

Introduction to Computer Arithmetic for Efficient Hardware Implementations

Arnaud Tisserand

CNRS, Lab-STICC











CEA-SPEC Seminar, Nov. 2019



Babylonian Arithmetic

Use of a **positional number system** with:

- primary **radix 60**
- auxiliary **radix 10**
- **digits** in the set:

1	2	3	4	5	6	7	8	9	10
									











Example:



Babylonian Arithmetic

Use of a **positional number system** with:

- primary **radix 60**
- auxiliary **radix 10**
- **digits** in the set:

1	2	3	4	5	6	7	8	9	10
									

Example:













$$= 33 \times 60$$

Babylonian Arithmetic

Use of a **positional number system** with:

- primary **radix 60**
- auxiliary **radix 10**
- **digits** in the set:

1	2	3	4	5	6	7	8	9	10
									

Example:













$$= 33 \times 60 + 39$$

Babylonian Arithmetic

Use of a **positional number system** with:

- primary **radix 60**
- auxiliary **radix 10**
- **digits** in the set:

1	2	3	4	5	6	7	8	9	10
									

Example:



$$= 33 \times 60 + 39 = 2019$$

Egyptian Multiplication

$$M(n, m, p) = n \times m + p$$

Rewriting rules:

$$R1 : M(0, m, p) \mapsto p$$

$$R2 : M(2n, m, p) \mapsto M(n, 2m, p)$$

$$R3 : M(2n + 1, m, p) \mapsto M(n, 2m, p + m)$$

Example:

$$12 \times 12$$

Egyptian Multiplication

$$M(n, m, p) = n \times m + p$$

Rewriting rules:

$$R1 : M(0, m, p) \mapsto p$$

$$R2 : M(2n, m, p) \mapsto M(n, 2m, p)$$

$$R3 : M(2n + 1, m, p) \mapsto M(n, 2m, p + m)$$

Example:

$$12 \times 12 = M(12, 12, 0)$$

Egyptian Multiplication

$$M(n, m, p) = n \times m + p$$

Rewriting rules:

$$R1 : M(0, m, p) \mapsto p$$

$$R2 : M(2n, m, p) \mapsto M(n, 2m, p)$$

$$R3 : M(2n + 1, m, p) \mapsto M(n, 2m, p + m)$$

Example:

$$\begin{aligned} 12 \times 12 &= M(12, 12, 0) \\ &= M(6, 24, 0) \end{aligned}$$

Egyptian Multiplication

$$M(n, m, p) = n \times m + p$$

Rewriting rules:

$$R1 : M(0, m, p) \mapsto p$$

$$R2 : M(2n, m, p) \mapsto M(n, 2m, p)$$

$$R3 : M(2n + 1, m, p) \mapsto M(n, 2m, p + m)$$

Example:

$$\begin{aligned} 12 \times 12 &= M(12, 12, 0) \\ &= M(6, 24, 0) \\ &= M(3, 48, 0) \end{aligned}$$

Egyptian Multiplication

$$M(n, m, p) = n \times m + p$$

Rewriting rules:

$$R1 : M(0, m, p) \mapsto p$$

$$R2 : M(2n, m, p) \mapsto M(n, 2m, p)$$

$$R3 : M(2n + 1, m, p) \mapsto M(n, 2m, p + m)$$

Example:

$$\begin{aligned} 12 \times 12 &= M(12, 12, 0) \\ &= M(6, 24, 0) \\ &= M(3, 48, 0) \\ &= M(1, 96, 48) \end{aligned}$$

Egyptian Multiplication

$$M(n, m, p) = n \times m + p$$

Rewriting rules:

$$R1 : M(0, m, p) \mapsto p$$

$$R2 : M(2n, m, p) \mapsto M(n, 2m, p)$$

$$R3 : M(2n + 1, m, p) \mapsto M(n, 2m, p + m)$$

Example:

$$\begin{aligned} 12 \times 12 &= M(12, 12, 0) \\ &= M(6, 24, 0) \\ &= M(3, 48, 0) \\ &= M(1, 96, 48) \\ &= M(0, 192, 144) \end{aligned}$$

Egyptian Multiplication

$$M(n, m, p) = n \times m + p$$

Rewriting rules:

$$R1 : M(0, m, p) \mapsto p$$

$$R2 : M(2n, m, p) \mapsto M(n, 2m, p)$$

$$R3 : M(2n + 1, m, p) \mapsto M(n, 2m, p + m)$$

Example:

$$\begin{aligned} 12 \times 12 &= M(12, 12, 0) \\ &= M(6, 24, 0) \\ &= M(3, 48, 0) \\ &= M(1, 96, 48) \\ &= M(0, 192, 144) \\ &= 144 \end{aligned}$$

Computer Arithmetic

representations
of numbers

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \simeq \mathbb{R}, \mathbb{F}_q$

algorithms

$\pm, \times, \div, \sqrt{}, \text{mod},$
 $\forall, \exists, e^x, \simeq f(x), \dots$

Computer Arithmetic

representations
of numbers

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \simeq \mathbb{R}, \mathbb{F}_q$

algorithms

$\pm, \times, \div, \sqrt{}, \text{mod},$
 $\forall \leq, e^x, \simeq f(x), \dots$

implementation

soft GPP/SP,
ASIC, FPGA

Computer Arithmetic

representations
of numbers

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \simeq \mathbb{R}, \mathbb{F}_q$

algorithms

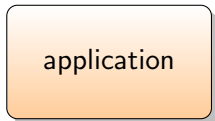
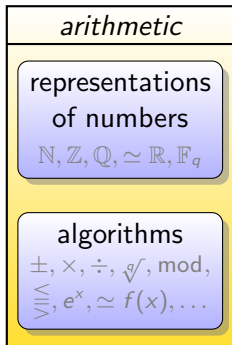
$\pm, \times, \div, \sqrt{}, \text{mod},$
 $\forall \llcorner, e^x, \simeq f(x), \dots$

implementation

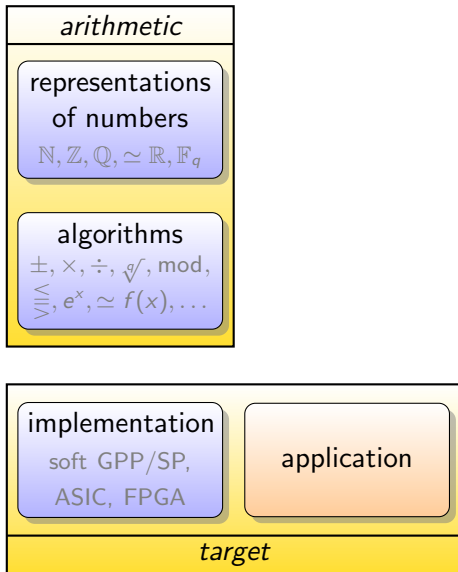
soft GPP/SP,
ASIC, FPGA

application

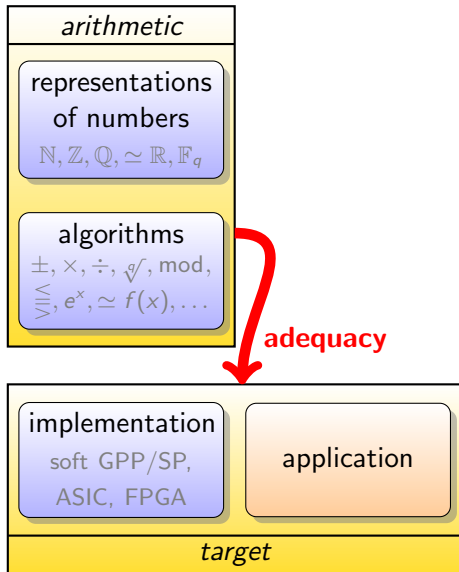
Computer Arithmetic



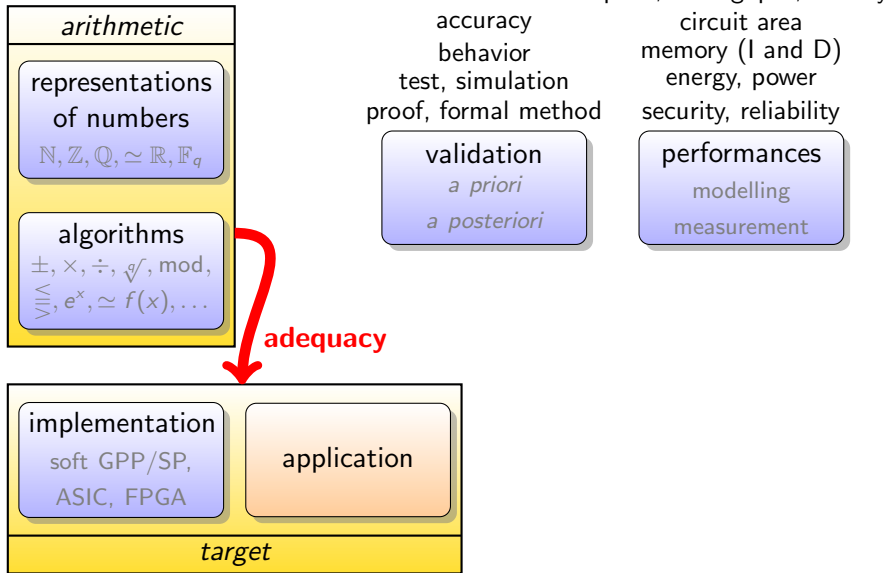
Computer Arithmetic



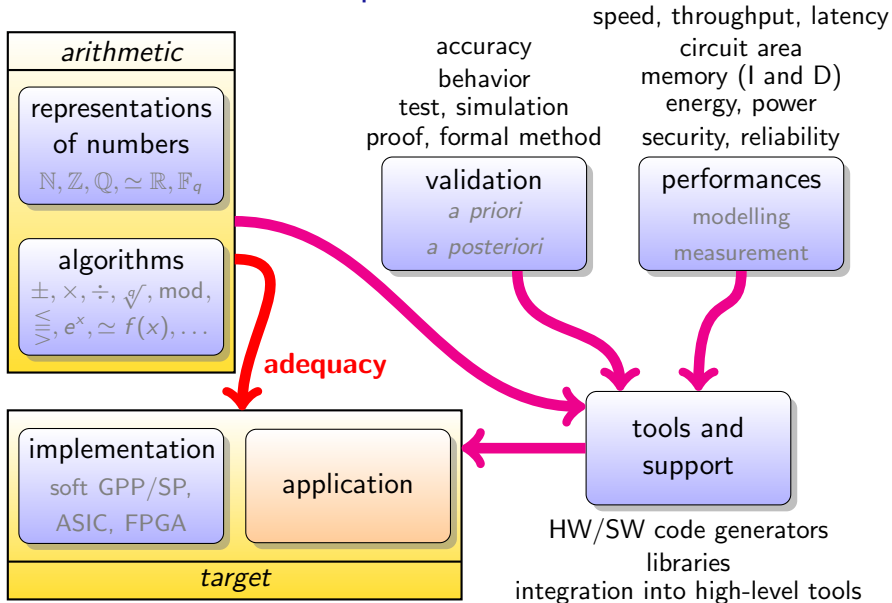
Computer Arithmetic



Computer Arithmetic



Computer Arithmetic



Number Systems

- **set** of represented numbers
 - ▶ integers: \mathbb{N}, \mathbb{Z}
 - ▶ rationals: \mathbb{Q}
 - ▶ real approximations: subset of \mathbb{R}
 - ▶ complex approximations: subset of \mathbb{C}
 - ▶ finite fields: $\mathbb{F}_p, \mathbb{F}_{2^m}, \mathbb{F}_{3^m}$
 - ▶ ...
- **system properties**
 - ▶ positional or non positional
 - ▶ redundant or non redundant
 - ▶ fixed precision or arbitrary precision (multiple precision)
 - ▶ completeness (in a finite set)
 - ▶ ...

Number system =

1. data format and encoding
2. a set of interpretation rules for the encoding

Positional Number System(s)

$$X = \sum_{i=-m}^{n-1} x_i \beta^i = (x_{n-1}x_{n-2} \cdots x_1x_0 \cdot x_{-1}x_{-2} \cdots x_{-m})$$

- **radix** β (usually a power of 2)
- **digits** x_i ($\in \mathbb{N}$) in the **digit set** \mathcal{D}
- **rank** or **position** i , **weight** β^i
- n **integer** digits, m **fractional** digits

Examples:

- $\beta = 10, \mathcal{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\beta = 2, \mathcal{D} = \{0, 1\}$

Positional Number System(s)

$$X = \sum_{i=-m}^{n-1} x_i \beta^i = (x_{n-1}x_{n-2} \cdots x_1x_0 \cdot x_{-1}x_{-2} \cdots x_{-m})$$

- **radix** β (usually a power of 2)
- **digits** x_i ($\in \mathbb{N}$) in the **digit set** \mathcal{D}
- **rank** or **position** i , **weight** β^i
- n **integer** digits, m **fractional** digits

Examples:

- $\beta = 10, \mathcal{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\beta = 2, \mathcal{D} = \{0, 1\}$
- carry save: $\beta = 2, \mathcal{D}_{\text{cs}} = \{0, 1, 2\}$
- borrow save: $\beta = 2, \mathcal{D}_{\text{bs}} = \{-1, 0, 1\}$

Positional Number System(s)

$$X = \sum_{i=-m}^{n-1} x_i \beta^i = (x_{n-1}x_{n-2} \cdots x_1x_0 \cdot x_{-1}x_{-2} \cdots x_{-m})$$

- **radix** β (usually a power of 2)
- **digits** x_i ($\in \mathbb{N}$) in the **digit set** \mathcal{D}
- **rank** or **position** i , **weight** β^i
- n **integer** digits, m **fractional** digits

Examples:

- $\beta = 10, \mathcal{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\beta = 2, \mathcal{D} = \{0, 1\}$
- carry save: $\beta = 2, \mathcal{D}_{cs} = \{0, 1, 2\}$
- borrow save: $\beta = 2, \mathcal{D}_{bs} = \{-1, 0, 1\}$
- signed digits: $\beta > 2, \mathcal{D}_{sd,\alpha,\beta} = \{-\alpha, \dots, \alpha\}$ with $2\alpha + 1 \geq \beta$

Positional Number System(s)

$$X = \sum_{i=-m}^{n-1} x_i \beta^i = (x_{n-1}x_{n-2} \cdots x_1x_0 \cdot x_{-1}x_{-2} \cdots x_{-m})$$

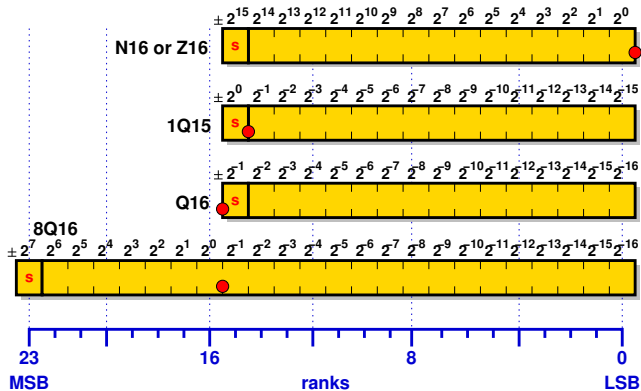
- **radix** β (usually a power of 2)
- **digits** x_i ($\in \mathbb{N}$) in the **digit set** \mathcal{D}
- **rank** or **position** i , **weight** β^i
- n **integer** digits, m **fractional** digits

Examples:

- $\beta = 10, \mathcal{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\beta = 2, \mathcal{D} = \{0, 1\}$
- carry save: $\beta = 2, \mathcal{D}_{cs} = \{0, 1, 2\}$
- borrow save: $\beta = 2, \mathcal{D}_{bs} = \{-1, 0, 1\}$
- signed digits: $\beta > 2, \mathcal{D}_{sd,\alpha,\beta} = \{-\alpha, \dots, \alpha\}$ with $2\alpha + 1 \geq \beta$
- theoretical systems: $\beta = \frac{1+\sqrt{5}}{2}, \beta = 1 + i \dots$

Fixed-Point Representations

Widely used in DSPs and digital integrated circuits for higher speed, lower silicon area and power consumption compared to floating point



Typical fixed-point formats: 16, 24, 32 and 48 bits

Representation(s) of Numbers and Power Consumption

Impact of the **representation of numbers**:

- operator speed
- circuit area
- **useful** and **useless** activity

cycle	value	2's complement	t_{c2}	sign/magnitude	t_{sm}
0	0	0000000000000000	0	0000000000000000	0
1	1	0000000000000001	1	0000000000000001	1
2	-1	1111111111111111	15	1000000000000001	1
3	8	0000000000001000	15	0000000000001000	3
4	-27	11111111111100101	15	1000000000011011	4
5	27	0000000000011011	15	0000000000011011	1
total			61		10

- sign/magnitude (absolute value):

$$A = (s_a a_{n-2} \dots a_1 a_0) = (-1)^{s_a} \times \sum_{i=0}^{n-2} a_i 2^i$$

- 2's complement:

$$A = (a_{n-1} a_{n-2} \dots a_1 a_0) = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

Floating-Point Representation(s)

Radix- β floating-point representation of x :

- **sign** s_x , 1-bit encoding: $0 \Rightarrow x > 0$ and $1 \Rightarrow x < 0$
- **exponent** $e_x \in \mathbb{N}$ on k digits and $e_{min} \leq e_x \leq e_{max}$
- **mantissa** m_x on $n + 1$ digits
- encoding:

$$x = (-1)^{s_x} \times m_x \times \beta^{e_x}$$

$$m_x = x_0 . x_1 x_2 x_3 \cdots x_n$$

$$x_i \in \{0, 1, \dots, \beta - 1\}$$

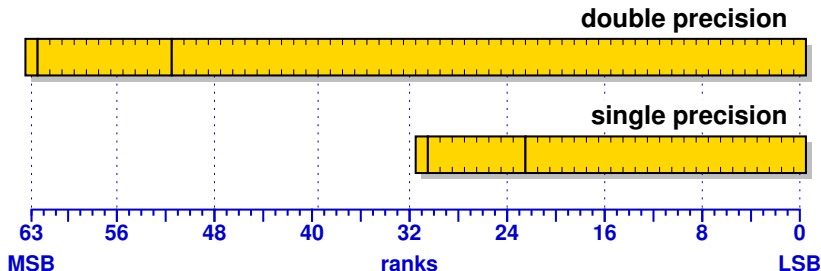
For accuracy purpose, the mantissa must be **normalized** ($x_0 \neq 0$)

Then $m_x \in [1, \beta[$ and a specific encoding is required for the number 0

IEEE-754: basic formats

Radix $\beta = 2$, the first bit of the normalized mantissa is always a “1”
(non-stored implicit bit)

format	number of bits			
	total	sign	exponent	mantissa
double precision	64	1	11	52 + 1
simple precision	32	1	8	23 + 1

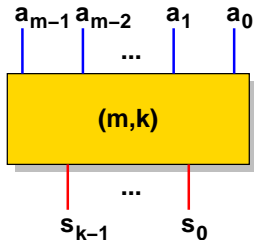


Basic Cells for Addition

Useful circuit element in computer arithmetic: **counter**

A **(m, k) -counter** is a cell that counts the number of 1 on its m inputs (result expressed as a k -bit integer)

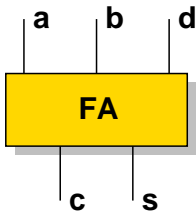
$$\sum_{i=0}^{m-1} a_i = \sum_{j=0}^{k-1} s_j 2^j$$



Standard counters:

- **half-adder** or **HA** is a $(2, 2)$ -counter
- **full-adder** or **FA** is a $(3, 2)$ -counter

FA Cell



<i>a</i>	<i>b</i>	<i>d</i>	<i>c</i>	<i>s</i>
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

Arithmetic equation:

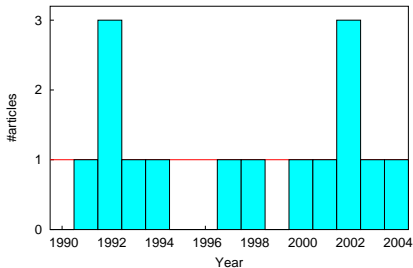
$$2c + s = a + b + d$$

Logic equation:

$$s = a \oplus b \oplus d$$

$$c = ab + ad + bd$$

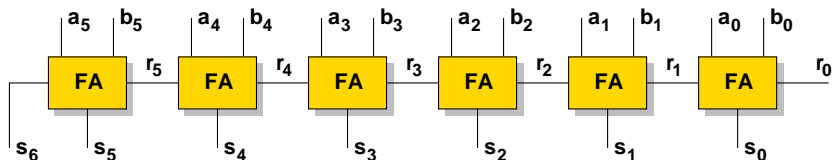
Articles about FA in IEEE Journals



There many implementations of the FA cell

Carry Ripple Adder (CRA)

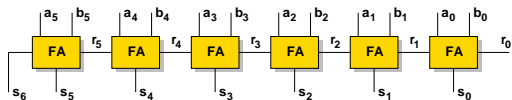
Very simple architecture: n FA cells connected in **series**



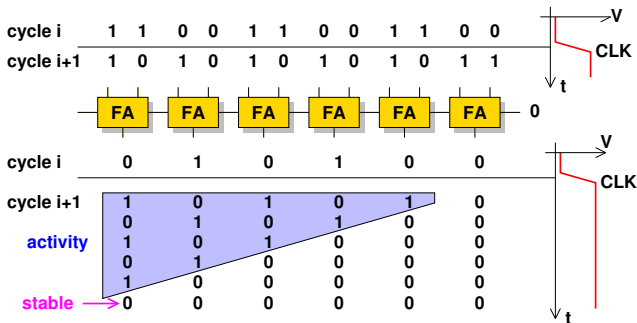
	complexity
delay	$O(n)$
area	$O(n)$

Warning: Sometimes a CRA is also called *Carry Propagate Adder* (CPA), **but** CPA also means a non-redundant adder (that propagates)

Useless Activity in a Carry Ripple Adder



Very simple architecture:
 n FA cells connected in **series**

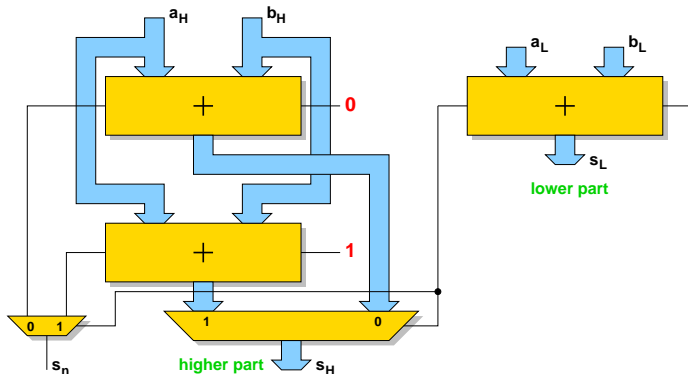


Theoretical models (equiprobable and uniform distribution of inputs):

- worst case $n^2/2$ transitions
- average $3n/2$ transitions and only $n/2$ useful

Carry-Select Adder

Idea: computation of the higher half part for the 2 possible input carries (0 and 1) and selection when the output carry from lower half part is known



Recursive version $\rightarrow O(\log n)$ delay

but there is a fanout problem...

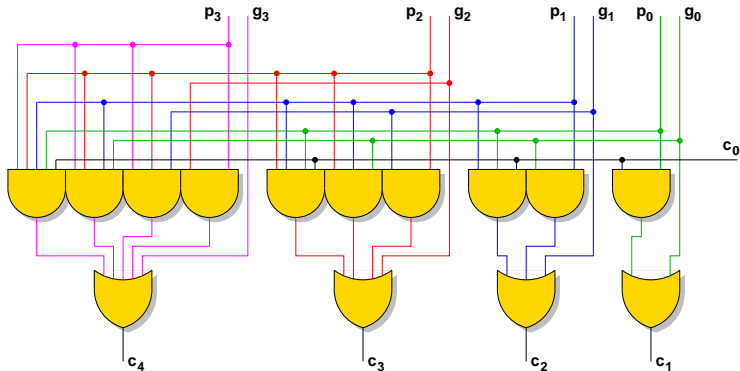
Carry Lookahead Adder: 4-Bit Example

$$c_1 = g_0 + p_0 c_0$$

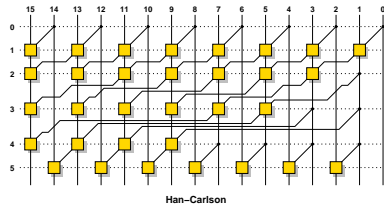
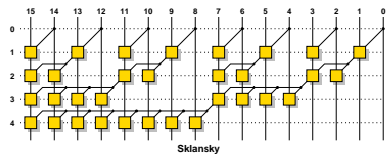
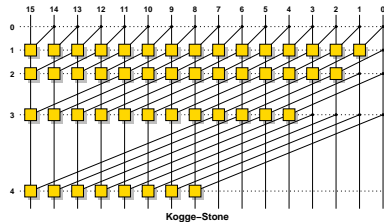
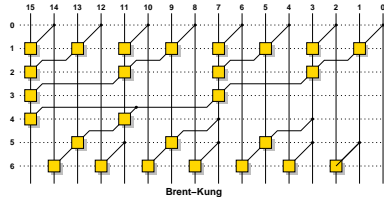
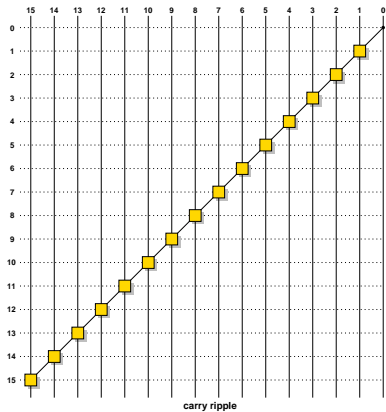
$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$



Parallel-Prefix Addition: Standard Architectures



Redundant or Constant Time Adders

To speed-up the addition, one solution consists in “saving” the carries and using them (this makes sense only in case of multiple additions)

In 1961, Avizienis suggested to represent numbers in radix β with digits in $\{-\alpha, -\alpha + 1, \dots, 0, \dots, \alpha - 1, \alpha\}$ instead of $\{0, 1, 2, \dots, \beta - 1\}$ with $\alpha \leq \beta - 1$

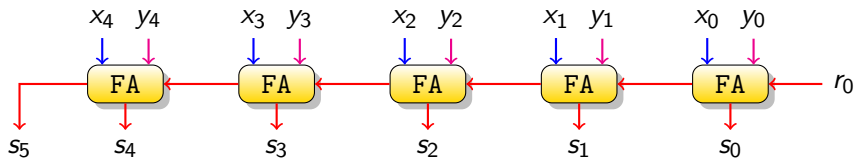
Using this representation, if $2\alpha + 1 > \beta$ some numbers have several possible representation at the bit level. For instance, the value 2345 (in the standard representation) can be represented in radix 10 with digits in $\{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$ by the values 2345, 235(-5) or 24(-5)(-5)

Such a representation is said **redundant**

In a redundant number system there is constant-time addition algorithm (without carry propagation) where all computations are done in parallel

Addition

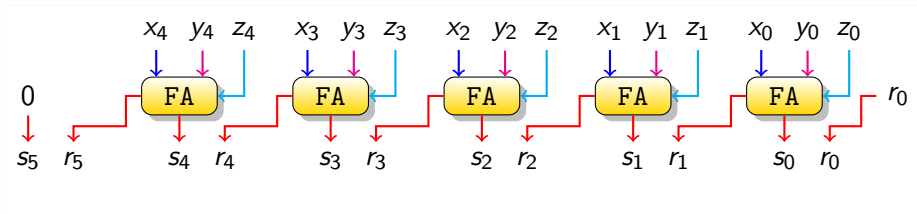
Q: How can we speed up addition?



Addition

Q: How can we speed up addition?

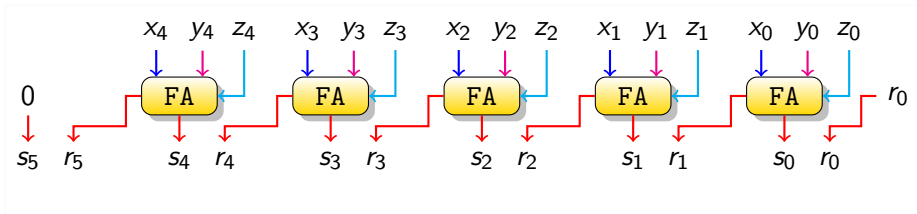
R: Save the carries!



Addition

Q: How can we speed up addition?

R: Save the carries!



$$X + Y + Z = S + R = \sum_{i=0}^n (s_i + r_i) 2^i$$

The computation time does not depend on n

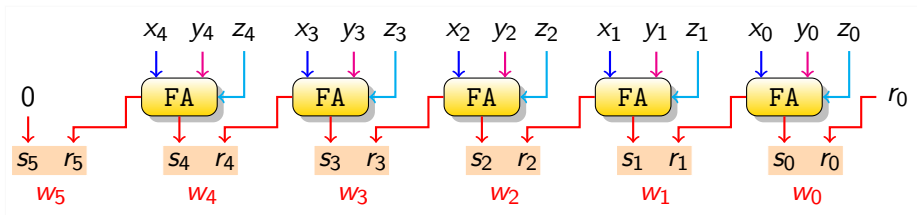


$$T(n) = O(1)$$

Addition using the *carry-save* representation

Q: How can we speed up addition?

R: **Save** the carries!



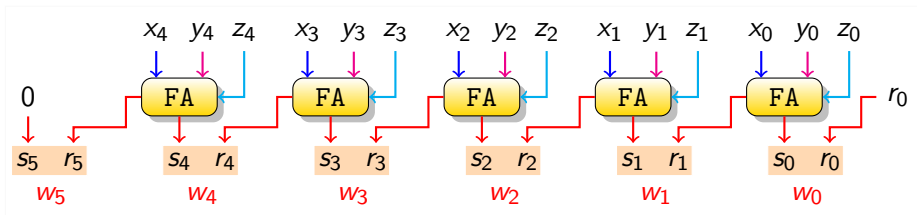
$$\begin{aligned} X + Y + Z &= S + R = \sum_{i=0}^n (s_i + r_i) 2^i \\ &= W = \sum_{i=0}^n w_i 2^i \quad \text{avec } w_i = s_i + r_i \in \{0, 1, 2\} \end{aligned}$$

The computation time does **not depend** on n \rightarrow $T(n) = O(1)$

Addition using the *carry-save* representation

Q: How can we speed up addition?

R: **Save** the carries!



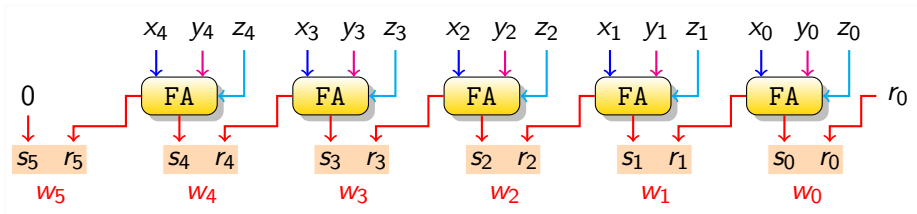
$$\begin{aligned} X + Y + Z &= S + R = \sum_{i=0}^n (s_i + r_i) 2^i \\ &= W = \sum_{i=0}^n w_i 2^i \quad \text{avec } w_i = s_i + r_i \in \{0, 1, 2\} \end{aligned}$$

The computation time does **not depend** on n \rightarrow $T(n) = O(1)$

Addition using the *carry-save* representation

Q: How can we speed up addition?

R: **Save** the carries!



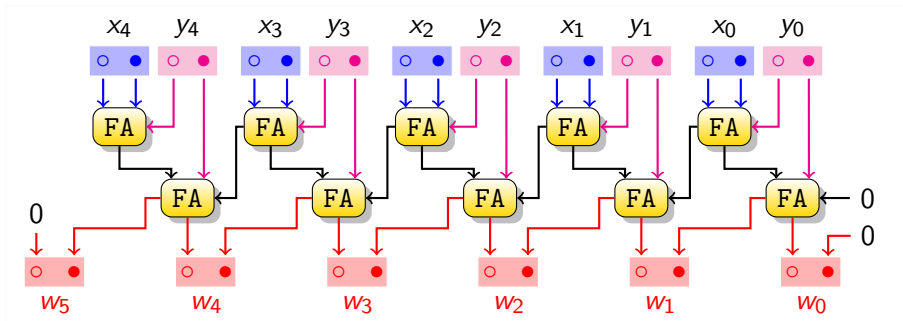
$$X + Y + Z = S + R = \sum_{i=0}^n (s_i + r_i) 2^i$$

$$= W = \sum_{i=0}^n w_i 2^i \quad \text{avec} \quad w_i = s_i + r_i \in \{0, 1, 2\}$$

$$= \left(w_n w_{n-1} \dots w_1 w_0 \right)_{cs} = \left(\begin{array}{|c|c|} \hline s_n & r_n \\ \hline \end{array} \begin{array}{|c|c|} \hline s_{n-1} & r_{n-1} \\ \hline \end{array} \dots \begin{array}{|c|c|} \hline s_1 & r_1 \\ \hline \end{array} \begin{array}{|c|c|} \hline s_0 & r_0 \\ \hline \end{array} \right)_{cs}$$

The computation time does **not depend** on n \rightarrow $T(n) = O(1)$

Addition of 2 Carry-Save Numbers



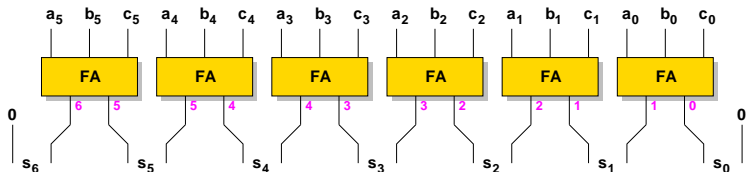
$$X = \sum_{i=0}^n x_i 2^i \quad \text{avec} \quad x_i = x_{s,i} + x_{r,i} = \circ + \bullet$$

$$Y = \sum_{i=0}^n y_i 2^i \quad \text{avec} \quad y_i = y_{s,i} + y_{r,i} = \circ + \bullet$$

$$X+Y = W = \sum_{i=0}^n w_i 2^i \quad \text{avec} \quad w_i = w_{s,i} + w_{r,i} = \circ + \bullet$$

Carry-Save Trees

Example with 3 inputs: A , B and C

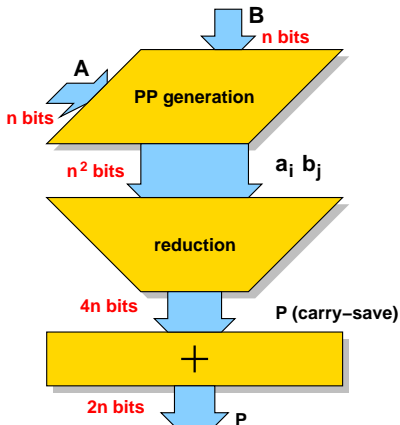


Carry-save reduction tree: $n(h)$ non-redundant inputs can be reduced by a h -level carry-save tree where $n(h) = \lfloor 3n(h-1)/2 \rfloor$ and $n(0) = 2$

h	1	2	3	4	5	6	7	8	9	10	11
$n(h)$	3	4	6	9	13	19	28	42	63	94	141

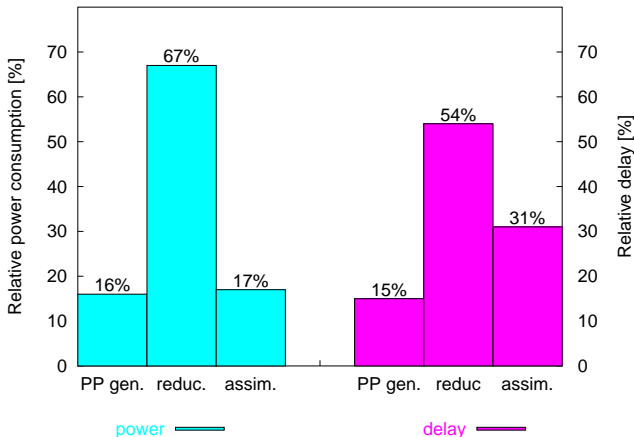
Fast Multipliers

1. partial products generation $a_i b_j$
(with or without recoding)
 \hookrightarrow delay in $O(1)$ (fanout a_i, b_j
 $O(\log n)$)
2. sum of the partial products using
a *carry-save* reduction tree
 \hookrightarrow delay in $O(\log n)$
3. assimilation of the carries using a
fast adder
 \hookrightarrow delay in $O(\log n)$



Multiplication delay $O(\log n)$, area $O(n^2)$

Power Consumption in Fast Multipliers



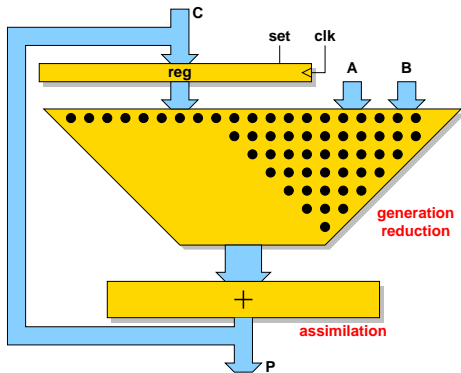
- 30% to 70% of redundant transitions (useless)
- place and route steps based on the internal arrival time
- add a pipeline stage

MAC and FMA

MAC: multiply and accumulate $P(t) = A \times B + P(t - 1)$

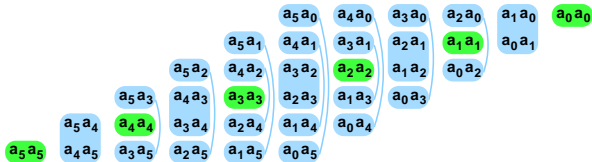
A, B are n -bit values and P a m -bit with $m \gg n$ (e.g., $16 \times 16 + 40 \rightarrow 40$ in some DSPs)

FMA: fused multiply and add $P = A \times B + C$ where A, B, C and P can be stored in different registers (recent general purpose processors, e.g., Itanium)



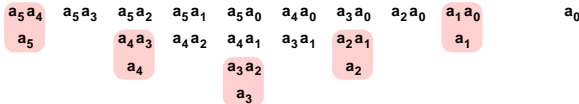
Squarer

$$\begin{array}{r} \times \\ a_5 \quad a_4 \quad a_3 \quad a_2 \quad a_1 \quad a_0 \\ a_5 \quad a_4 \quad a_3 \quad a_2 \quad a_1 \quad a_0 \end{array}$$

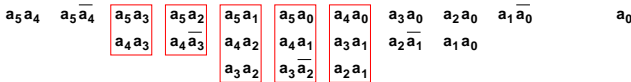


$$a_i a_i = a_i$$

$$a_i a_j + a_j a_i = 2a_i a_j$$



$$\begin{aligned} a_i a_j + a_i &= 2a_i a_j + a_i - a_i a_j \\ &= 2a_i a_j + a_i (1 - a_j) \\ &= 2a_i a_j + a_i \bar{a}_j \end{aligned}$$



15 AND + 5 IAND
3 FA + 2 HA



1 ADD(9 bits)

Multiplication by Constants (1/2)

Problem: substitute a complete multiplier by an optimized sequence of shifts and additions and/or subtractions

Example: $p = 111463 \times x$

algo.	$p = 111463 \times x =$	#op.
direct	$(x \ll 16) + (x \ll 15) + (x \ll 13) + (x \ll 12) + (x \ll 9) + (x \ll 8) + (x \ll 6) + (x \ll 5) + (x \ll 2) + (x \ll 1) + x$	10 \pm
CSD	$(x \ll 17) - (x \ll 14) - (x \ll 12) + (x \ll 10) - (x \ll 7) - (x \ll 5) + (x \ll 3) - x$	7 \pm
Bernstein	$((t_2 \ll 2) + x) \ll 3 - x$ where $t_1 = ((x \ll 3) - x) \ll 2 - x$ $t_2 = t_1 \ll 7 + t_1$	5 \pm
Our	$(t_2 \ll 12) + (t_2 \ll 5) + t_1$ where $t_1 = (x \ll 3) - x$ $t_2 = (t_1 \ll 2) - x$	4 \pm

CSD: canonical signed digit, $111463 = 11011001101100111_2 = 100\bar{1}0\bar{1}0100\bar{1}0\bar{1}0100\bar{1}_2$

Multiplication by Constants (2/2)

Power savings: 30 up to 60%

operator	init.	[1]	[2]	our
DCT 8b	300	94	73	56
DCT 12b	368	100	84	70
DCT 16b	521	129	114	89
DCT 24b	789	212	—	119

Power savings: 10%

operator	init.	[1]	[2]	our
8×8 Had.	56	24	—	24
(16, 11) R.-M.	61	43	31	31
(15, 7) BCH	72	48	47	44
(24, 12, 8) Golay	76	—	47	45

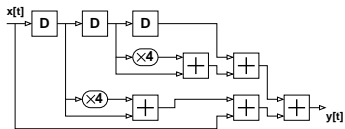
Power savings: up to 40%

operator	init.	[22]	our
8 bits	35	32	24
16 bits	72	70	46

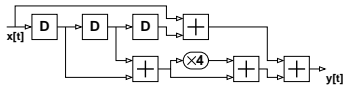
Parks-McClellan filter

`remez(25, [0 0.2 0.25 1], [1 1 0 0]).`

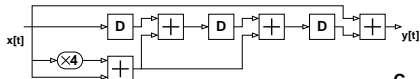
FIR (1, 5, 5, 1)



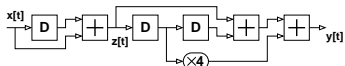
A



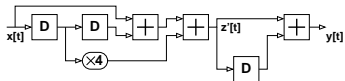
B



C



D



E

Error and Accuracy

Question: how many bits are correct ?

$$\begin{cases} x_t & = (1.000\ 000\ 00)_2 & \textit{theoretical value} \\ x_c & = (0.111\ 111\ 11)_2 & \textit{value in the circuit} \\ |x_t - x_c| & = (0.000\ 000\ 01)_2 = 2^{-8} \end{cases}$$

Error and Accuracy

Question: how many bits are correct ?

$$\begin{cases} x_t & = (1.000\ 000\ 00)_2 & \textit{theoretical value} \\ x_c & = (0.111\ 111\ 11)_2 & \textit{value in the circuit} \\ |x_t - x_c| & = (0.000\ 000\ 01)_2 = 2^{-8} \end{cases}$$

Error, ϵ : distance between 2 objects (e.g. $\epsilon = \|f(x) - p(x)\|$)

Accuracy, μ : (fractional) number of bits required to represent values with an error $\leq \epsilon$

$$\mu = -\log_2 |\epsilon|$$

Error and Accuracy

Question: how many bits are correct ?

$$\begin{cases} x_t & = (1.000\ 000\ 00)_2 & \textit{theoretical value} \\ x_c & = (0.111\ 111\ 11)_2 & \textit{value in the circuit} \\ |x_t - x_c| & = (0.000\ 000\ 01)_2 = 2^{-8} \end{cases}$$

Error, ϵ : distance between 2 objects (e.g. $\epsilon = \|f(x) - p(x)\|$)

Accuracy, μ : (fractional) number of bits required to represent values with an error $\leq \epsilon$

$$\mu = -\log_2 |\epsilon|$$

Notation: μ expressed in terms of correct or significant bits ([cb], [sb])

Error and Accuracy

Question: how many bits are correct ?

$$\begin{cases} x_t & = (1.000\ 000\ 00)_2 & \textit{theoretical value} \\ x_c & = (0.111\ 111\ 11)_2 & \textit{value in the circuit} \\ |x_t - x_c| & = (0.000\ 000\ 01)_2 = 2^{-8} \end{cases}$$

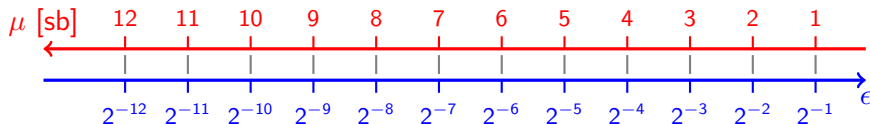
Error, ϵ : distance between 2 objects (e.g. $\epsilon = \|f(x) - p(x)\|$)

Accuracy, μ : (fractional) number of bits required to represent values with an error $\leq \epsilon$

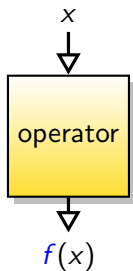
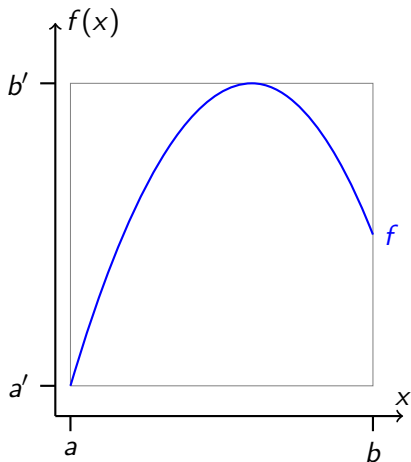
$$\mu = -\log_2 |\epsilon|$$

Notation: μ expressed in terms of correct or significant bits ([cb], [sb])

Example: error $\epsilon = 0.0000107$ is equivalent to accuracy $\mu = 16.5$ sb



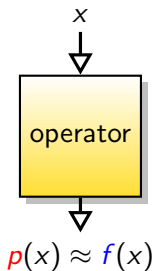
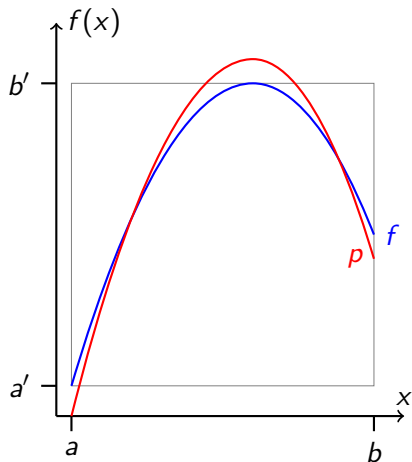
Polynomial Approximations



x argument
 $[a, b]$ domain

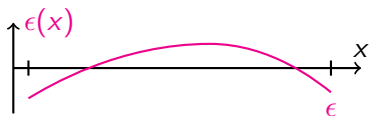
f function

Polynomial Approximations



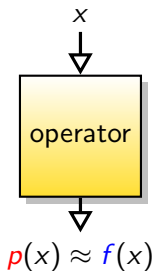
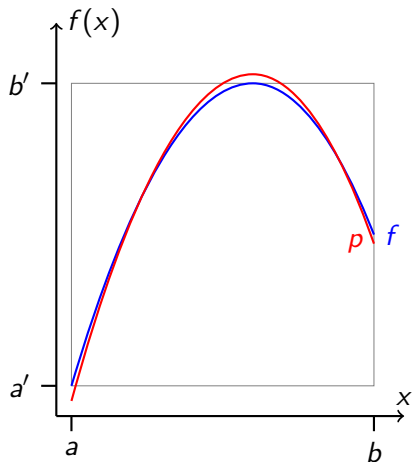
x argument
 $[a, b]$ domain

f function
 p polynomial



$$\epsilon(x) = f(x) - p(x) \quad \epsilon \text{ approx. error}$$

Polynomial Approximations



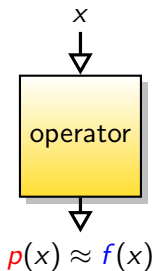
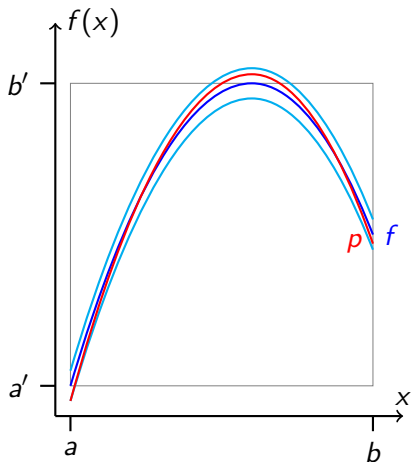
x argument
 $[a, b]$ domain

f function
 p polynomial



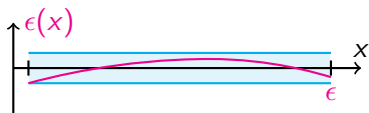
$$\epsilon(x) = f(x) - p(x) \quad \epsilon \text{ approx. error}$$

Polynomial Approximations



x argument
 $[a, b]$ domain

f function
 p polynomial



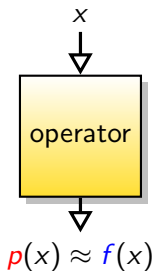
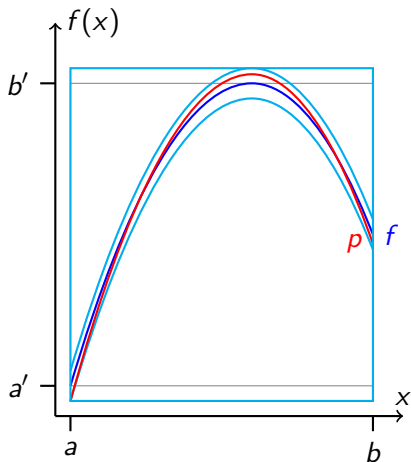
$$\epsilon(x) = f(x) - p(x)$$

$$\epsilon(x) \leq \epsilon_{\text{target}}$$

ϵ approx. error

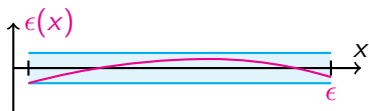
ϵ_{target} maximum
allowed error

Polynomial Approximations



x argument
 $[a, b]$ domain

f function
 p polynomial



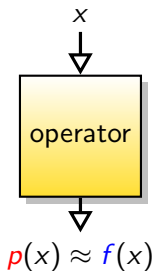
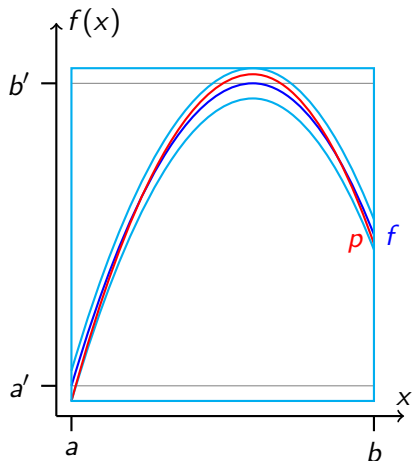
$$\epsilon(x) = f(x) - p(x)$$

$$\epsilon(x) \leq \epsilon_{\text{target}}$$

ϵ approx. error

ϵ_{target} maximum
allowed error

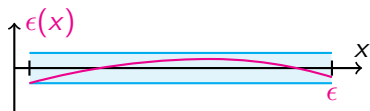
Polynomial Approximations



x argument
 $[a, b]$ domain

f function
 p polynomial

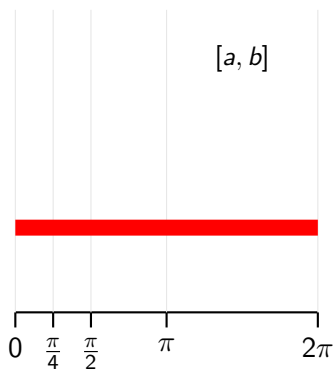
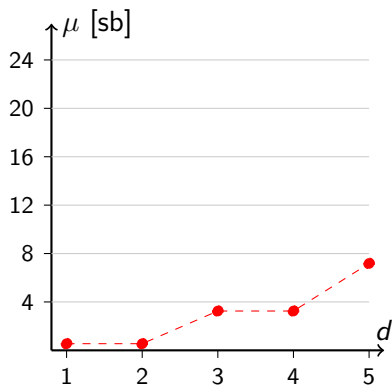
Question: what is the best p ?



$\epsilon(x) = f(x) - p(x)$ ϵ approx. error
 $\epsilon(x) \leq \epsilon_{\text{target}}$ ϵ_{target} maximum allowed error

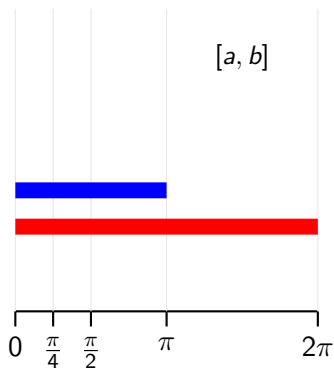
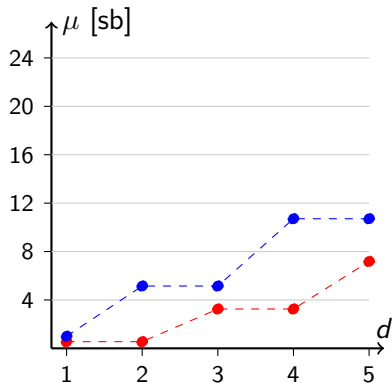
Accuracy, Degree and Evaluation Cost

Degree- d minimax approximation polynomials to $\sin(x)$ with $x \in [a, b]$:



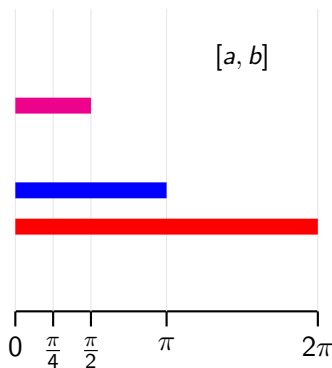
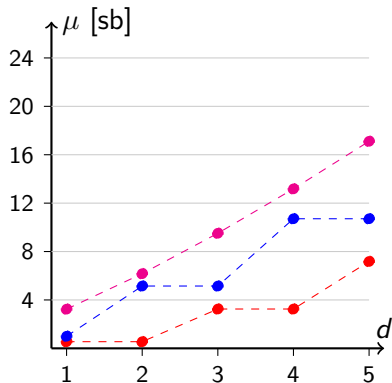
Accuracy, Degree and Evaluation Cost

Degree- d minimax approximation polynomials to $\sin(x)$ with $x \in [a, b]$:



Accuracy, Degree and Evaluation Cost

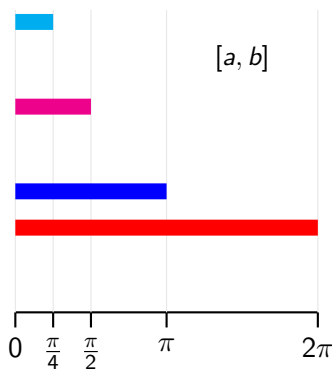
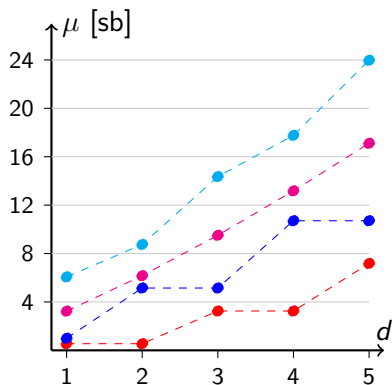
Degree- d minimax approximation polynomials to $\sin(x)$ with $x \in [a, b]$:



- higher accuracy \implies higher degree
- higher degree \implies more costly evaluation

Accuracy, Degree and Evaluation Cost

Degree- d minimax approximation polynomials to $\sin(x)$ with $x \in [a, b]$:



- higher accuracy \implies higher degree
- higher degree \implies more costly evaluation

Polynomial Evaluation Schemes

scheme	computations	# ±	# ×
direct	$p_0 + p_1x + p_2x^2 + p_3x^3$	3	5
Horner	$p_0 + (p_1 + (p_2 + p_3x)x)x$	3	3
Estrin	$p_0 + p_1x + (p_2 + p_3x)x^2$	3	4

Trade-off:

- direct scheme → high operation cost and smaller accuracy
- Horner scheme → smallest cost but sequential
- Estrin scheme → some internal parallelism

Polynomial Evaluation Schemes

scheme	computations	# ±	# ×
direct	$p_0 + p_1x + p_2x^2 + p_3x^3$	3	5
Horner	$p_0 + (p_1 + (p_2 + p_3x)x)x$	3	3
Estrin	$p_0 + p_1x + (p_2 + p_3x)x^2$	3	4

Trade-off:

- direct scheme → high operation cost and smaller accuracy
- Horner scheme → smallest cost but sequential
- Estrin scheme → some internal parallelism

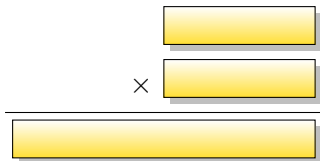
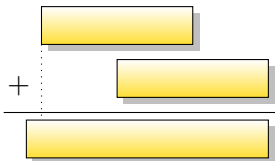
Question: what is the **best evaluation scheme**?

Round-off Errors

Round-off errors occur during most of computations:

- due to the **finite accuracy** during the computations
- small for a single operation (fraction of the LSB)
- **accumulation** of such errors may be a problem in long computation sequences
- **need** for a sufficient datapath width in order to limit round-off errors

Examples: $1/3 = 0.33333333 \dots \rightarrow 0.3333$ or 0.3334 in $1Q_{10}4$ format

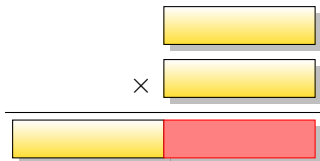
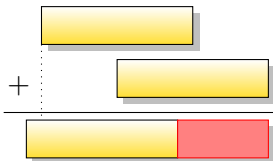


Round-off Errors

Round-off errors occur during most of computations:

- due to the **finite accuracy** during the computations
- small for a single operation (fraction of the LSB)
- **accumulation** of such errors may be a problem in long computation sequences
- **need** for a sufficient datapath width in order to limit round-off errors

Examples: $1/3 = 0.33333333\dots \rightarrow 0.3333$ or 0.3334 in $1Q_{10}4$ format

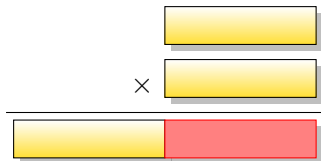
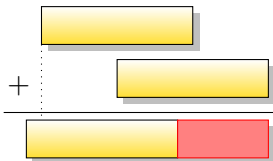


Round-off Errors

Round-off errors occur during most of computations:

- due to the **finite accuracy** during the computations
- small for a single operation (fraction of the LSB)
- **accumulation** of such errors may be a problem in long computation sequences
- **need** for a sufficient datapath width in order to limit round-off errors

Examples: $1/3 = 0.33333333\dots \rightarrow 0.3333$ or 0.3334 in $1Q_{10}4$ format

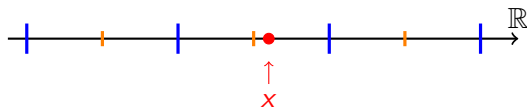


Question: what is the **best datapath width**?

Rounding Modes and Correct Rounding

Notations:

- \odot is an operation $\pm, \times, \div \dots$
- \diamond is the active **rounding mode** (or quantization mode)
IEEE-754: $\Delta(x)$ towards $+\infty$ (up), $\nabla(x)$ towards $-\infty$ (down), $\mathcal{Z}(x)$ towards 0, $\mathcal{N}(x)$ towards the nearest



representable values

midpoints

mathematical values

$$r_{math} = a \odot_{math} b$$

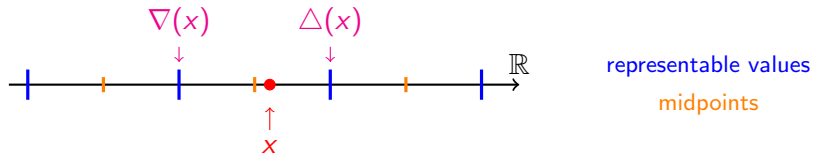
finite precision values

$$r_{finite} = a \odot_{finite} b$$

Rounding Modes and Correct Rounding

Notations:

- \odot is an operation $\pm, \times, \div \dots$
- \diamond is the active **rounding mode** (or quantization mode)
IEEE-754: $\Delta(x)$ towards $+\infty$ (up), $\nabla(x)$ towards $-\infty$ (down), $\mathcal{Z}(x)$ towards 0, $\mathcal{N}(x)$ towards the nearest



representable values

midpoints

mathematical values

$$r_{math} = a \odot_{math} b$$

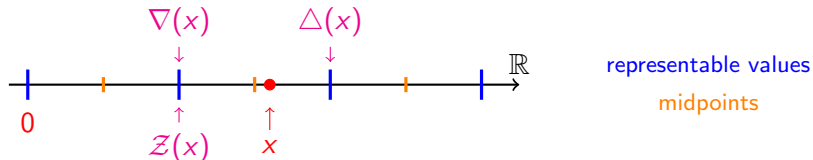
finite precision values

$$r_{finite} = a \odot_{finite} b$$

Rounding Modes and Correct Rounding

Notations:

- \odot is an operation $\pm, \times, \div \dots$
- \diamond is the active **rounding mode** (or quantization mode)
IEEE-754: $\Delta(x)$ towards $+\infty$ (up), $\nabla(x)$ towards $-\infty$ (down), $\mathcal{Z}(x)$ towards 0, $\mathcal{N}(x)$ towards the nearest



representable values

midpoints

mathematical values

$$r_{math} = a \odot_{math} b$$

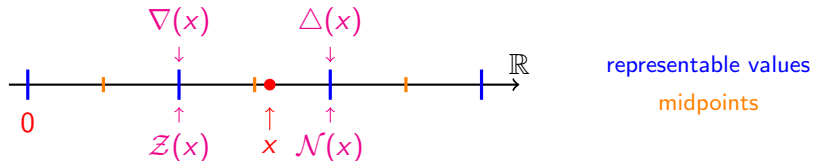
finite precision values

$$r_{finite} = a \odot_{finite} b$$

Rounding Modes and Correct Rounding

Notations:

- \odot is an operation $\pm, \times, \div \dots$
- \diamond is the active **rounding mode** (or quantization mode)
IEEE-754: $\Delta(x)$ towards $+\infty$ (up), $\nabla(x)$ towards $-\infty$ (down), $\mathcal{Z}(x)$ towards 0, $\mathcal{N}(x)$ towards the nearest



representable values

midpoints

mathematical values

$$r_{math} = a \odot_{math} b$$

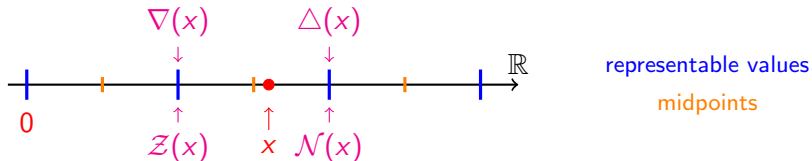
finite precision values

$$r_{finite} = a \odot_{finite} b$$

Rounding Modes and Correct Rounding

Notations:

- \odot is an operation $\pm, \times, \div \dots$
- \diamond is the active **rounding mode** (or quantization mode)
IEEE-754: $\Delta(x)$ towards $+\infty$ (up), $\nabla(x)$ towards $-\infty$ (down), $\mathcal{Z}(x)$ towards 0, $\mathcal{N}(x)$ towards the nearest



<i>mathematical values</i>
$r_{math} = a \odot_{math} b$

<i>finite precision values</i>
$r_{finite} = a \odot_{finite} b$
\downarrow
$r_{finite} = \diamond(a \odot_{math} b)$

Bounding Round-off Errors

Problem: it is very difficult to get **tight bounds**

Solutions:

- **worst case:** assume $1/2$ LSB error for each operation
 \rightsquigarrow simple but very pessimistic
- **qualification:** exhaustive or selected simulations
 \rightsquigarrow simple but only validated bounds for small systems
- **specific tools:** formal accurate analysis (and proof)
 \rightsquigarrow we use **gappa** developed by Guillaume Melquiond

Gappa Overview

- developed by Guillaume Melquiond
- goal: **formal verification of the correctness of numerical programs**:
 - ▶ software and **hardware**
 - ▶ integer, floating-point and **fixed-point** arithmetic (\pm , \times , \div , $\sqrt{}$)
- uses multiple-precision interval arithmetic, forward error analysis and expression rewriting to bound mathematical expressions (rounded and exact operators)
- generates a theorem and its **proof** which can be automatically checked using a **proof assistant** (e.g. Coq or HOL Light)
- reports **tight error bounds** for given expressions in a given domain
- C++ code and free software licence (CeCILL \simeq GPL)
- publication: ACM Transactions on Mathematical Software, n. 1, vol. 37, 2010, pp: 2:1–20, doi: 10.1145/1644001.1644003
- source code and doc: <http://gappa.gforge.inria.fr/>

Gamma Example

Degree-2 polynomial approximation to e^x over $[1/2, 1]$ and format 1Q9:

```
1 p0 = 571/512;    p1 = 275/512;    p2 = 545/512;
2
3 x = fixed<-9,dn>(Mx);
4
5 y1 fixed<-9,dn>= p2 * x + p1;
6 p  fixed<-9,dn>= y1 * x + p0;
7
8 Mp = (p2 * Mx + p1) * Mx + p0;
9
10 {
11     Mx in [0.5 , 1]    /\    |Mp-Mf| in [0 , 0.001385]
12 ->
13     |p-Mf| in ?
14 }
```

Gamma-0.14.0 result ($[a, b]$, $x\{(\approx x)_{10}, \log_2 x\}$, $xby = x2^y$):

Results for Mx in $[0.5, 1]$ and $|Mp - Mf|$ in $[0, 0.001385]$:

$|p - Mf|$ in $[0, 193518932894171697b-64 \{0.0104907, 2^{(-6.57475)}\}]$

Still Pending Questions

Question: what is the best (or a good) p ?

Question: what is the best (or a good) datapath width?

Question: what is the best (or a good) evaluation scheme?

Still Pending Questions

Question: what is the best (or a good) p ?

- mathematical p : *minimax approximations*
- implemented p : simple selection of representable coefficients
- links to other methods and tools

Question: what is the best (or a good) datapath width?

Question: what is the best (or a good) evaluation scheme?

Still Pending Questions

Question: what is the **best (or a good) p** ?

- mathematical p : *minimax approximations*
- implemented p : simple selection of representable coefficients
- links to other methods and tools

Question: what is the **best (or a good) datapath width**?

- basic optimization method
- better heuristics under development. . .

Question: what is the **best (or a good) evaluation scheme**?

Still Pending Questions

Question: what is the best (or a good) p ?

- mathematical p : *minimax approximations*
- implemented p : simple selection of representable coefficients
- links to other methods and tools

Question: what is the best (or a good) datapath width?

- basic optimization method
- better heuristics under development. . .

Question: what is the best (or a good) evaluation scheme?

- Horner or specific scheme examples. . .
- work still in progress. . .

Minimax Polynomial Approximations

- approximation error $\epsilon_{\text{app}} = \|f - p\|_{\infty} = \max_{a \leq x \leq b} |f(x) - p(x)|$
- **minimax** polynomial approximation to f over $[a, b]$ is p^* such that:

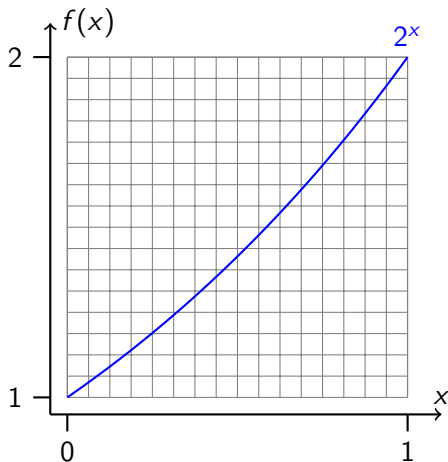
$$\|f - p^*\|_{\infty} = \min_{p \in \mathcal{P}_d} \|f - p\|_{\infty}$$

- \mathcal{P}_d set of polynomials with real coefficients and degree $\leq d$
- p^* computed using an algorithm from Remez (numerically implemented in Maple, Matlab, sollya. . .)

Problems:

- p^* coefficients in $\mathbb{R} \implies$ conversion to **finite precision**
- during p^* evaluation, some **round-off errors** add up to ϵ_{app}

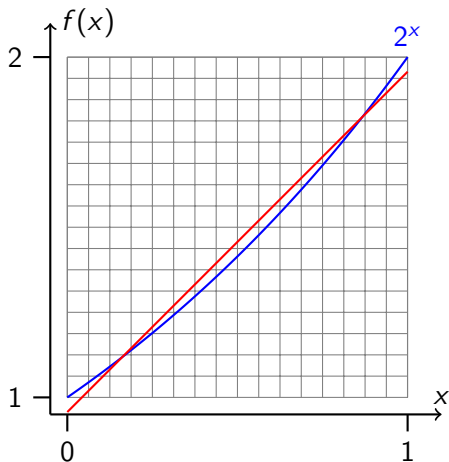
Example $f(x) = 2^x$ and $x \in [0, 1]$



p^* ?

d	μ [sb]	ϵ_{app}
1	4.53	4.31×10^{-2}
2	8.65	2.48×10^{-3}
3	13.18	1.08×10^{-4}
4	18.04	3.71×10^{-6}
5	23.15	1.07×10^{-7}

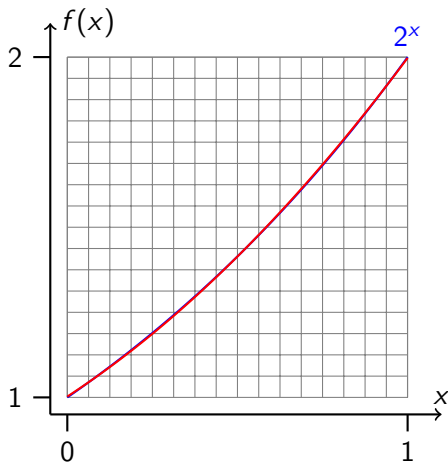
Example $f(x) = 2^x$ and $x \in [0, 1]$



d	μ [sb]	ϵ_{app}
1	4.53	4.31×10^{-2}
2	8.65	2.48×10^{-3}
3	13.18	1.08×10^{-4}
4	18.04	3.71×10^{-6}
5	23.15	1.07×10^{-7}

$$p^* = 0.956964333 + 1.000000000x$$

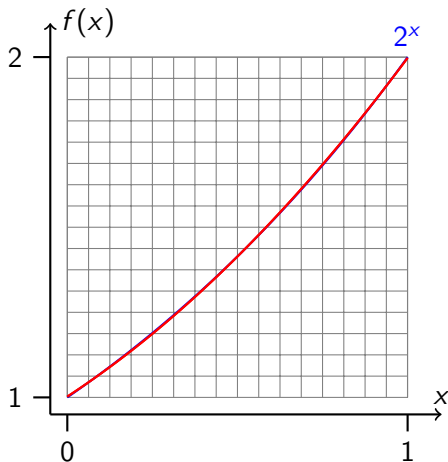
Example $f(x) = 2^x$ and $x \in [0, 1]$



d	μ [sb]	ϵ_{app}
1	4.53	4.31×10^{-2}
2	8.65	2.48×10^{-3}
3	13.18	1.08×10^{-4}
4	18.04	3.71×10^{-6}
5	23.15	1.07×10^{-7}

$$p^* = 1.002476056 + x \times (0.651046780 + x \times 0.344001106)$$

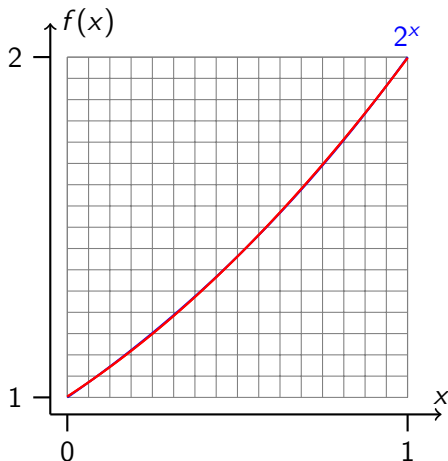
Example $f(x) = 2^x$ and $x \in [0, 1]$



d	μ [sb]	ϵ_{app}
1	4.53	4.31×10^{-2}
2	8.65	2.48×10^{-3}
3	13.18	1.08×10^{-4}
4	18.04	3.71×10^{-6}
5	23.15	1.07×10^{-7}

$$p^* = 0.999892965 + x \times (0.696457394 + x \times (0.224338364 + x \times 0.079204240))$$

Example $f(x) = 2^x$ and $x \in [0, 1]$



d	μ [sb]	ϵ_{app}
1	4.53	4.31×10^{-2}
2	8.65	2.48×10^{-3}
3	13.18	1.08×10^{-4}
4	18.04	3.71×10^{-6}
5	23.15	1.07×10^{-7}

$$p^* = 1.000003704 + x \times (0.692966122 + x \times (0.241638445 + x \times (0.051690358 + x \times 0.013697664)))$$

Finite Precision Coefficients Selection Problem

Example: $f(x) = e^x$ over $[1/2, 1]$ with $d = 2$, the `remez` function from `sollya` gives:

$$p^* = 1.116019297\dots + 0.535470348\dots \times x + 1.065407185\dots \times x^2$$

Finite Precision Coefficients Selection Problem

Example: $f(x) = e^x$ over $[1/2, 1]$ with $d = 2$, the `remez` function from `sollya` gives:

$$p^* = 1.116019297\dots + 0.535470348\dots \times x + 1.065407185\dots \times x^2$$

Question: what are “good” representable values for p_0 , p_1 and p_2 ?

Problem: p^* is the best **theoretical** approximation to f (i.e. $p_i \in \mathbb{R}$)

Need: find good approximations with “machine-representable” coefficients

Finite Precision Coefficients Selection Problem

Example: $f(x) = e^x$ over $[1/2, 1]$ with $d = 2$, the remez function from sollya gives:

$$p^* = 1.116019297\dots + 0.535470348\dots \times x + 1.065407185\dots \times x^2$$

Question: what are “good” representable values for p_0 , p_1 and p_2 ?

Problem: p^* is the best **theoretical** approximation to f (i.e. $p_i \in \mathbb{R}$)

Need: find good approximations with “machine-representable” coefficients

Above example with 1Q9 format (all values for domain $[1/2, 1]$):

- $\epsilon_{\text{app}} = \|f - p^*\|_{\infty} \simeq 1.385 \times 10^{-3} \rightsquigarrow \simeq 9.4 \text{ sb}$
- $\frac{571}{512} + \frac{137}{256}x + \frac{545}{512}x^2 \rightsquigarrow 8.1 \text{ sb} \quad (\forall i \text{ use } \mathcal{N}(p_i))$

Finite Precision Coefficients Selection Problem

Example: $f(x) = e^x$ over $[1/2, 1]$ with $d = 2$, the `remez` function from `sollya` gives:

$$p^* = 1.116019297\dots + 0.535470348\dots \times x + 1.065407185\dots \times x^2$$

Question: what are “good” representable values for p_0 , p_1 and p_2 ?

Problem: p^* is the best **theoretical** approximation to f (i.e. $p_i \in \mathbb{R}$)

Need: find good approximations with “machine-representable” coefficients

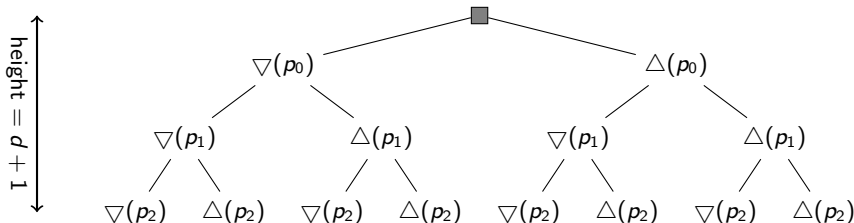
Above example with 1Q9 format (all values for domain $[1/2, 1]$):

- $\epsilon_{\text{app}} = \|f - p^*\|_{\infty} \simeq 1.385 \times 10^{-3} \rightsquigarrow \simeq 9.4 \text{ sb}$
- $\frac{571}{512} + \frac{137}{256}x + \frac{545}{512}x^2 \rightsquigarrow 8.1 \text{ sb} \quad (\forall i \text{ use } \mathcal{N}(p_i))$
- $\frac{571}{512} + \frac{275}{512}x + \frac{545}{512}x^2 \rightsquigarrow 9.3 \text{ sb} \quad (\text{best selection})$

Basic Coefficient Selection Method

Idea: search among all the rounding modes for all the p_i^*

- round up $p_i = \Delta(p_i^*)$, round down $p_i = \nabla(p_i^*)$
- 2 values per coeff. \implies total of 2^{d+1} values (but d is small)
- for each polynomial p evaluate $\epsilon_{\text{app}} = \|f - p\|_\infty$, then select polynomial(s) with the smallest ϵ_{app}

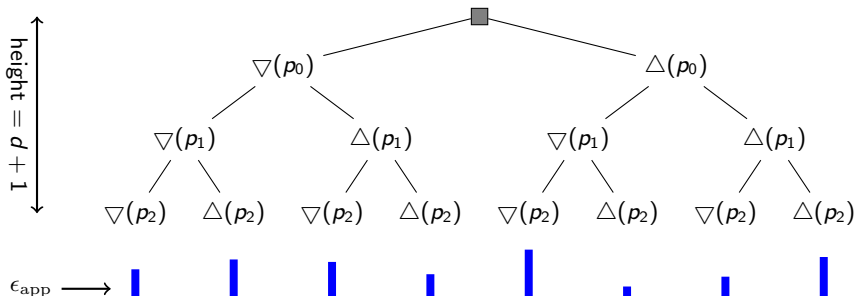


Result: $p(x) = \sum_{i=0}^d p_i x^i$ where all p_i are representable in target format

Basic Coefficient Selection Method

Idea: search among all the rounding modes for all the p_i^*

- round up $p_i = \Delta(p_i^*)$, round down $p_i = \nabla(p_i^*)$
- 2 values per coeff. \implies total of 2^{d+1} values (but d is small)
- for each polynomial p evaluate $\epsilon_{\text{app}} = \|f - p\|_\infty$, then select polynomial(s) with the smallest ϵ_{app}

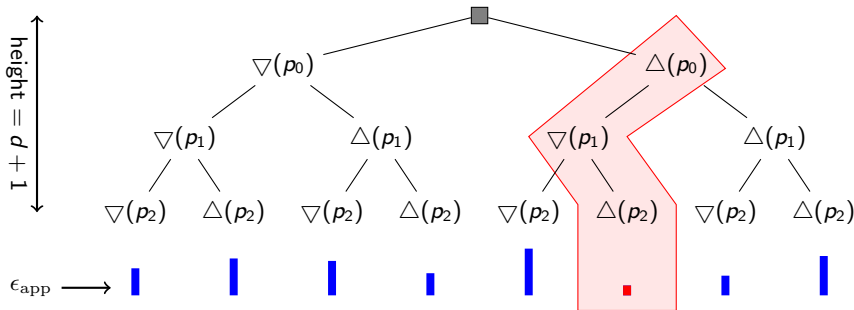


Result: $p(x) = \sum_{i=0}^d p_i x^i$ where all p_i are representable in target format

Basic Coefficient Selection Method

Idea: search among all the rounding modes for all the p_i^*

- round up $p_i = \Delta(p_i^*)$, round down $p_i = \nabla(p_i^*)$
- 2 values per coeff. \implies total of 2^{d+1} values (but d is small)
- for each polynomial p evaluate $\epsilon_{\text{app}} = \|f - p\|_\infty$, then select polynomial(s) with the smallest ϵ_{app}



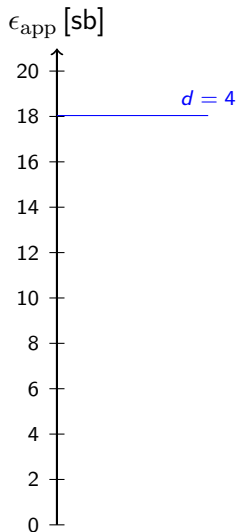
Result: $p(x) = \sum_{i=0}^d p_i x^i$ where all p_i are representable in target format

Example for $f(x) = 2^x$, $x \in [0, 1]$ and $d = 4$

$\epsilon_{\text{app}}(p^*) \rightsquigarrow 18.04$ sb

p	$\epsilon_{\text{app}}(p)$	p	$\epsilon_{\text{app}}(p)$
($\nabla, \nabla, \nabla, \nabla, \nabla$)	12.00	($\nabla, \nabla, \nabla, \nabla, \Delta$)	13.00
($\nabla, \nabla, \nabla, \Delta, \nabla$)	13.00	($\nabla, \nabla, \nabla, \Delta, \Delta$)	14.03
($\nabla, \nabla, \Delta, \nabla, \nabla$)	13.00	($\nabla, \nabla, \Delta, \nabla, \Delta$)	14.55
($\nabla, \nabla, \Delta, \Delta, \nabla$)	14.99	($\nabla, \nabla, \Delta, \Delta, \Delta$)	13.00
($\nabla, \Delta, \nabla, \nabla, \nabla$)	13.00	($\nabla, \Delta, \nabla, \nabla, \Delta$)	16.13
($\nabla, \Delta, \nabla, \Delta, \nabla$)	17.12	($\nabla, \Delta, \nabla, \Delta, \Delta$)	13.00
($\nabla, \Delta, \Delta, \nabla, \nabla$)	15.71	($\nabla, \Delta, \Delta, \nabla, \Delta$)	13.00
($\nabla, \Delta, \Delta, \Delta, \nabla$)	13.00	($\nabla, \Delta, \Delta, \Delta, \Delta$)	12.00
($\Delta, \nabla, \nabla, \nabla, \nabla$)	13.00	($\Delta, \nabla, \nabla, \nabla, \Delta$)	13.00
($\Delta, \nabla, \nabla, \Delta, \nabla$)	13.00	($\Delta, \nabla, \nabla, \Delta, \Delta$)	13.00
($\Delta, \nabla, \Delta, \nabla, \nabla$)	13.00	($\Delta, \nabla, \Delta, \nabla, \Delta$)	13.00
($\Delta, \nabla, \Delta, \Delta, \nabla$)	12.99	($\Delta, \nabla, \Delta, \Delta, \Delta$)	12.00
($\Delta, \Delta, \nabla, \nabla, \nabla$)	12.99	($\Delta, \Delta, \nabla, \nabla, \Delta$)	12.98
($\Delta, \Delta, \nabla, \Delta, \nabla$)	12.91	($\Delta, \Delta, \nabla, \Delta, \Delta$)	12.00
($\Delta, \Delta, \Delta, \nabla, \nabla$)	12.79	($\Delta, \Delta, \Delta, \nabla, \Delta$)	12.00
($\Delta, \Delta, \Delta, \Delta, \nabla$)	12.00	($\Delta, \Delta, \Delta, \Delta, \Delta$)	11.41

p represented by $(p_0, p_1, p_2, p_3, p_4)$

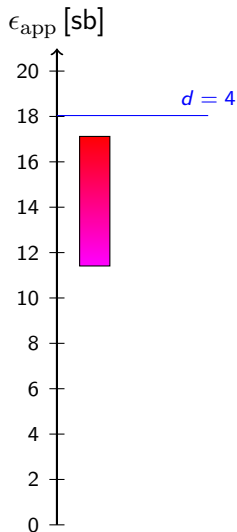


Example for $f(x) = 2^x$, $x \in [0, 1]$ and $d = 4$

$\epsilon_{\text{app}}(p^*) \rightsquigarrow 18.04$ sb

p	$\epsilon_{\text{app}}(p)$	p	$\epsilon_{\text{app}}(p)$
($\nabla, \nabla, \nabla, \nabla, \nabla$)	12.00	($\nabla, \nabla, \nabla, \nabla, \Delta$)	13.00
($\nabla, \nabla, \nabla, \Delta, \nabla$)	13.00	($\nabla, \nabla, \nabla, \Delta, \Delta$)	14.03
($\nabla, \nabla, \Delta, \nabla, \nabla$)	13.00	($\nabla, \nabla, \Delta, \nabla, \Delta$)	14.55
($\nabla, \nabla, \Delta, \Delta, \nabla$)	14.99	($\nabla, \nabla, \Delta, \Delta, \Delta$)	13.00
($\nabla, \Delta, \nabla, \nabla, \nabla$)	13.00	($\nabla, \Delta, \nabla, \nabla, \Delta$)	16.13
($\nabla, \Delta, \nabla, \Delta, \nabla$)	17.12	($\nabla, \Delta, \nabla, \Delta, \Delta$)	13.00
($\nabla, \Delta, \Delta, \nabla, \nabla$)	15.71	($\nabla, \Delta, \Delta, \nabla, \Delta$)	13.00
($\nabla, \Delta, \Delta, \Delta, \nabla$)	13.00	($\nabla, \Delta, \Delta, \Delta, \Delta$)	12.00
($\Delta, \nabla, \nabla, \nabla, \nabla$)	13.00	($\Delta, \nabla, \nabla, \nabla, \Delta$)	13.00
($\Delta, \nabla, \nabla, \Delta, \nabla$)	13.00	($\Delta, \nabla, \nabla, \Delta, \Delta$)	13.00
($\Delta, \nabla, \Delta, \nabla, \nabla$)	13.00	($\Delta, \nabla, \Delta, \nabla, \Delta$)	13.00
($\Delta, \nabla, \Delta, \Delta, \nabla$)	12.99	($\Delta, \nabla, \Delta, \Delta, \Delta$)	12.00
($\Delta, \Delta, \nabla, \nabla, \nabla$)	12.99	($\Delta, \Delta, \nabla, \nabla, \Delta$)	12.98
($\Delta, \Delta, \nabla, \Delta, \nabla$)	12.91	($\Delta, \Delta, \nabla, \Delta, \Delta$)	12.00
($\Delta, \Delta, \Delta, \nabla, \nabla$)	12.79	($\Delta, \Delta, \Delta, \nabla, \Delta$)	12.00
($\Delta, \Delta, \Delta, \Delta, \nabla$)	12.00	($\Delta, \Delta, \Delta, \Delta, \Delta$)	11.41

p represented by $(p_0, p_1, p_2, p_3, p_4)$

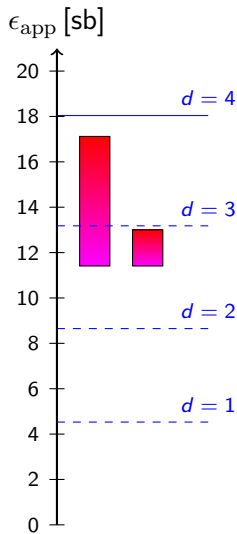


Example for $f(x) = 2^x$, $x \in [0, 1]$ and $d = 4$

$\epsilon_{\text{app}}(p^*) \rightsquigarrow 18.04 \text{ sb}$

p	$\epsilon_{\text{app}}(p)$	p	$\epsilon_{\text{app}}(p)$
($\nabla, \nabla, \nabla, \nabla, \nabla$)	12.00	($\nabla, \nabla, \nabla, \nabla, \Delta$)	13.00
($\nabla, \nabla, \nabla, \Delta, \nabla$)	13.00	($\nabla, \nabla, \nabla, \Delta, \Delta$)	14.03
($\nabla, \nabla, \Delta, \nabla, \nabla$)	13.00	($\nabla, \nabla, \Delta, \nabla, \Delta$)	14.55
($\nabla, \nabla, \Delta, \Delta, \nabla$)	14.99	($\nabla, \nabla, \Delta, \Delta, \Delta$)	13.00
($\nabla, \Delta, \nabla, \nabla, \nabla$)	13.00	($\nabla, \Delta, \nabla, \nabla, \Delta$)	16.13
($\nabla, \Delta, \nabla, \Delta, \nabla$)	17.12	($\nabla, \Delta, \nabla, \Delta, \Delta$)	13.00
($\nabla, \Delta, \Delta, \nabla, \nabla$)	15.71	($\nabla, \Delta, \Delta, \nabla, \Delta$)	13.00
($\nabla, \Delta, \Delta, \Delta, \nabla$)	13.00	($\nabla, \Delta, \Delta, \Delta, \Delta$)	12.00
($\Delta, \nabla, \nabla, \nabla, \nabla$)	13.00	($\Delta, \nabla, \nabla, \nabla, \Delta$)	13.00
($\Delta, \nabla, \nabla, \Delta, \nabla$)	13.00	($\Delta, \nabla, \nabla, \Delta, \Delta$)	13.00
($\Delta, \nabla, \Delta, \nabla, \nabla$)	13.00	($\Delta, \nabla, \Delta, \nabla, \Delta$)	13.00
($\Delta, \nabla, \Delta, \Delta, \nabla$)	12.99	($\Delta, \nabla, \Delta, \Delta, \Delta$)	12.00
($\Delta, \Delta, \nabla, \nabla, \nabla$)	12.99	($\Delta, \Delta, \nabla, \nabla, \Delta$)	12.98
($\Delta, \Delta, \nabla, \Delta, \nabla$)	12.91	($\Delta, \Delta, \nabla, \Delta, \Delta$)	12.00
($\Delta, \Delta, \Delta, \nabla, \nabla$)	12.79	($\Delta, \Delta, \Delta, \nabla, \Delta$)	12.00
($\Delta, \Delta, \Delta, \Delta, \nabla$)	12.00	($\Delta, \Delta, \Delta, \Delta, \Delta$)	11.41

p represented by $(p_0, p_1, p_2, p_3, p_4)$



Example: 2^x over $[0, 1]$ and $\mu \leq 12$ sb (1/2)

Let us try with $d = 3$ (max. theoretical accuracy 13.18 sb):

$$p^*(x) = 0.999892965 + 0.696457394x + 0.224338364x^2 + 0.079204240x^3$$

Coefficients (fractional part) size selection:

l	12	13	14	15	16
ϵ_{app}	12.38	12.45	13.00	13.00	13.02
# polynomials	0	0	2	2	7

Coefficients selection: for $n = k + l = 1 + 14$ bits, we get:

$(\nabla, \nabla, \nabla, \nabla)$	11.41	$(\nabla, \nabla, \nabla, \Delta)$	12.00
$(\nabla, \nabla, \Delta, \nabla)$	12.00	$(\nabla, \nabla, \Delta, \Delta)$	12.84
$(\nabla, \Delta, \nabla, \nabla)$	12.00	$(\nabla, \Delta, \nabla, \Delta)$	13.00
$(\nabla, \Delta, \Delta, \nabla)$	13.00	$(\nabla, \Delta, \Delta, \Delta)$	12.36
$(\Delta, \nabla, \nabla, \nabla)$	12.00	$(\Delta, \nabla, \nabla, \Delta)$	12.25
$(\Delta, \nabla, \Delta, \nabla)$	12.23	$(\Delta, \nabla, \Delta, \Delta)$	12.23
$(\Delta, \Delta, \nabla, \nabla)$	12.13	$(\Delta, \Delta, \nabla, \Delta)$	12.12
$(\Delta, \Delta, \Delta, \nabla)$	12.05	$(\Delta, \Delta, \Delta, \Delta)$	11.64

Example: 2^x over $[0, 1]$ and $\mu \leq 12$ sb (2/2)

Datapath size selection:

n'	14	15	16	17	18	19	20
ϵ_{eval} direct	11.24	11.86	12.32	12.62	12.79	12.89	12.94
ϵ_{eval} Horner	11.32	11.93	12.36	12.65	12.81	12.90	12.95

Solution: $d = 3$, $n = k + l = 1 + 14$ and $n' = 16$

Implementation results:

solution	area	period	#cycles	latency	power
wo. tools	1.00	1.00	4	1.00	1.00
w. tools	0.83	0.82	3	0.61	0.68

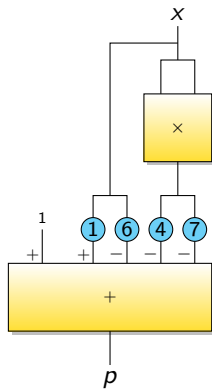
Example: \sqrt{x} over $[1, 2]$ and $\mu \leq 8$ sb

Selection of coefficients leading to sparse recodings

$$p^* = 1.00076383 + 0.48388463x - 0.071198745x^2$$

$$p = 1 + (0.10000\bar{1})_2 x - (0.0001001)_2 x^2$$

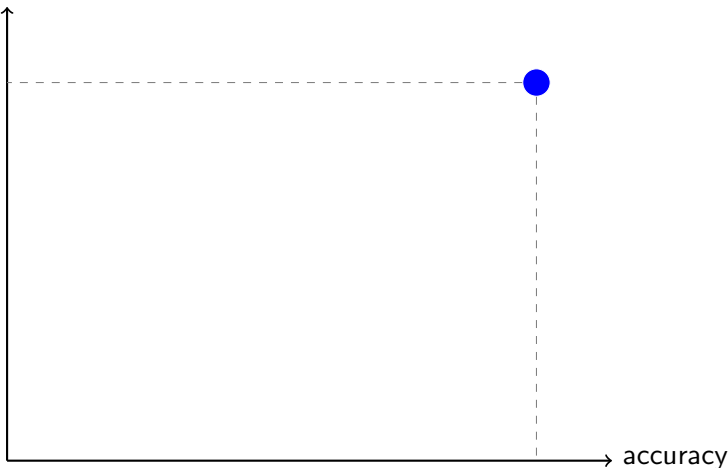
replace \times by a **small** number of \pm



solution	area	period	#cycles	latency	power
wo. tools	1.00	1.00	2	1.00	1.00
w. tools	0.59	0.97	1	0.48	0.45

Summary

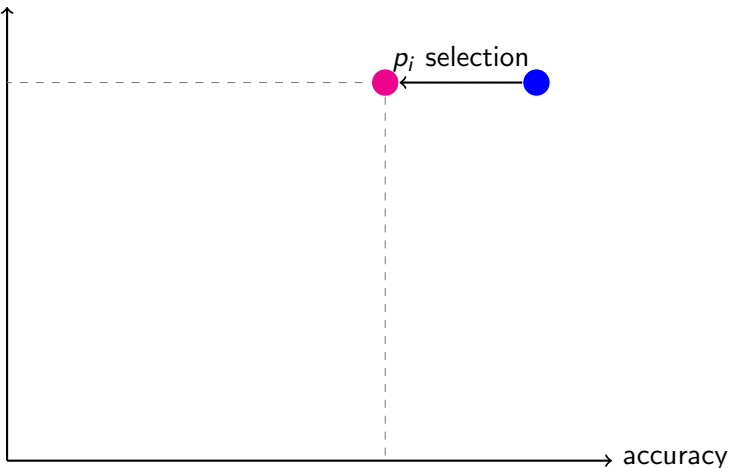
cost \approx delay \times area



Important: non-optimal solutions BUT very good ones in practice

Summary

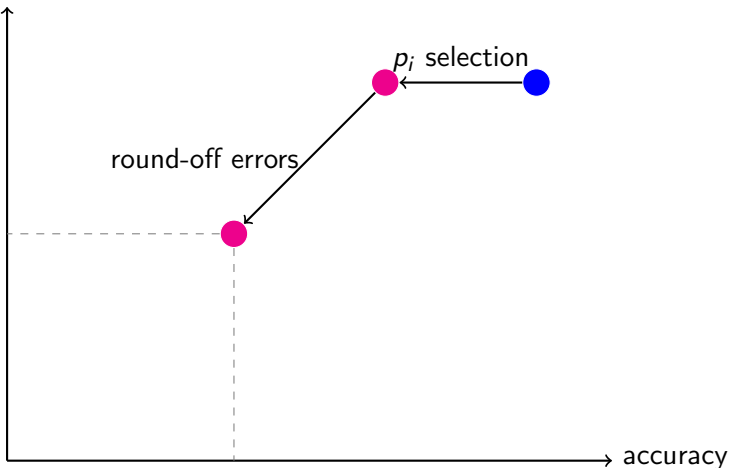
cost \approx delay \times area



Important: non-optimal solutions BUT very good ones in practice

Summary

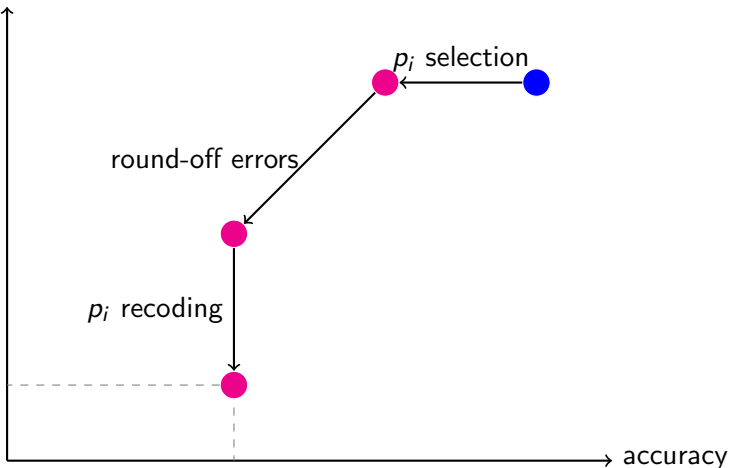
cost \approx delay \times area



Important: non-optimal solutions BUT very good ones in practice

Summary

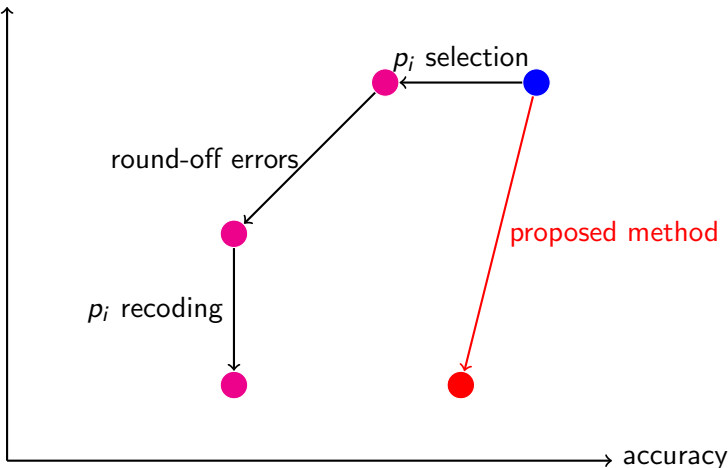
cost \approx delay \times area



Important: non-optimal solutions BUT very good ones in practice

Summary

cost \approx delay \times area



Important: non-optimal solutions BUT very good ones in practice

Conclusion

When designing circuits with arithmetic operators:

- use adequate **number system(s)**
- use adequate **algorithm(s)**
- use **specific operator(s)** when possible
- use **optimization** (open-source) **tool(s)**:
 - ▶ floating-point data-paths: **FloPoCo** flopoco.gforge.inria.fr
 - ▶ divider generator: **divgen**
<http://lipforge.ens-lyon.fr/www/divgen/>
 - ▶ polynomial approx.: **sollya** <http://sollya.gforge.inria.fr/>
 - ▶ rounding errors: **gappa** <http://gappa.gforge.inria.fr/>

The end, questions ?

Contact:

- <mailto:arnaud.tisserand@univ-ubs.fr>
- <http://www-labsticc.univ-ubs.fr/~tisseran>
- CNRS
Lab-STICC, Centre Recherche UBS
Rue St Maudé. BP 92116. 56321 Lorient cedex, France

Thank you