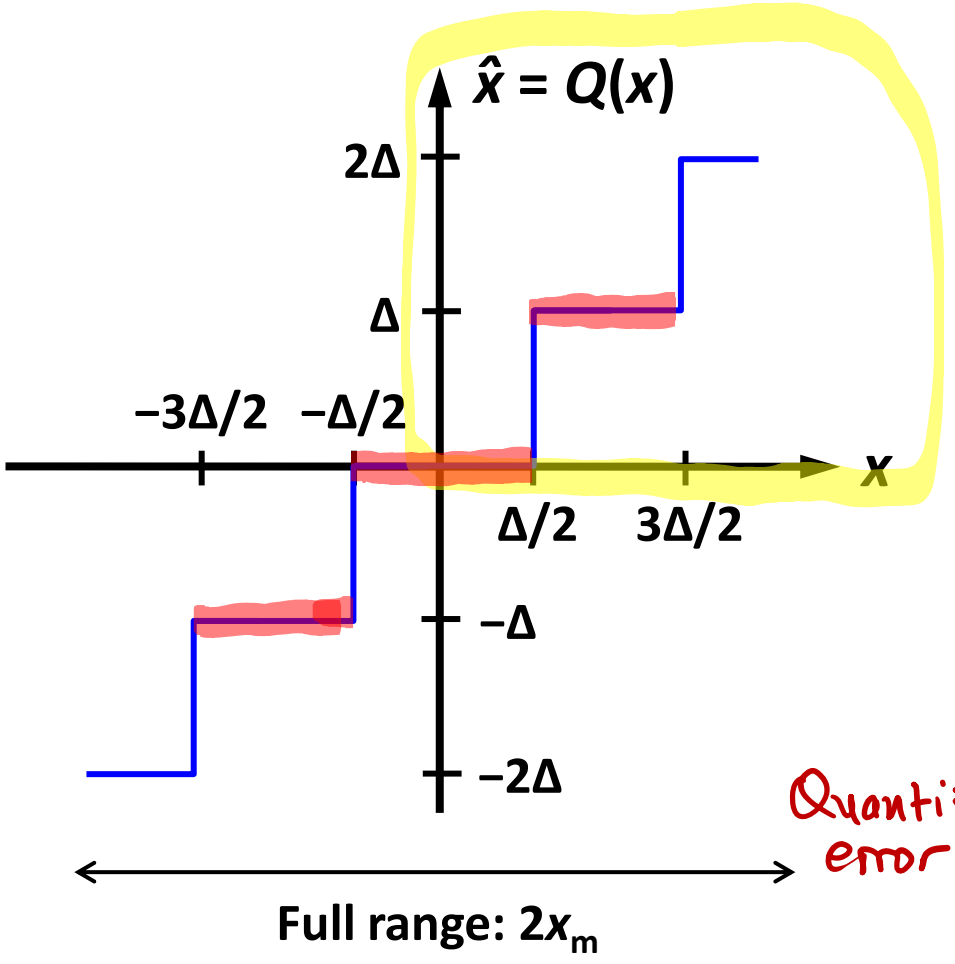
- **Property of 2's complement**
  - Negation = bit-by-bit complement + 1
    → C_{in} = 1, result: $a - b$

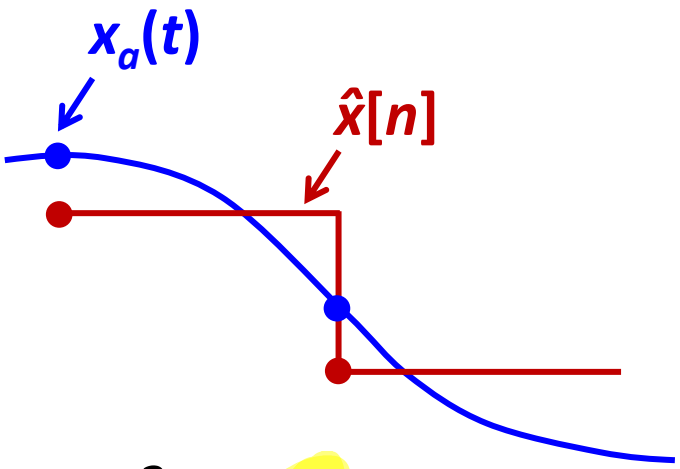
Quantization Effects





Quantization





$$\Delta = \frac{2x_m}{2^{B+1}} = \frac{x_m}{2^B}$$

B = # bits of quantization

Quantization error

$$e[n] = \hat{x}[n] - x[n]$$

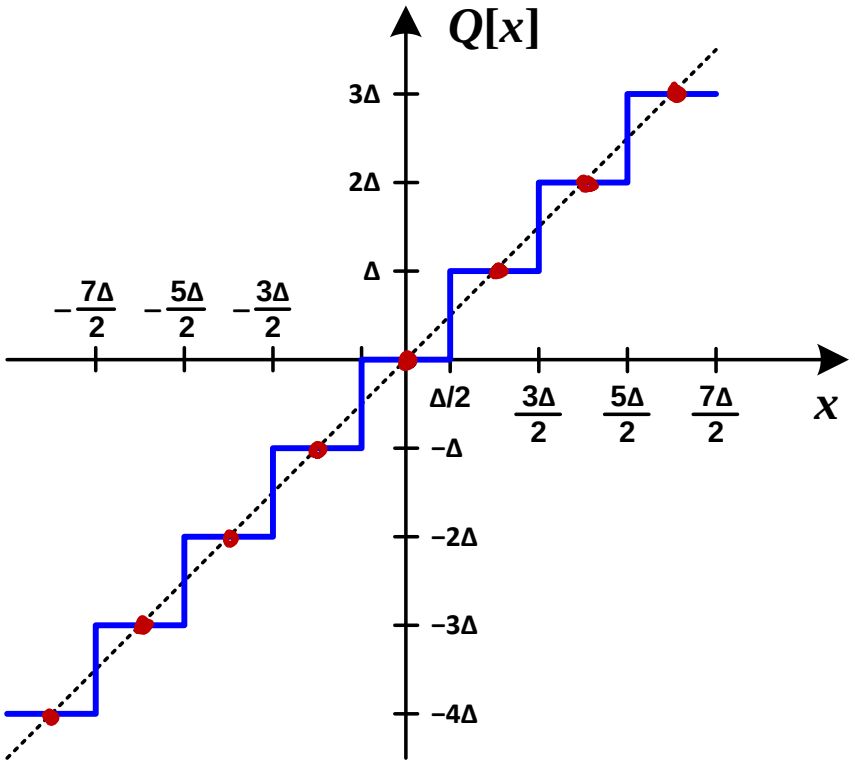
$$-\Delta/2 < e[n] \le \Delta/2$$

(AWN process)

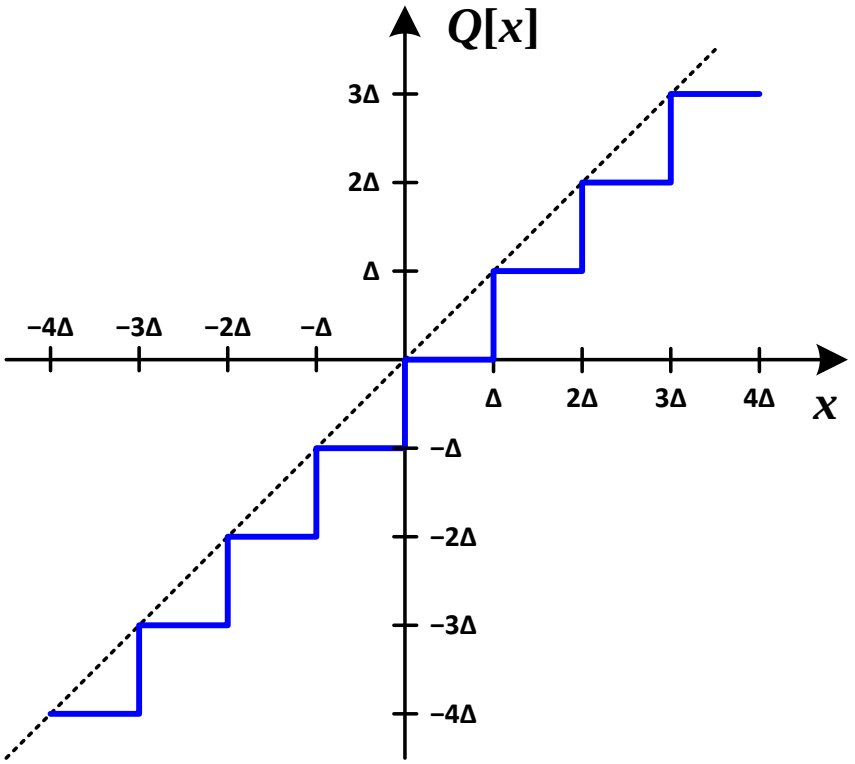
2's complement representation

Quantization Modes: Rounding, Truncation

Rounding

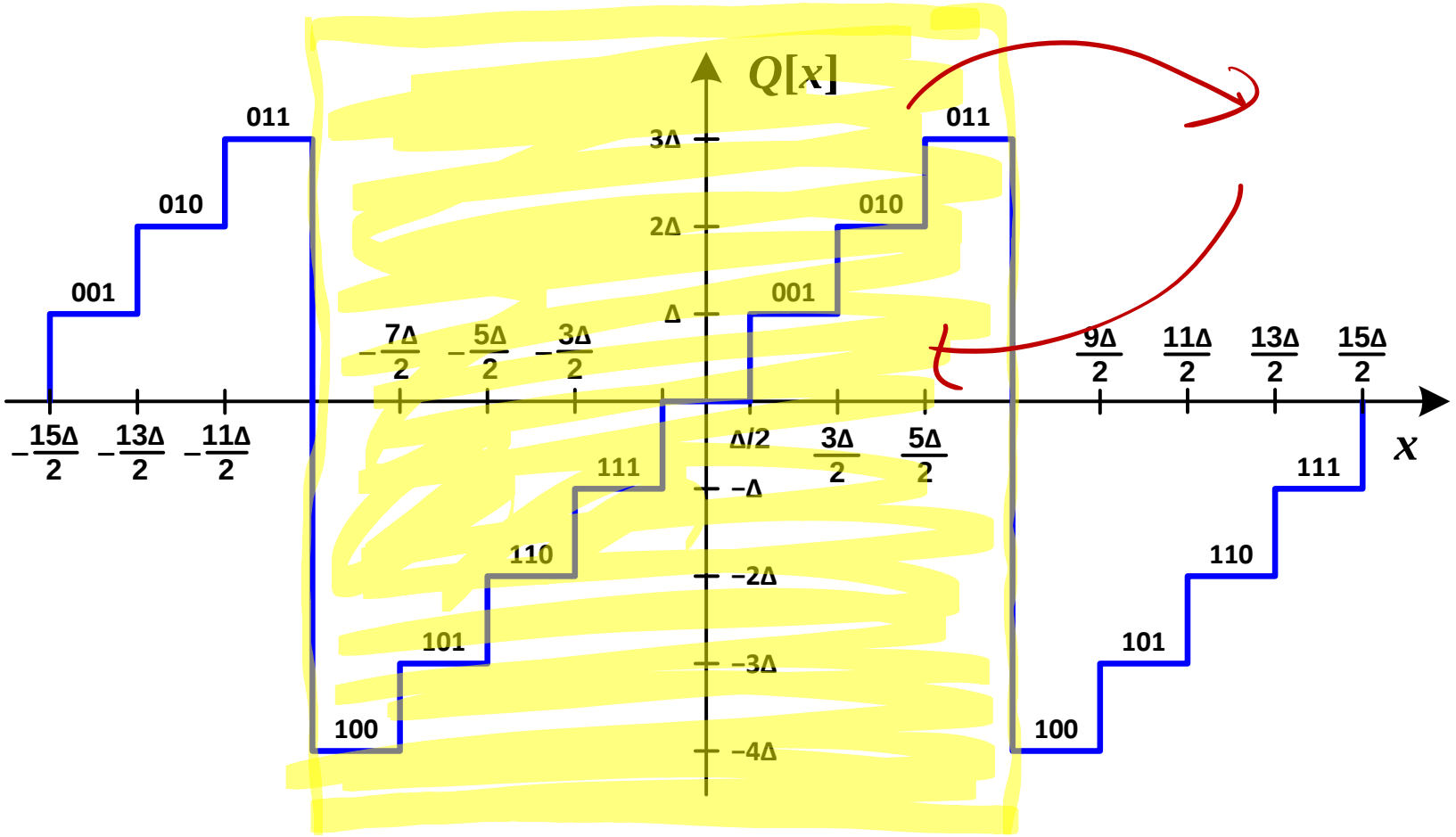


Truncation



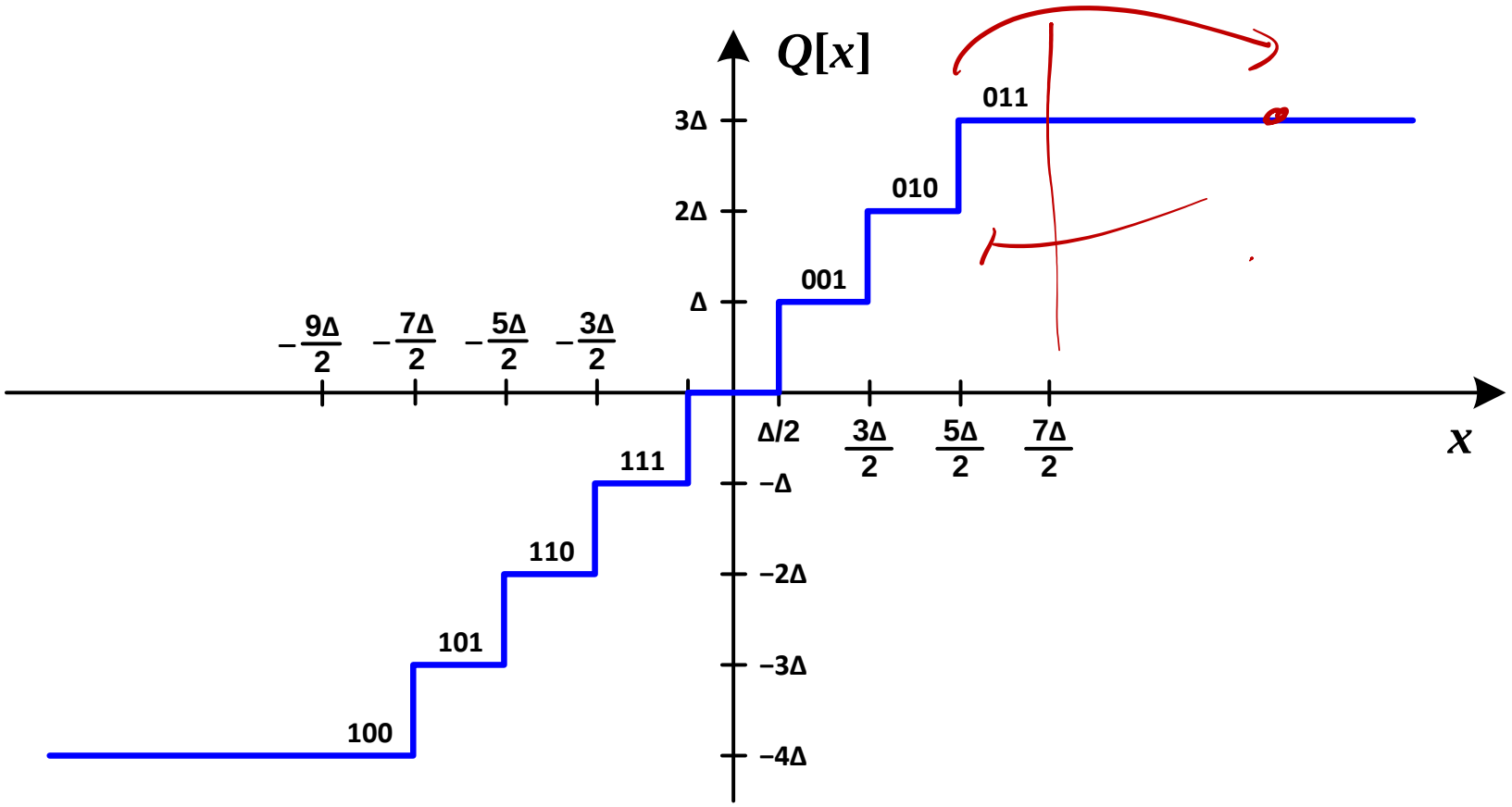
Feedback systems: rounding

Overflow Modes: Wrap Around



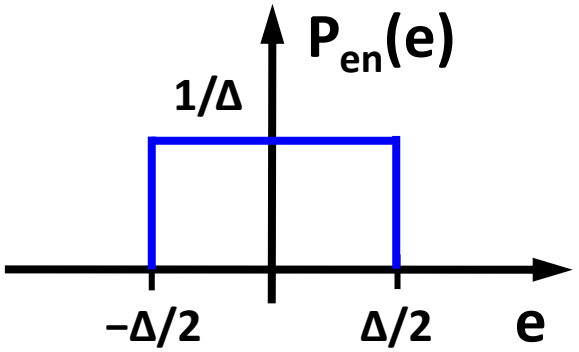
A.V. Oppenheim, R.W. Schafer, with J.R. Buck, Discrete-Time Signal Processing, (2nd Ed), Prentice Hall, 1998.

Overflow Modes: Saturation



Feedback systems: saturation

Quantization Noise



$$\sigma_e^2 = \int_{-\Delta/2}^{\Delta/2} e^2 \frac{1}{\Delta} de = \frac{\Delta^2}{12}$$

X_m: full-scale signal $= 2x_m$

B + 1 quantizer

$$\sigma_e^2 = \frac{2^{-2B} X_m^2}{12}$$

$$SQNR = 10log_{10}\left(\frac{\sigma_x^2}{\sigma_e^2}\right) = 10log_{10}\left(\frac{12 \cdot 2^{2B} \cdot \sigma_x^2}{X_m^2}\right)$$

$$= 6.02 \cdot B + 10.8 - 20log_{10}\left(\frac{X_m}{\sigma_x}\right)$$

Binary Multiplication

- **Arguments: *X*, *Y***

$$X = \sum_{i=0}^{M-1} X_i \cdot 2^i \qquad Y = \sum_{j=0}^{N-1} Y_j \cdot 2^j$$

- **Product: *Z***

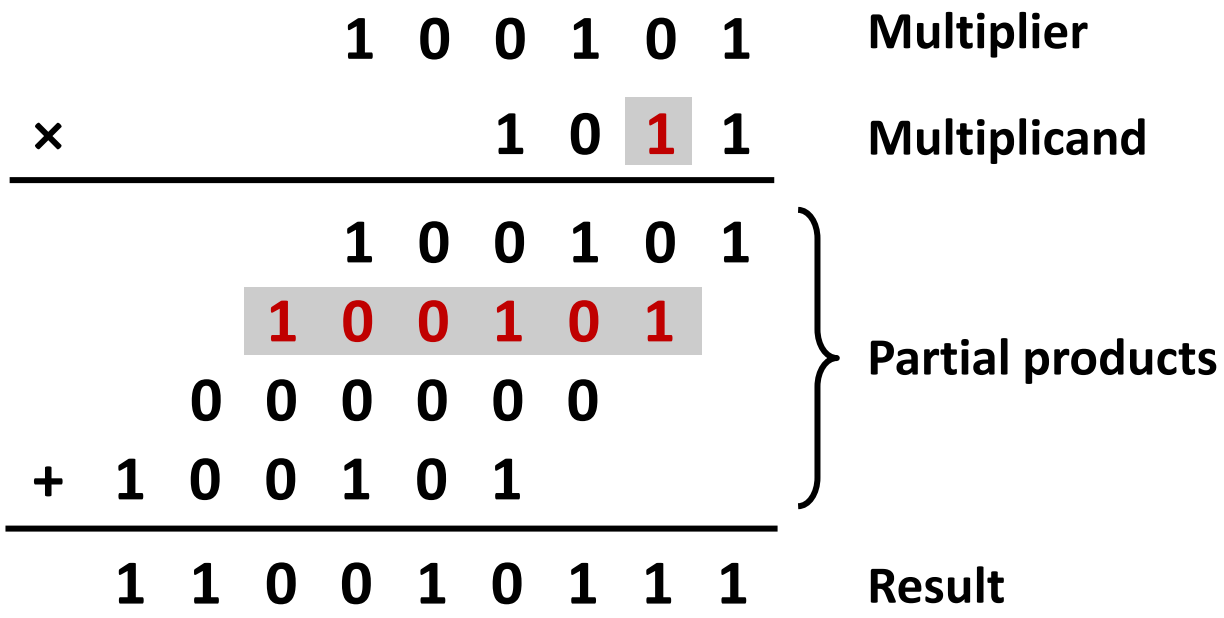
$$Z = X \cdot Y = \sum_{k=0}^{M+N-1} z_k \cdot 2^k = \left(\sum_{i=0}^{M-1} X_i \cdot 2^i \right) \cdot \left(\sum_{j=0}^{N-1} Y_j \cdot 2^j \right)$$

$$Z = \sum_{i=0}^{M-1} \left(\sum_{j=0}^{N-1} X_i \cdot Y_j \cdot 2^{i+j} \right)$$

J. Rabaey, A. Chandrakasan, B. Nikolić, Digital Integrated Circuits: A Design Perspective, (2nd Ed), Prentice Hall, 2003.

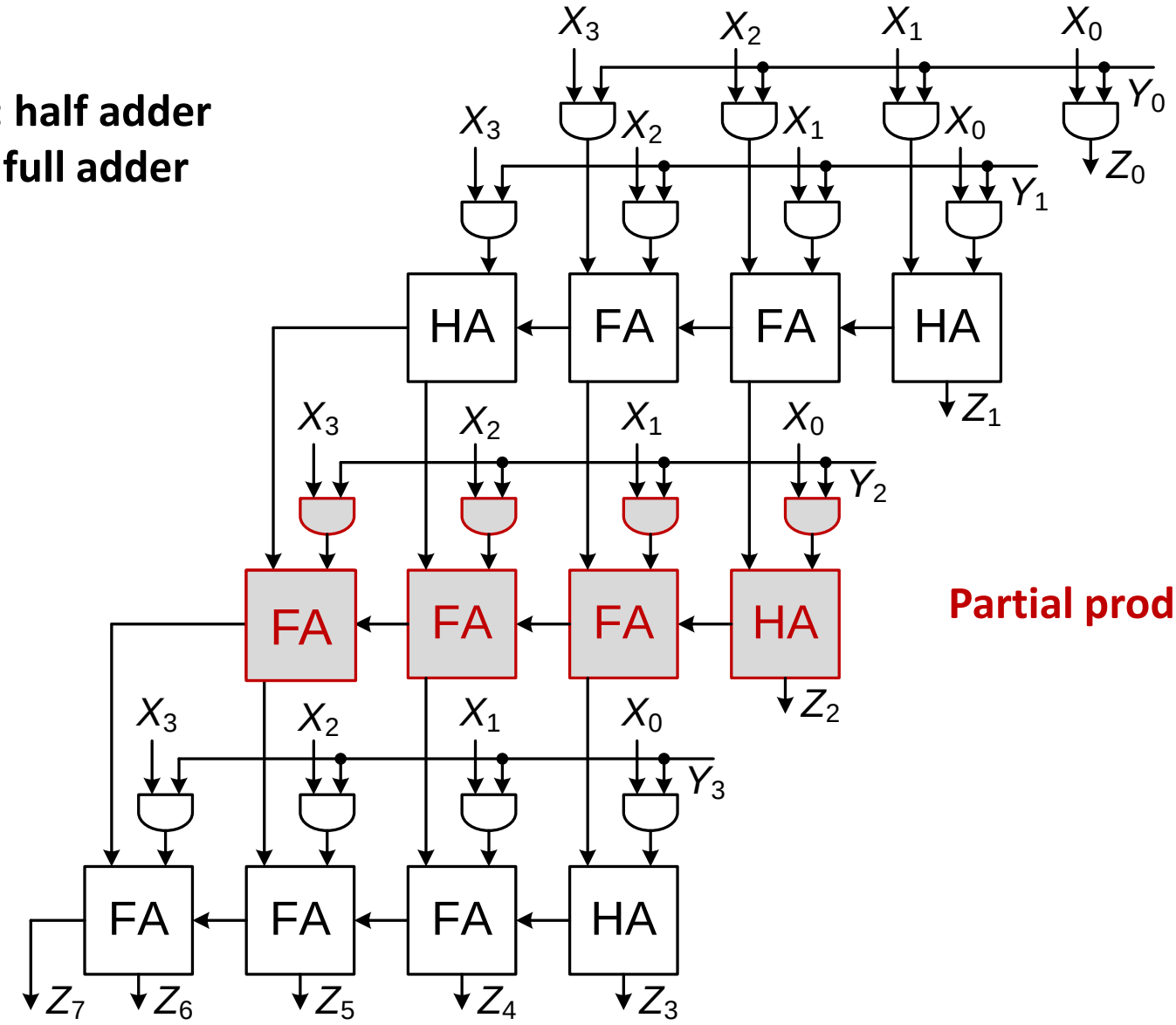
Binary Multiplication: Example

- **Multi-bit multiply**
  = bit-wise multiplies (partial products) + final adder

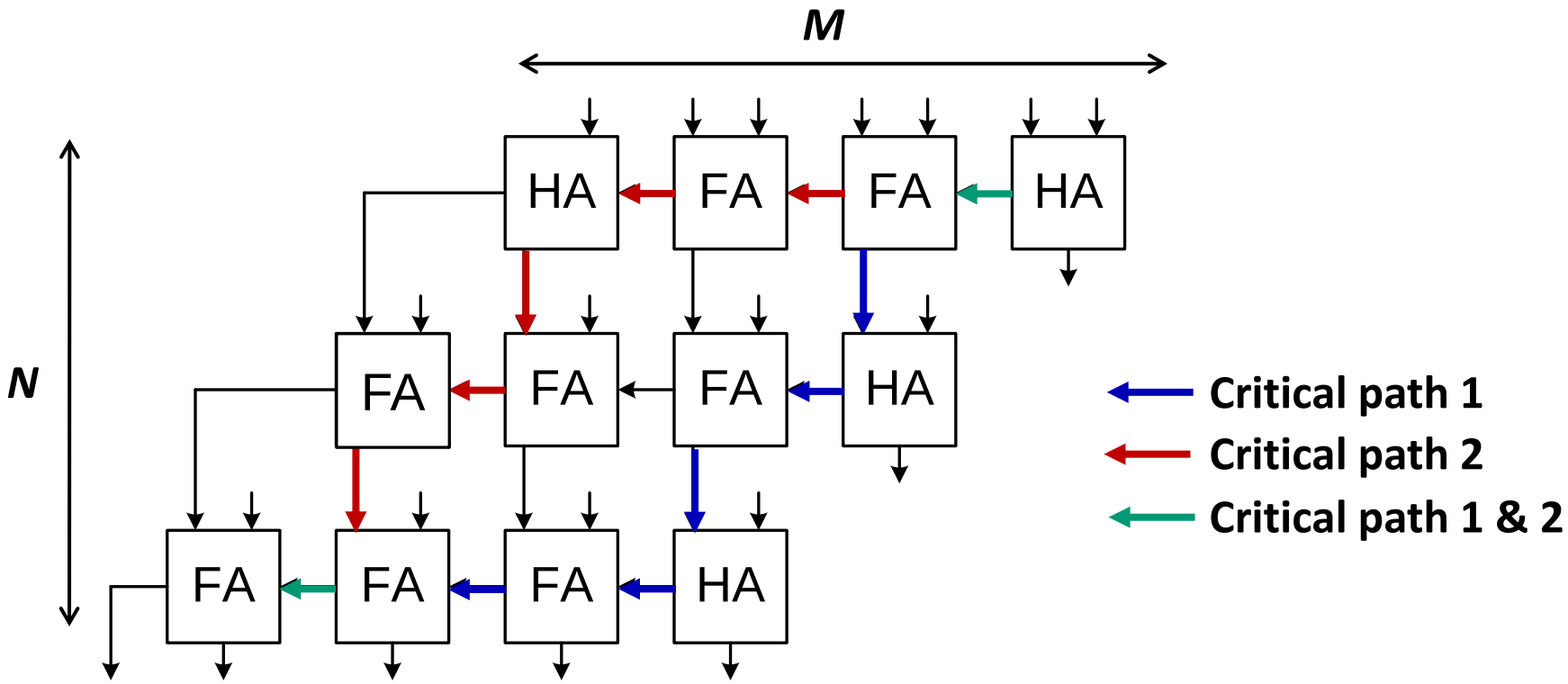


Array Multiplier

HA: half adder
FA: full adder



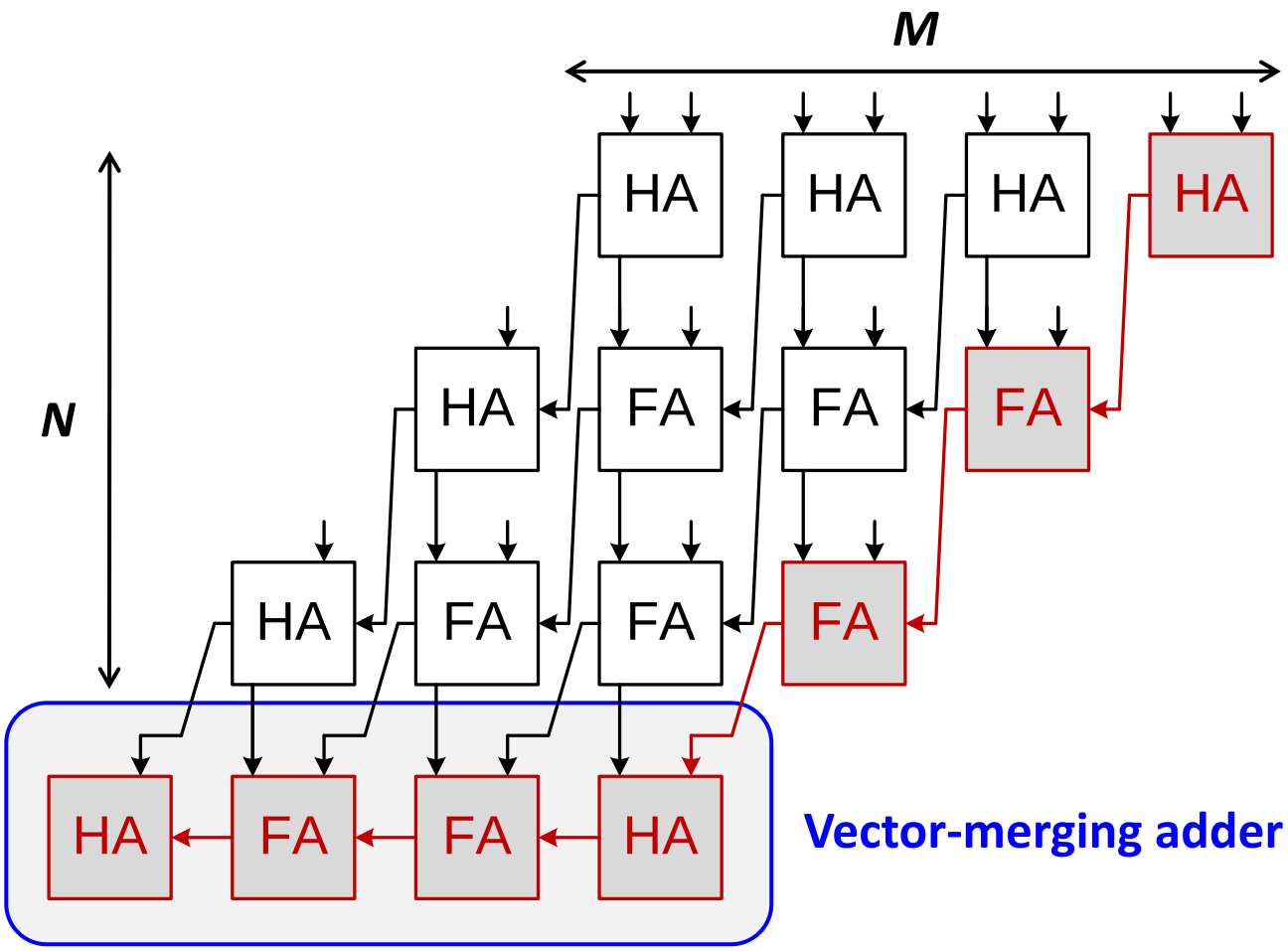
M-by-*N* Array Multiplier: Critical Path



Note: number of horizontal adder slices = $N - 1$

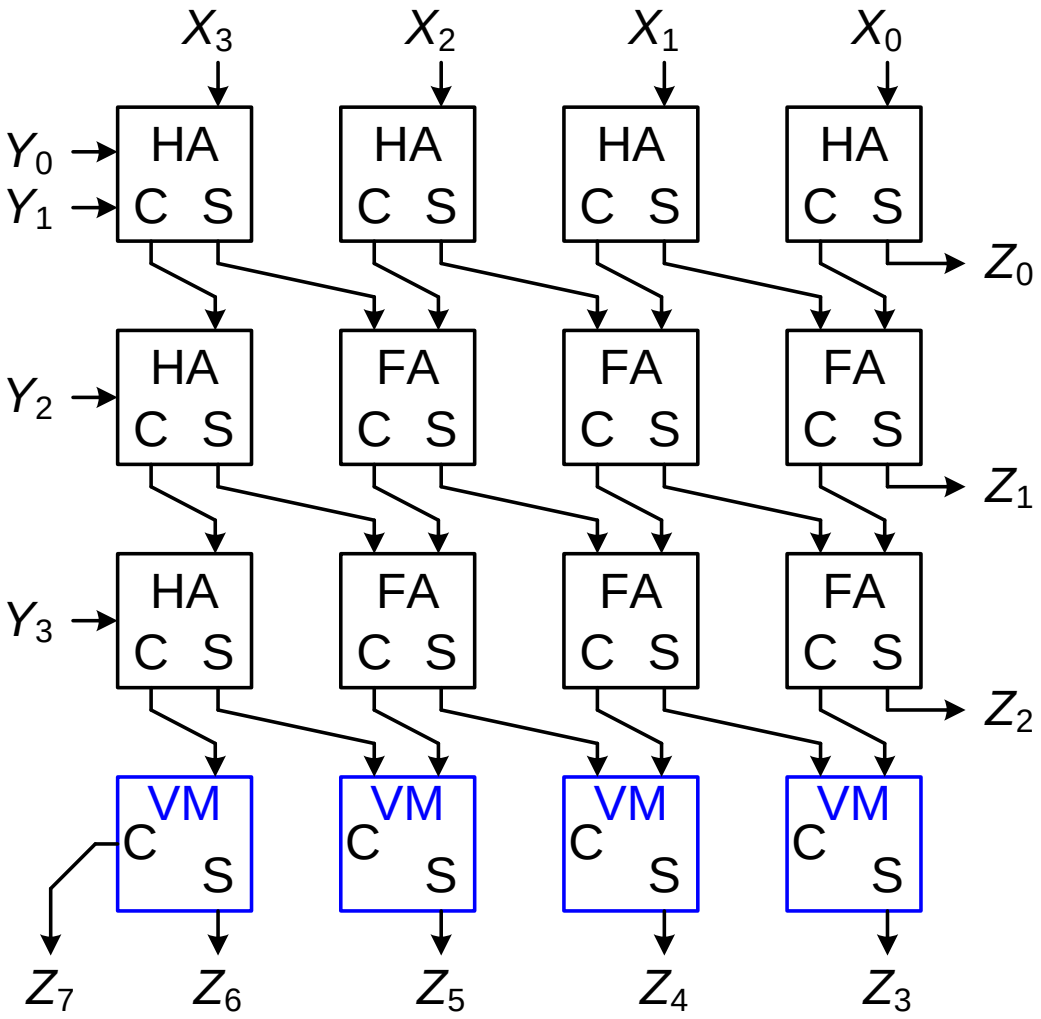
$$t_{mult} = \left[(M-1)+(N-2)\right] \cdot t_{carry} + (N-1) \cdot t_{sum} + t_{and}$$

Carry-Save Multiplier



$$t_{mult} = (N-1)\cdot t_{carry} + t_{and} + t_{merge}$$

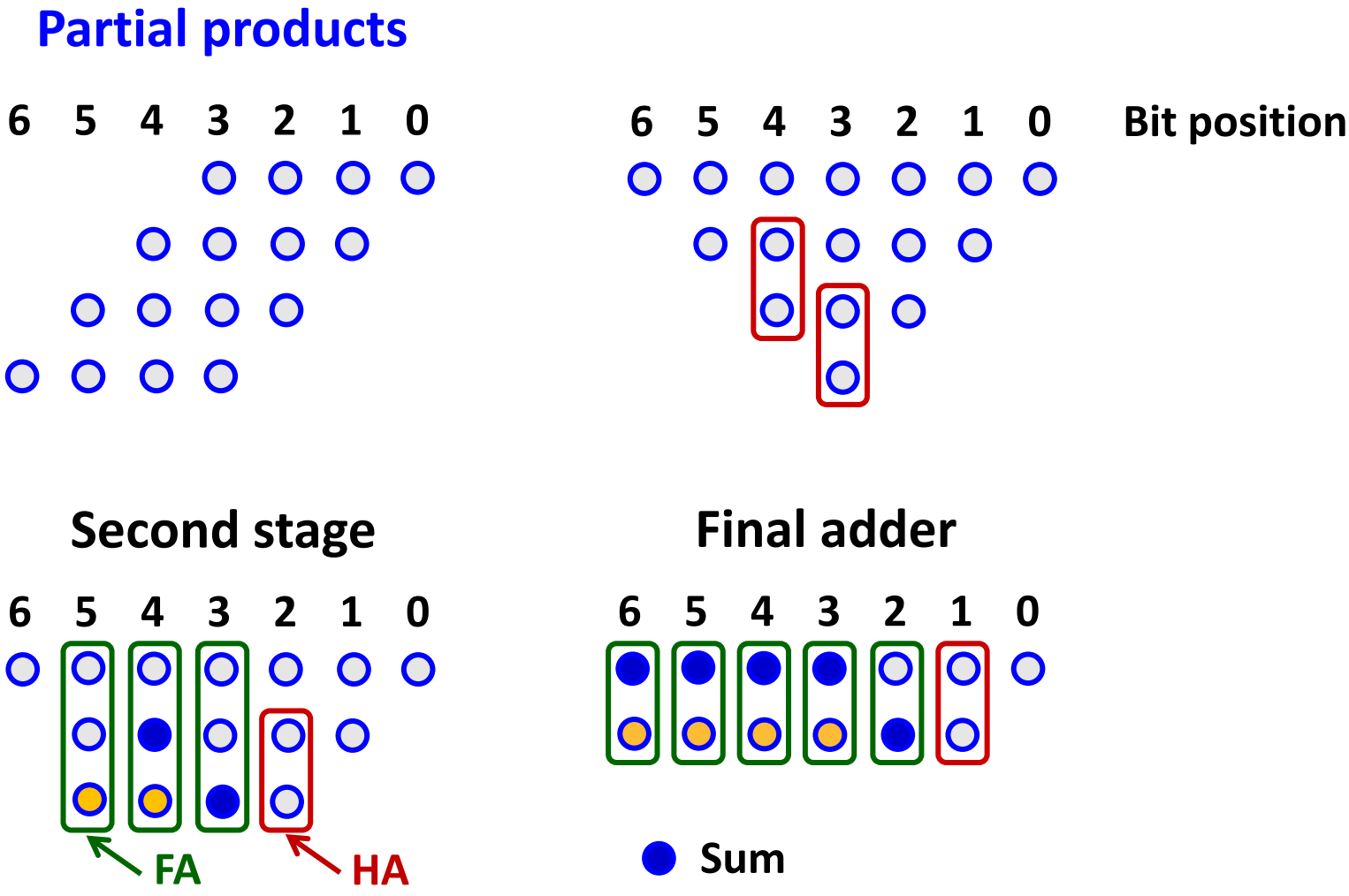
Multiplier Floorplan



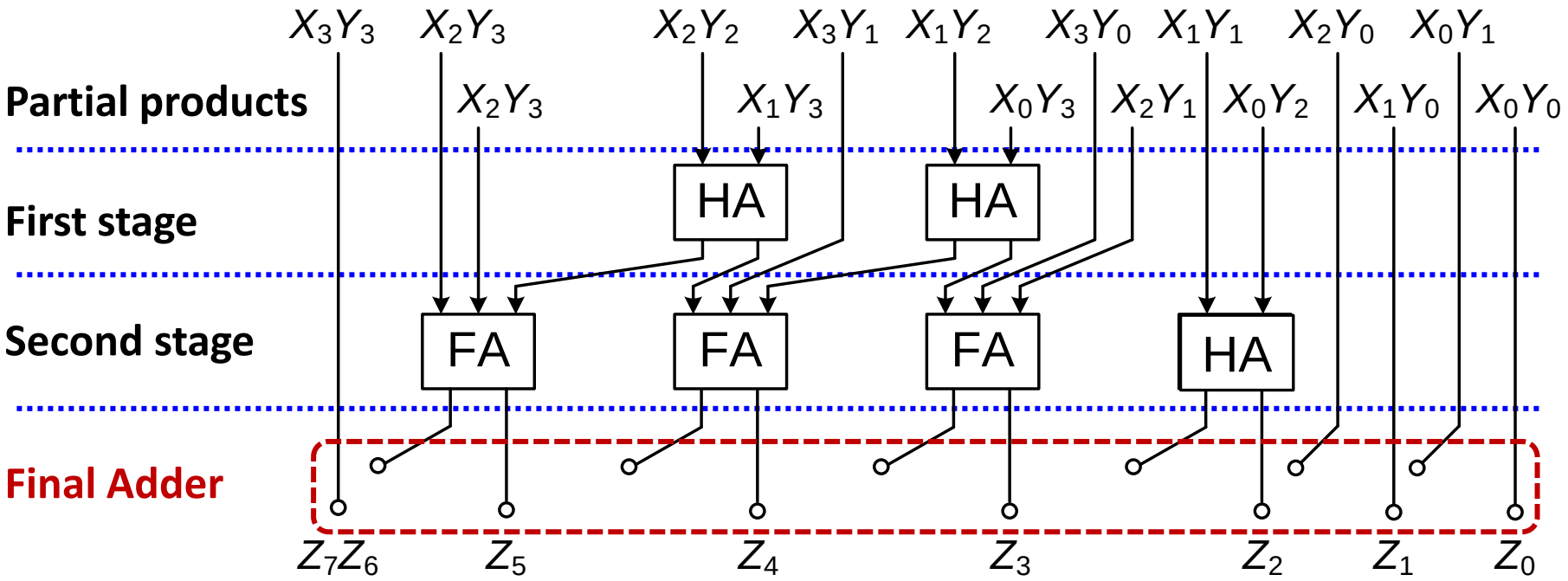
HA: half adder
FA: full adder
VM: vector-merging cell

***X* and *Y* signals are broadcasted through the complete array**

Wallace-Tree Multiplier



Wallace-Tree Multiplier

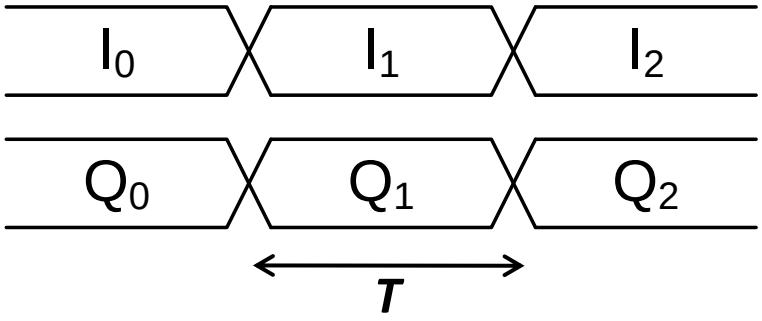


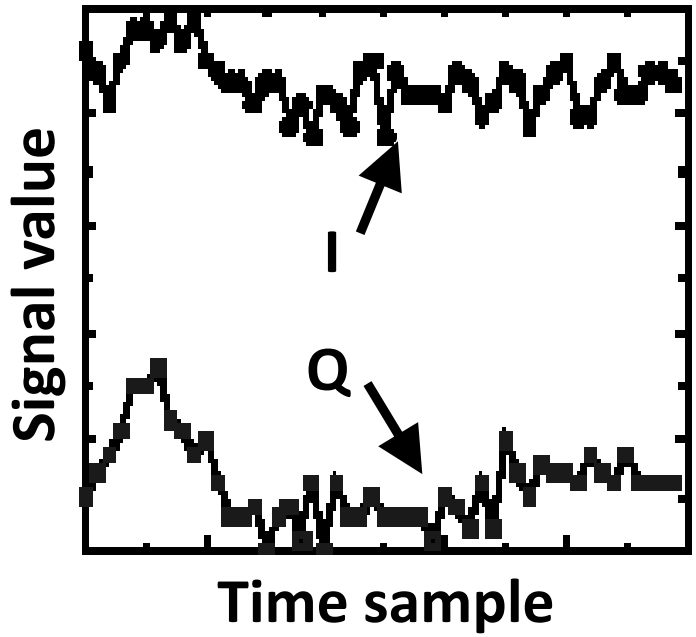
Multipliers: Summary

- **Optimization goals different than in binary adder**
- **Once again: Identify critical path**
- **Other possible techniques**
  - Logarithmic versus linear (Wallace-Tree multiplier)
  - Data encoding (Booth)
  - Pipelining

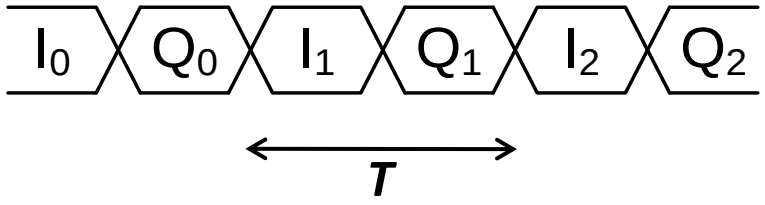
Time-Multiplexed Architectures

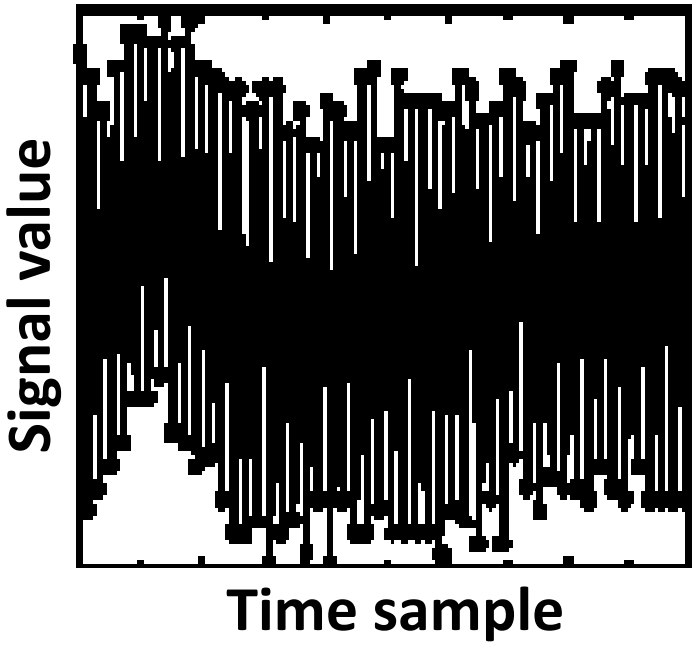
Parallel bus for I,Q





Time-shared bus for I,Q





Time sharing de-correlates I/Q and increases switching activity

Optimizing Multiplications

$A = in \times 0\ 0\ 1\ 1$

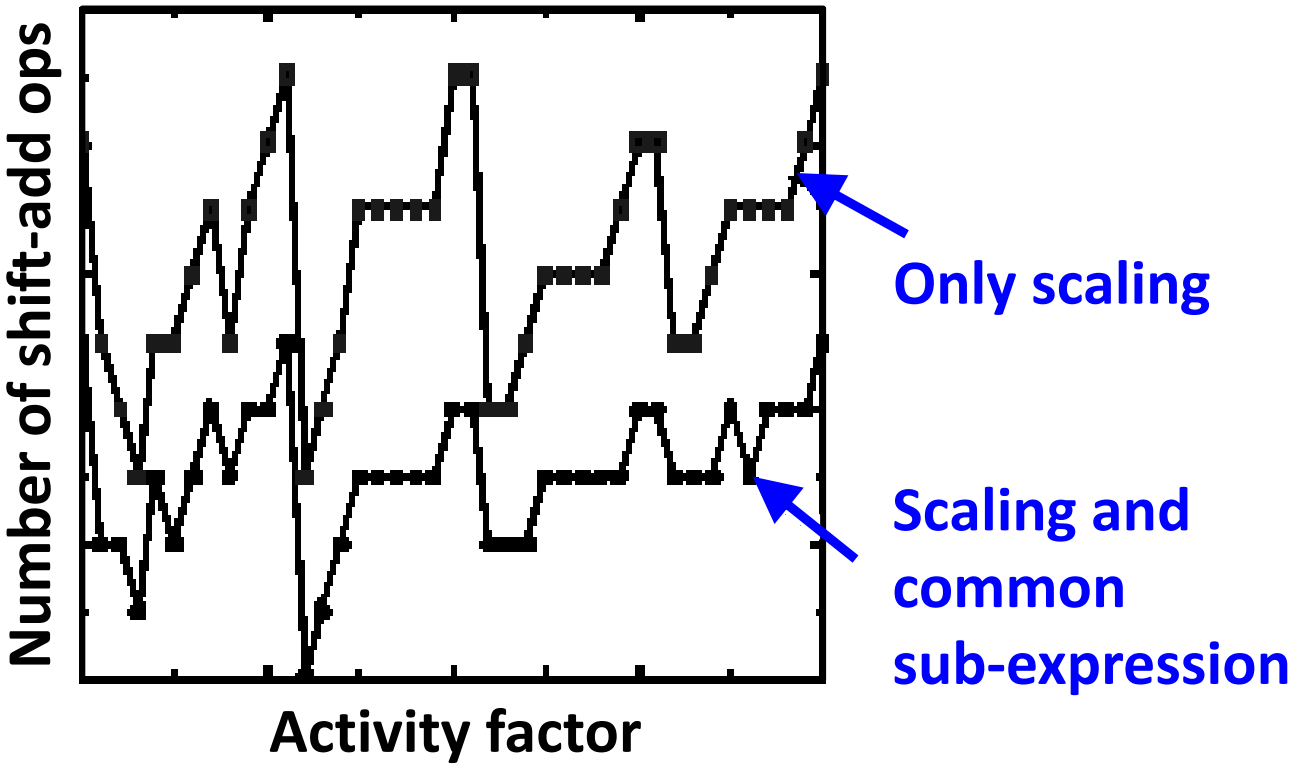
$B = in \times 0\ 1\ 1\ 1$

$A = (in >> 4 + in >> 3)$

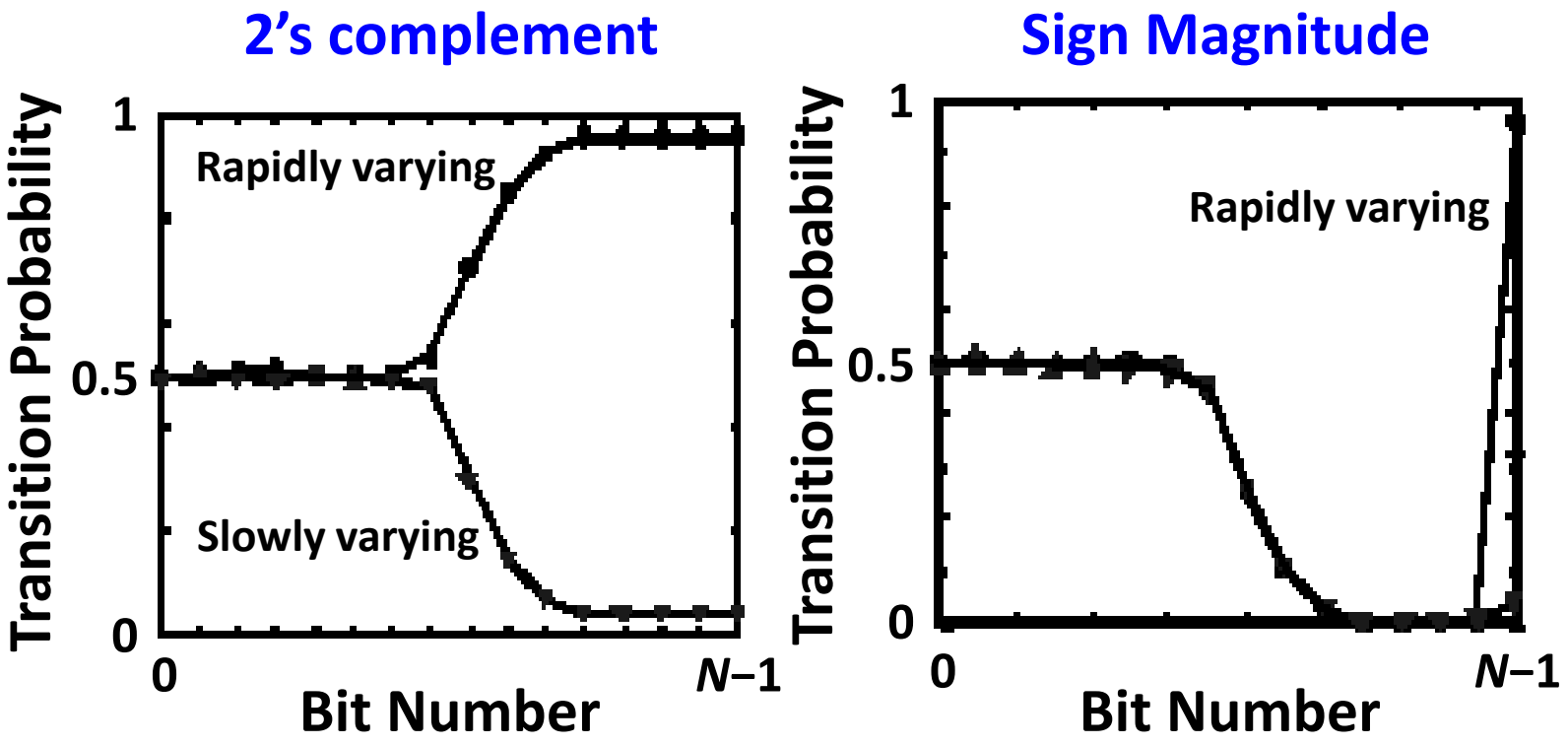
$B = (in >> 4 + in >> 3 + in >> 2)$

$A = (in >> 4 + in >> 3)$

$B = (A + in >> 2)$



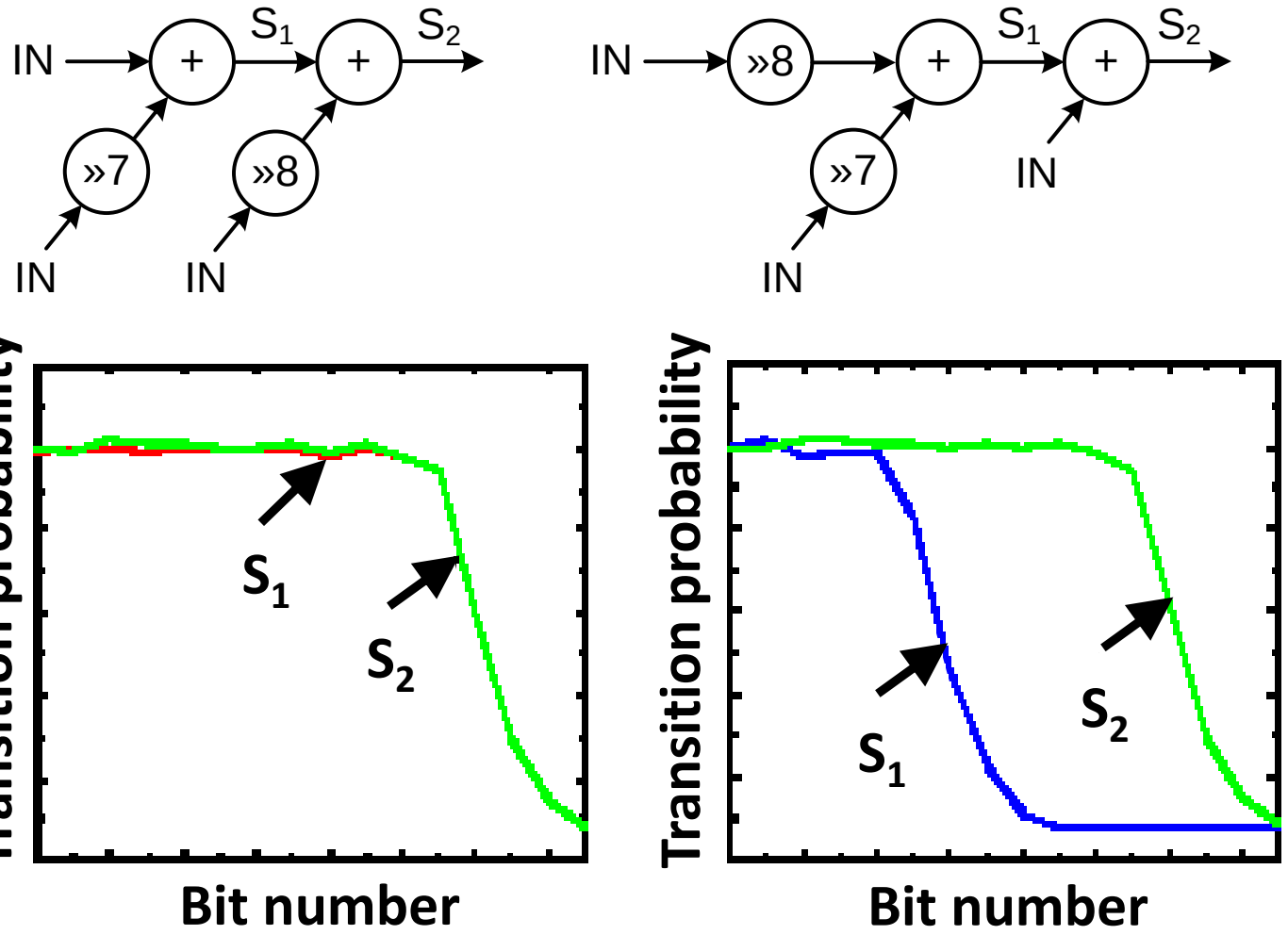
Number Representation



Sign-extension activity is significantly reduced using sign-magnitude representation

A. Chandrakasan, Low Power Digital CMOS Design, Ph.D. Thesis, UC Berkeley, 1994.

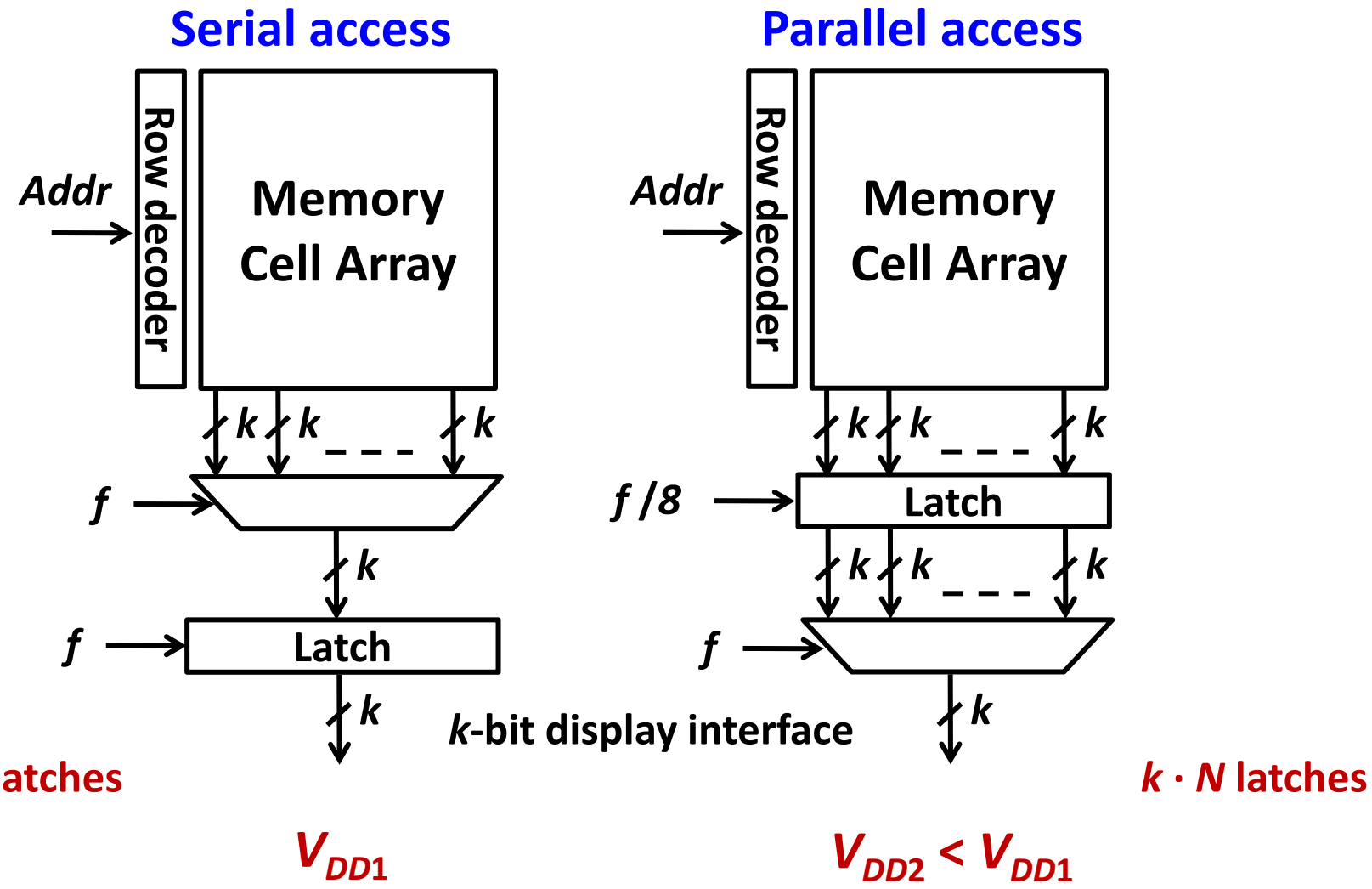
Reducing Activity by Reordering Inputs



30% reduction in switching energy

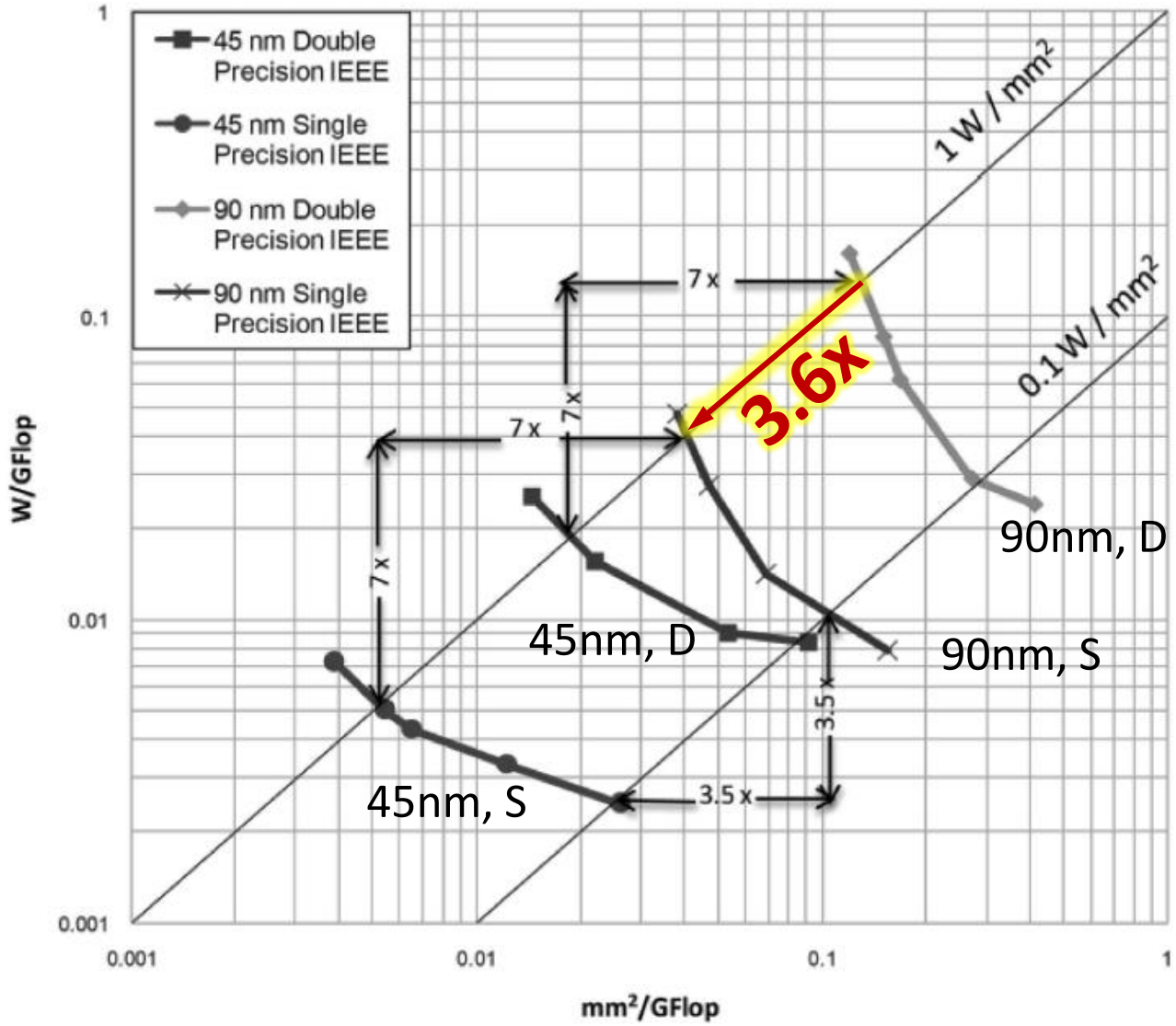
Memory Architecture

- **Pipelining and voltage scaling**



Efficiency: Single vs. Double Precision FPU

Diminishing efficiency gains with scaling at constant power density

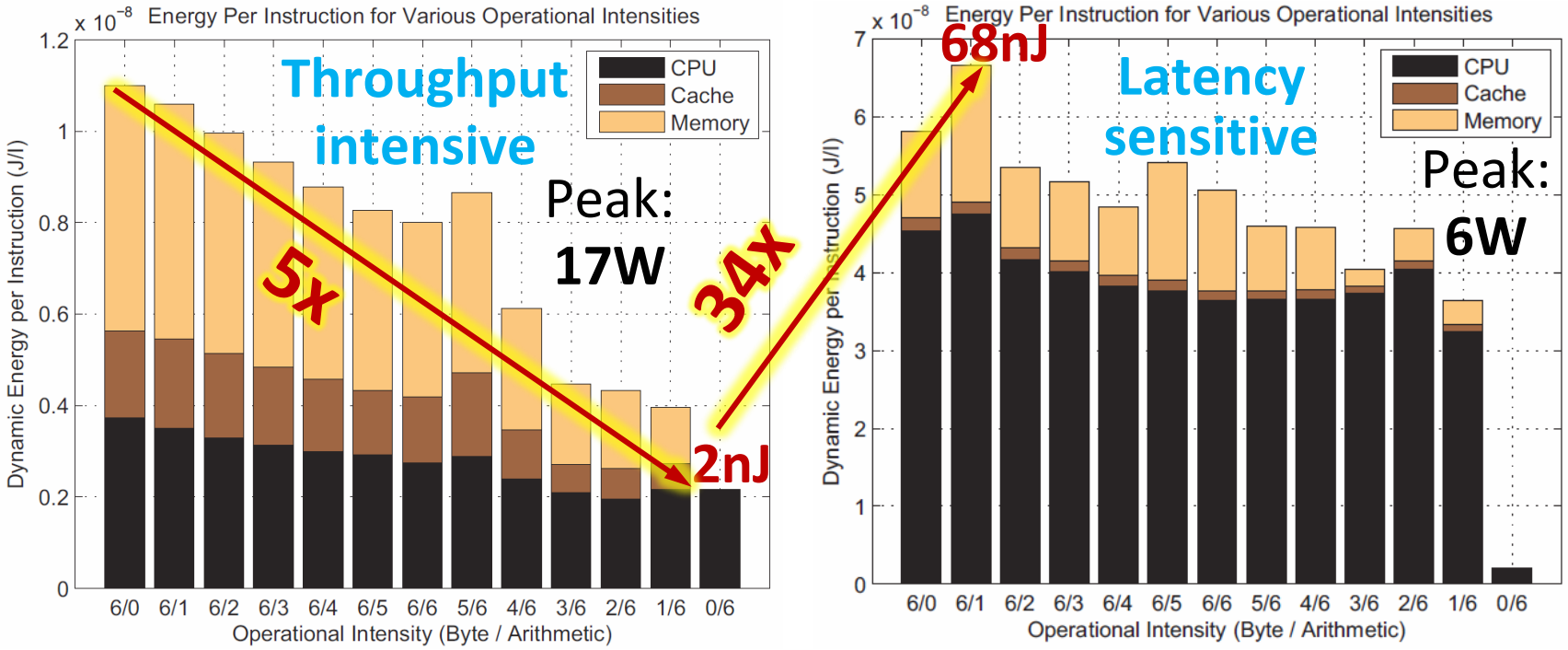


90nm → 45nm
• **1W/mm²** ▪ 7x speedup
• **0.1W/mm²** ▪ 3.5x speedup

From:
S. Galal, M. Horowitz, IEEE Trans. Computers, July 2011, pp. 913-922.

Efficiency: Memory Hierarchies

- **A study based on dual-socket 2.6GHz Xeon E5520**
  - **Throughput:** 4 instructions per clock cycle * 4 cores
  - **Memory:** 8M/256K/32K L3/L2/L1 cache + 12GB DDR3
- **Operational intensity** [Byte/Arithmetic ratio]



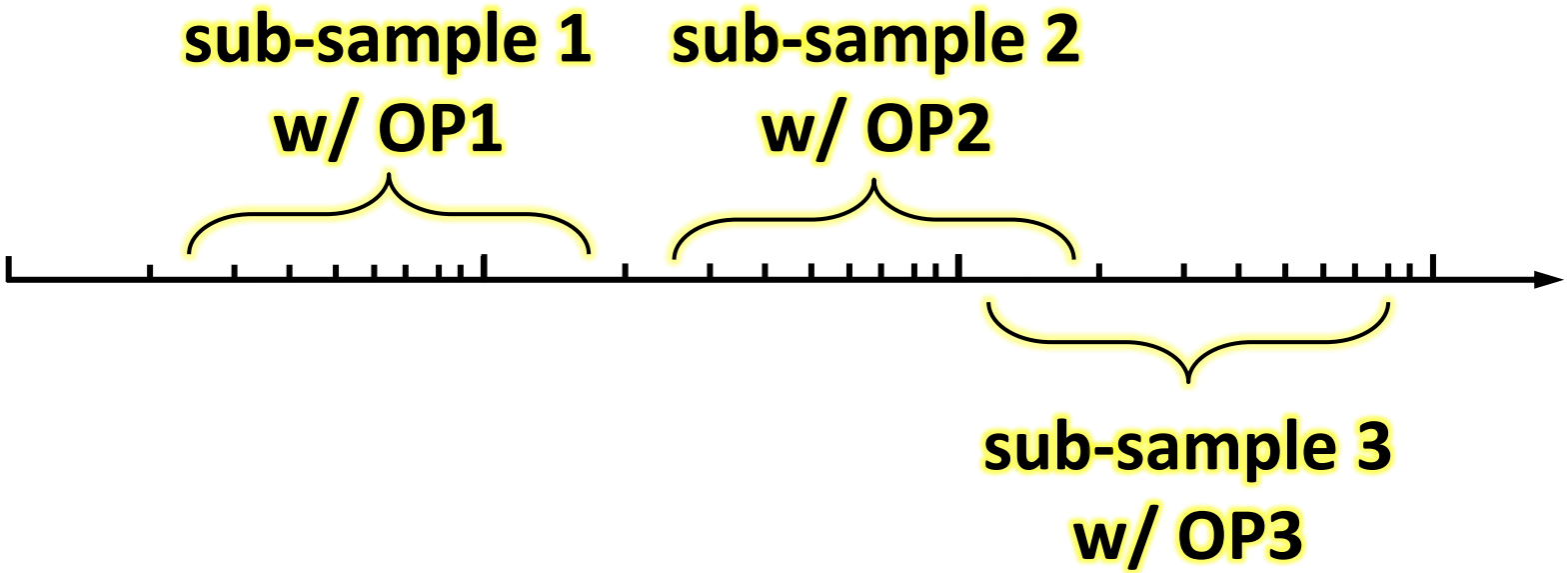
I. Manousakis, D.S. Nikolopoulos, IEEE Int. Symp. Computer Arch & HP Computing, 2012, pp 139-146.

Application Example:

Cognitive Radio Spectrum Sensing

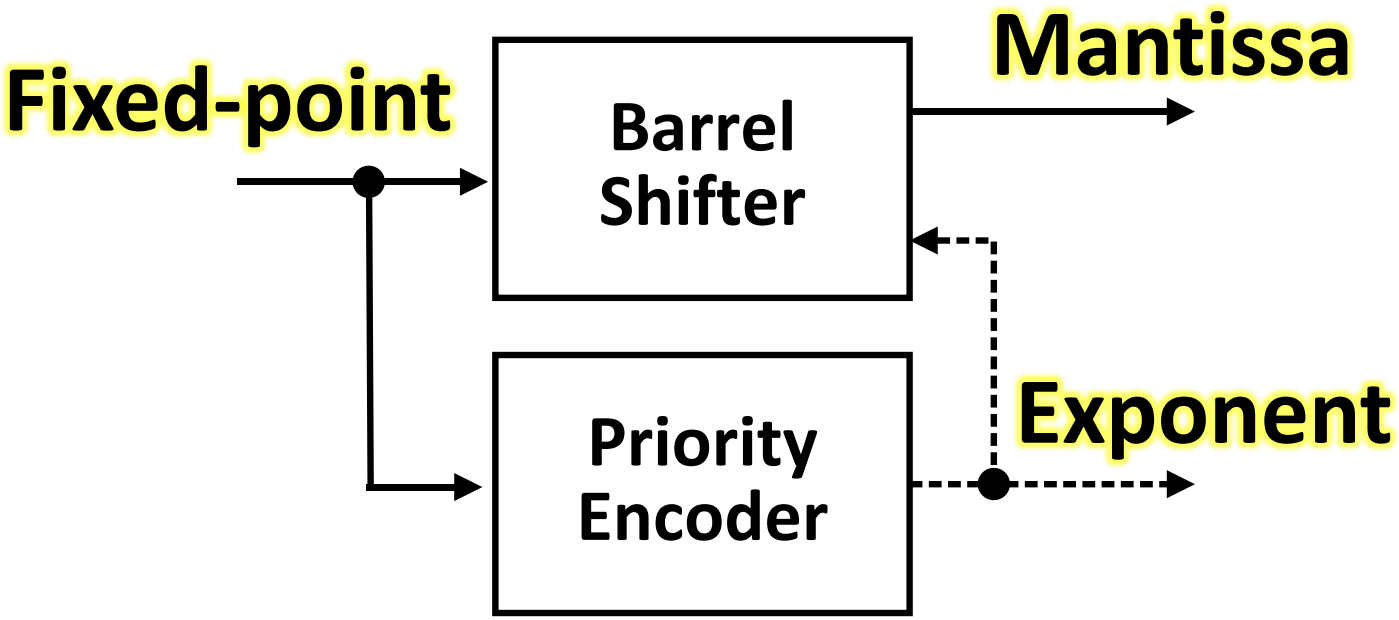
T.-H. Yu, et al., "A 7.4mW 200MS/s Wideband Spectrum Sensing Digital Baseband Processor for Cognitive Radios," IEEE JSSC, vol. 47, no. 9, pp. 2235-2245, Sep. 2012.

Mini Floating-Point

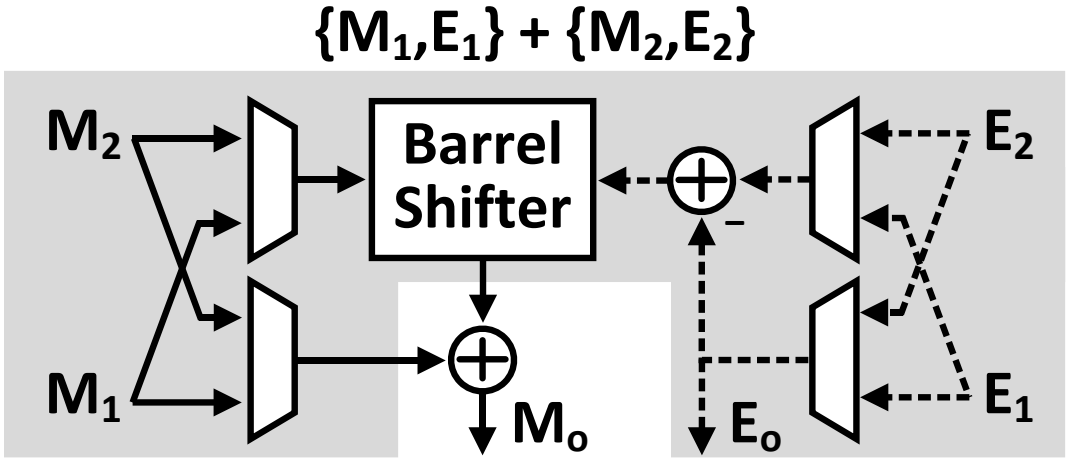


When large-dynamic-range samples can be partition into several small-dynamic-range operations, **mini floating-point** is area efficient

Fixed-Point to Floating-Point Converter

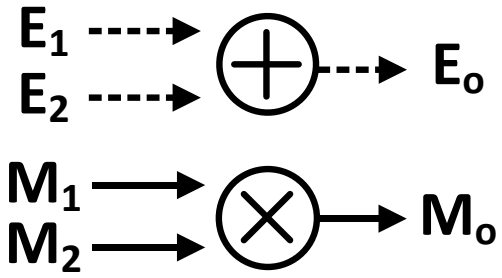


Mini Floating-Point Operation



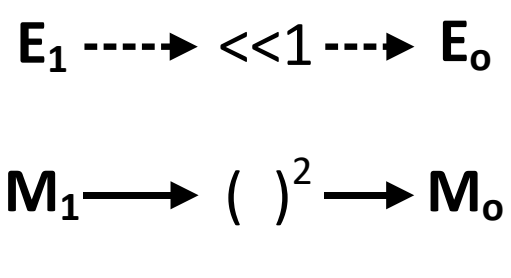
Addition

{M1,E1} x {M2,E2}



Multiplication

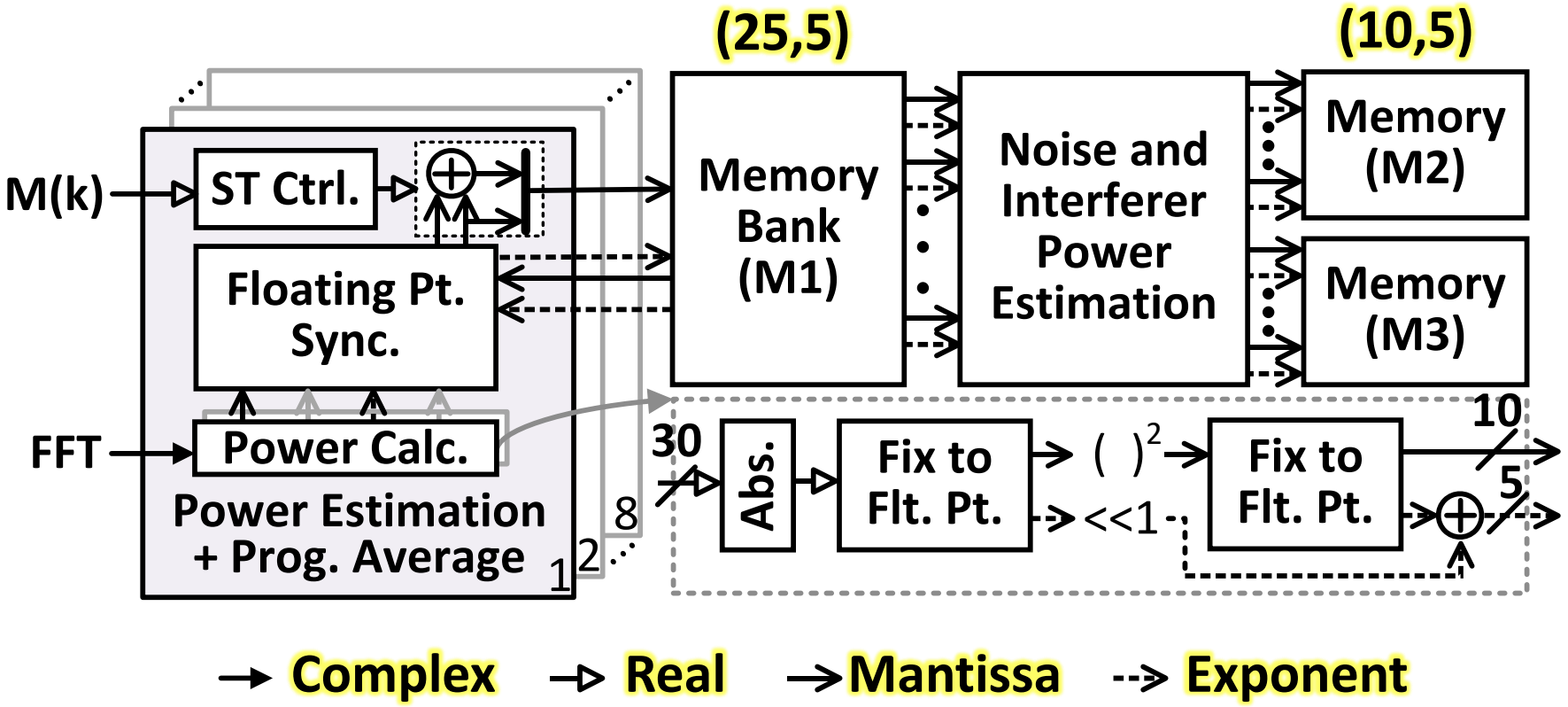
$\{M_1,E_1\}^2$



Squaring

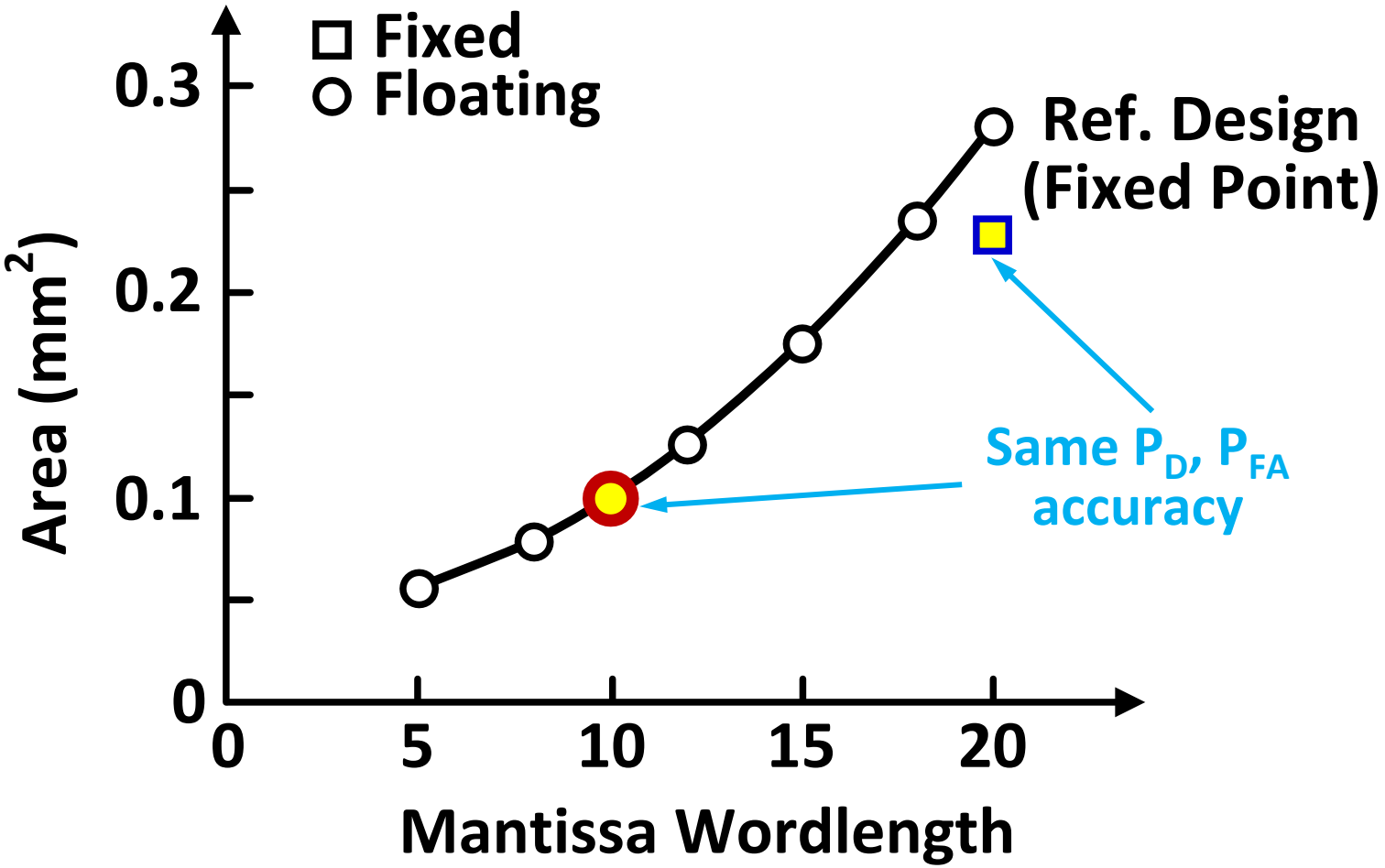
Mini-Float Example: PSD Estimation

- **Large-dynamic-range spectrum is channelized into independent small-dynamic range channels**
  - Mini floating point for area saving

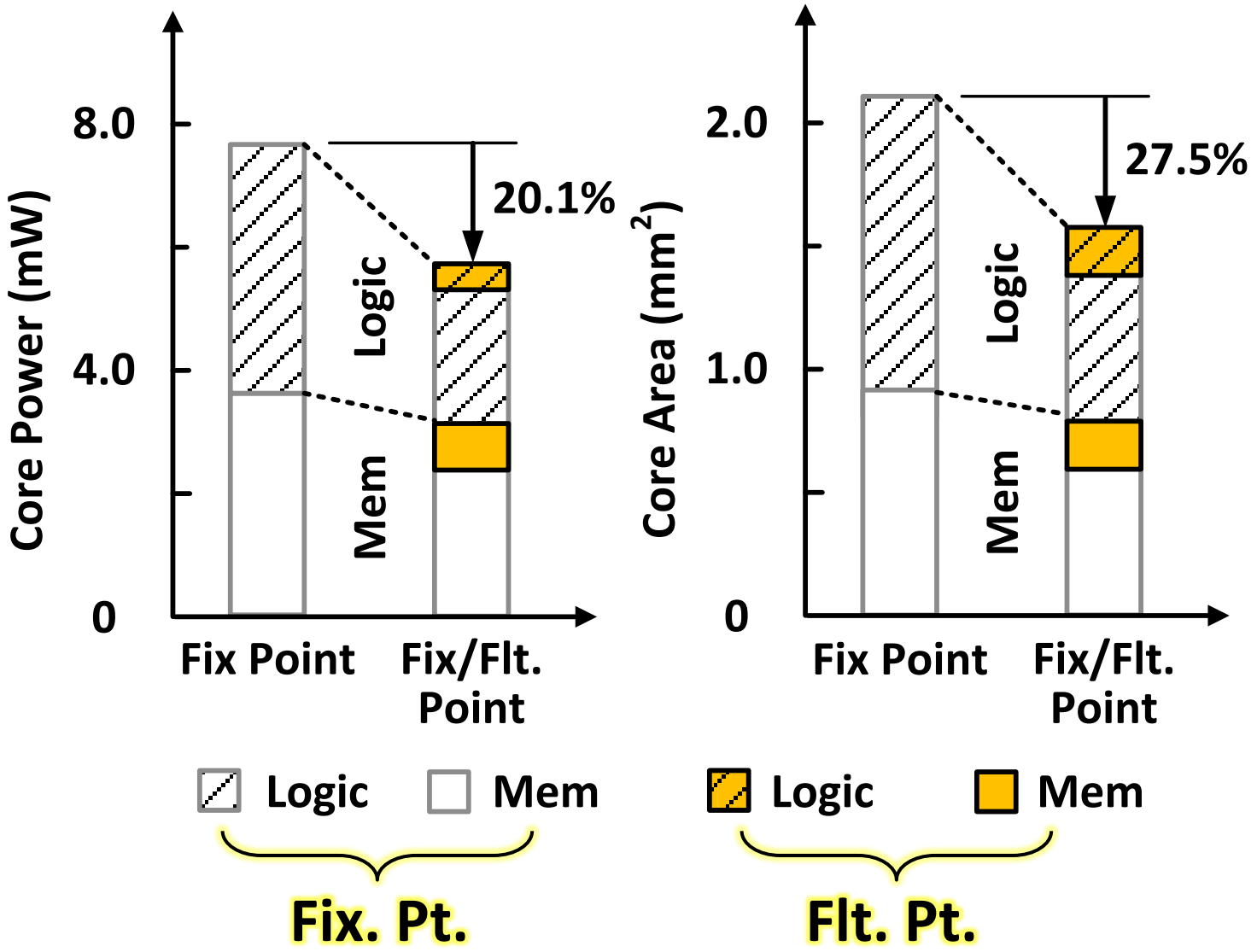


T.-H. Yu, et al., IEEE JSSC, pp. 2235-2245, Sep. 2012.

Optimal Multiplier Wordlength



Impact on Core Power/Area



Summary

- **Algorithm design:** algebraic form, floating-point verification
- **Fixed-point degrades performance due to quantization noise**
  - **Quantization** (round, trunc.) and **overflow** (sat., wrap-around)
    - **Rounding** has zero-mean error: suitable for recursive systems
    - **Saturation:** typically used in feedback systems
- **Data correlation impacts power consumption**
  - Time-shared buses increase activity (reduce correlation)
  - 2's complement has higher switching than sign-magnitude
- **Memory access (esp. DRAM) dominates energy in CPUs**
- **Mini-floating point suitable for some dedicated chips**
  - Reduced memory, lower compute energy