Low-Power Arithmetic Operators

Arnaud Tisserand

CNRS, IRISA laboratory, CAIRN research team

ECOFAC 2014
Lorient, ENSIBS / UBS
May 19th – 23th, 2014

Summary

Part I

Introduction

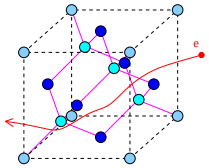
Electromigration

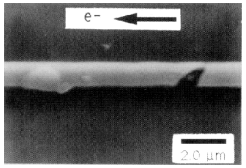
- high current density → movement of atoms in a conductor
- mean time to failure (MTTF) of a wire, Black's equation:

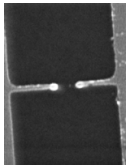
$$\text{MTTF} = A \times J^{-n} \times e^{\frac{E_a}{kT}}$$

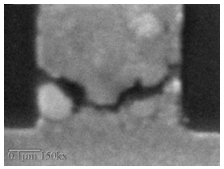
A section, J current density, $n \approx 2$ scale factor (cst), E_a activation energy (cst for a material), k Boltzmann's constant, T temperature

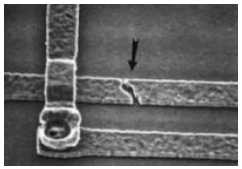
- decreases IC reliability (permanent and intermittent failures)

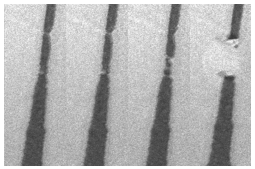




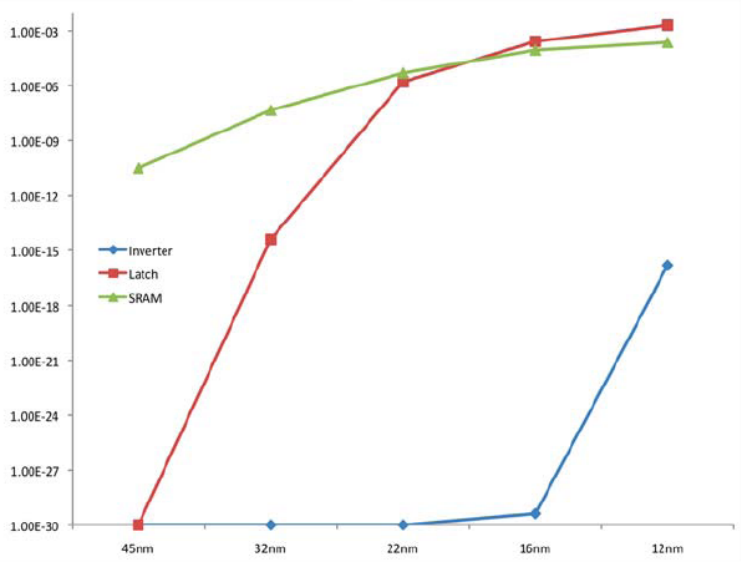








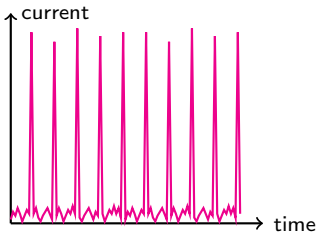
ITRS 2009: Failure Rate Evolution

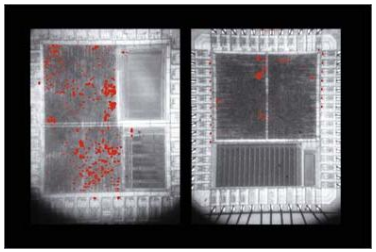


Source: `http://public.itrs.net/Links/2009ITRS/Home2009.htm`

Electromagnetic Interferences (EMI)

Electromagnetic emissions from a device or system (the culprit or attacker) that interfere with the normal operation of another device or system (the victim)





thermography 80C51 MCU by Philips
synchronous (left), asynchronous (right)

Electromagnetic compatibility (EMC):

- ability to avoid introducing intolerable electromagnetic disturbance
- circuit specific design rules

Cooling

Problems due to temperature:

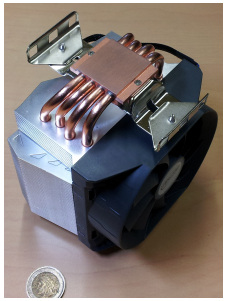
- performance decreases with temperature

  25 °C $\rightarrow$ 105 °C ➔ 30 % performance reduction
- reliability decreases with temperature

  IC temperature $>$ 125 °C ➔ faults and characteristics damage

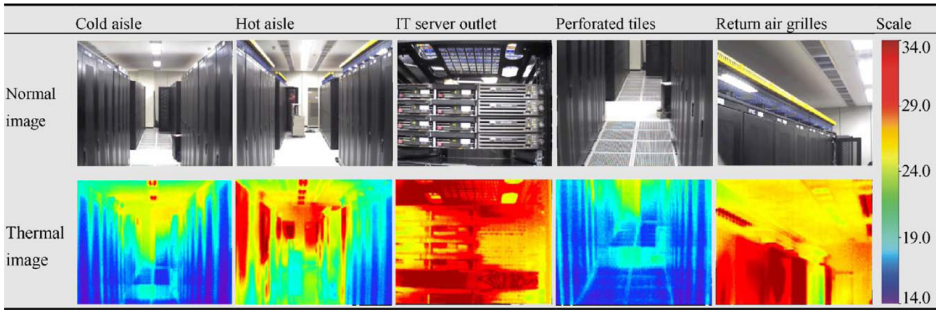
Solutions:

- reduce power consumption
- cool circuits (air, water,...)



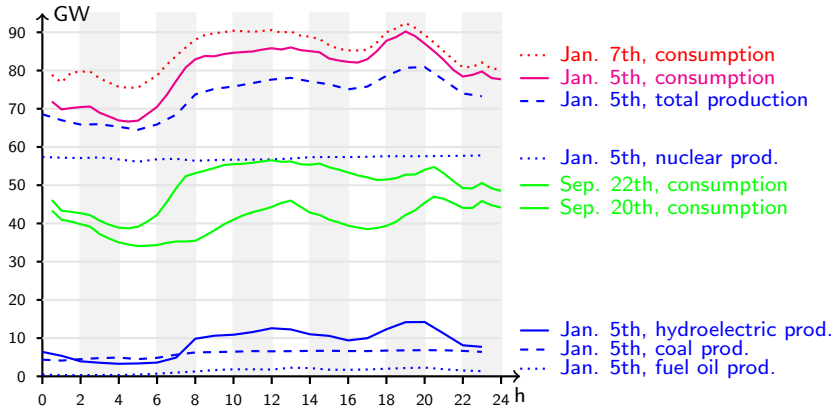
Cooling in Data Centers

- cooling is a significant challenge
- $<$ 50 % power for electronic equipments (30–40 % in some cases)
- problem: keeping the hardware cool and humidified



Source: J. Cho, T. Lim, B. S. Kim. *Measurements and predictions of the air distribution systems in high compute density (Internet) data centers*. Energy and Buildings, vol. 41, pp. 1107-1115, 2009

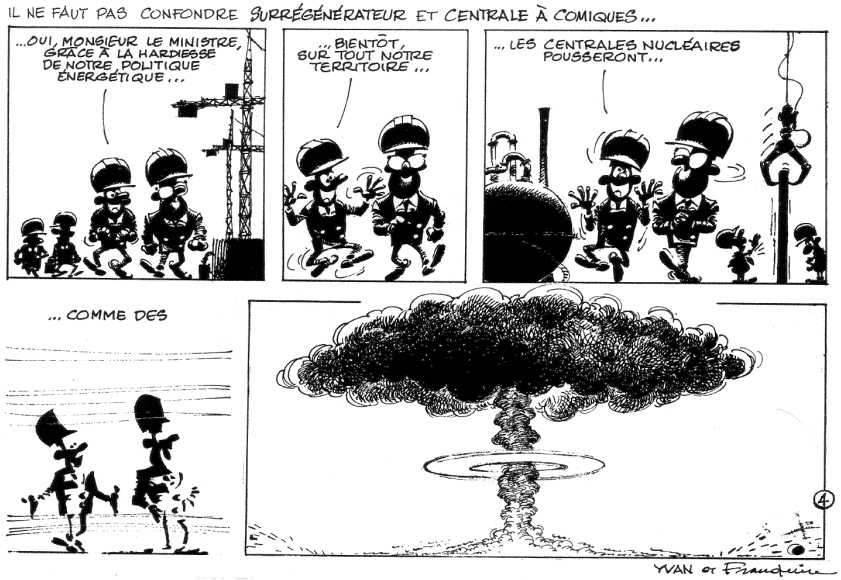
Power Consumption and Production in France (GW)



date	nuclear	coal	fuel oil	hydroelectric	total
Tue. 2009-01-05 20h	57.5	6.8	2.2	14.2	80.9
Sun. 2009-09-20 20h	39.1	2.0	0.0	3.9	45.1
Tue. 2009-09-22 20h	42.5	4.8	0.0	4.3	51.7

Source: *RTE (Réseau de transport d'électricité)* `http://www.rte-france.com/`

Another Good Reason for Reducing Power Consumption



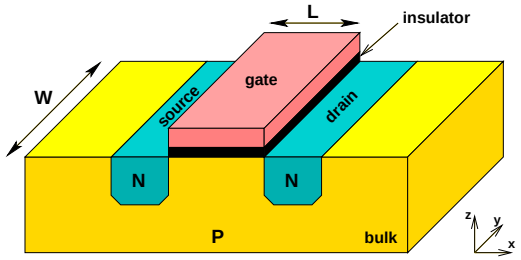
Idées Noires. Franquin. p. 10, Fluide Glacial, ISBN 2-85815-042-7

MOS Transistor: N and P transistors

MOS = *metal oxide semiconductor*

N transistors are made of:

- bulk (Si), P-type doping
- drain and source, N-type doping
- insulator
- gate or grid



In N-type doping area, the majority carriers are electrons (holes in a P-type area)

P transistor: bulk is N while source and drain are P areas

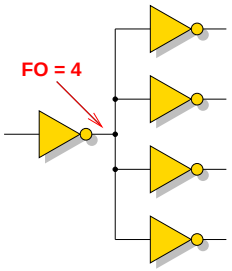
Fast Circuit Design: Basic Ideas

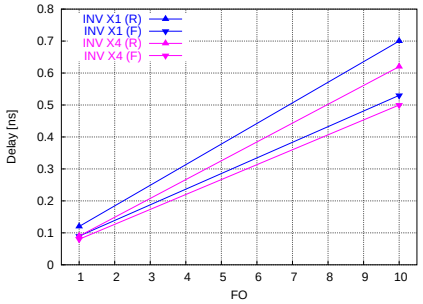
- $V_{DD} \nearrow \implies$ speed $\nearrow$ but limited by the technology
- Transistor size :
  - $W \nearrow \implies$ speed $\nearrow$ GOOD
  - $L \nearrow \implies$ speed $\searrow$ BAD
  - but $W \nearrow \implies C \nearrow \implies$ speed $\searrow$

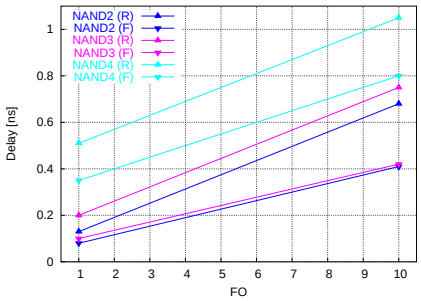
  Transistor Sizing
- Topology
- Logic optimizations
- Place and route optimizations
- Algorithms, data coding...

Fanout

The gate delay (change output state) depends on the output load. Fanout measures this load as the number of inputs of gate connected to the output (normalized w.r.t. an inverter)







Power Consumption: Basic Definitions

Instantaneous power:

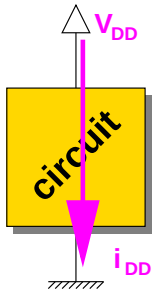
$$P(t) = i_{DD}(t)\, V_{\mathrm{DD}}$$

Energy over some time interval T:

$$E = \int_0^T i_{DD}(t)\, V_{\mathrm{DD}}\, dt$$

Average power over interval T:

$$P_{avg} = \frac{E}{T} = \frac{1}{T} \int_0^T i_{DD}(t)\, V_{\mathrm{DD}}\, dt$$



Units:

- current A
- voltage V
- power W
- energy J or Wh

Power Consumption: Components

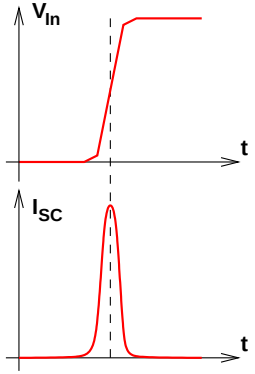
Power dissipation in CMOS circuits comes from 2 main components:

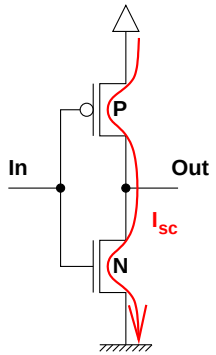
- static dissipation:
  - sub-threshold conduction through OFF transistors
  - leakage current through P-N junctions
  - tunneling current through gate oxide
  - ...
- dynamic dissipation:
  - charging and discharging of load capacitances (useful + parasitic)
  - short-circuit current

$$P_{\mathrm{total}} = P_{\mathrm{static}} + P_{\mathrm{dynamic}}$$

Short-Circuit Current in CMOS Gates

Occurs when both N and P transistors are ON while the input switches

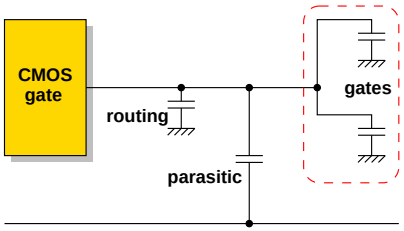




Solution : short transition (crisp edges)

Charging and Discharging Load Capacitances

There are capacitances everywhere in the circuit: transistor gate, routing, parasitics...



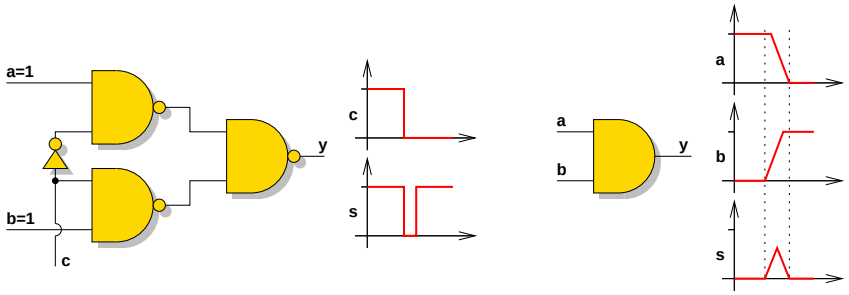
Solutions:

- design small circuits (small transistor, short wires, technology shrinking)
- reduce the activity (algorithms, data coding, sleep mode)
- reduce V_{DD} (without lowering speed)

Transitions

There are 2 kinds of transitions:

- useful transitions (data switching)
- redundant or parasitic transitions (imperfections)



Simple Power Consumption Model

Average dynamic power dissipation (no leakage, no short circuit):

$$P = \alpha \times C \times f \times V_{DD}^2$$

where

- α is the activity factor
- C is the average switched capacitance (at each cycle)
- f is the circuit frequency
- V_{DD} is the supply voltage

Remark: the gate delay is $d = \gamma \times \frac{C \times V_{DD}}{(V_{DD} - V_T)^2} \approx \frac{1}{V_{DD}}$

Part II

Number systems

Basic Number Systems

Exotic Number Systems

Positional Number System(s)

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

- 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: $\beta = 2, \mathcal{D}_{\text{bs}} = \{-1,0,1\}$
- signed digits: $\beta > 2, \mathcal{D}_{\text{sd},\alpha,\beta} = \{-\alpha,\ldots,\alpha\}$ with $2\alpha + 1 \geq \beta$
- theoretical systems: $\beta = \frac{1+\sqrt{5}}{2}$, $\beta = 1 + i \ldots$

Radix-2 Signed Integers

- sign and magnitude (absolute value)

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

- 2's complement

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

- biased (usually $B = 2^{n-1} - 1$)

$$A = A_{math} + B$$

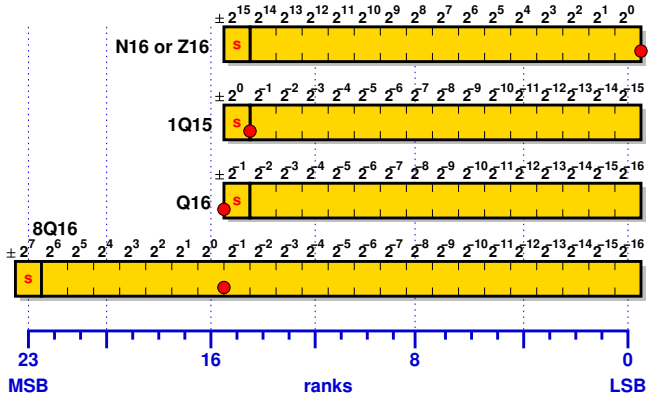
- ...

Signed Integers

integer	representations		
	sign/magnitude	2's complement	biased (B=7)
–8	—	1000	—
–7	1111	1001	0000
–6	1110	1010	0001
–5	1101	1011	0010
–4	1100	1100	0011
–3	1011	1101	0100
–2	1010	1110	0101
–1	1001	1111	0110
0	0000	0000	0111
1	0001	0001	1000
2	0010	0010	1001
3	0011	0011	1010
4	0100	0100	1011
5	0101	0101	1100
6	0110	0110	1101
7	0111	0111	1110
8	—	—	1111

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

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:

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

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

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

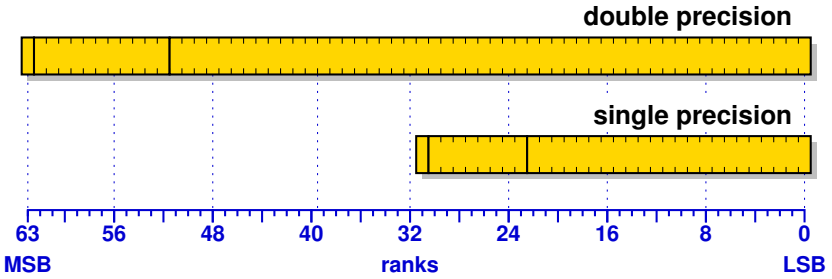
For accuracy purpose, the mantissa must be normalized ($x_0 \neq 0$)

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	$23+1$



Logarithmic Number System (LNS)

Representation of x:
(sign of x, fixed-point approximation of $\log_2 x$)

LNS operations:

$$\begin{aligned}
\log_2(a \times b) &= \log_2 a + \log_2 b \\
\log_2(a \div b) &= \log_2 a - \log_2 b \\
\log_2(a \pm b) &= \log_2 a + \log_2(1 \pm 2^{\log_2 b - \log_2 a}) \\
\log_2(a^q) &= q \times \log_2 a
\end{aligned}$$

where the functions $\log_2(1+2^x)$ and $\log_2(1-2^x)$ are approximated (tables or polynomials)

Applications in digital signal processing and digital control

Part III

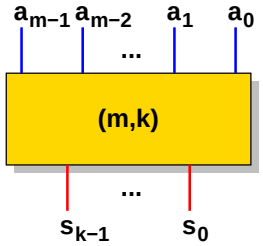
Basic operations (over integers)

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)

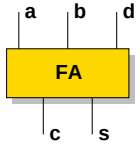
$$\sum_{i=0}^{m-1} a_i = \sum_{j=0}^{k-1} s_j 2^j$$



Standard counters:

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

FA Cell



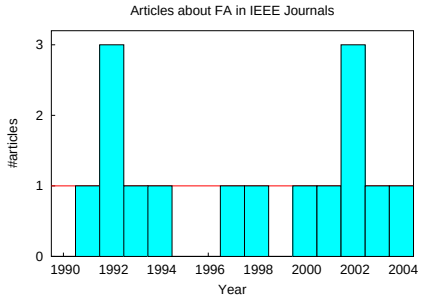
a	b	d	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Arithmetic equation:

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

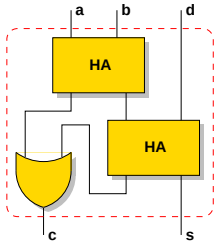
Logic equation:

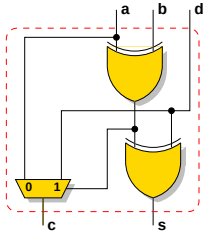
$$\begin{aligned} s &= a \oplus b \oplus d \\ c &= ab + ad + bd \end{aligned}$$

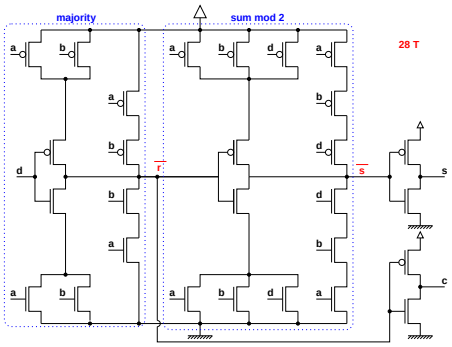


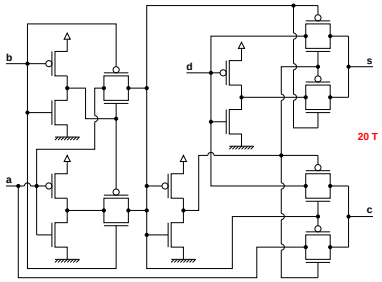
There many implementations of the FA cell

FA Implementations



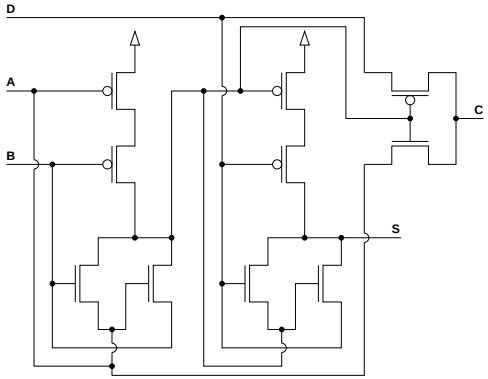






Optimized FA Cell

10-transistor solution[1] (some output signals are weak signals):

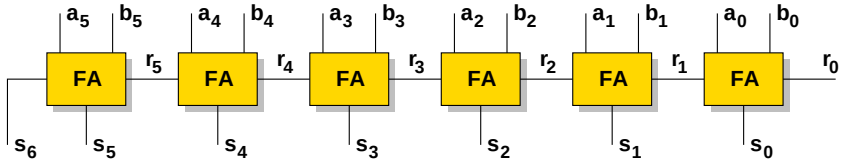


A	B	D	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0_w	1
0	1	1	1	0
1	0	0	0_w	1
1	0	1	1_w	0
1	1	0	1	0
1	1	1	1_w	1_w

[1] H. T. Bui, Y. Wang et Y. Jiang. *Design and analysis of low-power 10-transistor full adders using novel XOR–XNOR gates*. IEEE Trans. CaS, jan. 2002.

Carry Ripple Adder (CRA)

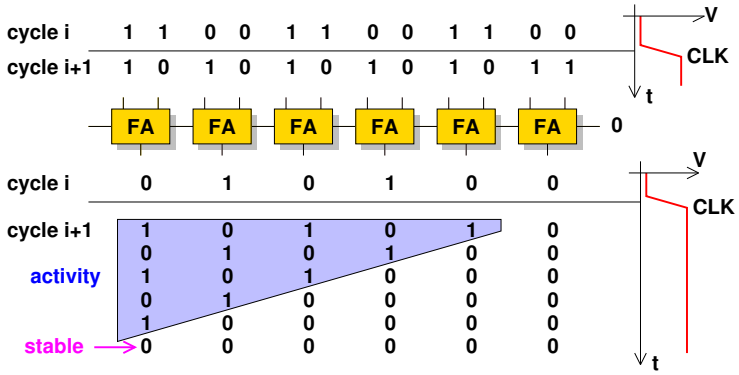
Very simple architecture: n FA cells connected in series



	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



Theoretical models (equiprobable and uniform distribution of inputs):

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

Subtraction

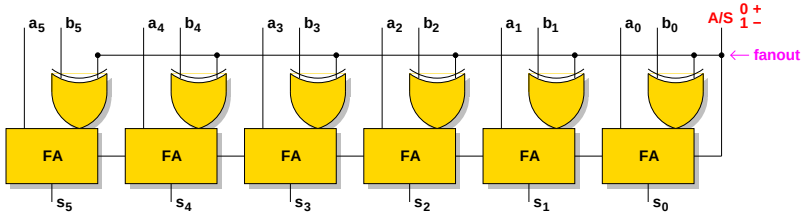
Using 2's complement representation:

$$A - B = A + (-B)$$

Implementation:

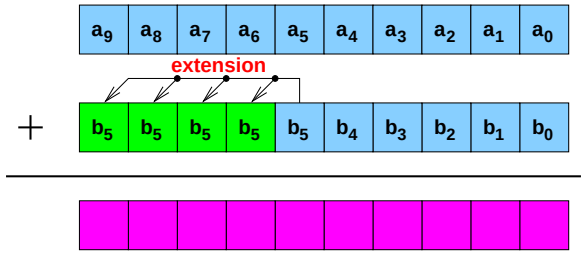
$$B + \overline{B} = \underbrace{111\ldots111}_{n \text{ bits}} = -2^{n-1} + \sum_{i=0}^{n-2} 2^i = -1$$

$$\Rightarrow -B = \overline{B} + 1$$



Sign Extension

Required for the addition of different size operands in 2's complement



Warning:

- fanout
- order in case of multiple additions

Representation(s) of Numbers and Power Consumption

Impact of the representation of numbers:

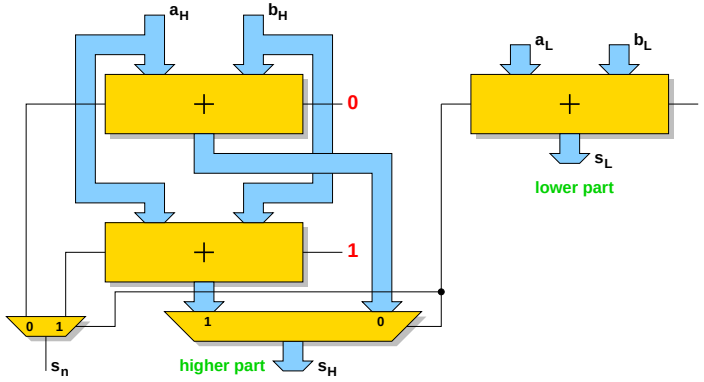
- operator speed
- circuit area
- useful and useless activity

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	1111111111100101	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:
$$A = (a_{n-1} a_{n-2} \dots a_1 a_0) = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

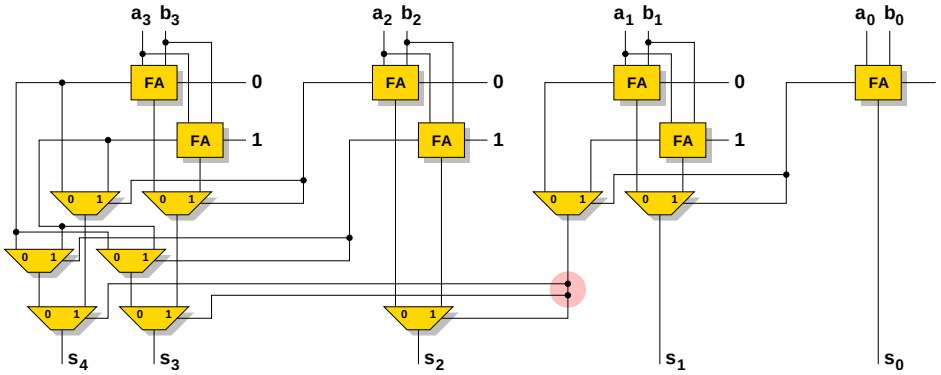
Carry-Select Adder

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



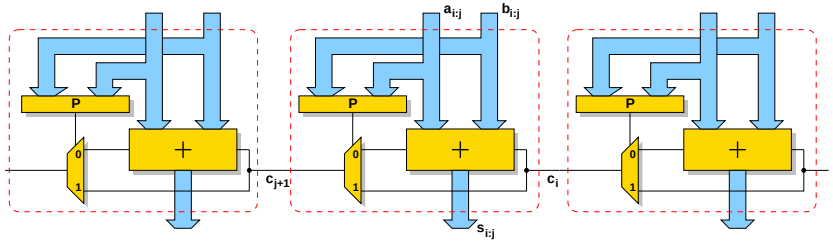
Recursive version $\longrightarrow$ $O(\log n)$ delay but...

Carry-Select Adder: Fanout Problem



Carry Skip Adder

Idea: split in blocks, fast detection of the block propagation in each block (all ranks of the block propagate the block input carry)



Questions:

- delay with uniform block size? → analytical models
- delay with non-uniform block size? → heuristics (papers)

Carry Look-ahead Adder

Idea: compute all carries as fast as possible (instead of propagating them)

At rank i, the input carry c_i is 1 in the following cases:

- rank $i-1$ generates a carry
  $\hookrightarrow g_{i-1} = 1$
- rank $i-1$ propagates a carry generated at rank $i-2$
  $\hookrightarrow p_{i-1} = g_{i-2} = 1$
- ranks $i-1$ and $i-2$ propagate a carry generated at rank $i-3$
  $\hookrightarrow p_{i-1} = p_{i-2} = g_{i-3} = 1$

$\vdots$

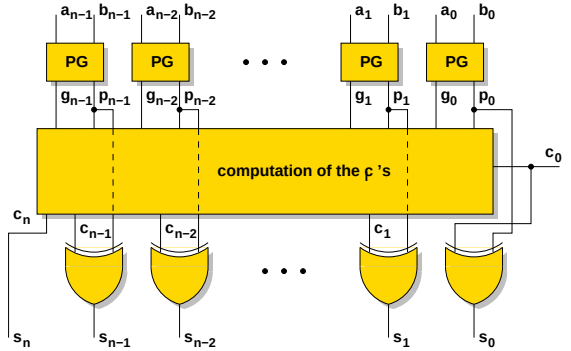
- ranks $i-1$ to 0 propagate the adder input carry c_0 (set to 1)
  $\hookrightarrow p_{i-1} = p_{i-2} = \ldots = p_1 = p_0 = c_0 = 1$

All carries can be computed using the relation ($c_i = g_{i-1} + c_{i-1}p_{i-1}$):

$$c_i = g_{i-1} + p_{i-1}g_{i-2} + p_{i-1}p_{i-2}g_{i-3} + \ldots + p_{i-1}\cdots p_1 g_0 + p_{i-1}\cdots p_0 c_0$$

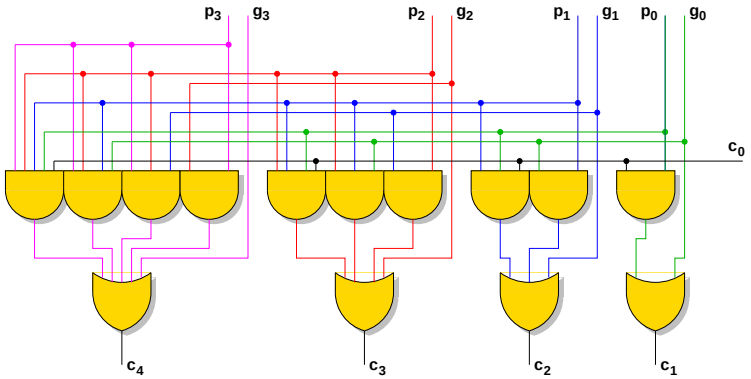
CLA architecture: parallel evaluation of

- (g_i, p_i) for all i
- carries c_i for all i using the above equation
- sums using $s_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i$



Carry Look-ahead Adder: 4-Bit Example

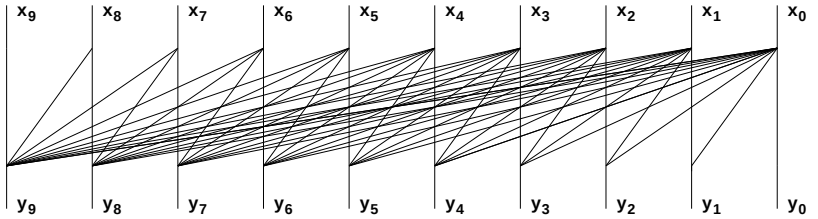
$$c_1 = g_0 + p_0 c_0$$
$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$
$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$
$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$



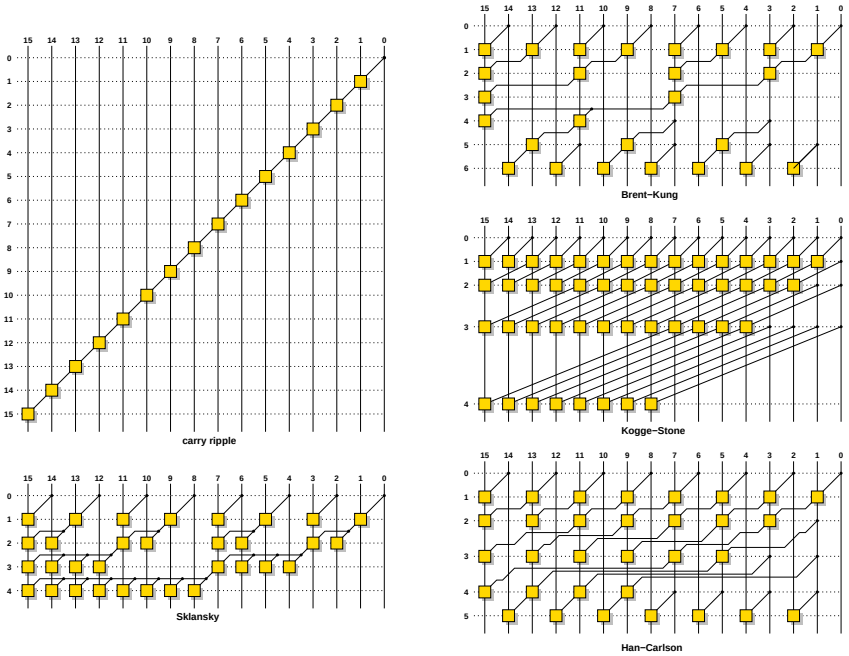
Parallel-Prefix Problems

The n outputs (y_{n-1}, y_{n-2},$\cdots$, y_0) are computed using the n inputs (x_{n-1}, x_{n-2},$\cdots$, x_0) and the associative operator $\square$:

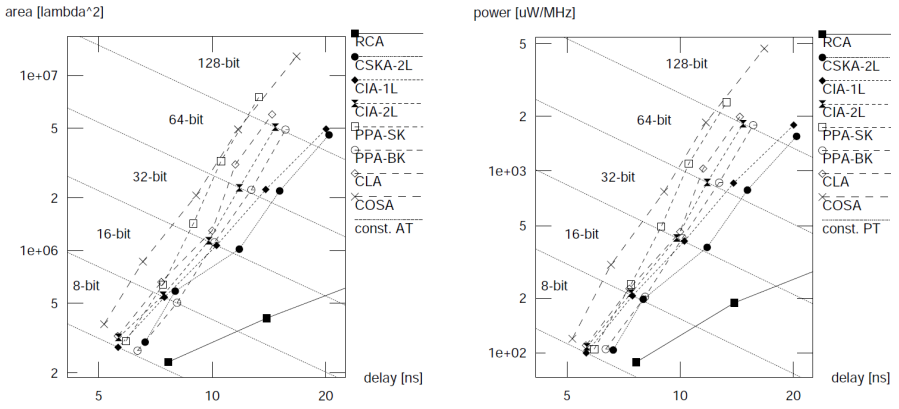
$$\begin{aligned} y_0 &= x_0 \\ y_1 &= x_1 \square x_0 \\ y_2 &= x_2 \square x_1 \square x_0 \\ &\vdots \\ y_{n-1} &= x_{n-1} \square x_{n-2} \square \cdots \square x_1 \square x_0 \end{aligned}$$



Parallel-Prefix Addition: Standard Architectures



Comparison of Adders



Source: PhD R. Zimmermann [11]

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 $\{-\alpha, -\alpha+1, \ldots, 0, \ldots, \alpha-1, \alpha\}$ instead of $\{0, 1, 2, \ldots, \beta-1\}$ 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)

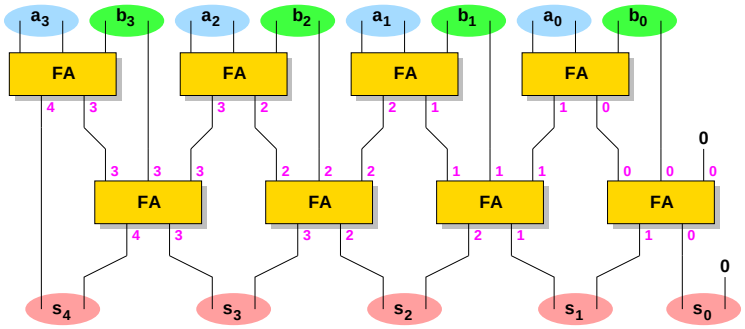
Such a representation is said redundant

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

Carry-Save Adder

In carry-save, the number A is represented in radix 2 using digits $a_i \in \{0, 1, 2\}$ coded by 2 bits such that $a_i = a_{i,c} + a_{i,s}$ where $a_{i,c} \in \{0, 1\}$ and $a_{i,s} \in \{0, 1\}$

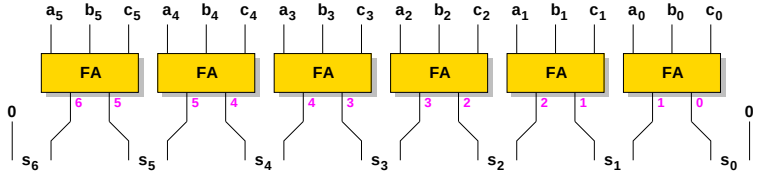
$$A = \sum_{i=0}^{n-1} a_i 2^i = \sum_{i=0}^{n-1} (a_{i,c} + a_{i,s}) 2^i$$



Carry-save addition: delay of 2 FA cells ($T = 0(1)$)

Carry-Save Trees

Example with 3 inputs: A, B and C



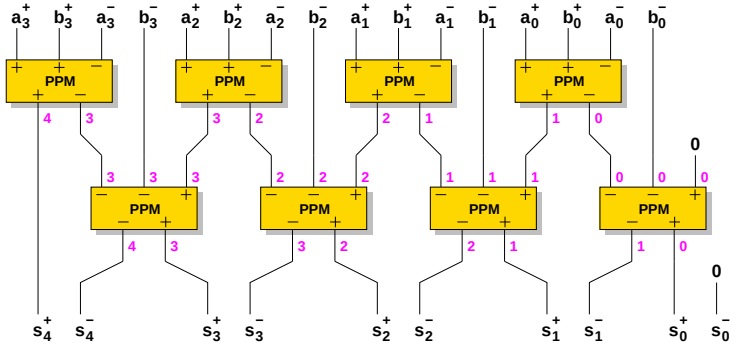
Carry-save reduction tree: $n(h)$ non-redundant inputs can be reduced by a h-level carry-save tree where $n(h) = \lfloor 3n(h-1)/2 \rfloor$ and $n(0) = 2$

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

Borrow-Save Addition

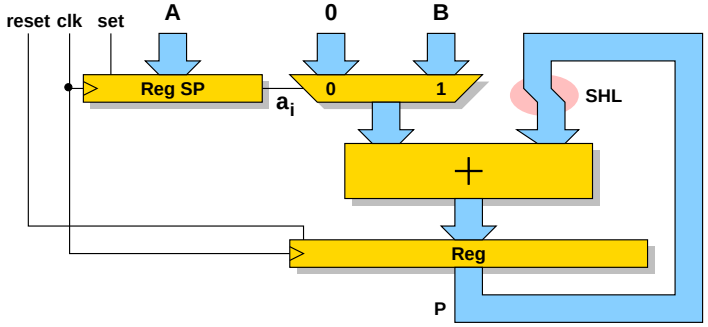
In borrow-save, the number A is represented in radix 2 using digits $a_i \in \{-1, 0, 1\}$ coded by 2 bits such that $a_i = a_i^+ - a_i^-$ where $a_i^+ \in \{0, 1\}$ and $a_i^- \in \{0, 1\}$

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



Borrow-save addition: delay of 2 PPM cells ($T = 0(1)$)

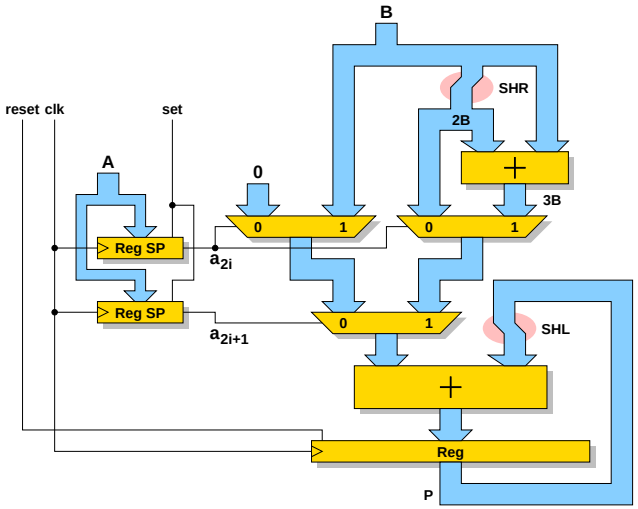
Shift-And-Add Multiplication: Implementation



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

Shift-And-Add Multiplication: High-Radix Method

Idea: use high-radix digits for the multiplier



Problem: multiple generation

Booth Recoding

In 1951, Booth proposed to increase the number of 0s in the multiplier using the digit set $\{-1 = \overline{1}, 0, 1\}$

Recoding based on the identity:
$2^{i+k} + 2^{i+k-1} + 2^{i+k-2} + \cdots + 2^i = 2^{i+k+1} - 2^i$

Example: the integer 60 is represented by $00111100 = 0100\overline{1}00$

The recoding replaces strings of 1s by a representation with more 0s

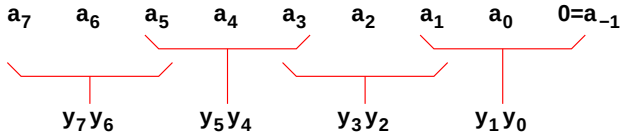
But, in some cases, this basic method leads to more 1 (or $\overline{1}$)!

Example: the value 01010101 is recoded to $\overline{1}1\overline{1}1\overline{1}1\overline{1}1$

$\Longrightarrow$ modified Booth's recoding

Modified Booth's Recoding

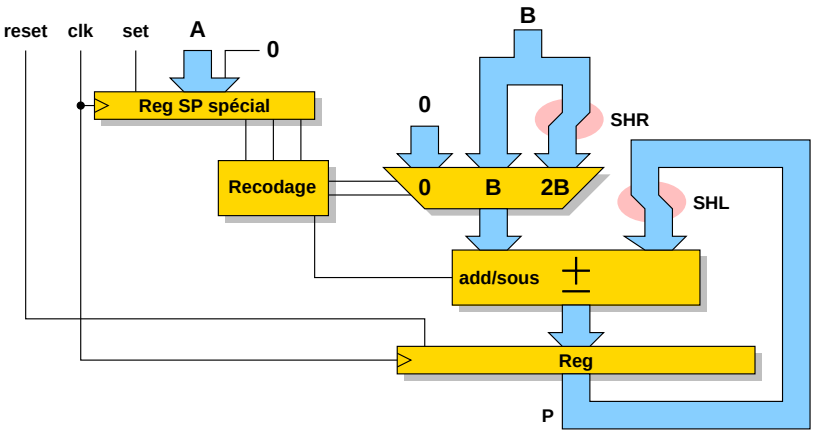
Idea: do not recode isolated 1 but only strings of 1



a_i	a_{i-1}	a_{i-2}	y_i	y_{i-1}	meaning	operation
0	0	0	0	0	string of 0s	$+0$
0	0	1	0	1	end of a string of 1s	$+B$
0	1	0	0	1	isolated 1	$+B$
0	1	1	1	0	end of a string of 1s	$+2B$
1	0	0	$\overline{1}$	0	beginning of a string of 1s	$-2B$
1	0	1	1	$\overline{1}$	isolated 0	$-B$
1	1	0	0	$\overline{1}$	beginning of a string of 1s	$-B$
1	1	1	0	0	middle of a string of 1s	$+0$

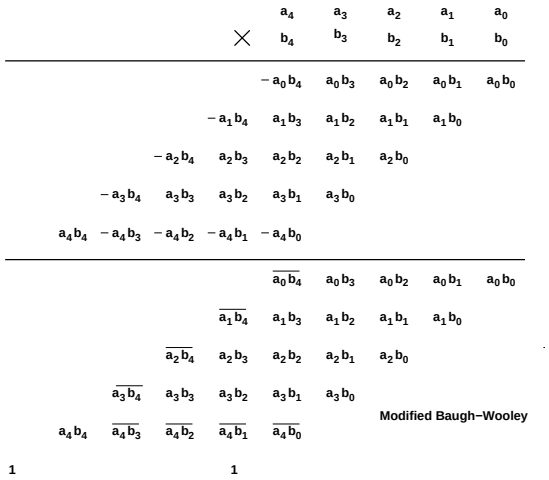
Improvement: leads to a n-product with $\lfloor n/2 \rfloor + 1$ additions and shifts at most

Modified Booth Multiplier



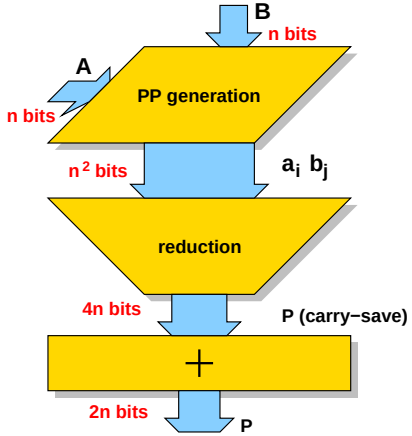
2's Complement Product

If A and B are represented using 2's complement, then some partial products have a negative weight



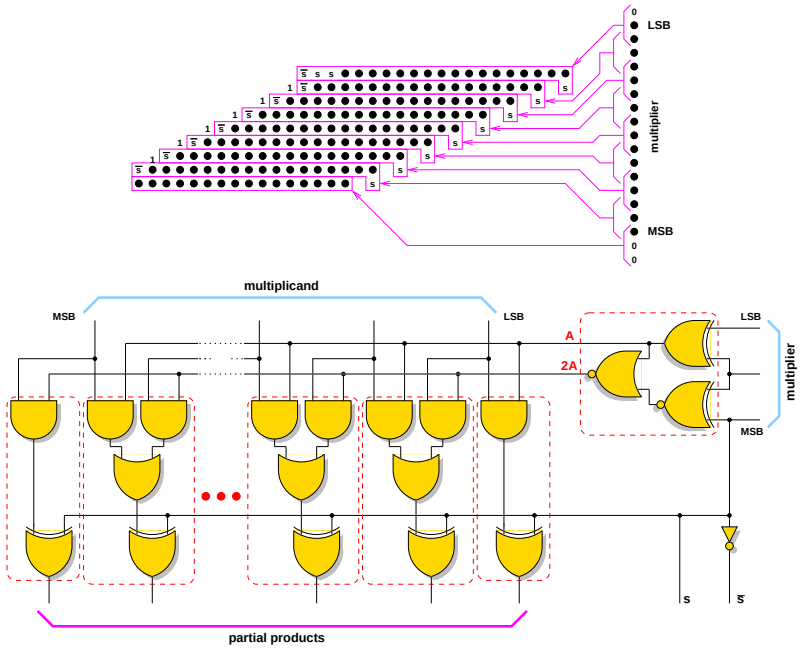
Tree Multipliers

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 $O(\log n)$

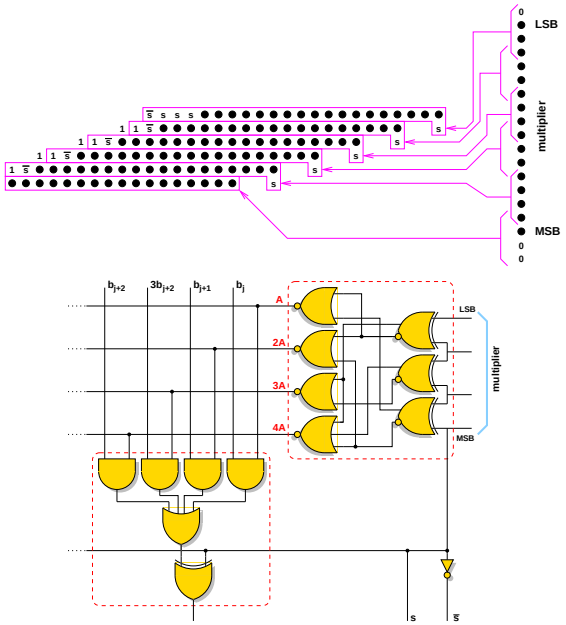


Multiplication delay $O(\log n)$, area $O(n^2)$

Partial Product Generation: Booth-2



Partial Product Generation: Booth-3



Booth's Recoding Benefit

Booth-2 case:

- Speed: 1 or 2 levels removed from the reduction tree but this is balanced by the recoding step
- Area: true benefit (30%), recoding cells limit the fanout problem and their implementation is efficient in CMOS

Booth-3 case:

It is rarely used because of the very complex recoding and partial product cells

Partial Products Addition: Reduction Trees

Goal: compute the addition of the $n/2+1$ partial products in *carry-save*

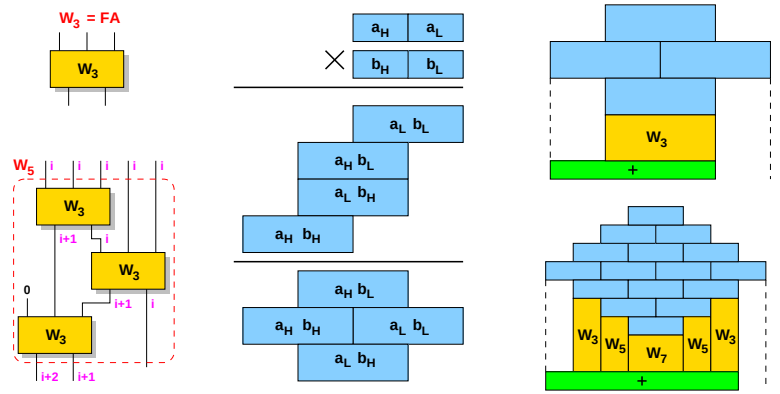
Several reduction trees can be used:

- Trees based on FA cells:
  - Wallace trees
  - Dadda trees
  - fast reduction trees
- Trees based on "4 to 2" cells
- Trees based on counters or compressors

Wallace Trees

Wallace trees[2] are p-input counters ($\lceil log_2 p \rceil$ outputs)

A Wallace tree with $2^{p+1}-1$ inputs can be built based on 2^p-1-input Wallace trees (a 3-input Wallace tree is a FA)

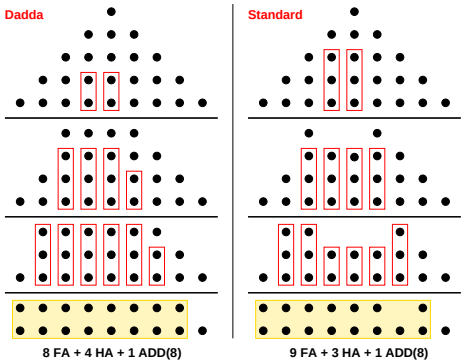


[2] C.S. Wallace. *A suggestion for a fast multiplier*. IEEE Transactions on Computers, Feb. 1964.

Dadda Trees

Idea: minimal reduction at each level of the tree (just enough to reach the next level in $n(h) = \lfloor 3n(h-1)/2 \rfloor$ with $n(0) = 2$)

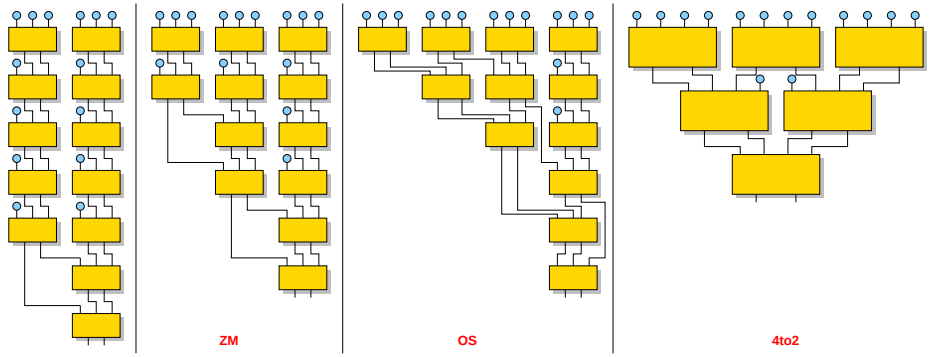
h	1	2	3	4	5	6	7	8	9	10	11
$n(h)$	3	4	6	9	13	19	28	42	63	94	141



Area benefit on large multipliers. Example: $n = 12$ bits $\Rightarrow$ 11% less gates compared to a Wallace tree

Placement and Routing Problems at the Gate Level

What is the best topology?

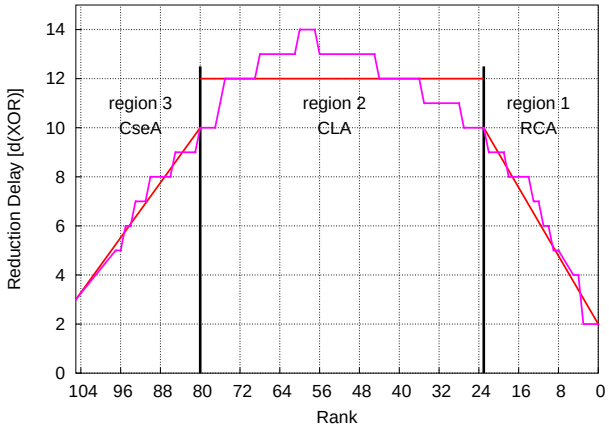


Example: 14-bit reduction

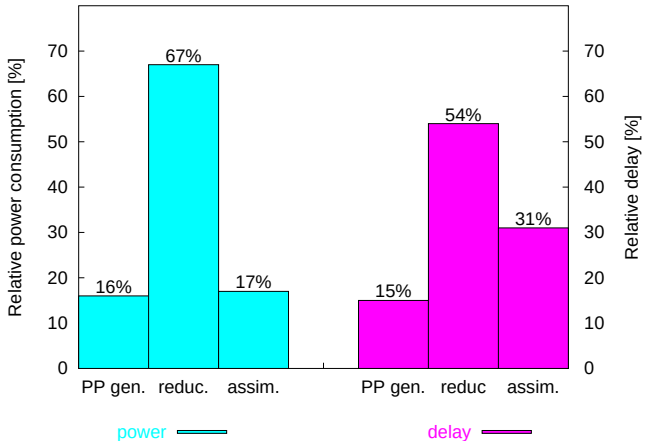
Final Addition in Multipliers

The final addition or assimilation of the carries is performed using a fast adder

Based on the relative arrival time of the partial products, some (small) optimizations can be done: use of several adder types depending on the rank region



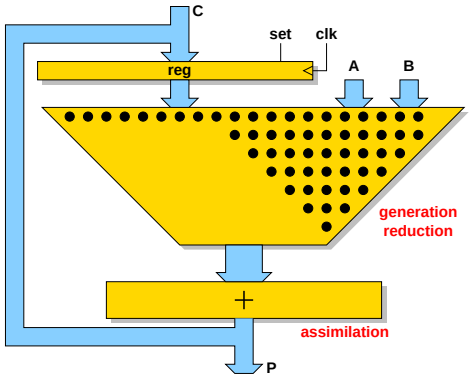
Power Consumption in Fast Multipliers



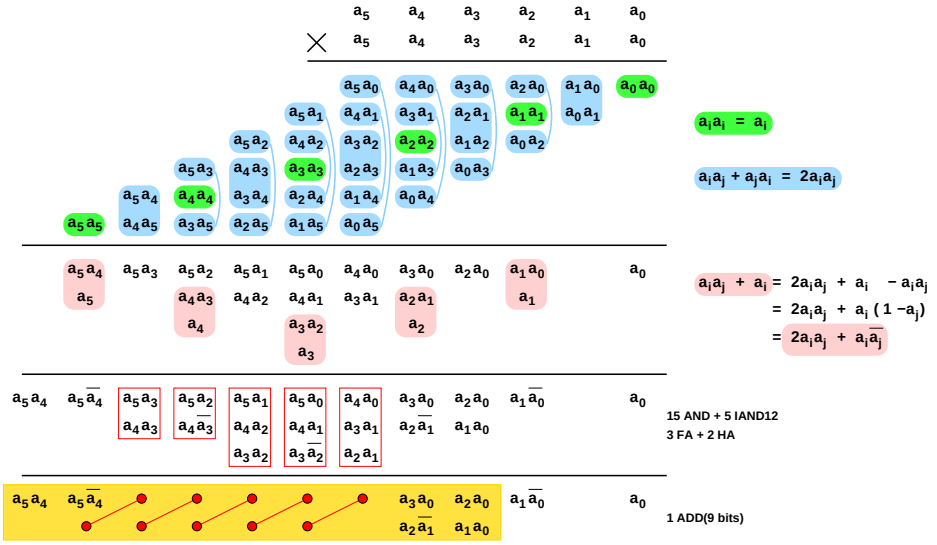
- 30% to 70% of redundant transitions (useless)
- place and route steps based on the internal arrival time
- add a pipeline stage

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 $m >> n$ (e.g., $16 \times 16 + 40 \longrightarrow 40$ 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



Multiplication by Constants (1/2)

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

Papers: [2, 1]

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 ±
CSD	$(x \ll 17)-(x \ll 14)-(x \ll 12)+(x \ll 10)$ $-(x \ll 7)-(x \ll 5)+(x \ll 3)-x$	7 ±
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 ±
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 ±

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%

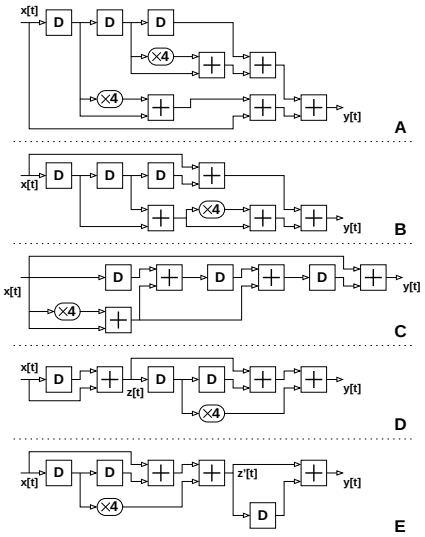
operator	init.	[1]	[2]	our
8×8 Had.	56	24	—	24
$(16, 11)$ R.-M.	61	43	31	31
$(15, 7)$ BCH	72	48	47	44
$(24, 12, 8)$ Golay	76	—	47	45

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 $(1, 5, 5, 1)$



Part IV

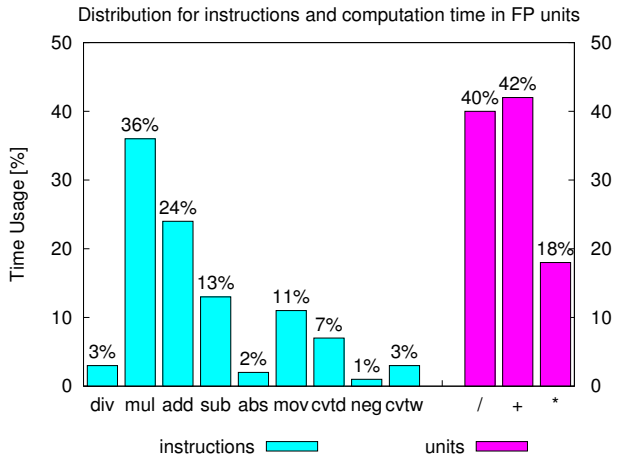
More advanced operations

Division (square root)

Elementary Functions

Division Not an Important Operation? Not So Clear!

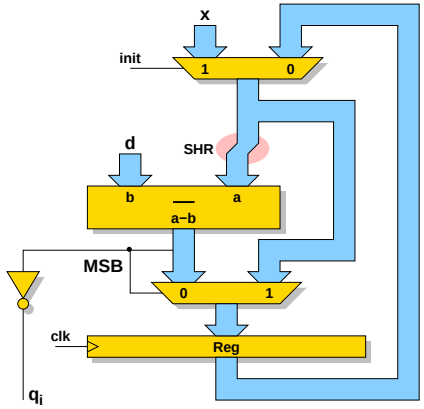
Technical report[3] from S. Oberman and M. Flynn : "*Design issues in floating-point division*", CSL-TR-94-647 Stanford University.



[3] SPECfp92 DECstation MIPS R3000 (latency : 2c add, 5c mul, 19c div), compil. O3.

Restoring Division

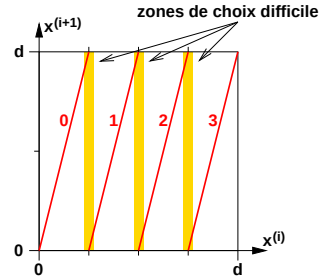
```
1  for i from 1 to n do
2      x ⟵ 2x
3      x ⟵ x − d
4      if x ≥ 0 then
5          q_i ⟵ 1
6      else
7          q_i ⟵ 0
8          x ⟵ x + d
```

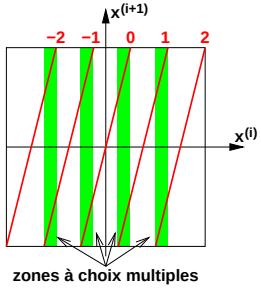


How to Speed-Up Division ?

Idea : high radix iterations

Problem: accurate comparisons with divisor multiples



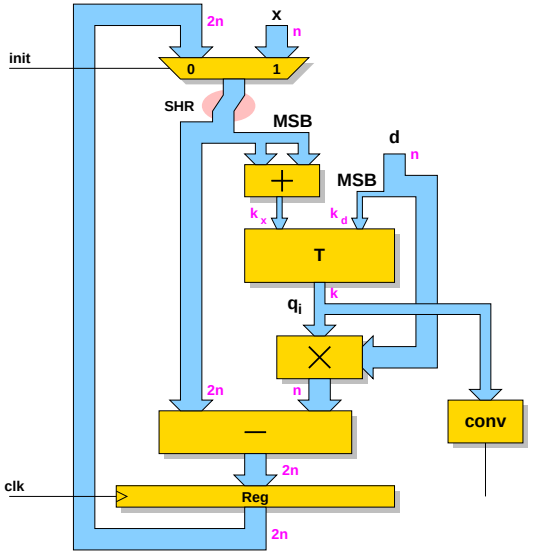


Solution: redundant representation for quotient digits

↪ approximate comparisons

SRT Division Architecture

SRT: Sweeney, Robertson and Tocher, 1958

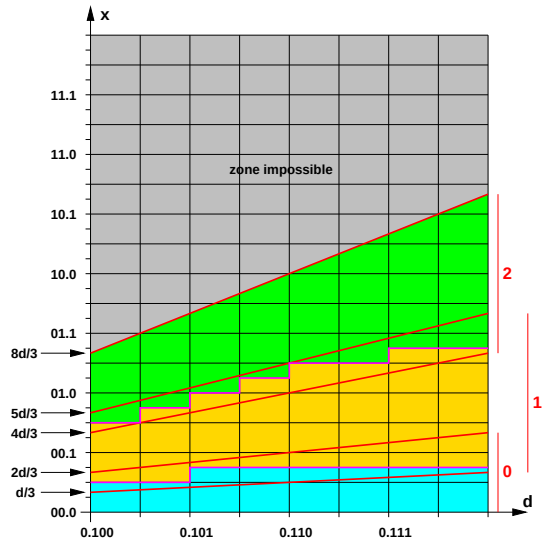


SRT Division Table

Diagram remainder-divisor :

- radix $\beta = 4$
- $q_i \in \{-2, -1, 0, 1, 2\}$
- input :
  - d on 3 bits
  - x on 5 bits

Remark : symmetric diagram (w.r.t. d axis)



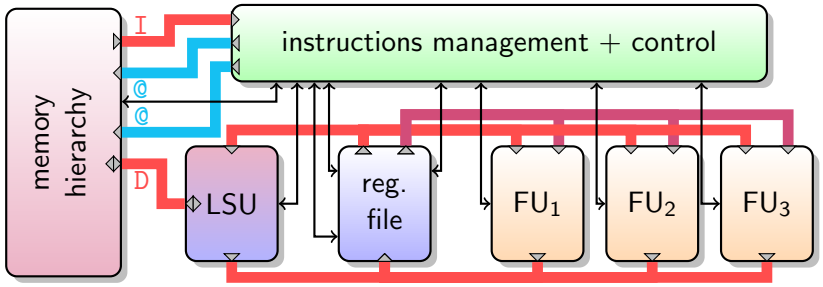
Micro-Coded Division using Newton-Raphson

Specific iteration with quadratic convergence to $1/x$:

$$x_{i+1} \quad = \quad x_i \cdot (2 - d \cdot x_i)$$

Then, use $d \cdot (1/x)$ (and a few op. for rounding if needed)

Motivation: simplified processor architecture



The Return of SRT Dividers in Processors...

Problems in Newton-Raphson iterations: power consumption and thermal hot-spot!

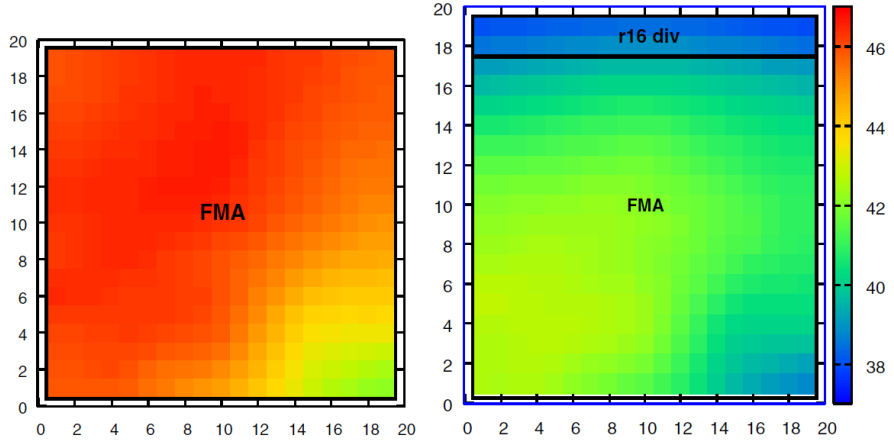
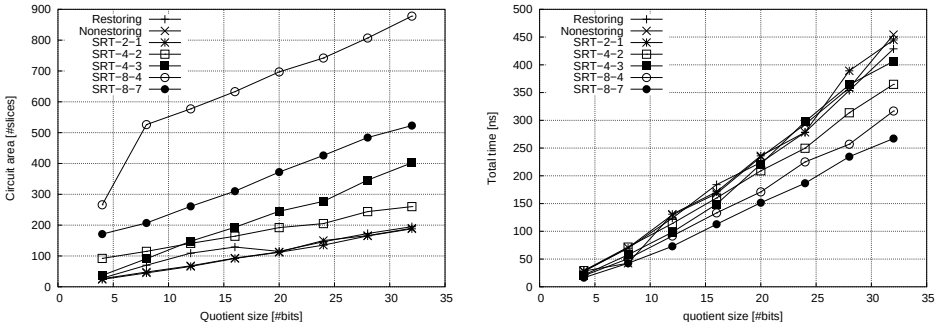


Fig. 6. Comparison of thermal profiles: FMA alone (left) and FMA plus **r16div** (right). Temperatures are in °C.

Source: W. Liu and A. Nannarelli. *Power Dissipation Challenges in Multicore Floating-Point Units*. Proc. ASAP 2010.

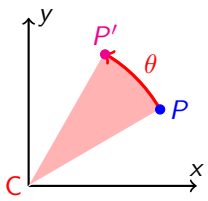
Divgen: a Division Unit Generator

- VHDL output
- supported algorithms: restoring, non-restoring and SRT
- partial remainder: 2's complement or carry-save
- # guard bits, SRT table folding, gray encoding



Publi. [5], web: `http://lipforge.ens-lyon.fr/www/divgen/`

Motivations for More Advanced Operations

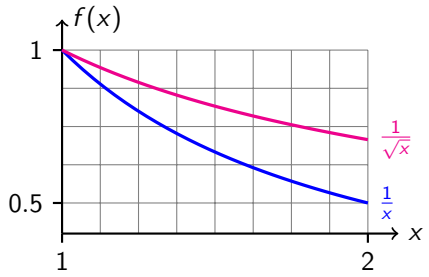


$$P' = \mathcal{R}(P, C, \theta)$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Initial approximations for floating-point units:

- reciprocal $\frac{1}{x}$ for division
- inverse square root $\frac{1}{\sqrt{x}}$ for square root



and many other functions: $\exp(x), \log(x), \arctan(x), \cosh(x), \ldots$

Error and Accuracy

Question: how many bits are correct ?

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

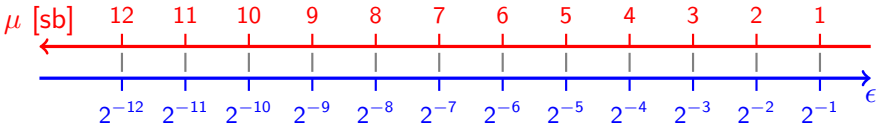
Error, ϵ: distance between 2 objects (e.g. $\epsilon = ||f(x) - p(x)||$)

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

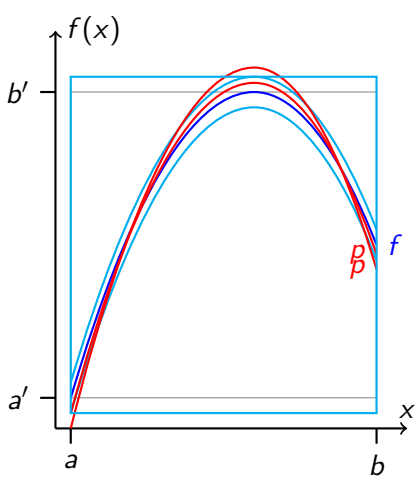
$$\mu = -\log_2 |\epsilon|$$

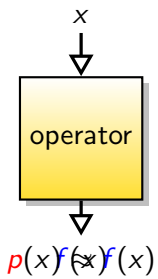
Notation: μ expressed in terms of correct or significant bits ([cb], [sb])

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



Polynomial Approximations





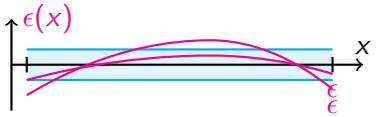
x argument

[a, b] domain

f function

p polynomial

Question: what is the best p?

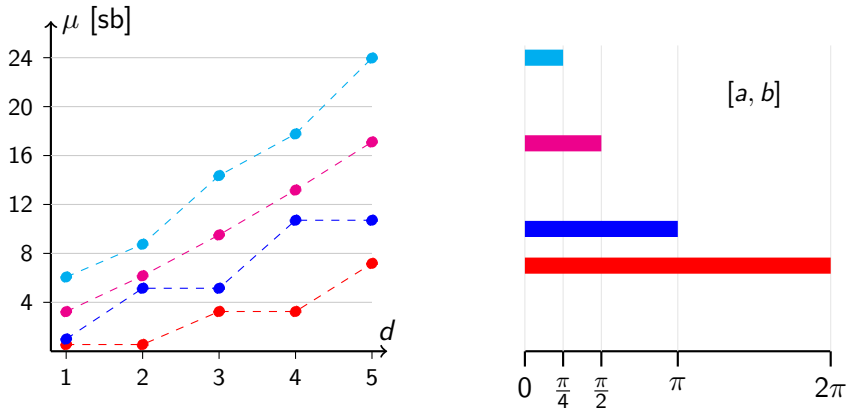


$\epsilon(x) = f(x) - p(x)$ *ϵ approx. error*

$\epsilon(x) \leq \epsilon_{target}$ *ϵ_{target} maximum allowed error*

Accuracy, Degree and Evaluation Cost

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



- higher accuracy $\Longrightarrow$ higher degree
- higher degree $\Longrightarrow$ 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 $\longrightarrow$ high operation cost and smaller accuracy
- Horner scheme $\longrightarrow$ smallest cost but sequential
- Estrin scheme $\longrightarrow$ 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 data-path width in order to limit round-off errors

Examples: $1/3 = 0.33333333\ldots \rightarrow 0.3333$ or 0.3334 in $1\mathrm{Q}_{10}4$ format

Question: what is the best data-path width?

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 license (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/`

Gappa Example

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

```
p0 = 571/512;     p1 = 275/512;     p2 = 545/512;

x = fixed<-9,dn>(Mx);

y1 fixed<-9,dn>= p2 * x + p1;
p  fixed<-9,dn>= y1 * x + p0;

Mp = (p2 * Mx + p1) * Mx + p0;

{
    Mx in [0.5,1]   /\   |Mp-Mf| in [0,0.001385]
->
    |p-Mf| in ?
}
```

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

```
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)}]
```

Minimax Polynomial Approximations

- approximation error $\epsilon_{\text{app}} = ||f - p||_\infty = \max_{a \le x \le b} |f(x) - p(x)|$
- 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 $\le d$
- p^* computed using an algorithm from Remez (numerically implemented in Maple, Matlab, sollya...)

Problems:

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

Finite Precision Coefficients Selection Problem

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

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

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

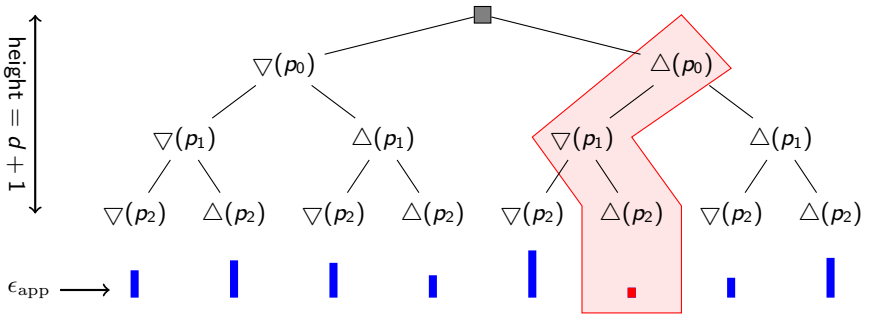
Above example with 1Q9 format (all values for domain $[1/2, 1]$):

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

Basic Coefficient Selection Method

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

- round up $p_i = \triangle(p_i^*)$, round down $p_i = \triangledown(p_i^*)$
- 2 values per coeff. $\Longrightarrow$ 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}



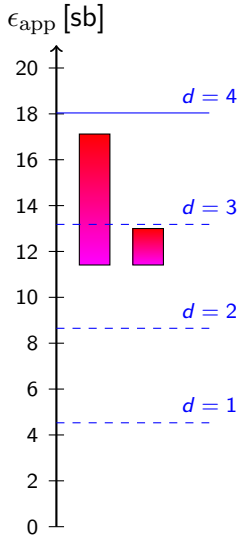
Result: $p(x) = \sum_{i=0}^{d} p_i x^i$ where all p_i are representable in target format

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

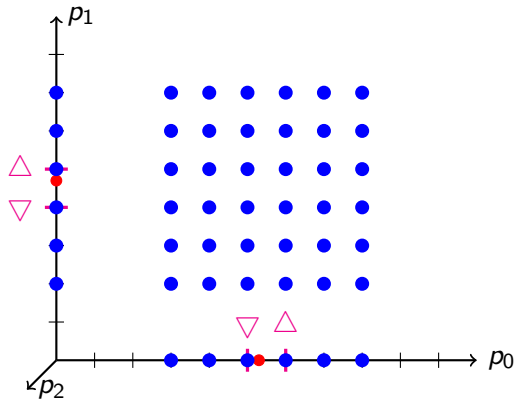
$\epsilon_{\text{app}}(p^*)$ $\rightsquigarrow$ 18.04 sb

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

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



Improved Coefficient Selection Methods



Other selection methods:

- linear programming methods, e.g. `meplib` software
  `https://lipforge.ens-lyon.fr/projects/meplib/`
- euclidean lattices reduction (LLL), e.g. `sollya` software
  `http://sollya.gforge.inria.fr/`

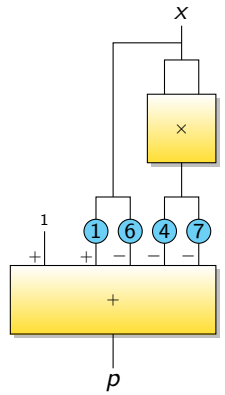
Example: $\sqrt{x}$ over $[1,2]$ and $\mu \leq 8$ sb

Selection of coefficients leading to sparse recodings

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

$p = 1 + (0.100001\overline{1})_2 x - (0.0001001)_2 x^2$

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

Part V

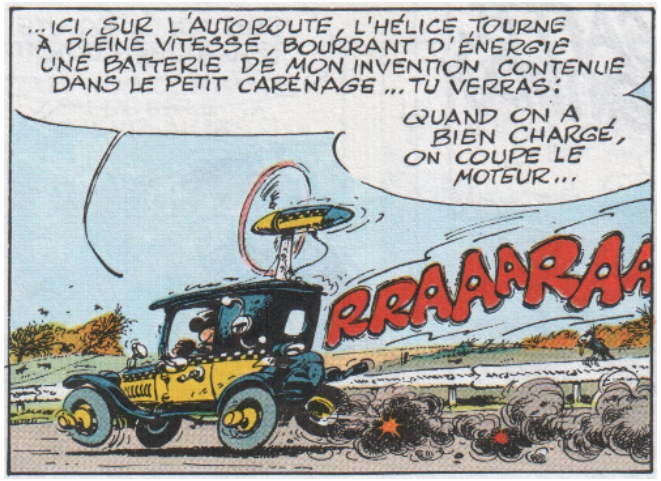
Conclusion & references

Conclusion

When designing arithmetic operators for low-power applications:

- 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/`

Take Care to Not So Good Ideas. . .



Gaston 14. La sage des gaffes. Page 5. Par Franquin.
Dupuis, 1982 (réédition 1993). ISBN 2-8001-0955-6

Arithmetic Operators for Circuits in 20 Years ???

Fluide Glacial
Numéro HS 46
(20/04/2009)
Où serons nous en 2040 ?
Couverture de Goossens

References I

N. Boullis and A. Tisserand.
Some optimizations of hardware multiplication by constant matrices.
In J.-C. Bajard and M. Schulte, editors, *Proc. 16th Symposium on Computer Arithmetic (ARITH)*, pages 20–27, Santiago de Compostela, Spain, June 2003. IEEE Computer Society.

N. Boullis and A. Tisserand.
Some optimizations of hardware multiplication by constant matrices.
IEEE Transactions on Computers, 54(10):1271–1282, October 2005.

H. T. Bui, Y. Wang, and Y. Jiang.
Design and analysis of low-power 10-transistor full adders using novel XOR-XNOR gates.
IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 49(1):25–30, January 2002.

L. Dadda.
Some schemes for parallel multipliers.
Alta Frequenza, 34:349–356, 1965.

R. Michard, A. Tisserand, and N. Veyrat-Charvillon.
Divgen: a divider unit generator.
In F. T. Luk, editor, *Proc. Advanced Signal Processing Algorithms, Architectures and Implementations XV*, volume 5910, pages 1–12, San Diego, California, U.S.A., August 2005. SPIE.

References II

A. Tisserand.
Low-power arithmetic operators.
In C. Piguet, editor, *Low Power Electronics Design*, chapter 9. CRC Press, November 2004.

A. Tisserand.
Introduction aux représentations des nombres et opérateurs arithmétiques à basse consommation d'énergie.
Technique et Science Informatiques, 26(5):639–646, May 2007.

A. Tisserand.
Unités de calcul flottant.
Cours École Thématique ARCHI07, March 2007.

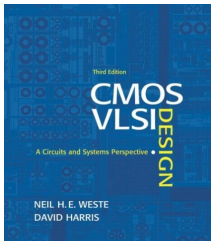
A. Tisserand.
Power analysis and cryptosystem security: Attacks and countermeasures.
Cours École Thématique ECOFAC 2012, May 2012.

C.S. Wallace.
A suggestion for a fast multiplier.
IEEE Transactions on Electronic Computers, EC-13:14–17, 1964.

R. Zimmermann.
Binary Adder Architectures for Cell-Based VLSI and their Synthesis.
Phd thesis, Swiss Federal Institute of Technology Zurich, 1998.

Good Books

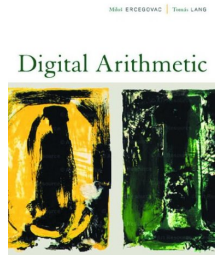
CMOS VLSI Design
A Circuits and Systems Perspective
Neil Weste and David Harris
3rd edition, 2004
Addison Wesley
ISBN: 0–321–14901–7



Micro et nano-électronique
Bases, Composants, Circuits
Hervé Fanet
2006
Dunod
ISBN: 2–10–049141–5

Good Books

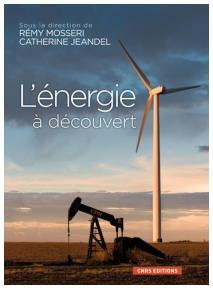
Digital Arithmetic
Milos Ercegovac and Tomas Lang
2003
Morgan Kaufmann
ISBN: 1–55860–798–6



Arithmétique des ordinateurs
Jean-Michel Muller
1989
Masson
ISBN: 2–225–81689–1
(web version)

Good Books

L'Énergie à découvert
Sous la direction de R. Mosseri et C. Jeandel
2013
CNRS Éditions
ISBN: 978-2-271-07678-6



The end, questions ?

Contact:

- `mailto:arnaud.tisserand@irisa.fr`
- `http://people.irisa.fr/Arnaud.Tisserand/`
- CAIRN Group `http://www.irisa.fr/cairn/`
- IRISA Laboratory, CNRS–INRIA–Univ. Rennes 1
  6 rue Kerampont, CS 80518, F-22305 Lannion cedex, France

Thank you