Hardware Arithmetic Operators

Arnaud Tisserand

CNRS, IRISA laboratory, CAIRN research team

École ARCHI
Mont-Louis, Pyrénées Orientales
June 13–17th 2011

Summary

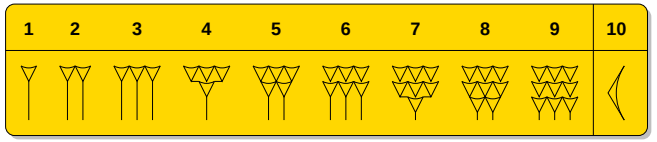
Part I

Introduction

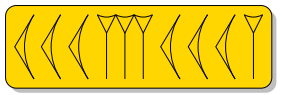
Babylonian Arithmetic

Use of a positional number system with:

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



Example:



$= 33 \times 60 + 31 = 2011$

Egyptian Multiplication

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

Rewriting rules:

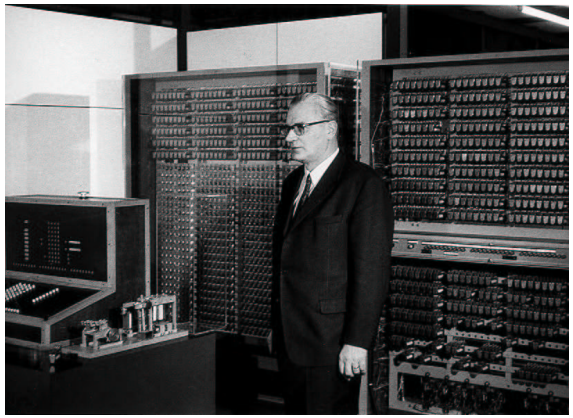
$$M(0, m, p) \longmapsto p$$
$$M(2n, m, p) \longmapsto M(n, 2m, p)$$
$$M(2n + 1, m, p) \longmapsto M(n, 2m, p + m)$$

Example:

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

First Computer with a Floating-Point Unit

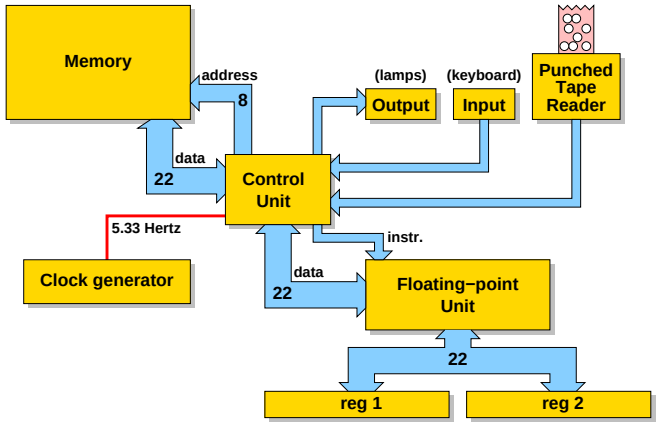
Z3 designed by Konrad Zuse (1910–1995) in 1941, Berlin



picture of the version rebuilt in 1961

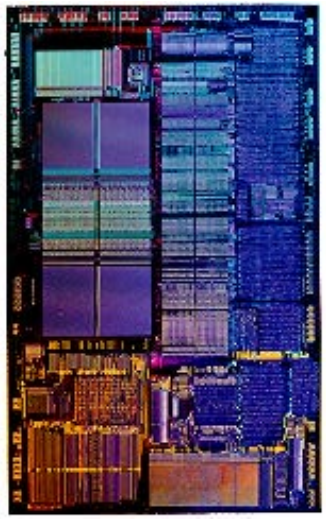
Source: `http://www.epemag.com/zuse/`

Z3: Architecture and Characteristics



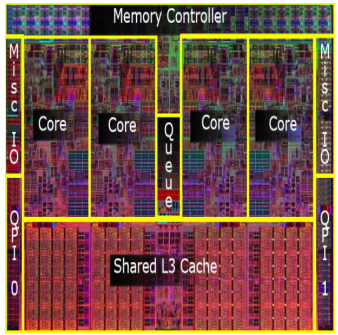
size	5 m × 2 m × 0.8 m
weight	≈ 1000 kg
frequency	5.33 Hz
technology	elect. relays (cpu.: 600, mem.: 1400)
power consumption	≈ 4000 W

Intel 486 Processor



- 32-bit processor
- embedded arithmetic co-processor
- first commercialization in 1989
- frequencies: 50, 33 and 25,MHz
- transistors number: 1.2×10^6
- technology: CMOS 0.8 or 1.0 μm
- silicon area: 81 mm^2
- power supply: 5 V
- package: 168 PGA
- pipeline: 5 stages
- L1 cache: 8 KB (4w SA, WT)
- 1 ALU

Intel Xeon Core i7 (code name: Bloomfield)



- 64-bit processor
- quad core with 2 threads
- hyper-threading comm. 6.4 GT/s
- 3 DDR3-1066 links $\rightarrow$ 25.6 GB/s
- first commercialization in Q4.2008
- frequencies: 2.66 to 3.33 GHz
- transistors number: 731×10^6
- technology: CMOS 45 nm
- silicon area: 263 mm^2
- power supply: 130 W under 1.375 V
- socket/package: 1366 LGA
- caches: 32 kB I + 32 kB D L1, 4×256 kB L2, 8 MB L3

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)

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

- 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 $B = 2^{n-1} - 1$)

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

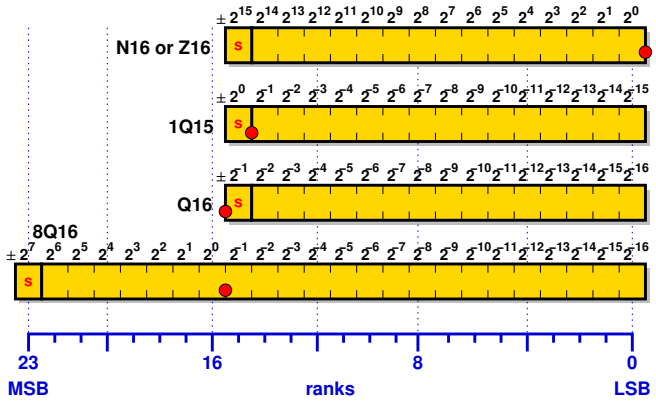
- ...

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:

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

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

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

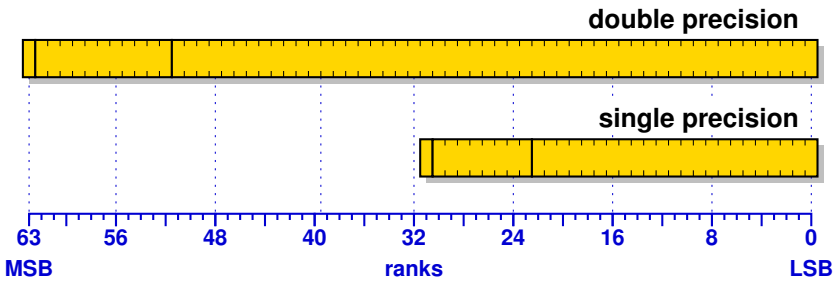
For accuracy purpose, the mantissa must be normalized ($x_0 \neq 0$)

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	$23 + 1$



IEEE-754: Exponent and Special Values

format	size k	bias b	unbiased e_{min}	unbiased e_{max}	biased e_{min}	biased e_{max}
SP	8	127 $(= 2^{8-1}-1)$	-126	127	1	254
DP	11	1023 $(= 2^{11-1}-1)$	-1022	1023	1	2046

-0	1	00000000	00000000000000000000000
$+0$	0	00000000	00000000000000000000000
$-\infty$	1	11111111	00000000000000000000000
$+\infty$	0	11111111	00000000000000000000000
NaN	0	11111111	00000000000000000000001 (for instance)

Not a Number (NaN) is the result of invalid operations such as $0/0$, $\sqrt{-1}$ or $0 \times \infty$

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

Double-Base Number Systems (DBNS) (1/2)

Source: L. Imbert

Redundant representation based the sum of powers of 2 AND 3:

$$x = \sum_{i=1}^{n} x_i 2^{a_i} 3^{b_i}, \text{ with } x_i \in \{-1, 1\},\ a_i, b_i \geq 0$$

Example: $127 = 108 + 16 + 3 = 72 + 54 + 1 = \ldots$

	1	2	4	8	16
1					1
3	1				
9					
27			1		

	1	2	4	8
1	1			
3				
9				1
27		1		

Double-Base Number Systems (DBNS) (2/2)

Smallest $x > 0$ with n DBNS terms in its decomposition:

n	unsigned	signed
2	5	5
3	23	105
4	431	(4985)
5	18,431	?
6	3,448,733	
7	1,441,896,119	
8	?	

DBNS is a very sparse and redundant representation

Example: 127 has 783 DBNS representations among which 6 are canonic: $127 = (108 + 18 + 1) = (108 + 16 + 3) = (96 + 27 + 4) = (72 + 54 + 1) = (64 + 54 + 9) = (64 + 36 + 27)$

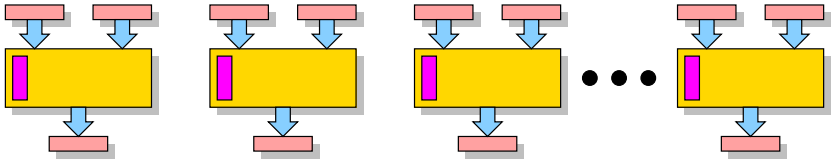
Residue Number System (RNS)

- Base $\mathcal{B} = (m_1, m_2, \ldots, m_k)$ of k relatively prime moduli
- Size of the base: k

$$A = \{a_1, a_2, \ldots, a_k\}, \quad \forall i \; a_i = A \bmod m_i$$

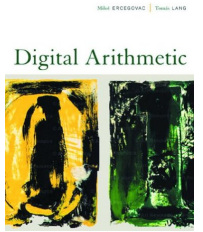
Operations:

$$A \pm B = (|a_1 \pm b_1|_{m_1}, \ldots, |a_k \pm b_k|_{m_k})$$
$$A \times B = (|a_1 \times b_1|_{m_1}, \ldots, |a_k \times b_k|_{m_k})$$



Reference

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



Part II

Circuit Design

Transistors

Logic Gates

Electrical Characteristics

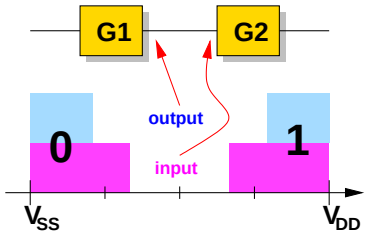
(Low) Power Consumption

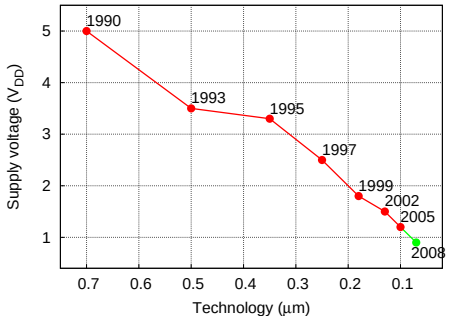
Logic Values: Representation

The logic values $\{0, 1\}$ are represented using voltages:

- $0 \iff$ reference voltage or ground (V_{SS}, ⏚)
- $1 \iff$ supply voltage ($V_{DD} > 0$ or △)

Due to the noise in the circuit (from many sources), the logic values must be represented using voltage intervals (noise margins): digital vs. analog



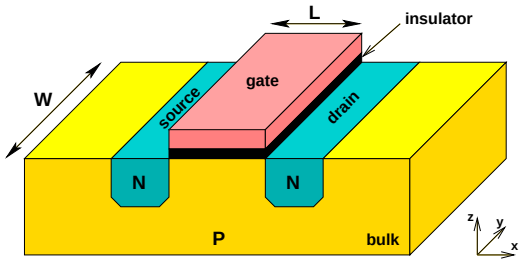


MOS Transistor: N and P transistors

MOS = *metal oxide semiconductor*

N transistors are made of:

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

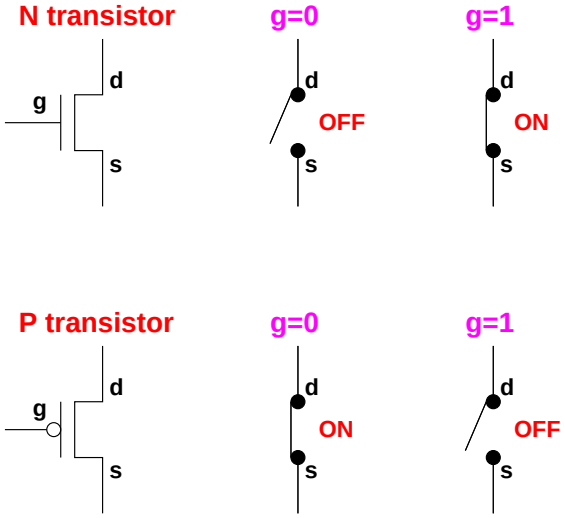


In N-type doping area, the majority carriers are electrons (holes in a P-type area)

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

MOS Transistor: Logic Model

Simple logic behavior (≈ switch)



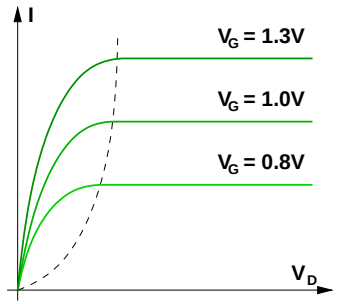
MOS Transistor: Simple Model

Current/Voltage (I/V) characteristic:

$$I = \begin{cases} 0 & V_G < V_T \\ \beta\left((V_G - V_T)V_D - \frac{{V_D}^2}{2}\right) & 0 < V_D < V_G - V_T \\ \frac{\beta}{2}(V_G - V_T)^2 & 0 < V_G - V_T < V_D \end{cases}$$

where

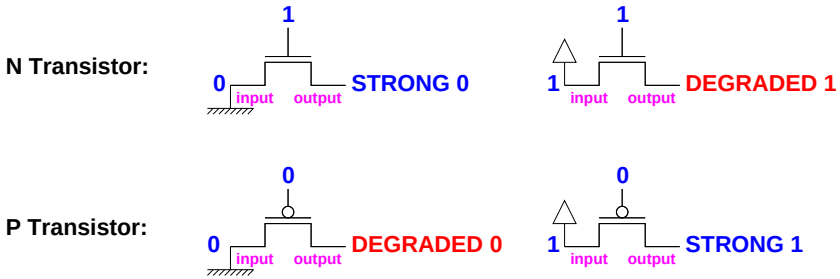
$$\beta = C_{\text{techno}} \times \frac{W}{L}$$



Threshold voltage: V_T

3 modes of operation: *cutoff*, *linear* and *saturation*

MOS Transistor: Imperfect Switch



	0	1
STRONG	close to V_{SS}	close to V_{DD}
DEGRADED	greater than V_{SS}	less than V_{DD}

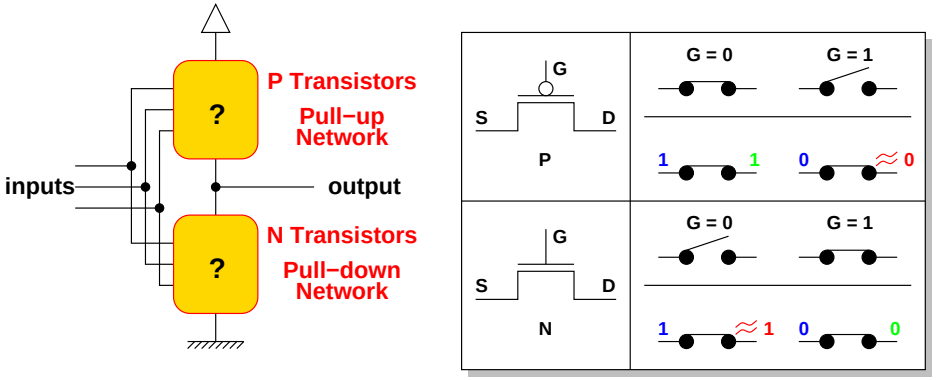
N transistor pull no higher than $V_{DD} - V_{T_N}$

P transistor pull no lower than $|V_{T_P}|$

CMOS Logic

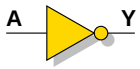
CMOS = *complementary MOS*

N and P transistors are only used for passing strong signals

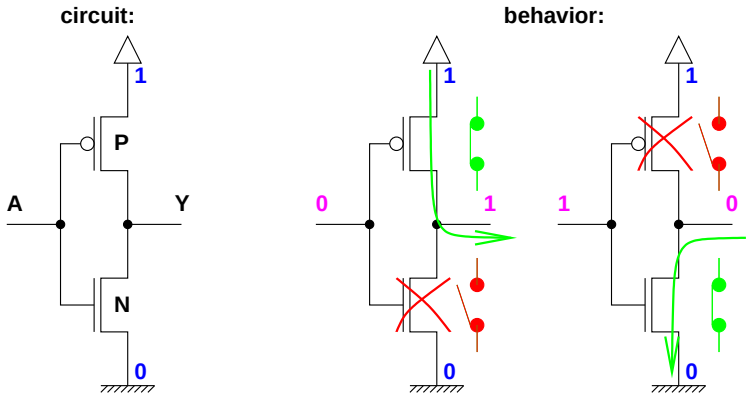


Logic Gate: Inverter

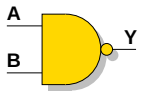
The simplest gate: only 2 transistors (1 N and 1 P)



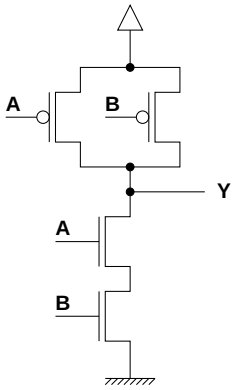
A	Y
0	1
1	0



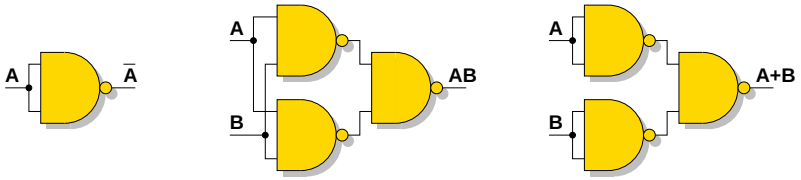
Logic Gate: NAND2 (2-input not–and)



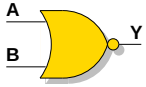
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



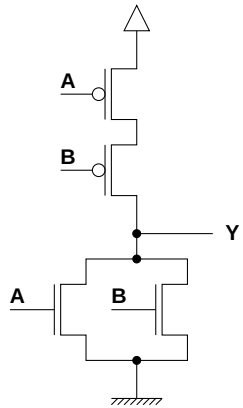
All logic functions can be built using only NAND gates:



Logic Gate: NOR2 (2-input not–or)

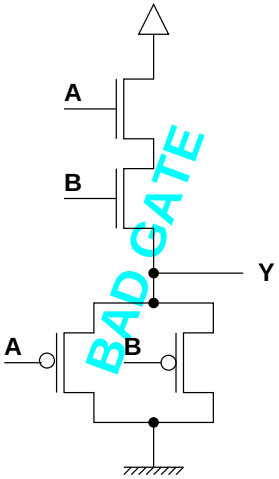


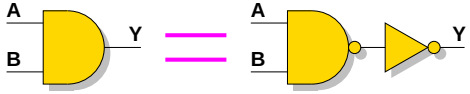
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



Logic Gate: AND2

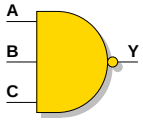
There is a very bad solution and a good one...



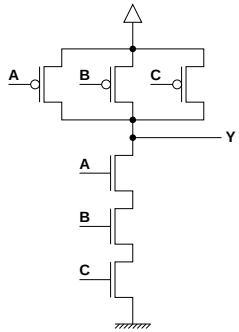


- the bad one (left side): some output levels are degraded
- the good one (right side): $AB = \overline{\overline{AB}}$ (6-transistor gate)

Logic Gate: NAND3 (3-input NAND)



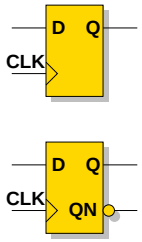
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



The number of transistors in series is limited (3 to 5)

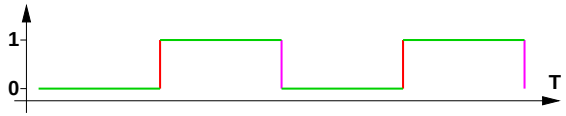
Memory Elements

There are many types of memory elements. Here, we will only focus on standard flip-flops

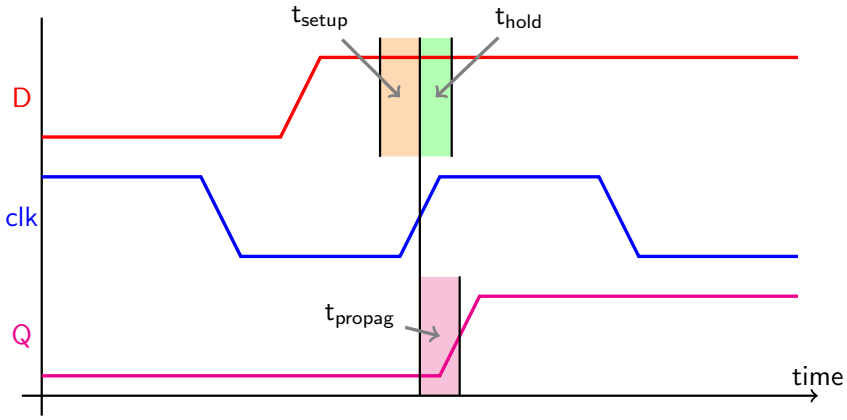


CLK	D	Q(t+1)	QN(t+1)
1	X	Q(t)	QN(t)
0	X	Q(t)	QN(t)
↑	0	0	1
↑	1	1	0

Remark:
↑ is the rising clock edge



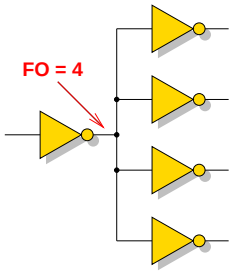
Setup, Hold and Propagation Delays

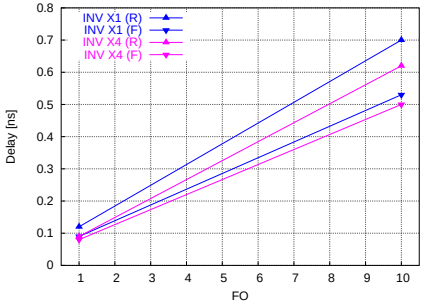


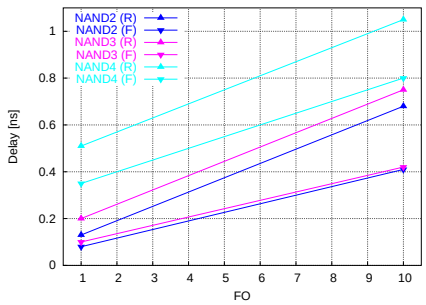
- **setup** delay (t_{setup}): data should be held steady *before* clock edge
- **hold** delay (t_{hold}): data should be held steady *after* clock edge
- **propagation** delay (t_{propag}): propagation time from D to Q

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)

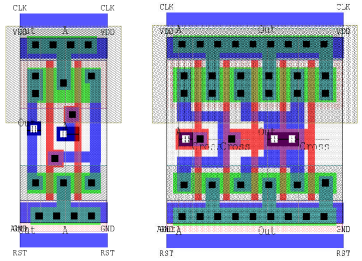


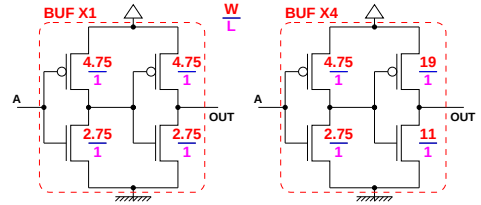




Signal regeneration

Buffers are used to regenerated the signal levels ($f(x) = x$)

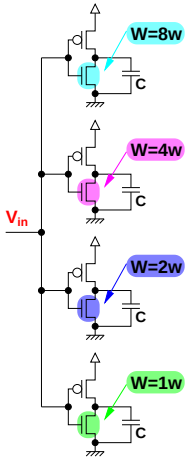


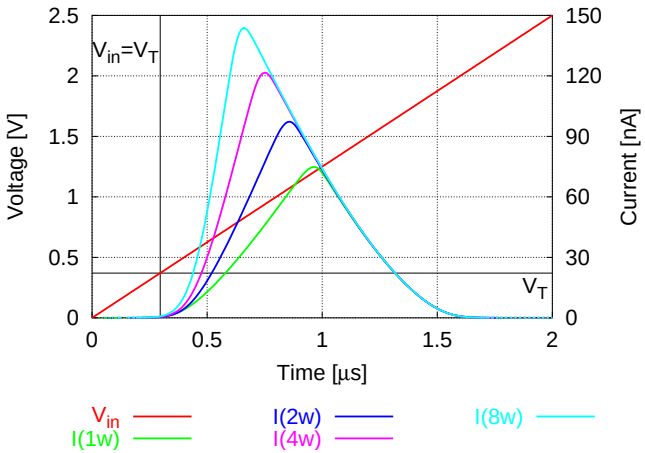


	BUF X1	BUF X4
size (h×l) [λ]	53 × 25	53 × 50
capacitance [fF]	5.89	5.89
$T_{0\to1}$	$11 + 439 \times C_{out}$	$17 + 132 \times C_{out}$
$T_{1\to0}$	$12 + 318 \times C_{out}$	$21 + 137 \times C_{out}$

Transistor Sizing

Current behavior (higher $I \Longrightarrow$ faster gate): $I = \beta \times f(V_G, V_D, V_T)$
where the transistor gain is: $\beta = C_{techno} \times \frac{W}{L}$





Fast Circuit Design: Basic Ideas

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

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

Transistor Sizing: Gate Level Solution

Full custom: transistor sizing for all elements $\rightsquigarrow$ very complex

Standard cells: use cells from a library (several gates for one function with different output transistor sizes)

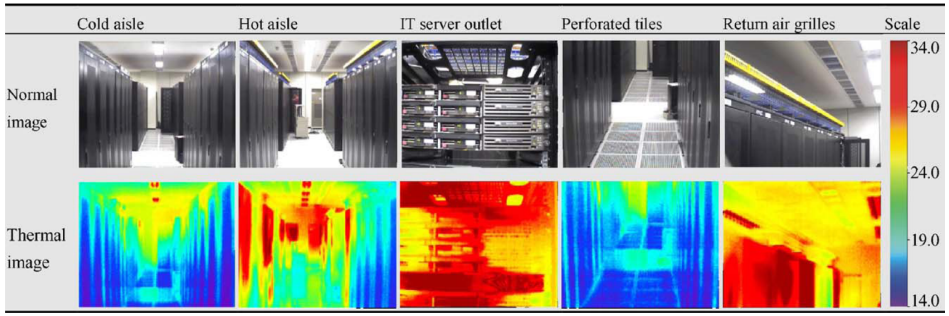
gate	drive	size [μm]		delay [ns]		power [μW/MHz]
		H	W	$\uparrow$	$\downarrow$	
INVX1	1X	5.04	1.32	0.0253	0.0146	0.0117
INVX2	2X	5.04	1.98	0.0228	0.0140	0.0218
INVX4	4X	5.04	2.64	0.0206	0.0125	0.0394
INVX8	8X	5.04	3.96	0.0198	0.0125	0.0773

Estimated or Measured Characteristics

- Time
  - delay or clock period [s] [ns] [ps] [FO4] [#NAND2]
  - latency [#cycles]
- Speed
  - clock frequency [Hz] [MHz] [GHz]
  - throughput [evt/s] [MIPS] [MFLOPS]
- Area
  - real area [mm^2] [μm^2] [λ^2]
  - arbitrary unit [#transistor] [#NAND2]
- Power consumption and energy
  - power [W] [μW/MHz]
  - energy [J] [A/h]
  - battery duration
- Compound: $A.T$, $A.T^2$, MIPS/W, MIPS^2/W, MIPS^3/W, ...

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

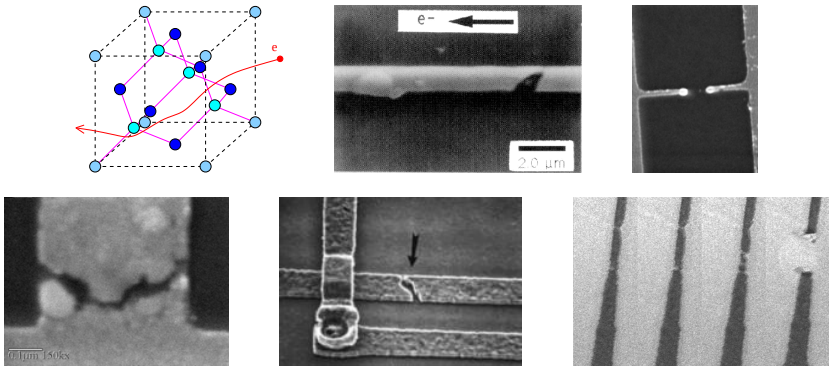
Electromigration

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

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

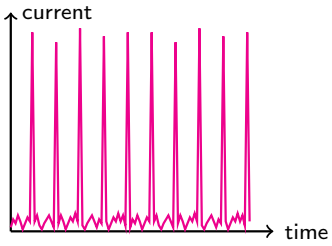
A section, J current density, $n \approx 2$ scale factor (cst), E_a activation energy (cst for a material), k Boltzmann's constant, T temperature

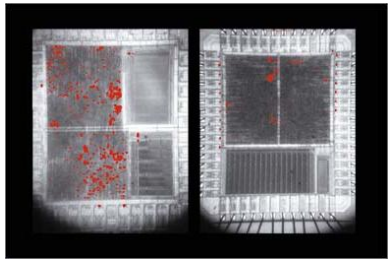
- decreases IC reliability (permanent and intermittent failures)



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

Power Consumption: Basic Definitions

Instantaneous power:

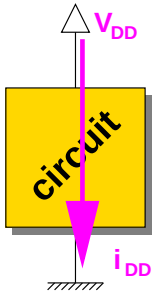
$$P(t) = i_{DD}(t)\, V_{\mathrm{DD}}$$

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

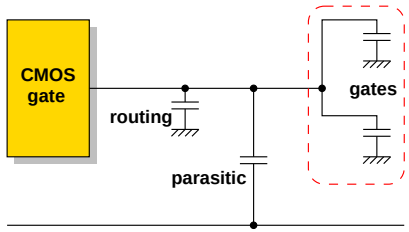
Power dissipation in CMOS circuits comes from 2 main components:

- 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

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

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)

Simple Power Consumption Model

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

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

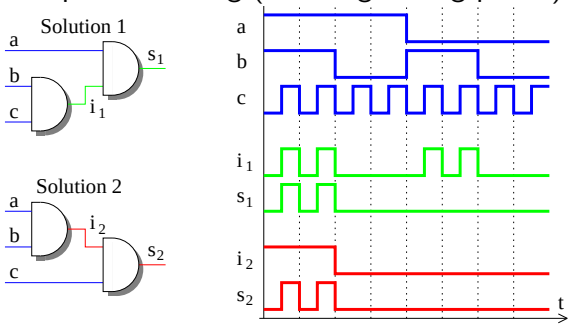
where

- 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

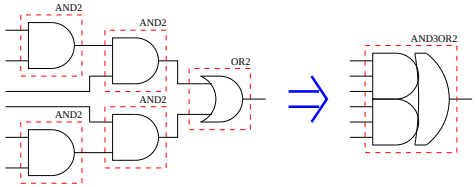
Remark: the gate delay is $d = \gamma \times \frac{C \times V_{DD}}{(V_{DD} - V_T)^2} \approx \frac{1}{V_{DD}}$

Power Reduction at Gate Level

- gate and/or input reordering (reduce glitching power):



- use complex gates (reduce internal capacitances and area):



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

Predicting the Future: Roadmap

International Technology Roadmap for Semiconductors (ITRS)
`http://public.itrs.net/`

Semiconductor Industry Association (SIA)

Source: CMOS VLSI Design: A Circuits and Systems Perspective. N. Weste and D. Harris, 3rd ed., 2004, Addison Wesley. (extract from ITRS 2002 edition)

year	2001	2004	2007	2010	2013	2016
size [nm]	130	90	65	45	32	22
V_{DD}[V]	1.1–1.2	1–1.2	0.7–1.1	0.6–1.0	0.5–0.9	0.4–0.9
MT/die	193	385	773	1564	3092	6184
Wire levels	8–10	9–13	10–14	10–14	11–15	11–15
I/O signals	1024	1024	1024	1280	1408	1472
Frequency [MHz]	1684	3990	6739	11511	19348	28751
FO4 delays/cycle	13.7	8.4	6.8	5.8	4.8	4.7
Max. power [W]	130	160	190	218	251	288
DRAM size [Gb]	0.5	1	4	8	32	64

ITRS 2009 Edition

High-performance processor and ASIC expected features:

year	2009	2010	2011	2012	2013	2014	2015	2020
size [nm]	29	27	24	22	20	18	17	10.7
HP V_{DD}[V]	1.0	0.97	0.93	0.9	0.87	0.84	0.81	0.68
LP V_{DD}[V]	0.95	0.95	0.85	0.85	0.80	0.80	0.75	0.65
MT/chip	2 212	2 212	4 424	4 424	8 848	8 848	8 848	35 391
ML	12	12	12	12	13	13	13	14
# pckg pins	4 620	4 851	5 094	5 348	5 616	5 896	6 191	7 902
Freq. [MHz]	5 454	5 875	6 329	6 817	7 344	7 911	8 522	12 361
MaxPw [W]	143	146	161	158	149	152	143	130
DRAM [Gb]	2.15	2.15	4.29	4.29	4.29	8.59	8.59	34.36

size: physical gate length, HP: high performance, LP: low power, ML: metal layers, Freq.: local clock, MaxPw: maximum power with heat-sink

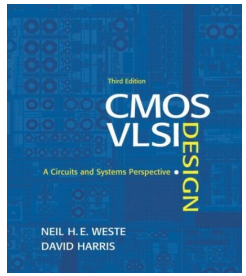
Source: `http://public.itrs.net/Links/2009ITRS/Home2009.htm`

References

CMOS VLSI Design
A Circuits and Systems Perspective

N. Weste and D. Harris

3rd Edition, 2004, Addison Wesley
ISBN: 0–321–14901–7



Micro et nanoélectronique
Bases, Composants, Circuits

H. Fanet

2006, Dunod
ISBN: 2–10–049141–5

Part III

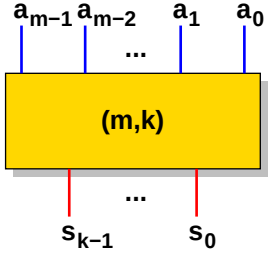
Arithmetic Operators

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)

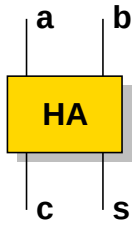
$$\sum_{i=0}^{m-1} a_i = \sum_{j=0}^{k-1} s_j 2^j$$



Standard counters:

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

HA Cell



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

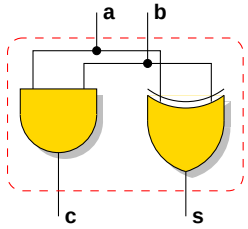
Arithmetic equation:

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

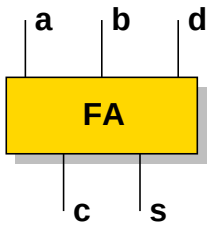
Logic equation:

$$\begin{aligned} s &= a \oplus b \\ c &= ab \end{aligned}$$

Gate-level implementation of the HA:



FA Cell



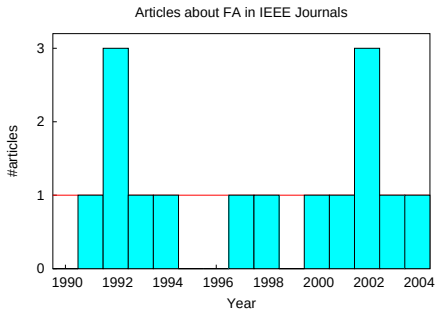
a	b	d	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Arithmetic equation:

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

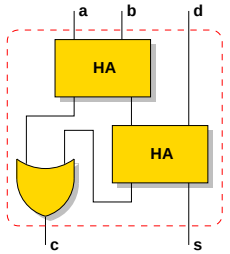
Logic equation:

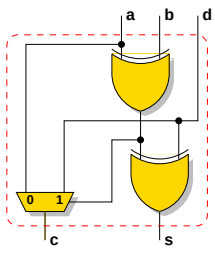
$$\begin{aligned} s &= a \oplus b \oplus d \\ c &= ab + ad + bd \end{aligned}$$

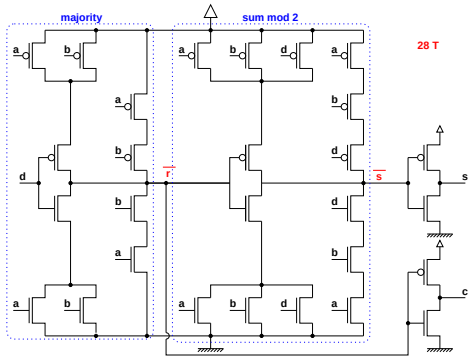


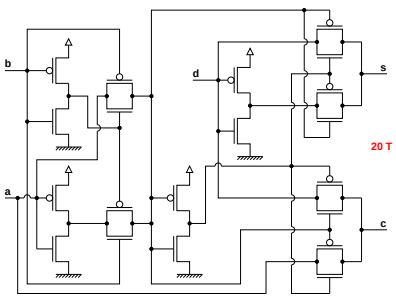
There many implementations of the FA cell

FA Implementations



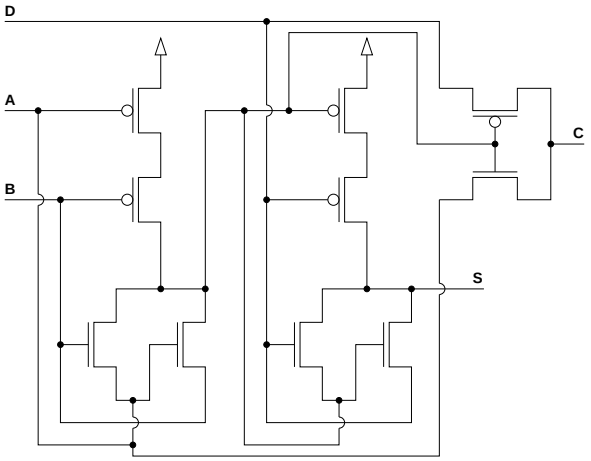






Optimized FA Cell

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

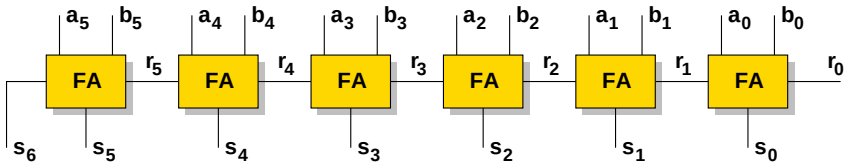
[1] H. T. Bui, Y. Wang et Y. Jiang. *Design and analysis of low-power 10-transistor full adders using novel XOR–XNOR gates*. IEEE Trans. CaS, jan. 2002.

Main Gates Used in Computer Arithmetic Operators

gate	area	input capa.	delay [ns] ↗		delay [ns] ↘		power	path
	[μm^2]	[pF]	C_L	$10C_L$	C_L	$10C_L$	[μW/MHz]	
HA 1X	164	0.015	0.26	0.81	0.33	0.80	0.93	A → C
			0.35	0.90	0.35	0.79		A → S
		0.015	0.26	0.82	0.35	0.82		B → C
			0.28	0.83	0.35	0.79		B → S
FA 1X	364	0.030	0.39	0.96	0.45	0.95	1.28	A → C
			0.44	1.01	0.59	1.08		A → S
		0.028	0.40	0.97	0.42	0.94		B → C
			0.44	1.01	0.59	1.08		B → S
		0.025	0.34	0.90	0.38	0.89		D → C
			0.47	1.03	0.54	1.01		D → S
INV 1X	36	0.007	0.12	0.70	0.09	0.53	0.23	
NAND2 1X	55	0.007	0.13	0.68	0.08	0.42	0.28	
AND2 1X	73	0.004	0.22	0.77	0.27	0.72	0.42	
NAND3 1X	91	0.007	0.20	0.75	0.10	0.41	0.43	
XOR2 1X	146	0.019	0.28	0.79	0.14	0.49	0.74	
MUX 1X	127	0.006	0.27	0.82	0.31	0.77	0.50	A,B → S
		0.012	0.24	0.79	0.36	0.82		C → S

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)

Subtraction

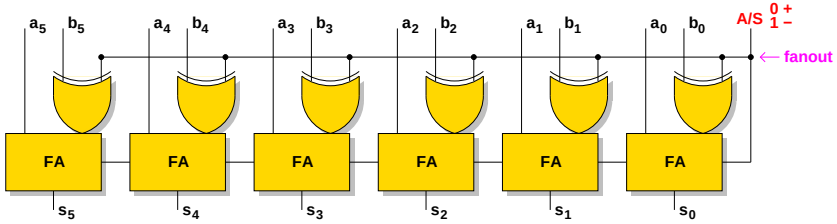
Using 2's complement representation:

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

Implementation:

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

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



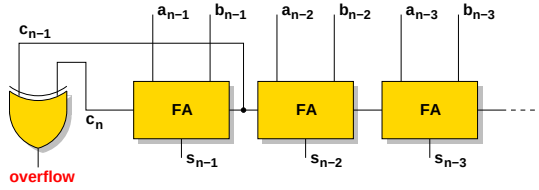
Overflow

- n bits $\pm$ n bits $\longrightarrow$ $n+1$ bits then no possible overflow
- n bits $\pm$ n bits $\longrightarrow$ n bits then possible overflow (need test)

Overflow detection in 2's complement:

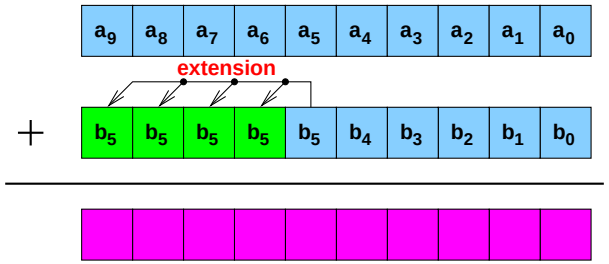
- Operands with opposite signs: no overflow
- Operands with same sign: there is an overflow iff the sign of the result is different from the sign of the operands

a_{n-1}	b_{n-1}	c_{n-1}	c_n	s_{n-1}
+	+	0	0	+
+	+	1	0	−
+	−	0	0	−
+	−	1	1	+
−	+	0	0	−
−	+	1	1	+
−	−	0	1	+
−	−	1	1	−



Sign Extension

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



Warning:

- fanout
- order in case of multiple additions

Carry Chains in a CRA

The computation time depends on the longest carry chain:

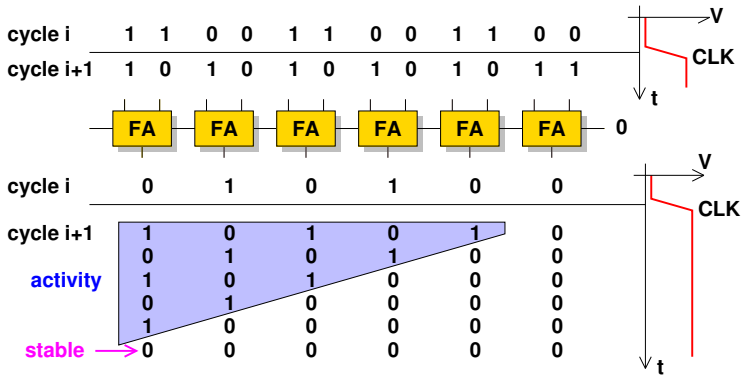
- worst case $\rightsquigarrow n$ propagations (ex: $000000 + 111111$)
- best case $\rightsquigarrow 0$ propagation (ex: $111111 + 111111$)
- average case?

A. Burks, H. Goldstine et J. Von Neumann "*preliminary discussion of the logical design of an electronic instrument*", 1946: study of the average computation time of the CRA

If $L(n)$ is the average length of the longest carry chain in a n-bit CRA (assuming uniform distribution and 1/2 probability), then

$$L(n) \leq \log_2 n$$

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

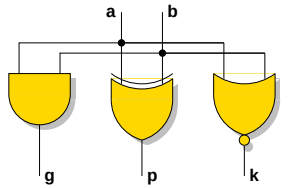
Carry Propagation and Generation

a	b	c	s	function
0	0	0	d	generate
0	1	d	$\overline{d}$	propagate
1	0	d	$\overline{d}$	propagate
1	1	1	d	generate

a, b	c	s	function
$a = b$	a	d	generate
$a \neq b$	d	$\overline{d}$	propagate

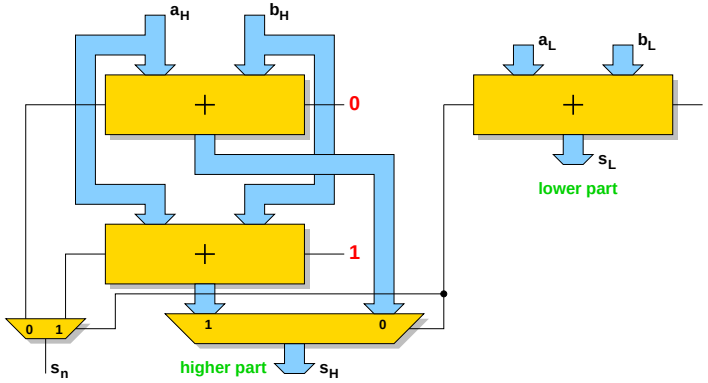
Sometimes kill is used for generating 0 for the output carry ($a = b = 0$)

$$\begin{aligned} p &= a \oplus b \\ g &= ab \\ k &= \overline{a}\overline{b} = \overline{a + b} \end{aligned}$$



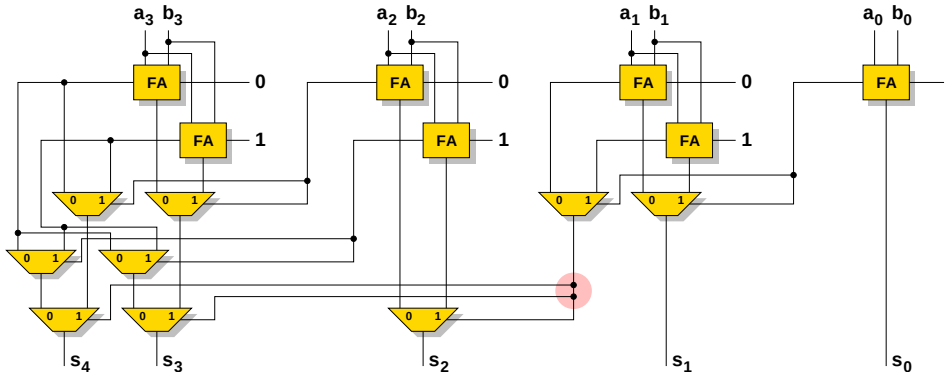
Carry-Select Adder

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



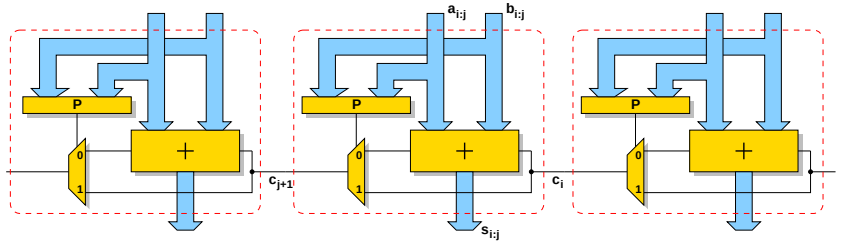
Recursive version $\longrightarrow$ $O(\log n)$ delay but...

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?
- delay with non-uniform block size?

Carry Skip Adder: Uniform Block Size

- n-bit CSkA with k-bit blocks (assuming n/k is an integer)
- τ_1 propagation delay on 1 bit, τ_2 skip delay over 1 block ($\tau_2 < \tau_1$)

Worst case: propagation from rank 1 to rank $n-2$ (gen. at ranks 0 and $n-1$)

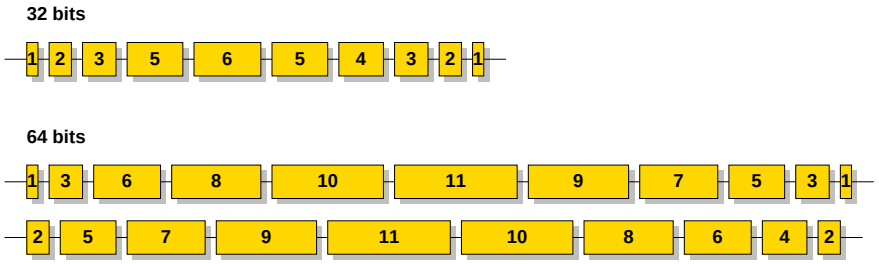
$$\begin{aligned} T(k) &= 2(k-1)\tau_1 + (\frac{n}{k} - 2)\tau_2 \\ T'(k) &= 2\tau_1 - \frac{n\tau_2}{k^2} \\ T''(k) &= \frac{2n\tau_2}{k^3} \end{aligned}$$

$$k_{opt} = \sqrt{\frac{n\tau_2}{2\tau_1}} \quad \longrightarrow \quad T(k_{opt}) = O(\sqrt{n})$$

Carry Skip Adder: Non-Uniform Block Size

Very complex problem, many possible heuristics

Examples from *A Simple Strategy for Optimized Design of One-Level Carry-Skip Adders*. M. Alioto et G. Palumbo. IEEE Trans. CaS I, jan. 03.



Carry Lookahead 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 $i-3$
  $\hookrightarrow p_{i-1} = p_{i-2} = g_{i-3} = 1$

⋮

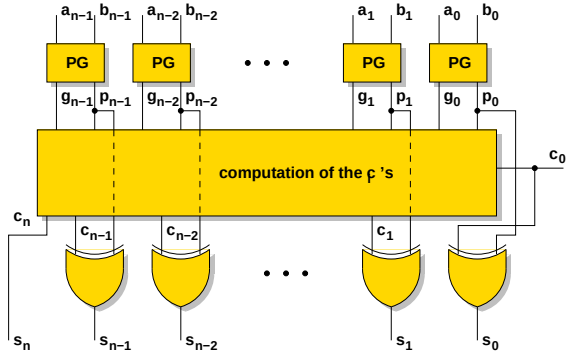
- ranks $i-1$ to 0 propagate the adder input carry c_0 (set to 1)
  $\hookrightarrow p_{i-1} = p_{i-2} = \ldots = p_1 = p_0 = c_0 = 1$

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

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

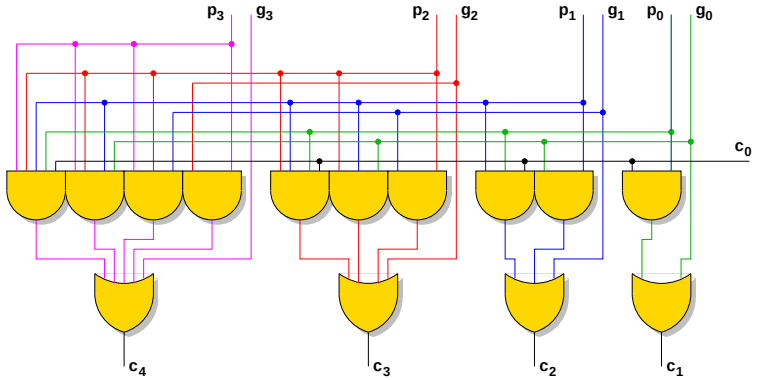
CLA architecture: parallel evaluation of

- (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 Lookahead Adder: 4-Bit Example

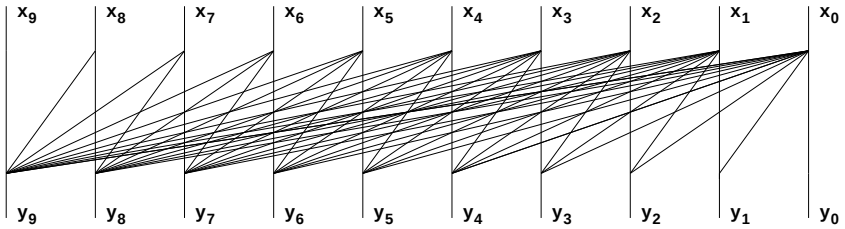
$$c_1 = g_0 + p_0 c_0$$
$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$
$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$
$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$



Parallel-Prefix 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

1. Computation of the generate and propagate signals for all inputs:

$$g_i = a_i b_i \quad \text{and} \quad p_i = a_i \oplus b_i \qquad \forall i = 0, 1, \ldots, n-1$$

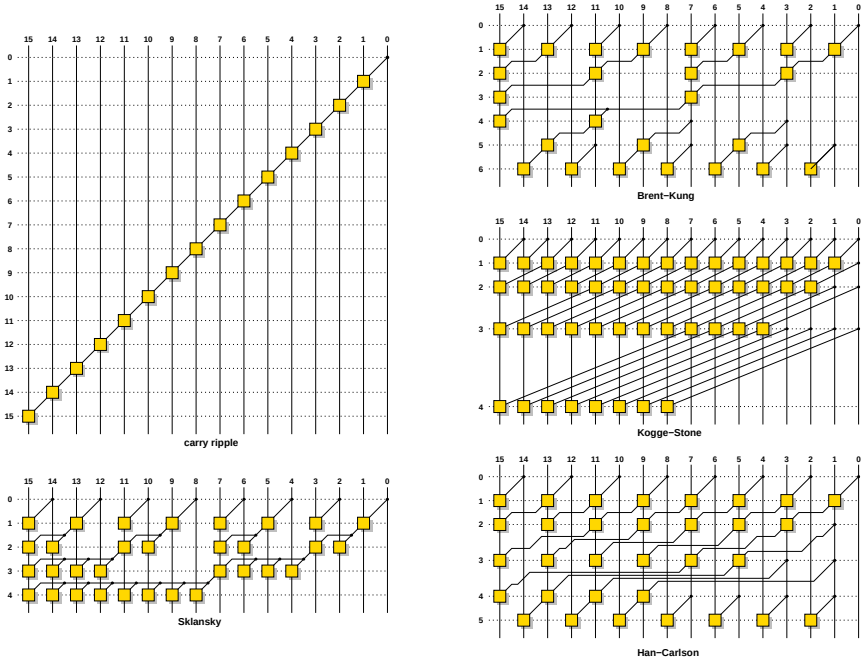
2. Computation of the carries c_i using a m-level parallel-prefix architecture:

$$\begin{aligned} (G^0_{i:i}, P^0_{i:i}) &= (g_i, p_i) \\ (G^l_{i:k}, P^l_{i:k}) &= (G^{l-1}_{i:j}, P^{l-1}_{i:j}) \square (G^{l-1}_{j:k}, P^{l-1}_{j:k}) \qquad k \le j \le i \text{ and } l = 1, \ldots, m \\ &= (G^{l-1}_{i:j} + P^{l-1}_{i:j} G^{l-1}_{j:k}, P^{l-1}_{i:j} P^{l-1}_{j:k}) \\ c_{i+1} &= G^m_{i:0} + P^m_{i:0} c_0 \qquad \forall i = 0, 1, \ldots, n-1 \end{aligned}$$

3. Computation of the sums:

$$s_i = p_i \oplus c_i \qquad \forall i = 0, 1, \ldots, n-1$$

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

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)

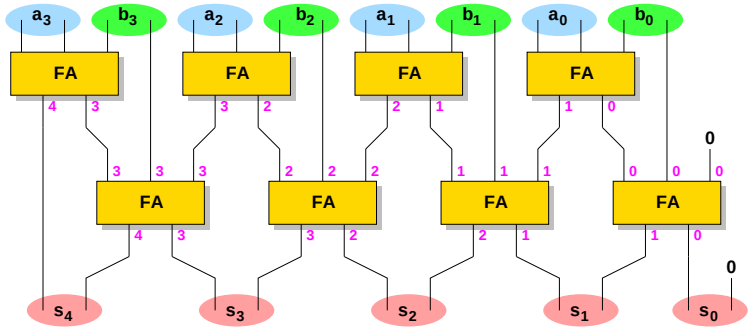
Such a representation is said redundant

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 $a_{i,s} \in \{0,1\}$

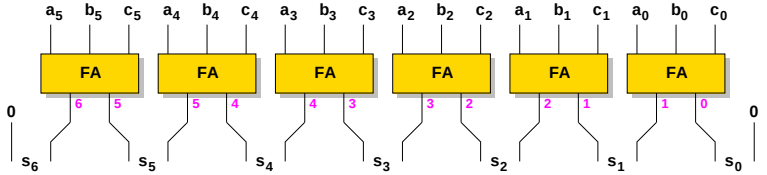
$$A = \sum_{i=0}^{n-1} a_i 2^i = \sum_{i=0}^{n-1} (a_{i,c} + a_{i,s}) 2^i$$



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

Carry-Save Trees

Example with 3 inputs: A, B and C



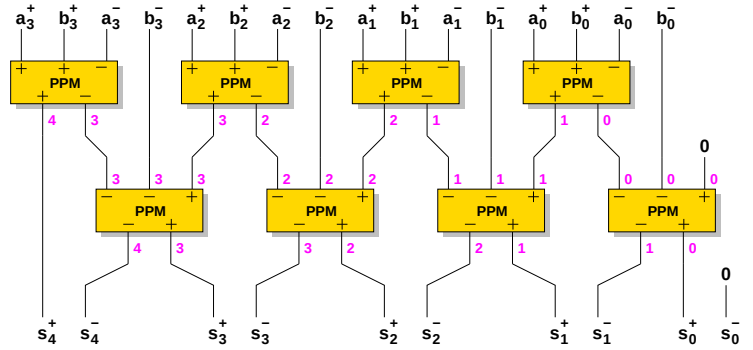
Carry-save reduction tree: $n(h)$ non-redundant inputs can be reduced by a h-level carry-save tree where $n(h) = \lfloor 3n(h-1)/2 \rfloor$ and $n(0) = 2$

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 ($T = 0(1)$)

PPM Cell

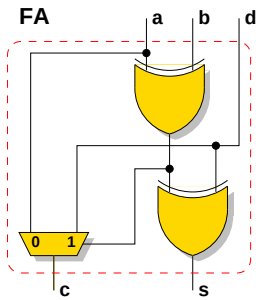
a^+	b^+	d^-	c^+	s^-
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

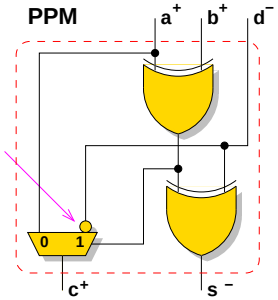
Arithmetic equation:

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

Logic equation:

$$\begin{aligned} s &= a^+ \oplus b^+ \oplus d^- \\ c &= a^+ b^+ + a^+ \overline{d^-} + b^+ \overline{d^-} \end{aligned}$$





Shift-And-Add Multiplication

The product $P = A \times B$ can be performed using additions and shifts with the following (parallel-serial) algorithm:

```
1  P ⟵ 0
2  for i from 0 to n−1 do
3      P ⟵ P + a_i B 2^i
```

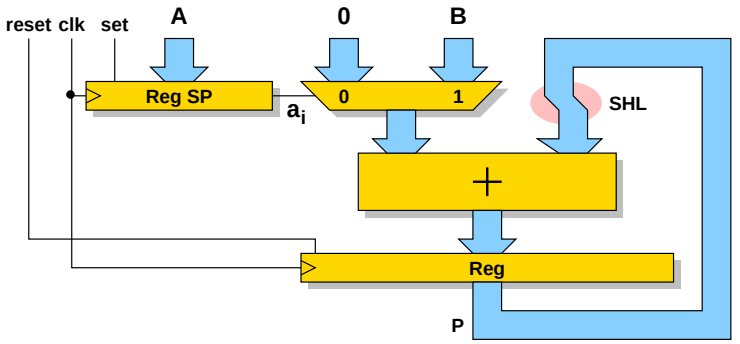
Remark: This algorithm requires a shifter operator (variable shift amount)

Simplification (constant shift):

```
1  P ⟵ 0
2  for i from 0 to n−1 do
3      P ⟵ (P + a_i B) × 2^{−1}
4  P ⟵ P 2^n
```

Operation on line 4 is virtual

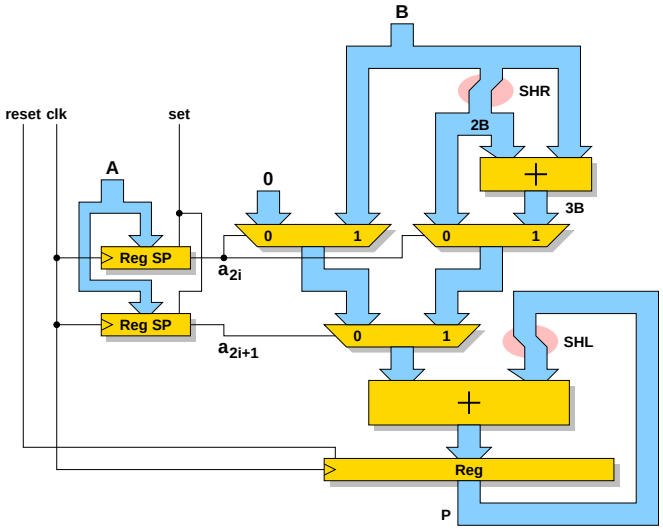
Shift-And-Add Multiplication: Implementation



	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

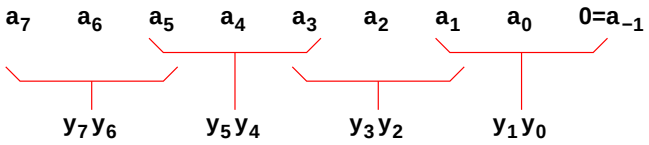
But, in some cases, this basic method leads to more 1 (or $\overline{1}$)!

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

$\Longrightarrow$ modified Booth's recoding

Modified Booth's Recoding

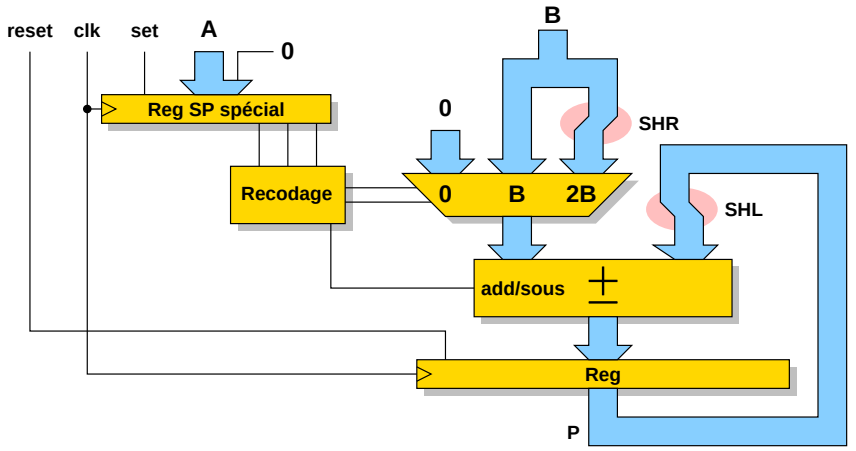
Idea: do not recode isolated 1 but only strings of 1



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	$+0$

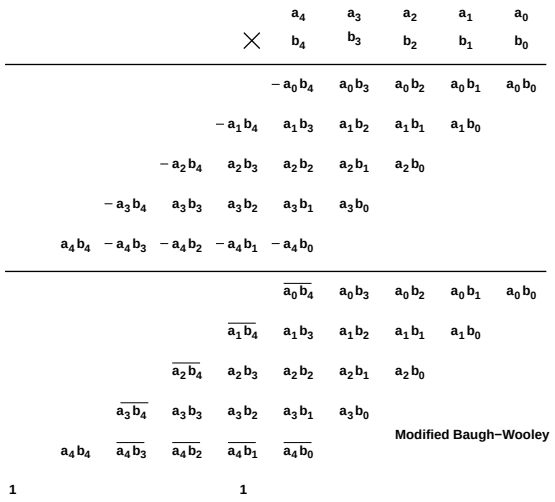
Improvement: leads to a n-product with $\lfloor n/2 \rfloor + 1$ additions and shifts at most

Modified Booth Multiplier



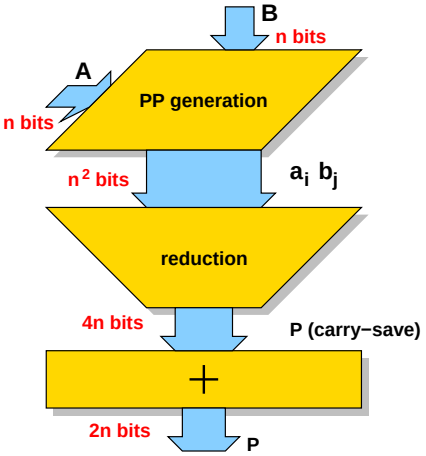
2's Complement Product

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

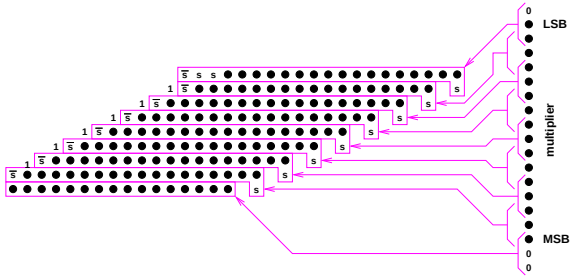
During the computation of the product $P = A \times B$ where A and B are n-bit integers, there are n^2 AND cells

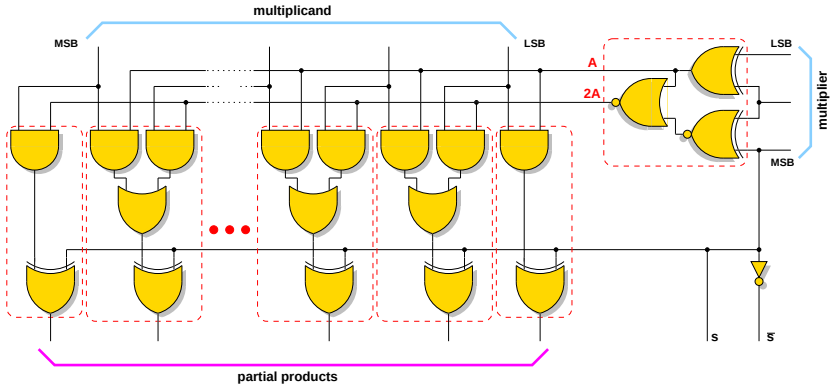
Fanout problem: a_i is used in all $a_i b_j$ for $j \in \{0, 1, 2, \ldots, n-1\}$ (solution bufferization during the recoding)

Modified Booth recoding is used to diminish the number of partial products

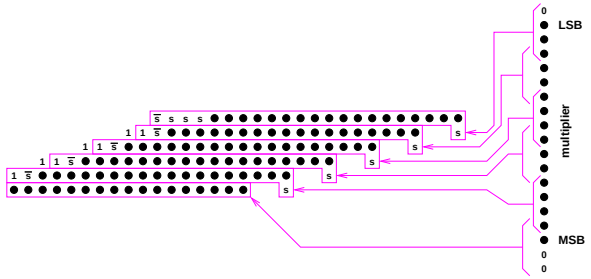
- Booth-2 : recoding of the a_is in radix 4 with digits in $\{0, \pm 1, \pm 2\}$.
- Booth-3 : recoding of the a_is in radix 8 with digits in $\{0, \pm 1, \pm 2, \pm 3, \pm 4\}$.

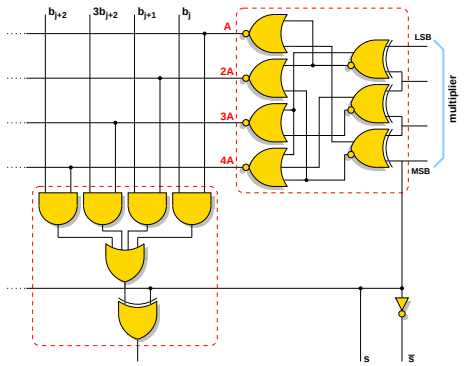
Partial Product Generation: Booth-2





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*

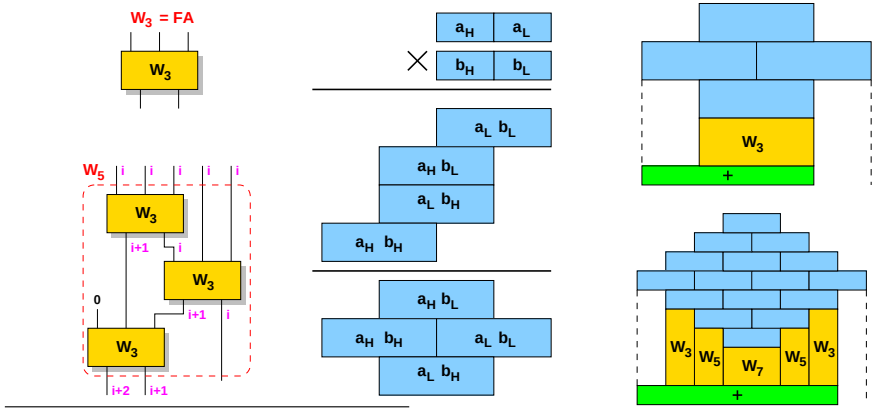
Several reduction trees can be used:

- 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)

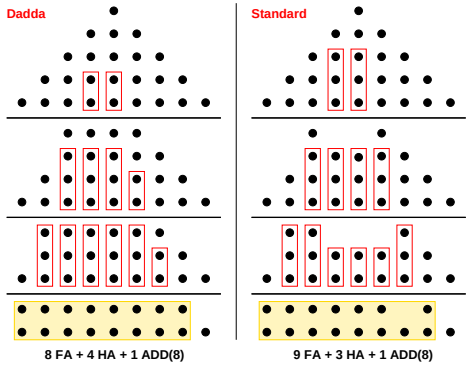


[2] C.S. Wallace. *A suggestion for a fast multiplier*. IEEE Transactions on Computers, Feb. 1964.

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

Automatic Generation Algorithms for Reduction Trees

There are generalizations and improvements to Dadda's method using different kinds of counters and delay models (depending on the implementation target)

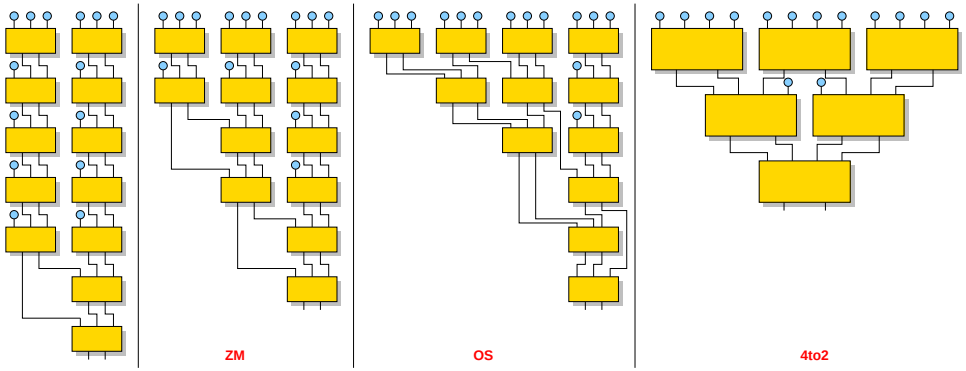
Example: The TDM[3] method leads to 30% faster and 15% smaller circuits compared to a solution based on "4 to 2" cells

The automatic generation of optimized multipliers is very complex problem: speed, area, regular layout, activity, specific cell requirements...

[3] V. Oklobdzija, D. Villeger and S. Liu. *A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach*. IEEE Transactions on Computers, mars 1996.

Placement and Routing Problems at the Gate Level

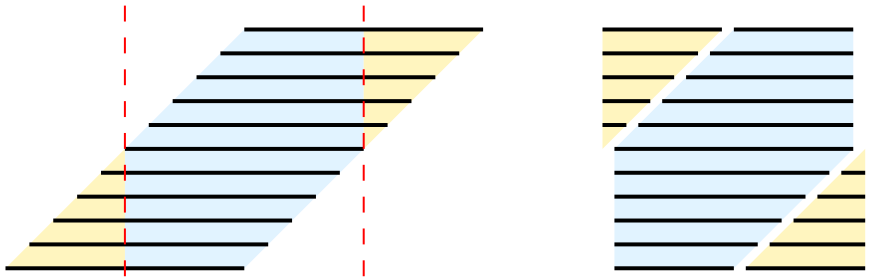
What is the best topology?



Example: 14-bit reduction

Layout Problem

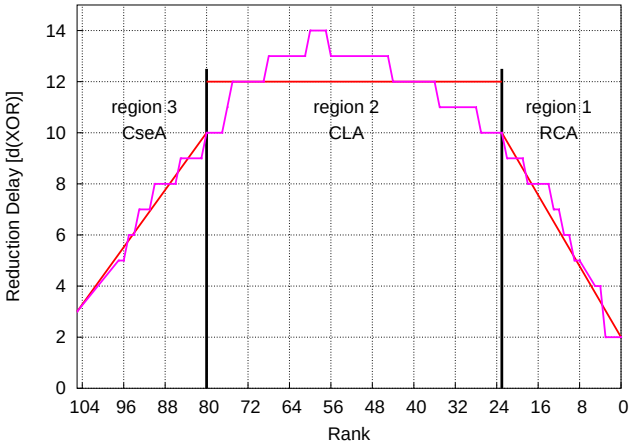
The best circuits have a square shape



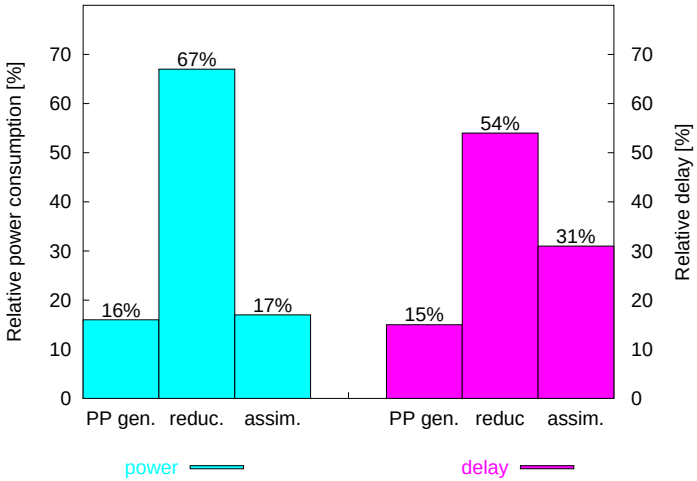
Final Addition in Multipliers

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



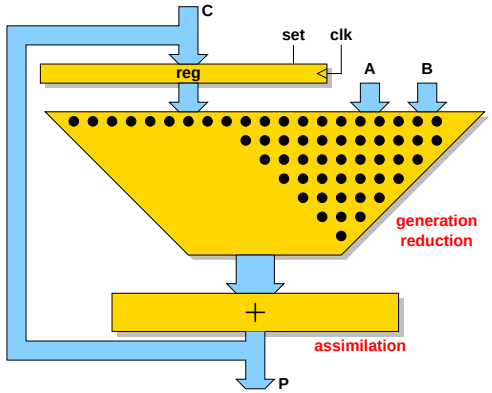
Power Consumption in Fast Multipliers



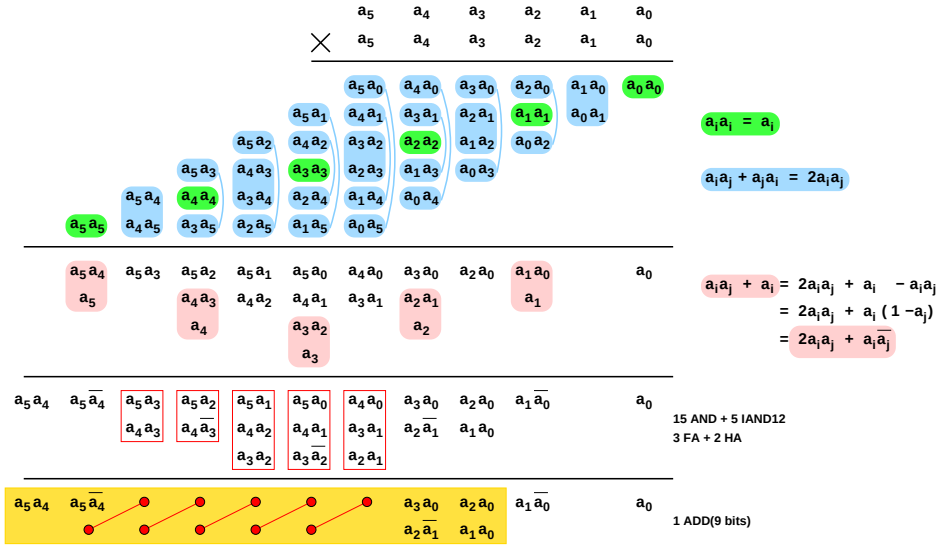
- 30% to 70% of redundant transitions (useless)
- place and route steps based on the internal arrival time
- add a pipeline stage

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
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%

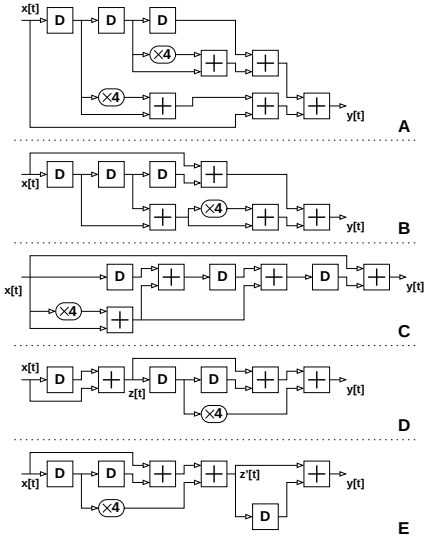
operator	init.	[1]	[2]	our
8×8 Had.	56	24	—	24
(16, 11) R.-M.	61	43	31	31
(15, 7) BCH	72	48	47	44
(24, 12, 8) Golay	76	—	47	45

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])`.

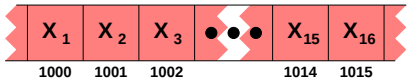
FIR $(1, 5, 5, 1)$



Specific Addressing Modes in Early DSPs

Loop indexes and pointers computation may require many cycles

```
1  a ⟵ 0
2  for i from 1 to 16 do
3      a ⟵ a + X_i
```

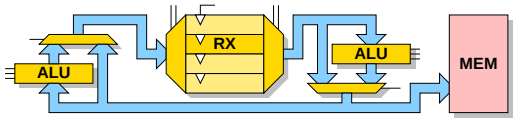



```
1      CLR A   // accu ← 0
2      LD RX, 1000
3  .L: ADD A, A, X(RX)
4      INC RX
5      BNE RX, 1016, .L
```

Specific addressing modes (and hardware resources):

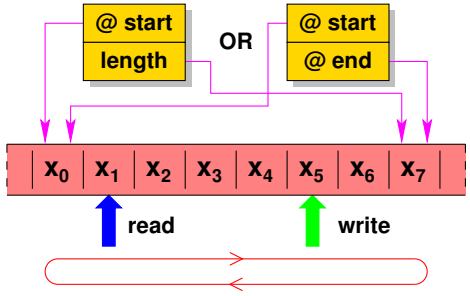
- with pre-increment or post-increment
- with pre-decrement or post-decrement

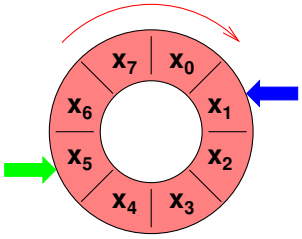
```
3  .L: ADD A, A, X(RX)+
4      BNE RX, 1016, .L
```



Circular Addressing Mode in DSPs

Idea: only the n last values are kept in the memory (emulation of a FIFO but without explicit memory management)

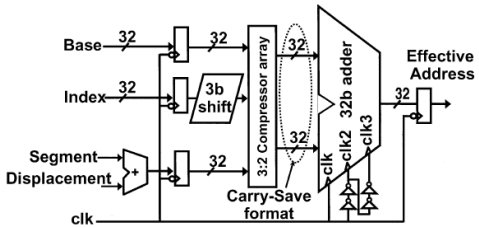


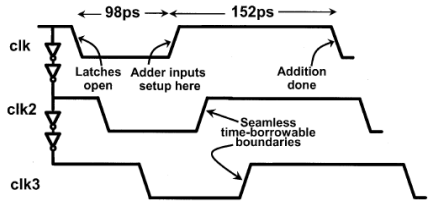


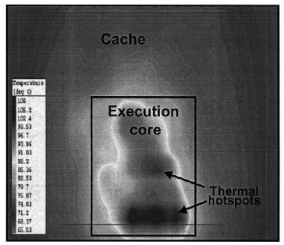
- Version with (start address and length) [4]: TMS320C3x or TMS320C4x
- Version (start and end addresses): TMS320C5x or DSP16xx

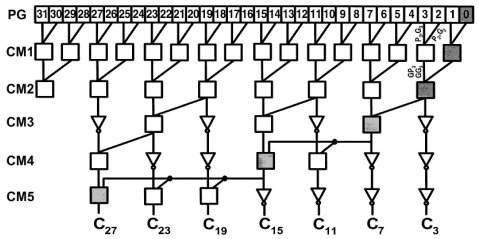
[4] With some restrictions on the start address

AGU Example #1





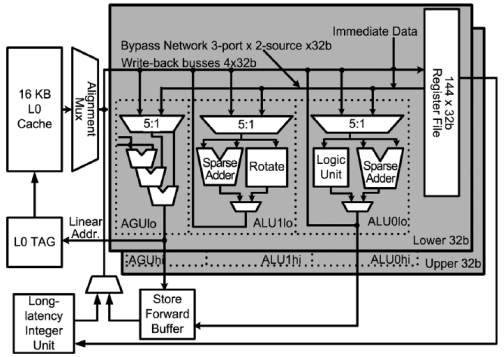


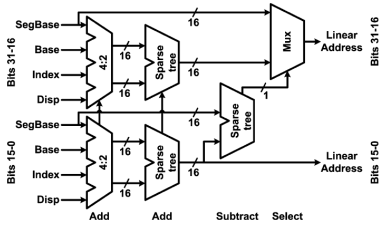


effective address = segment + displacement + base + index × scale

S. Mathew, *et al.*. A 4-GHz 130-nm Address Generation Unit With 32-bit Sparse-Tree Adder Core. IEEE Journal of Solid-State Circuits, vol. 38, n. 5, pp 689–695, May 2005

AGU Example #2





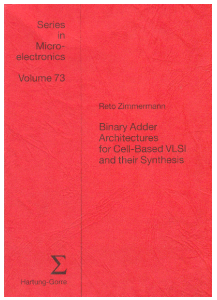
S. B. Wijeratne, *et al.* A 9-GHz 65-nm Intel Pentium 4 Processor Integer Execution Unit, IEEE Journal of Solid-State Circuits, vol. 42, n. 1, pp. 26–37, Jan. 2007.

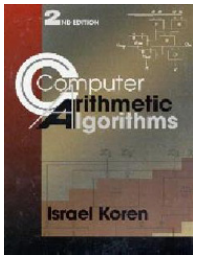
References

Binary Adder Architectures for Cell-Based VLSI and their Synthesis

Reto Zimmermann

1998
Hartung–Gorre
ISBN: 3–89649–289–6





Computer Arithmetic Algorithms

Israel Koren

2002 (2nd edition)
A. K. Peters
ISBN: 1-56881-160-8

The end, some questions ?

Contact:

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

Thank you