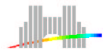


Algorithms and Number Systems for Hardware Computer Arithmetic

Arnaud Tisserand
Arénaire INRIA LIP

ISSAC 2005, Tutorial
Beijing, China



- 1 A “short” introduction to **digital integrated circuits**

Short break

- 2 **Hardware computer arithmetic** basics

Short break

- 3 **Examples** in cryptography

Part 1

A Short Introduction to Digital Integrated Circuits

Contents

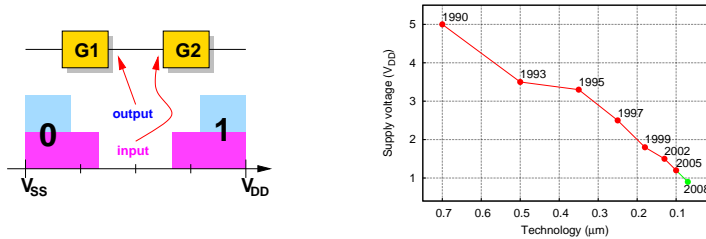
- Logic values: *representation, standard levels*
- Basic elements: *wires, capacitors, transistors, logic gates, memory elements*
- High-speed design: *limits, fanout, transistor sizing*
- Hardware complexity: *definitions, examples*
- Power consumption: *definitions, low-power design*
- Examples: *gate libraries*
- New technologies: *roadmap*
- Bibliography

Logic Values: Representation

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

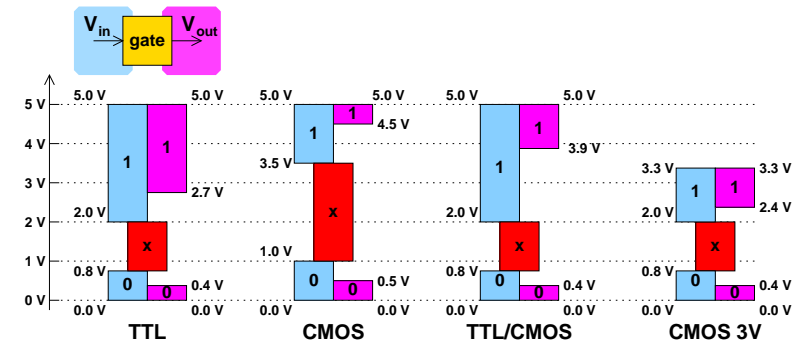
- 0 \iff reference voltage or **ground** (V_{SS} , $\underline{\quad}$)
- 1 \iff **supply voltage** ($V_{DD} > 0$ or $\hat{\quad}$)

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



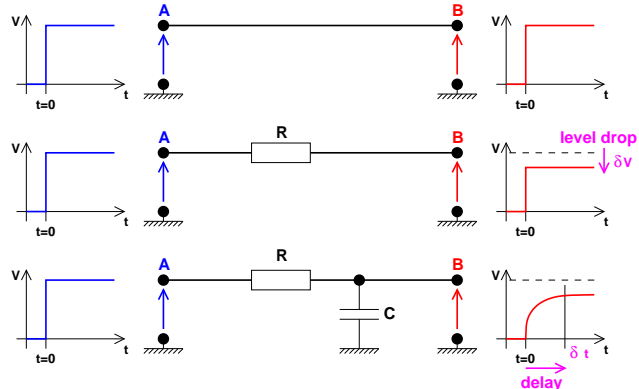
Logic Values: Standard Levels

Examples of standard levels used for integrated circuit interconnection:



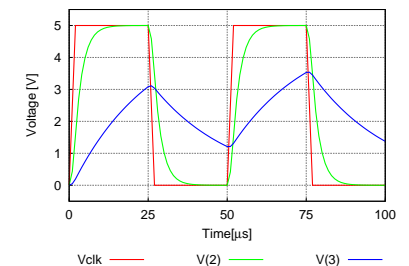
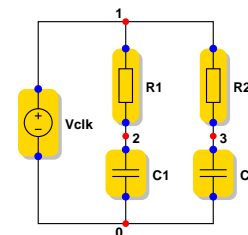
Wires

- Signal quality diminution
- May limit the speed (if too long and in some new technologies)



Capacitor Loading

```
Vclk 1 0 pulse(0V 5V 0us 2us 2us 23us 50us)
R1 1 2 25k
C1 2 0 100p
R2 1 3 25k
C2 3 0 1n
.print tr v(1,2,3)
.transient 100us 1us basic quiet > val.tmp
.end
```

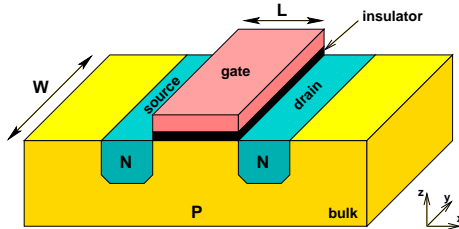


MOS Transistor: N and P transistors

MOS = *metal oxide semiconductor*

N transistors are made of:

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

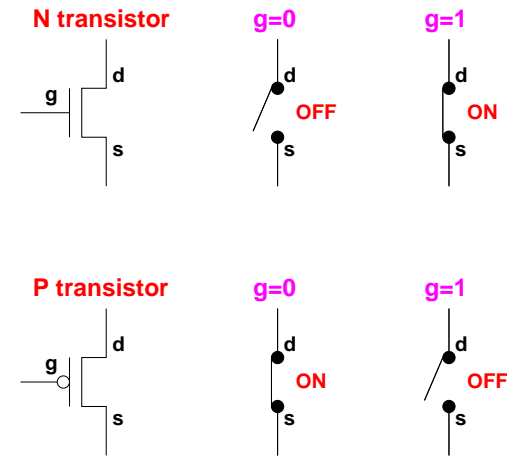


In N-type doping area, the **majority carriers** are **electrons** (holes in a P-type area)

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

MOS Transistor: Logic Model

Simple logic behavior (\approx switch)



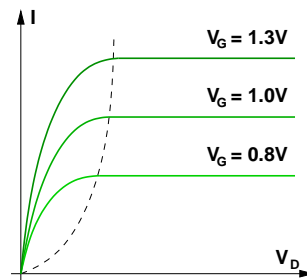
MOS Transistor: Simple Model

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: Electrical Simulation Model

Spice¹ model for an N transistor in technology 0.18 μm from TSMC (106 parameters, source: www.mosis.org) :

```

.MODEL CMOSN NMOS (
+VERSION = 3.1          TNOM = 27          TOX = 4E-9
+XJ = 1E-7             NCH = 2.3549E17     VTH0 = 0.3618568
+K1 = 0.5821674       K2 = 2.962352E-3      K3 = 1E-3
+K3B = 3.1746246     W0 = 1E-7          NLX = 1.784411E-7
+DVT0W = 0           DVT1W = 0          DVT2W = 0
+DVT0 = 1.0776375   DVT1 = 0.3574214    DVT2 = 0.0606977
+U0 = 257.825805    UA = -1.445098E-9   UB = 2.280431E-18
+UC = 5.132975E-11  VSAT = 1.002296E5   A0 = 1.9572227
+AGS = 0.4279783    B0 = 1.291312E-8    B1 = 6.025607E-7
+KETA = -0.0112723  A1 = 3.225587E-4    A2 = 0.8886833
+RDSW = 105         PRWG = 0.5         PRWB = -0.2
+WR = 1             WINT = 0          LINT = 1.345391E-8
+XL = 0             XV = -1E-8       DWG = -1.012269E-8
+DMB = 8.38965E-9  VOFF = -0.090305   NFACTOR = 2.2452365
+CIT = 0            CDSC = 2.4E-4       CDSCD = 0
+CDSCB = 0          ETA0 = 3.37666E-3  ETAB = 1.141951E-5
    
```

¹electrical simulator program

+DSUB = 0.017061	PCLM = 0.7636672	PDIBLC1 = 0.1793189
+PDIBLC2 = 2.914511E-3	PDIBLCB = -0.1	DROUT = 0.7552449
+PSCBE1 = 4.184752E10	PSCBE2 = 2.410517E-9	PVAG = 0.0261218
+DELTA = 0.01	RSH = 6.7	MOBMOD = 1
+PRT = 0	UTE = -1.5	KT1 = -0.11
+KT1L = 0	KT2 = 0.022	UA1 = 4.31E-9
+UB1 = -7.61E-18	UC1 = -5.6E-11	AT = 3.3E4
+WL = 0	WLN = 1	VWV = 0
+WMN = 1	WML = 0	LL = 0
+LLN = 1	LW = 0	LWN = 1
+LWL = 0	CAPMOD = 2	XPART = 0.5
+CGDO = 7.51E-10	CGSO = 7.51E-10	CGBO = 1E-12
+CJ = 9.520232E-4	PB = 0.8	MJ = 0.3763097
+CJSW = 2.543816E-10	PBSW = 0.8	MJSW = 0.1472251
+CJSWG = 3.3E-10	PBSWG = 0.8	MJSWG = 0.1472251
+CF = 0	PVTH0 = -6.376792E-4	PRDSW = -0.5939392
+PK2 = 1.01238E-3	WKETA = 4.251478E-3	LKETA = -7.831209E-3
+PU0 = 8.6592416	PUA = 5.50172E-12	PUB = 0
+PVSAT = 1.405109E3	PETA0 = 1.003159E-4	PKETA = 1.134176E-3

MOS Transistor: Imperfect Switch

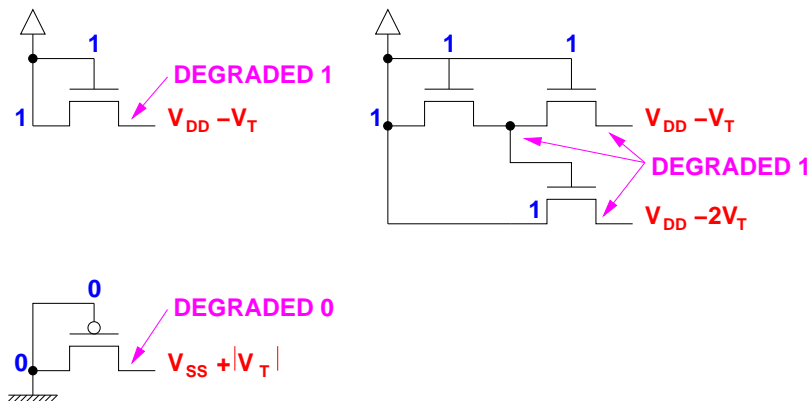


	0	1
STRONG	close to V_{SS}	close to V_{DD}
DEGRADED	greater than V_{SS}	less than V_{DD}

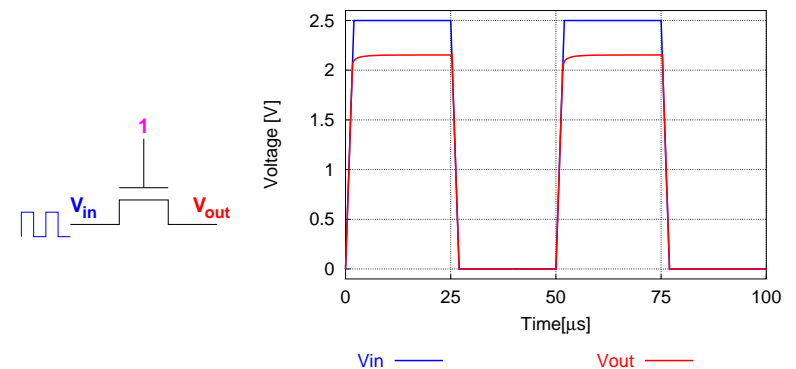
N transistor pull no higher than $V_{DD} - V_{TN}$

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

MOS Transistor: Imperfect Switch

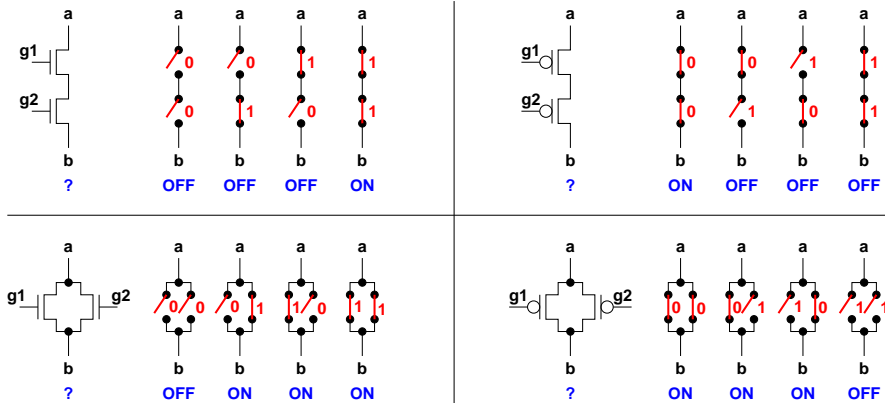


MOS Transistor: Imperfect Switch Simulation



Techno.: $0.25 \mu\text{m}$, $V_{DD} = 2.5 \text{ V}$, $W = 0.72 \mu\text{m}$, $L = 0.24 \mu\text{m}$, $V_{TN} \approx 0.37 \text{ V}$

MOS Transistors: Series and Parallel



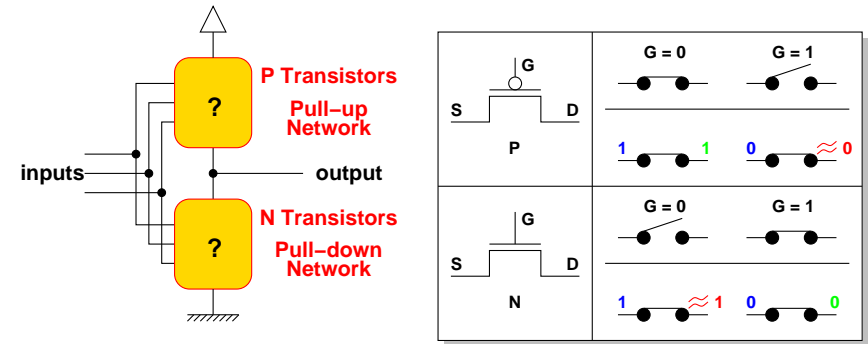
N: (1=ON, 0=OFF), P: (1=OFF, 0=ON)

Series: **both** must be ON, Parallel: **either** can be ON

CMOS Logic

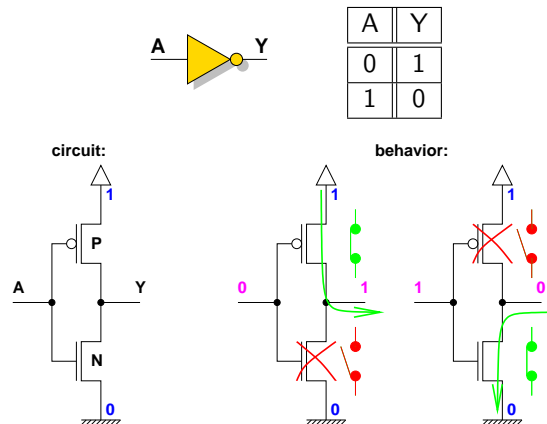
CMOS = complementary MOS

N and P transistors are only used for passing **strong** signals

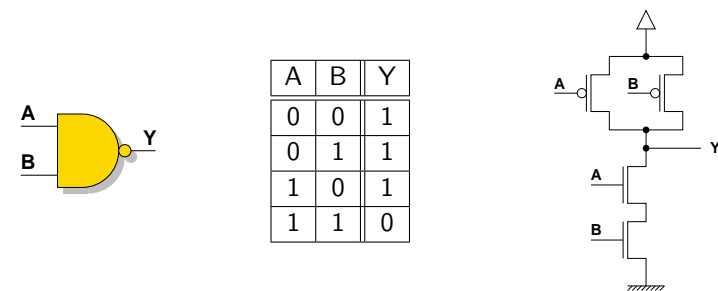


Logic Gate: Inverter

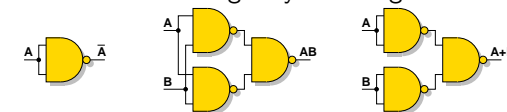
The simplest gate: only 2 transistors (1 N and 1 P)



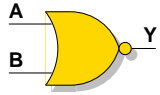
Logic Gate: NAND2 (2-input not-and)



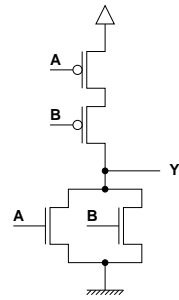
All logic functions can be built using only NAND gates:



Logic Gate: NOR2 (not-or)

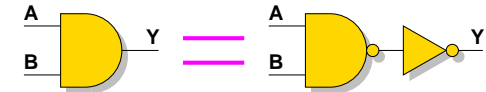
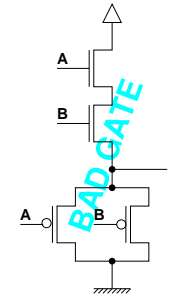


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



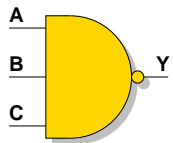
Logic Gate: AND2

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

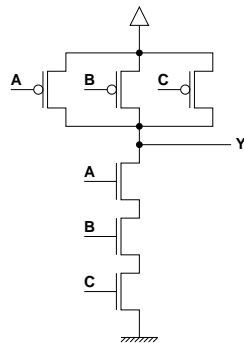


- left solution (the **bad** one): some output levels are degraded
- right solution (the **good** one): $AB = \overline{\overline{AB}}$ (6-transistor gate)

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)

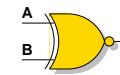
Logic Gate: Other Gates

XOR (exclusive or)



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

XNOR



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

MUX (multiplexer)

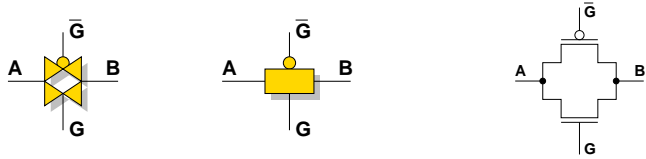


S	Y
0	A
1	B

Pass Gate

The *water tap* for circuits. . .

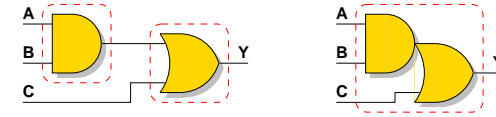
The nodes A and B are connected when $G=1$ (disconnected when $G=0$)



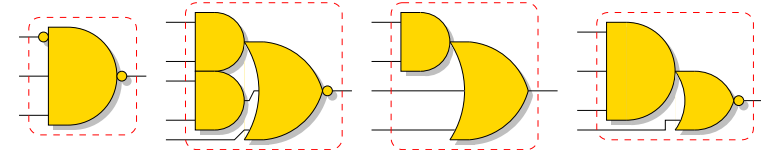
The state of nodes A and B are not **forced** by V_{DD} or V_{SS} , it can be degraded \rightsquigarrow signal bufferization or regeneration

Complex Gates

- AO21 gate (area -25% and speed +40%)

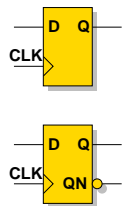


- Standard complex gates



Memory Elements

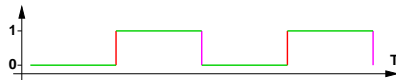
There are many types of memory elements. Here, we will only focus on standard **flip-flops**



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

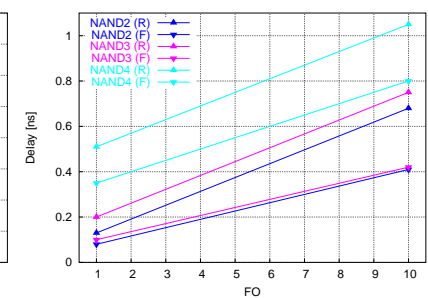
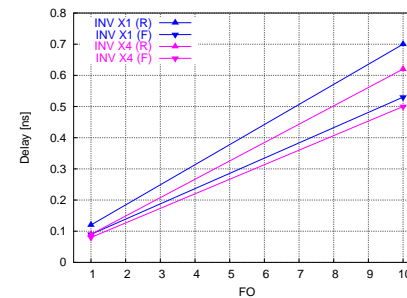
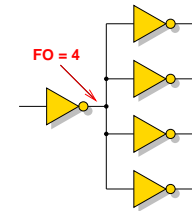
Remark:

↑ is the rising clock **edge**



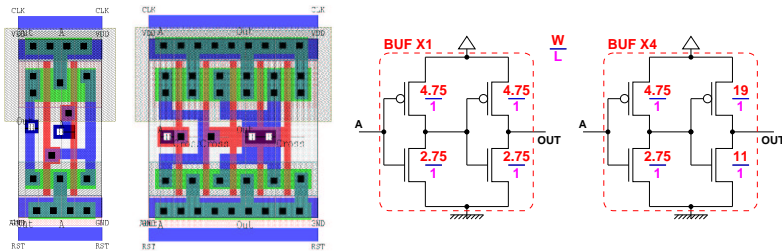
Fanout

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 regenerate the signal levels ($f(x) = x$)



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

Fast Circuit Design: Basic Ideas

• $V_{DD} \nearrow \Rightarrow$ speed \nearrow but limited by the technology

• Transistor size :

▶ $W \nearrow \Rightarrow$ speed \nearrow GOOD

▶ $L \nearrow \Rightarrow$ speed \searrow BAD

▶ but $W \nearrow \Rightarrow C \nearrow \Rightarrow$ speed \searrow

Transistor Sizing

• Topology

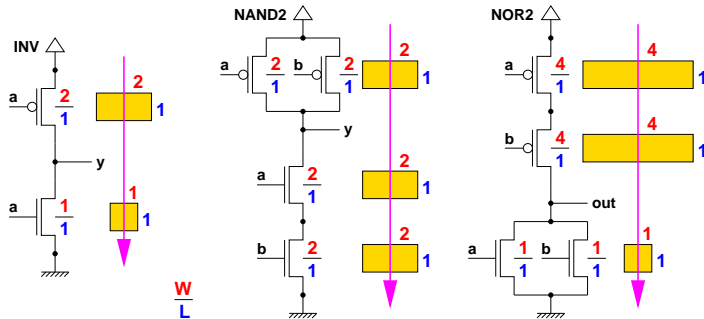
• Logic optimizations

• Place and route optimizations

• Algorithms, data coding. . .

Transistor Sizing: N and P Mobility Difference

The mobility of holes in silicon is typically lower than that of electrons
 $\mu = \mu_N / \mu_P \in [2, 3]$

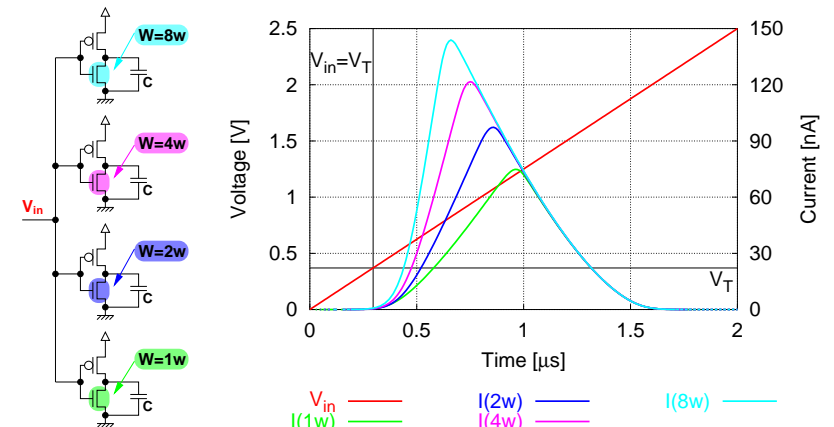


Symmetric behavior: same current for both N and P transistor networks
 (\rightsquigarrow same rising and falling times): $\frac{W}{L} = \mu = \frac{\mu_N}{\mu_P}$

Transistor Sizing

Current behavior (higher $I \Rightarrow$ faster gate): $I = \beta \times f(V_G, V_D, V_T)$

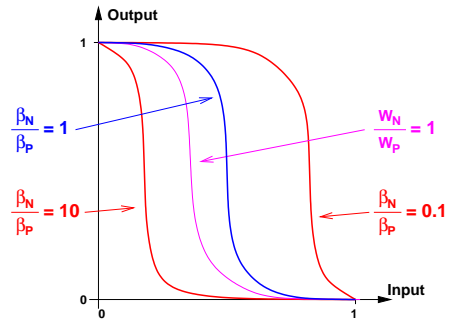
where the transistor gain is: $\beta = C_{techno} \times \frac{W}{L}$



Transistor Sizing: β Ratio Effects

Example in the inverter case:

- Sizing such that $\frac{\beta_N}{\beta_P} = 1$ ($T_{0 \rightarrow 1} \approx T_{1 \rightarrow 0}$)
- minimal L for all transistors



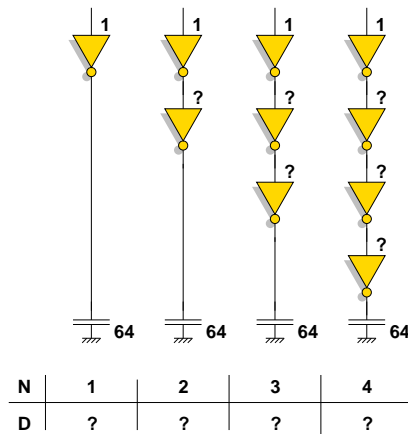
Transistor Sizing: Gate Level Solution

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)

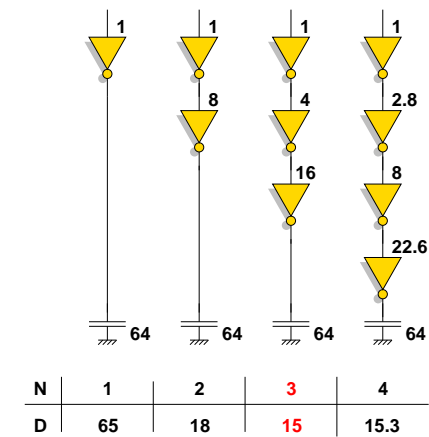
gate	drive	size [μm]		delay [ns]		power [$\mu\text{W}/\text{MHz}$]
		H	W	\uparrow	\downarrow	
INVX1	1X	5.04	1.32	0.0253	0.0146	0.0117
INVX2	2X	5.04	1.98	0.0228	0.0140	0.0218
INVX4	4X	5.04	2.64	0.0206	0.0125	0.0394
INVX8	8X	5.04	3.96	0.0198	0.0125	0.0773

Transistor Sizing: Simple Case

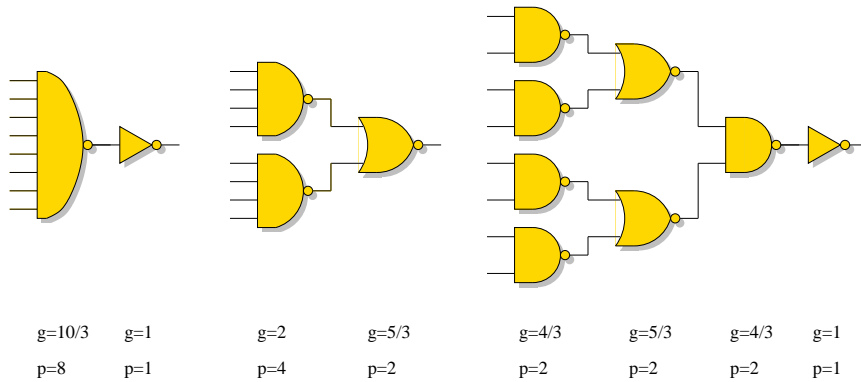


Question: what is the fastest architecture (#stages and delay)?

Transistor Sizing: Simple Case (Solution)



Transistor Sizing: 8-Input AND Gate



Question: what is the fastest architecture?

Transistor Sizing: 8-Input AND Gate (Solution)

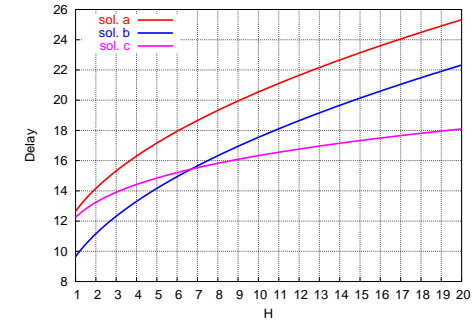
Solution a : $\hat{D} = 2(3.33H)^{1/2} + 9$

Solution b : $\hat{D} = 2(3.33H)^{1/2} + 6$

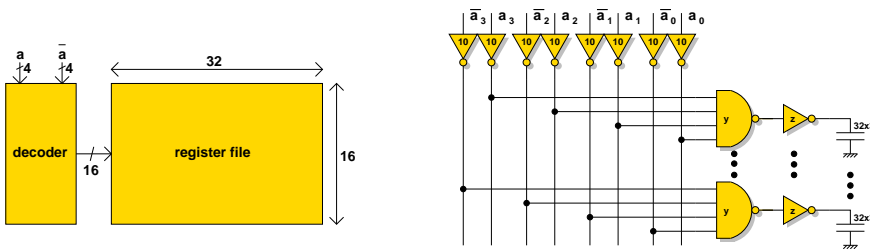
Solution c : $\hat{D} = 4(2.96H)^{1/4} + 7$

Delay value \hat{D} for output load H

	sol. a	sol. b	sol. c
$H = 1$	12.65	9.65	12.25
$H = 12$	21.64	18.64	16.77



Transistor Sizing: 4→16 decoder



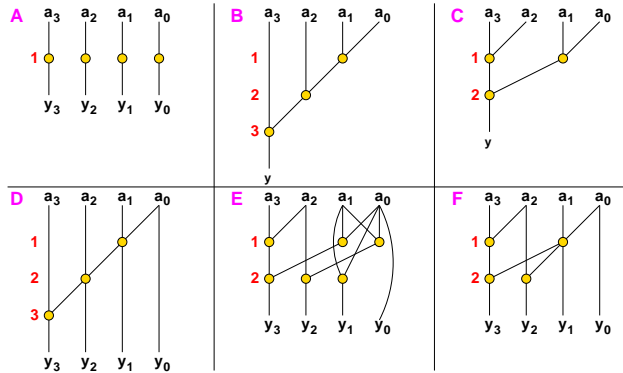
solution	N	G	P	D
NAND4, INV	2	2	5	29.8
INV, NAND4, INV	3	2	6	22.1
INV, NAND4, INV, INV	4	2	7	21.1
NAND2, INV, NAND2, INV	4	16/9	6	19.7
INV, NAND2, INV, NAND2, INV	5	16/9	7	20.4
INV, NAND2, INV, NAND2, INV, INV	6	16/9	8	21.6
INV, NAND2, INV, NAND2, INV, INV, INV	7	16/9	9	23.1

Hardware Complexity

Estimated or measured values:

- Time
 delay [s] [ns] [ps] [1] [FO4] [#NAND2], clock period, latency [#cycles]
- Speed
 clock frequency [Hz] [MHz] [GHz], throughput [evt/s] [MIPS]
- Area
 real area [mm²] [μm²] [λ²], arbitrary unit [1] [#NAND2] [#transistor]
- Power consumption and energy
 power [W] [μW/MHz], energy [J] [A/h]
- Compound: AT, AT², MIPS/W, MIPS²/W, MIPS³/W
- . . .

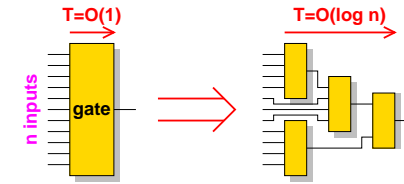
Hardware Complexity: Basic Examples



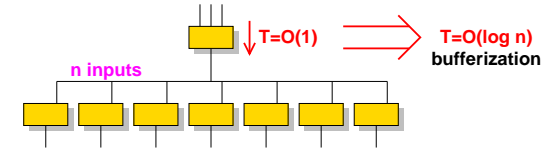
Case	A	B	C	D	E	F
Area $A(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n \log n)$
Time $T(n)$	$O(1)$	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(\log n)$

Hardware Complexity: Pitfalls

- Fanin problem:



- Fanout problem:



Power Consumption: Basic Definitions

Instantaneous power:

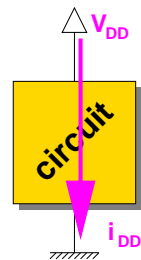
$$P(t) = i_{DD}(t) V_{DD}$$

Energy over some time interval T:

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

Average power over interval T:

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



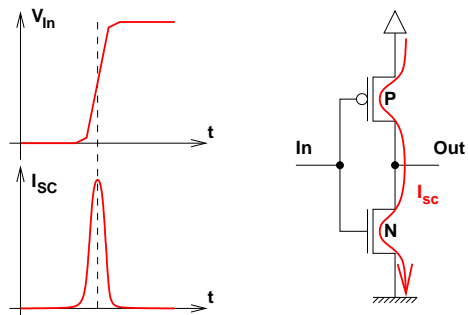
Power Consumption: Components

Power dissipation in CMOS circuits comes from 2 main components:

- Static dissipation:
 - ▶ sub-threshold conduction through OFF transistors
 - ▶ leakage current through diodes (reverse biased)
 - ▶ tunneling current through gate oxide
 - ▶ ...
- Dynamic dissipation:
 - ▶ charging and discharging of load capacitances (useful + parasitic)
 - ▶ short-circuit current

Short-Circuit Current in CMOS Gates

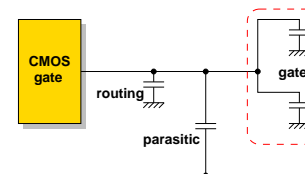
Occurs when both N and P transistors are **ON** while the input switches



Solution : short transition (crisp edges)

Charging and Discharging Load Capacitances

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



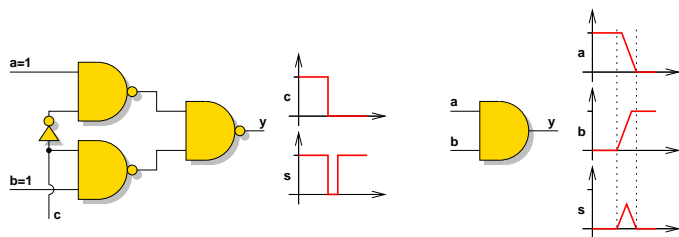
Solutions:

- 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):

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

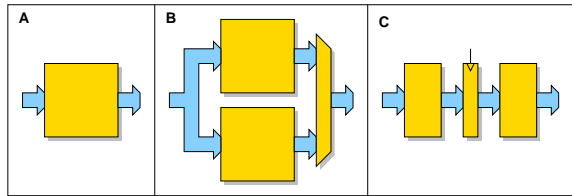
where

- α is the activity factor
- C the average switched capacitance (at each cycle)
- f the circuit frequency

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

Low-Power Design: Parallelism

Idea: reduce the supply voltage V_{DD} without speed degradation (parallelism)



solution	total capacitance	supply voltage	circuit frequency	power consumption
standard (A)	C	V	f	$P_A = CV^2f$
parallel (B)	$2.2C$	$0.6V$	$0.5f$	$P_B = 2.2C(0.6V)^2 \cdot 0.5f = 0.396 P_A$
pipeline (C)	$1.2C$	$0.6V$	f	$P_C = 1.2C(0.6V)^2 f = 0.432 P_A$

Gate Examples: 0.35 μm AMS Library

gate	area [μm^2]	input capacitance [pF]	delay [ns]				power consumption [$\mu\text{W}/\text{MHz}$]
			C_L	$10C_L$	C_L	$10C_L$	
INV 1X	36	0.007	0.12	0.70	0.09	0.53	0.23
INV 2X	55	0.010	0.10	0.63	0.09	0.51	0.47
INV 4X	73	0.021	0.09	0.62	0.08	0.50	0.83
BUF 1X	73	0.006	0.18	0.73	0.19	0.62	0.41
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
NOR2 1X	55	0.009	0.14	0.66	0.15	0.58	0.39
OR2 1X	73	0.009	0.21	0.76	0.21	0.66	0.47
NAND3 1X	91	0.007	0.20	0.75	0.10	0.41	0.43
NOR3 1X	91	0.012	0.13	0.65	0.15	0.59	0.38
XOR2 1X	146	0.019	0.28	0.79	0.14	0.49	0.74
XOR3 1X	273	0.019	0.27	0.78	0.14	0.49	0.68
MUX 1X (A)	127	0.006	0.27	0.82	0.31	0.77	0.50
(C)		0.012	0.24	0.79	0.36	0.82	

Source: Austriamicrosystems <http://asic.austriamicrosystems.com/>
Technology: 0.35 μm , $V_{DD} = 3.3\text{V}$, $T = 25^\circ\text{C}$, $C_L = 0.015\text{pF}$

0.35 μm AMS Library: Gate List

type	gate list
Half Adders	ADD21 ADD22
Full Adders	ADD31 ADD32
AND-OR-INVERT	AOI210 AOI211 AOI212 AOI2110 AOI2111 AOI2112 AOI220 AOI221 AOI222 AOI310 AOI311 AOI312
Tri-State Buffers	BUFE2 BUFE4 BUFE6 BUFE8 BUFE10 BUFE12 BUFE15
Tri-State Buffers	BUFT2 BUFT4 BUFT6 BUFT8 BUFT10 BUFT12 BUFT15
Buffers	BUF2 BUF4 BUF6 BUF8 BUF12 BUF15
Bus Holder	BUSHD
Clock Buffers	CLKBU2 CLKBU4 CLKBU6 CLKBU8 CLKBU12 CLKBU15
Clock Inverters	CLKIN0 CLKIN1 CLKIN2 CLKIN3 CLKIN4 CLKIN6 CLKIN8 CLKIN10 CLKIN12 CLKIN15
D-Type Flip-Flops	DF1 DF3 DF31 DF32 DF33 DF34 DF35 DF36 DF37 DF38 DF39 DF40 DF41 DF42 DF43 DF44 DF45 DF46 DF47 DF48 DF49 DF50 DF51 DF52 DF53 DF54 DF55 DF56 DF57 DF58 DF59 DF60 DF61 DF62 DF63 DF64 DF65 DF66 DF67 DF68 DF69 DF70 DF71 DF72 DF73 DF74 DF75 DF76 DF77 DF78 DF79 DF80 DF81 DF82 DF83 DF84 DF85 DF86 DF87 DF88 DF89 DF90 DF91 DF92 DF93 DF94 DF95 DF96 DF97 DF98 DF99 DF100
Data Latches	DL1 DL3 DL31 DL32 DL33 DL34 DL35 DL36 DL37 DL38 DL39 DL40 DL41 DL42 DL43 DL44 DL45 DL46 DL47 DL48 DL49 DL50 DL51 DL52 DL53 DL54 DL55 DL56 DL57 DL58 DL59 DL60 DL61 DL62 DL63 DL64 DL65 DL66 DL67 DL68 DL69 DL70 DL71 DL72 DL73 DL74 DL75 DL76 DL77 DL78 DL79 DL80 DL81 DL82 DL83 DL84 DL85 DL86 DL87 DL88 DL89 DL90 DL91 DL92 DL93 DL94 DL95 DL96 DL97 DL98 DL99 DL100
Delay Buffers	DLY12 DLY22 DLY32 DLY42
Inverting Majority	IMAJ30 IMAJ31
Inverting Multiplexers	IMUX20 IMUX21 IMUX22 IMUX23 IMUX24 IMUX30 IMUX31 IMUX32 IMUX33 IMUX40 IMUX41 IMUX42
Inverters	INV0 INV1 INV2 INV3 INV4 INV6 INV8 INV10 INV12 INV15
JK Flip-Flops	JK1 JK3 JK31 JK32 JK33 JK34 JK35 JK36 JK37 JK38 JK39 JK40 JK41 JK42 JK43 JK44 JK45 JK46 JK47 JK48 JK49 JK50 JK51 JK52 JK53 JK54 JK55 JK56 JK57 JK58 JK59 JK60 JK61 JK62 JK63 JK64 JK65 JK66 JK67 JK68 JK69 JK70 JK71 JK72 JK73 JK74 JK75 JK76 JK77 JK78 JK79 JK80 JK81 JK82 JK83 JK84 JK85 JK86 JK87 JK88 JK89 JK90 JK91 JK92 JK93 JK94 JK95 JK96 JK97 JK98 JK99 JK100
Tie-Up/Down	TIE0 TIE1
Majority	MAJ31 MAJ32
Multiplexers	MUX21 MUX22 MUX24 MUX26 MUX31 MUX32 MUX33 MUX34 MUX41 MUX42 MUX43
NAND	NAND20 NAND21 NAND22 NAND23 NAND24 NAND26 NAND28 NAND30 NAND31 NAND32 NAND33 NAND34 NAND40 NAND41 NAND42 NAND43
NOR	NOR20 NOR21 NOR22 NOR23 NOR24 NOR30 NOR31 NOR32 NOR33 NOR40 NOR41 NOR42
OR-AND-INVERT	OAI210 OAI211 OAI212 OAI2110 OAI2111 OAI2112 OAI220 OAI221 OAI222 OAI310 OAI311 OAI312
Toggle Flip-Flops	TFEC1 TFEC3 TFEC31 TFEC32 TFEC33 TFEC34 TFEC35 TFEC36 TFEC37 TFEC38 TFEC39 TFEC40 TFEC41 TFEC42 TFEC43 TFEC44 TFEC45 TFEC46 TFEC47 TFEC48 TFEC49 TFEC50 TFEC51 TFEC52 TFEC53 TFEC54 TFEC55 TFEC56 TFEC57 TFEC58 TFEC59 TFEC60 TFEC61 TFEC62 TFEC63 TFEC64 TFEC65 TFEC66 TFEC67 TFEC68 TFEC69 TFEC70 TFEC71 TFEC72 TFEC73 TFEC74 TFEC75 TFEC76 TFEC77 TFEC78 TFEC79 TFEC80 TFEC81 TFEC82 TFEC83 TFEC84 TFEC85 TFEC86 TFEC87 TFEC88 TFEC89 TFEC90 TFEC91 TFEC92 TFEC93 TFEC94 TFEC95 TFEC96 TFEC97 TFEC98 TFEC99 TF100 TF101 TF102 TF103 TF104 TF105 TF106 TF107 TF108 TF109 TF110 TF111 TF112 TF113 TF114 TF115 TF116 TF117 TF118 TF119 TF120 TF121 TF122 TF123 TF124 TF125 TF126 TF127 TF128 TF129 TF130 TF131 TF132 TF133 TF134 TF135 TF136 TF137 TF138 TF139 TF140 TF141 TF142 TF143 TF144 TF145 TF146 TF147 TF148 TF149 TF150 TF151 TF152 TF153 TF154 TF155 TF156 TF157 TF158 TF159 TF160 TF161 TF162 TF163 TF164 TF165 TF166 TF167 TF168 TF169 TF170 TF171 TF172 TF173 TF174 TF175 TF176 TF177 TF178 TF179 TF180 TF181 TF182 TF183 TF184 TF185 TF186 TF187 TF188 TF189 TF190 TF191 TF192 TF193 TF194 TF195 TF196 TF197 TF198 TF199 TF200
XOR	XOR20 XOR21 XOR22 XOR30 XOR31 XOR40 XOR41
XNOR	XNR20 XNR21 XNR22 XNR30 XNR31 XNR40 XNR41

2002 ITRS Roadmap

International Technology Roadmap for Semiconductors (from the Semiconductor Industry Association), <http://public.itrs.net/>

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
MTr/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

Source: book *CMOS VLSI Design*, N. Weste and D. Harris

Bibliography

CMOS VLSI Design

A Circuits and Systems Perspective

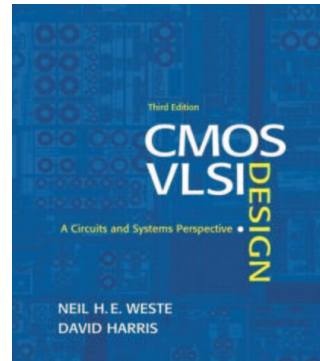
N. Weste and D. Harris

3rd Edition

2004

Addison Wesley

ISBN: 0-321-14901-7



Part 2

Hardware Computer Arithmetic Basics

Contents

- Basic number systems: *signed and unsigned integer, carry-save*
- Addition
 - ▶ Basic cells and adders: *HA, FA, counters, ripple, propagate/generate*
 - ▶ Fast adders: *select, skip, lookahead, parallel-prefix*
 - ▶ Redundant adders and multi-operand adders: *carry-save, borrow-save*
 - ▶ Bibliography
- Multiplication
 - ▶ Basic algorithms: *shift-and-add, high-radix, Braun, Baugh-Wooley*
 - ▶ Recoding: *Booth, modified Booth*
 - ▶ Fast multipliers: *partial product generation, reduction, fast assimilation*
 - ▶ Modified multipliers: *MAC, FMA, squarer, \times_{cst}*
 - ▶ Bibliography

Basic Number Systems

Representation of n -bit **unsigned integers** in **radix-2**:

$$A = a_{n-1}a_{n-2} \cdots a_1a_0 = \sum_{i=0}^{n-1} a_i 2^i$$

Representation of **signed integers** using **2's complement** or **sign magnitude**:

$$A = a_{n-1}a_{n-2} \cdots a_1a_0 = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

$$A = sa_{n-2} \cdots a_1a_0 = (-1)^s \sum_{i=0}^{n-2} a_i 2^i$$

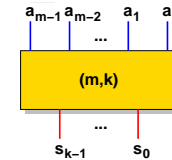
Radices used in practice: 2 or a power of 2 (2^k with $k \in \{2, 3, \dots, 10\}$)

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)

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

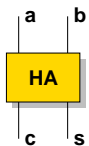


Standard counters:

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

Addition

HA Cell



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

Arithmetic equation:

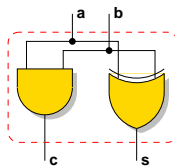
$$2c + s = a + b$$

Logic equation:

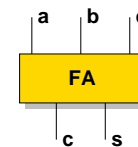
$$s = a \oplus b$$

$$c = ab$$

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

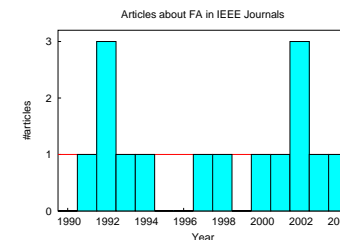
Arithmetic equation:

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

Logic equation:

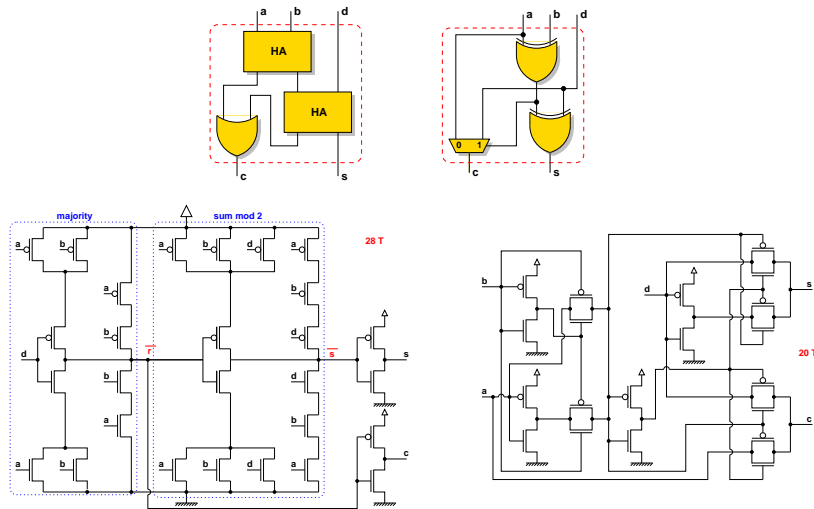
$$s = a \oplus b \oplus d$$

$$c = ab + ad + bd$$



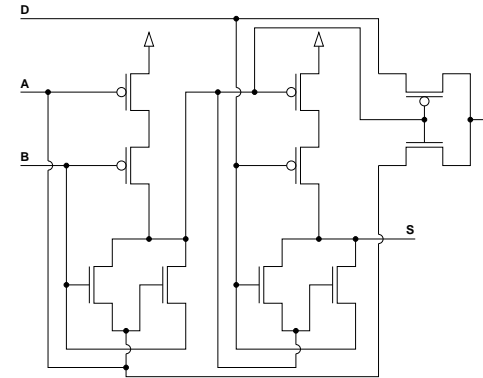
There many implementations of the FA cell

FA Implementations



Recently Published FA Cell

10-transistor solution² (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

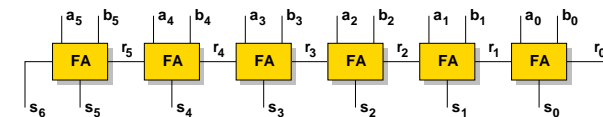
²H. T. Bui, Y. Wang et Y. Jiang. *Design and analysis of low-power 10-transistor full adders using novel XOR-XNOR gates*. IEEE Trans. CaS, jan. 2002.

Main Gates Used in Computer Arithmetic Operators

gate	area [μm^2]	input capacitance [pF]	delay [ns]				power [$\mu\text{W}/\text{MHz}$]	path			
			C_L	$10C_L$	C_L	$10C_L$					
HA 1X	164	0.015	0.26	0.81	0.33	0.80	0.93	A \rightarrow C			
			0.35	0.90	0.35	0.79		A \rightarrow S			
		0.015	0.26	0.82	0.35	0.82		B \rightarrow C			
			0.28	0.83	0.35	0.79		B \rightarrow S			
FA 1X	364	0.030	0.39	0.96	0.45	0.95	1.28	A \rightarrow C			
			0.44	1.01	0.59	1.08		A \rightarrow S			
		0.028	0.40	0.97	0.42	0.94		B \rightarrow C			
			0.44	1.01	0.59	1.08		B \rightarrow S			
		0.025	0.34	0.90	0.38	0.89		D \rightarrow C			
			0.47	1.03	0.54	1.01		D \rightarrow 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 \rightarrow S			
		0.012	0.24	0.79	0.36	0.82	C \rightarrow 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)

Carry Propagation and Generation

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

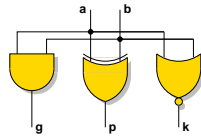
a, b	c	s	function
a = b	a	d	generate
a ≠ b	d	\bar{d}	propagate

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

$$p = a \oplus b$$

$$g = ab$$

$$k = \bar{a}\bar{b} = \overline{a+b}$$



Carry Chains in a CRA

The computation time depends on the **longest carry chain**

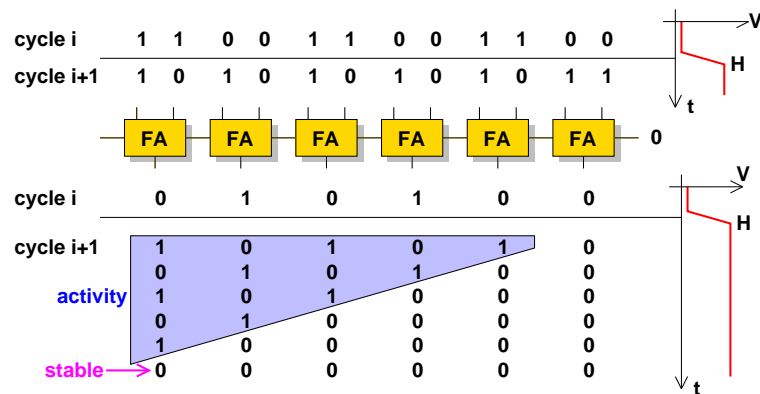
- worst case $\rightsquigarrow n$ propagations (ex: 000000 + 111111)
- best case $\rightsquigarrow 0$ propagation (ex: 111111 + 111111)
- average case?

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

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

Activity in a CRA



Worst case activity $\approx n^2/2$ (transitions), average activity $\approx \frac{3n}{2}$ (but only $\frac{n}{2}$ useful)

Subtraction

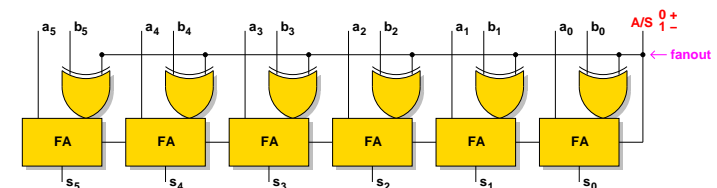
Using 2's complement representation:

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

Implementation:

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

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



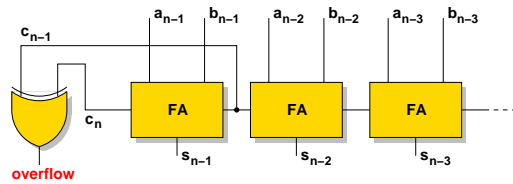
Overflow

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

Overflow detection in 2's complement:

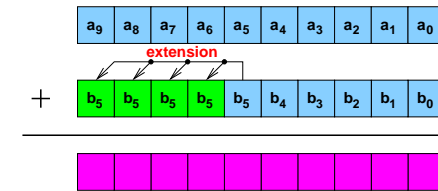
- Operands with opposite signs: no overflow
- Operands with same sign: there is an overflow iff the sign of the result is different from the sign of the operands

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

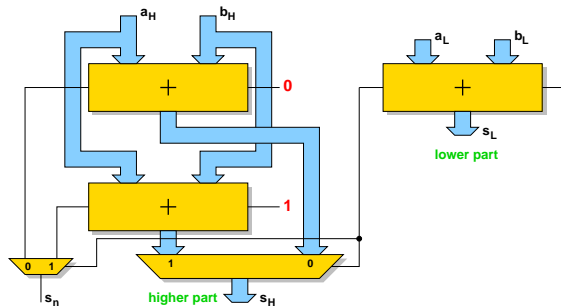


Warning:

- fanout
- order in case of multiple additions

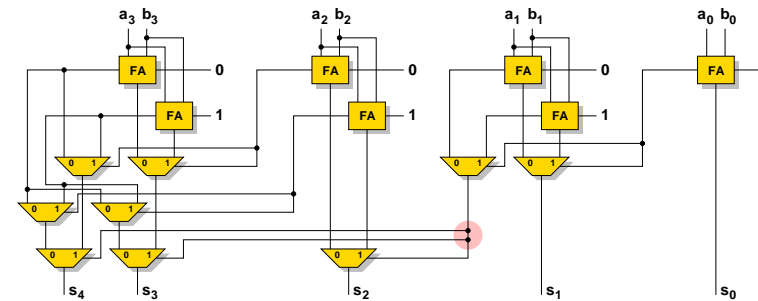
Carry-Select Adder

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



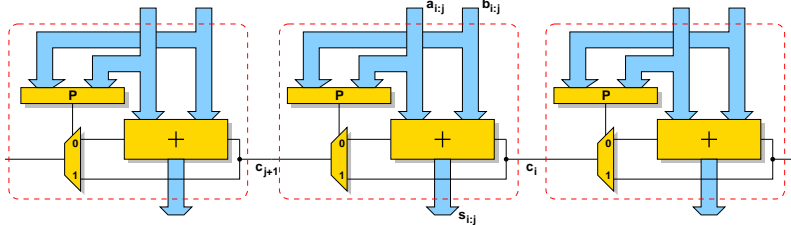
Recursive version $\rightarrow O(\log n)$ delay but...

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 $n-1$)

$$T(k) = 2(k-1)\tau_1 + \left(\frac{n}{k} - 2\right)\tau_2$$

$$T'(k) = 2\tau_1 - \frac{n\tau_2}{k^2}$$

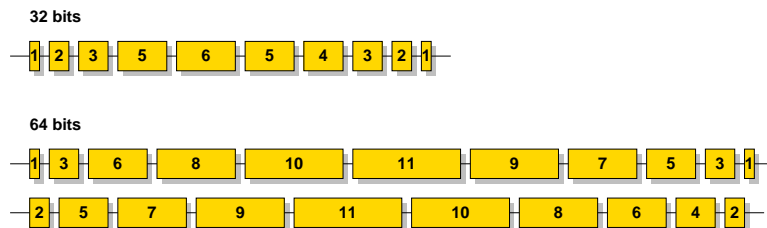
$$T''(k) = \frac{2n\tau_2}{k^3}$$

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

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:

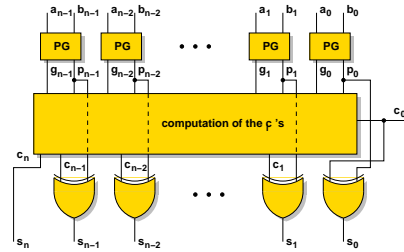
- rank $i-1$ generates a carry
 $\hookrightarrow g_{i-1} = 1$
- rank $i-1$ propagates a carry generated at rank $i-2$
 $\hookrightarrow p_{i-1} = g_{i-2} = 1$
- ranks $i-1$ and $i-2$ propagate a carry generated at rank $i-3$
 $\hookrightarrow p_{i-1} = p_{i-2} = g_{i-3} = 1$
 \vdots
- ranks $i-1$ to 0 propagate the adder input carry c_0 (set to 1)
 $\hookrightarrow p_{i-1} = p_{i-2} = \dots = p_1 = p_0 = c_0 = 1$

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

$$c_i = g_{i-1} + p_{i-1}g_{i-2} + p_{i-1}p_{i-2}g_{i-3} + \dots + p_{i-1} \dots p_1g_0 + p_{i-1} \dots p_0c_0$$

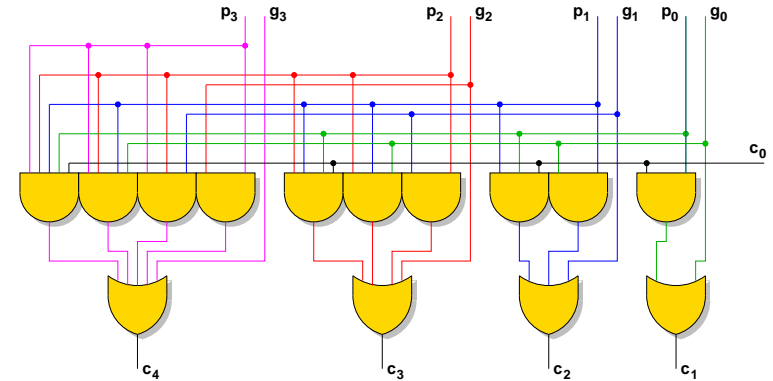
CLA architecture: parallel evaluation of

- (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

$$\begin{aligned} c_1 &= g_0 + p_0c_0 & c_3 &= g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0 \\ c_2 &= g_1 + p_1g_0 + p_1p_0c_0 & c_4 &= g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0 \end{aligned}$$



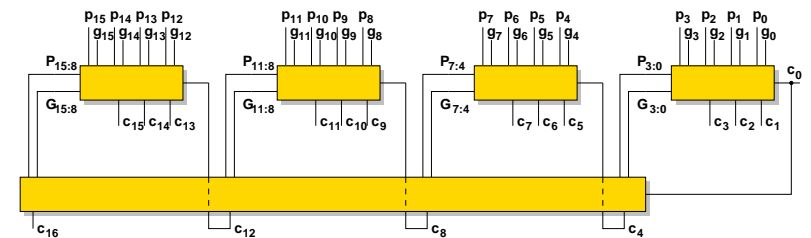
Carry Lookahead Adder: Remarks

- Non-bounded fanout
- Non-bounded fanin
- Theoretical delay in $O(\log n)$
- Small CLA (such as 4-bit CLAs) are used as basic elements for larger adders (CLA or other schemes)

Carry Lookahead Adder: Multiple Level CLA

The k -bit block, ranks i to j ($i > j$ et $i - j + 1 = k$), computes:

- carries c_i, c_{i-1}, \dots to c_j
- propagation over the block $P_{i:j} = p_i p_{i-1} \dots p_j$
- generation into the block $G_{i:j} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \dots + p_i \dots p_{j+1} g_j$

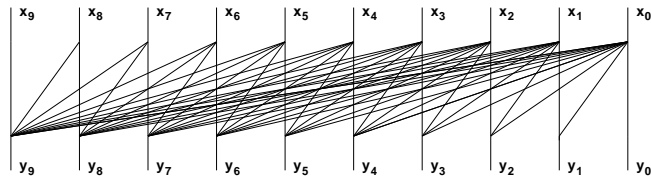


Delay: $O(\log_k n)$ (but very complex architecture: multi-tree mapping)

Parallel-Prefix Problems

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

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



Parallel-Prefix Addition

- Computation of the generate and propagate signals for all inputs:

$$g_i = a_i b_i \quad \text{et} \quad p_i = a_i \oplus b_i \quad \forall i = 0, 1, \dots, n-1$$

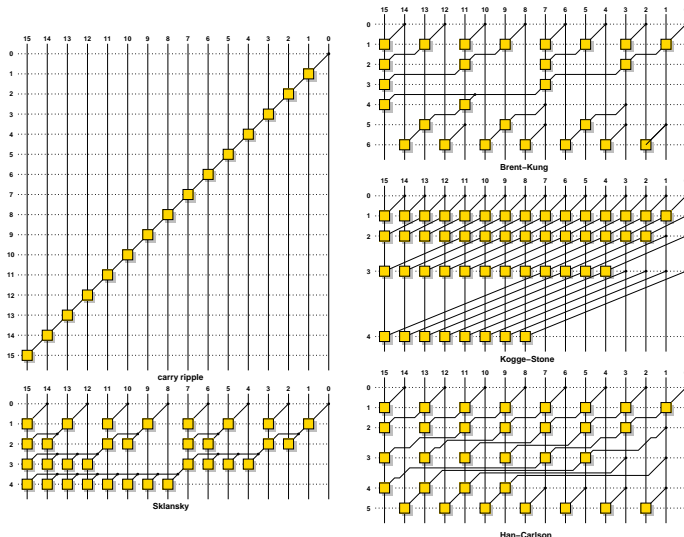
- Computation of the carries c_i using a m -level parallel-prefix architecture:

$$\begin{aligned} (G_{i,i}^0, P_{i,i}^0) &= (g_i, p_i) \\ (G_{i:k}^l, P_{i:k}^l) &= (G_{i:j}^{l-1}, P_{i:j}^{l-1}) \square (G_{j:k}^{l-1}, P_{j:k}^{l-1}) \quad k \leq j \leq i \text{ et } l = 1, \dots, m \\ &= (G_{i:j}^{l-1} + P_{i:j}^{l-1} G_{j:k}^{l-1}, P_{i:j}^{l-1} P_{j:k}^{l-1}) \\ c_{i+1} &= G_{i:0}^m + P_{i:0}^m c_0 \quad \forall i = 0, 1, \dots, n-1 \end{aligned}$$

- Computation of the sums:

$$s_i = p_i \oplus c_i \quad \forall i = 0, 1, \dots, n-1$$

Parallel-Prefix Addition: Standard Architectures

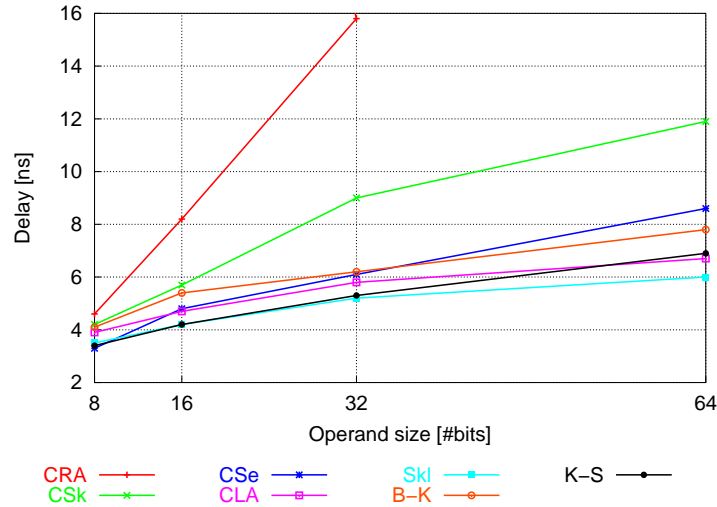


Parallel-Prefix Addition: Standard Architectures

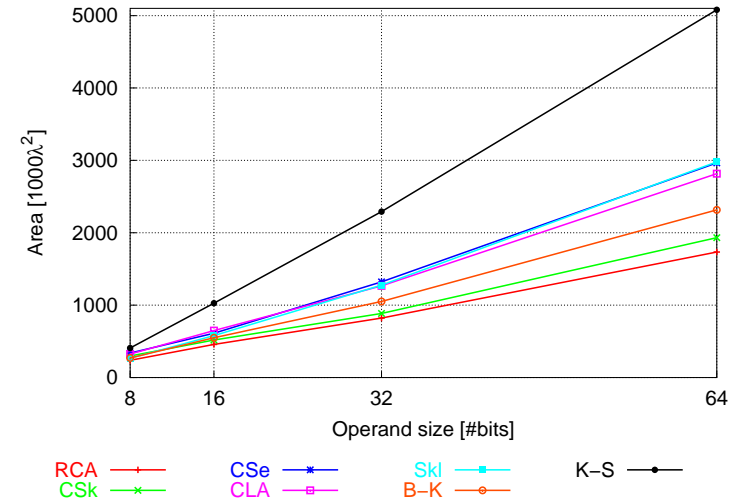
Values for the computation of the carries c_i only (step ②, steps ① and ③ are $T = O(1)$ and $A = O(n)$). T_{\square} et A_{\square} are the delay and the area (unit = $\#\square$). HT is the number of horizontal routing tracks and FO the maximal fanout

adder	T_{\square}	S_{\square}	HT	FO
carry ripple	$n-1$	$n-1$	1	2
Sklansky	$\log n$	$\frac{1}{2}n \log n$	$\log n$	$\frac{1}{2}n$
Brent-Kung	$2 \log n - 2$	$2n - \log n - 2$	$2 \log n - 1$	$\log n$
Kogge-Stone	$\log n$	$n \log n - n + 1$	$n-1$	2
Han-Carlson	$\log n + 1$	$\frac{1}{2}n \log n$	$\frac{1}{2}n + 1$	2

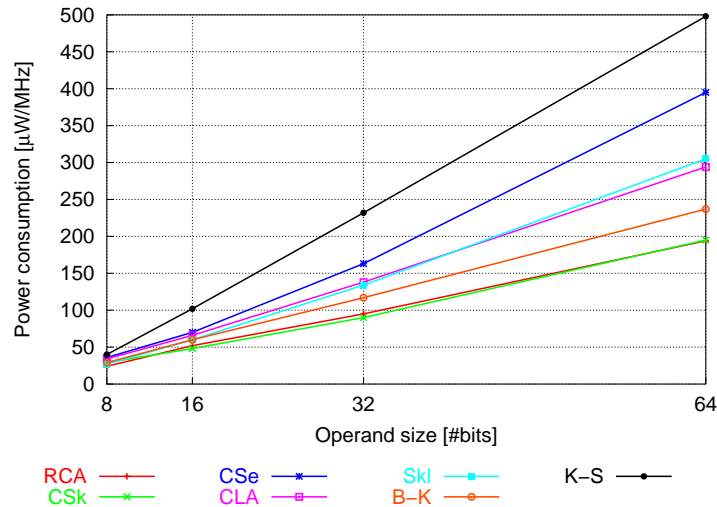
Comparison of Adders: Speed



Comparison of Adders: Area



Comparison of Adders: Power Consumption



Redundant or Constant Time Adders

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

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

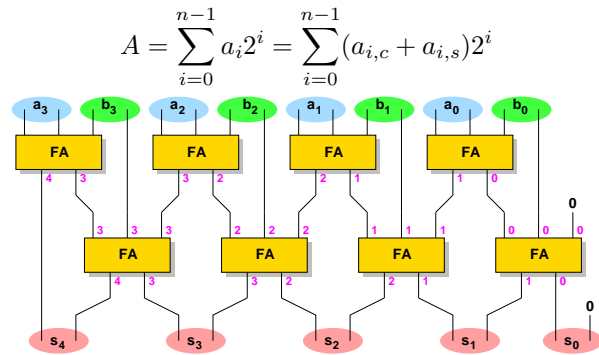
Using this representation, if $2\alpha + 1 > \beta$ some numbers have several possible representation at the bit level. For instance, the value 2345 (in the standard representation) can be represented in radix 10 with digits in $\{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$ by the values 2345, 235(-5) or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

Such a representation is said **redundant**

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

Carry-Save Adder

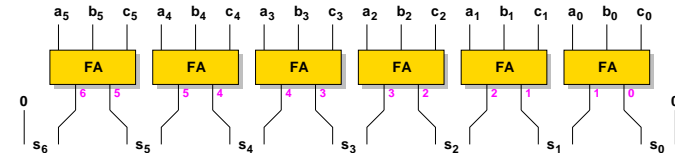
In **carry-save**, the number A is represented in radix 2 using digits $a_i \in \{0, 1, 2\}$ coded by 2 bits such that $a_i = a_{i,c} + a_{i,s}$ where $a_{i,c} \in \{0, 1\}$ and $a_{i,s} \in \{0, 1\}$



A carry-save addition is performed with the delay of 2 FA cells ($T = 0(1)$)

Carry-Save Trees

Example with 3 inputs: A , B and C

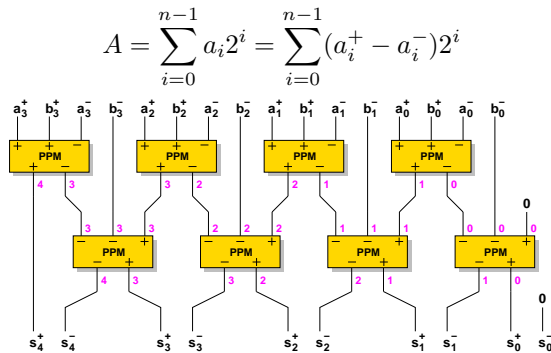


Carry-save reduction tree: $n(h)$ non-redundant inputs can be reduced by a h -level carry-save tree where $n(h) = \lfloor 3n(h-1)/2 \rfloor$ and $n(0) = 2$

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 borrow-save addition is performed with the delay of 2 PPM cells ($T = 0(1)$)

PPM Cell

Arithmetic equation:

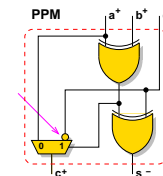
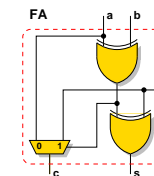
$$2c^+ - s^- = a^+ + b^+ - d^-$$

Logic equation:

$$s = a^+ \oplus b^+ \oplus d^-$$

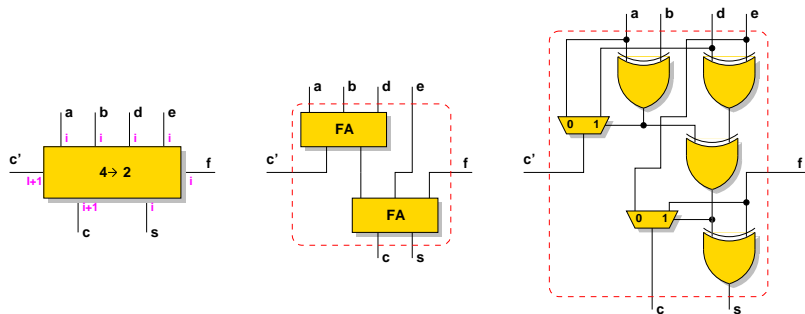
$$c = a^+ b^+ + a^+ \overline{d^-} + b^+ \overline{d^-}$$

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



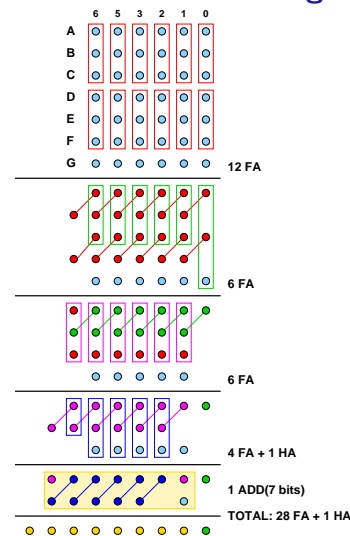
“4 to 2” Cell

Arithmetic equation: $a + b + d + e + f = 2(c + c') + s$

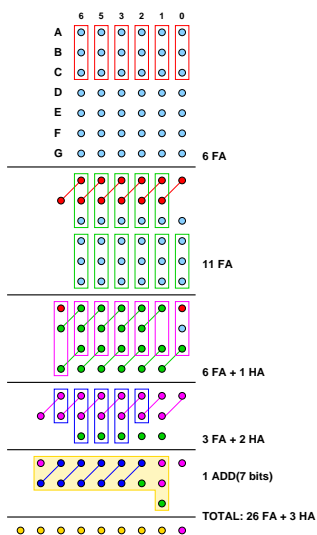


Interest: 4-2 cell based trees are more regular than FA based trees (placement and routing are simpler and more efficient)

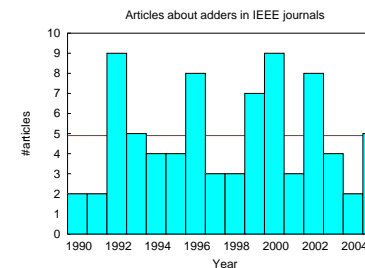
Multiperand Addition: Dot Diagram Reduction



Multiperand Addition: Dot Diagram Reduction



Research Activities on Adders



- 1994 : NEC, CMOS 0.4 μm and 3.3 V
ALU based on a 32-bit CLA (500 MHz)
- 2002 : Intel, CMOS 0.13 μm and 1.5 V
ALU based on a 32-bit Han-Carlson adder (5 GHz)

Bibliography

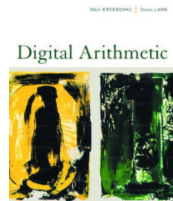
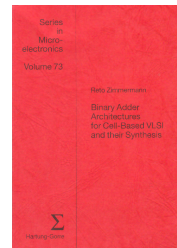
Binary Adder Architectures for Cell-Based VLSI and their Synthesis

Reto Zimmermann

1998

Hartung-Gorre

ISBN: 3-89649-289-6



Digital Arithmetic

Milos Ercegovac and Tomas Lang

2003

Morgan Kaufmann

ISBN: 1-55860-798-6

Multiplication

Shift-And-Add Multiplication

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

```

1  $P \leftarrow 0$ 
2 for  $i$  from 0 to  $n - 1$  do
3    $P \leftarrow P + a_i B 2^i$ 

```

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

Simplification (constant shift):

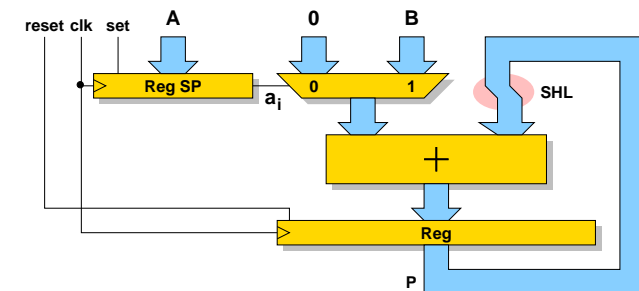
```

1  $P \leftarrow 0$ 
2 for  $i$  from 0 to  $n - 1$  do
3    $P \leftarrow (P + a_i B) \times 2^{-1}$ 
4  $P \leftarrow P 2^n$ 

```

Operation on line 4 is virtual

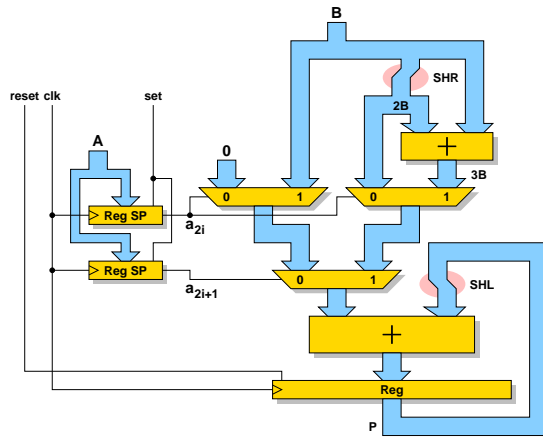
Shift-And-Add Multiplication: Implementation



	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 $\{-1 = \bar{1}, 0, 1\}$

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

Example: the integer 60 is represented by 00111100 = 01000 $\bar{1}$ 00

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

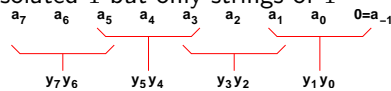
But, in some cases, this basic method leads to more 1 (or $\bar{1}$)!

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

⇒ modified Booth's recoding

Modified Booth's Recoding

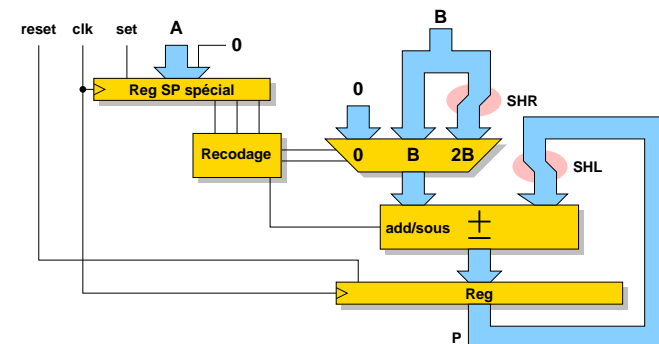
Idea: do not recode isolated 1 but only strings of 1



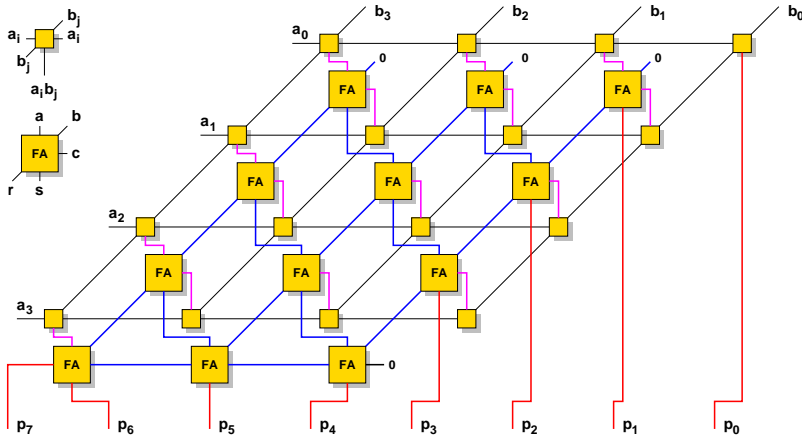
a_i	a_{i-1}	a_{i-2}	y_i	y_{i-1}	meaning	operation
0	0	0	0	0	string of 0s	+0
0	0	1	0	1	end of a string of 1s	+B
0	1	0	0	1	isolated 1	+B
0	1	1	1	0	end of a string of 1s	+2B
1	0	0	$\bar{1}$	0	beginning of a string of 1s	-2B
1	0	1	1	$\bar{1}$	isolated 0	-B
1	1	0	0	$\bar{1}$	beginning of a string of 1s	-B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



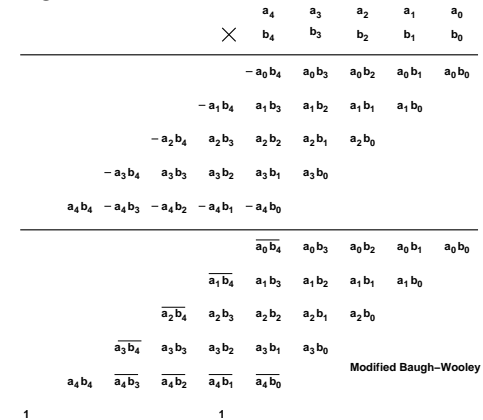
Braun Multiplier



Delay in $O(n)$ (area in $O(n^2)$)

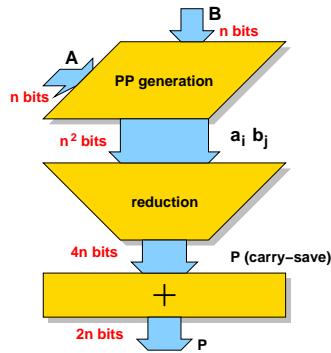
2's Complement Product

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

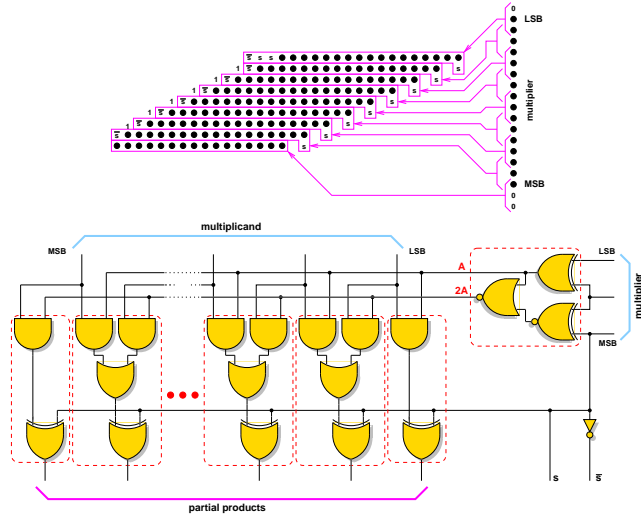
During the computation of the product $P = A \times B$ where A and B are n -bit integers, there are n^2 AND cells

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

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

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

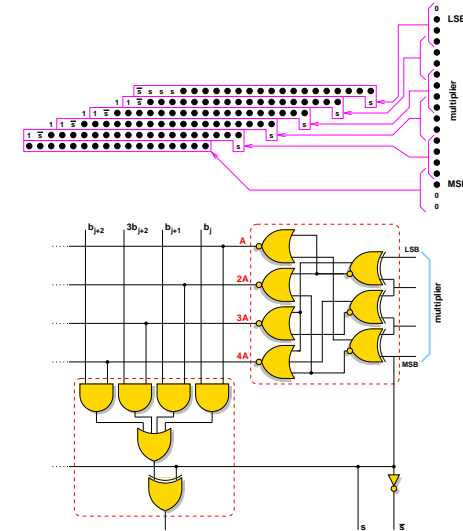
Partial Product Generation: Booth-2



A. Tisserand – ISSAC 2005 – Tutorial: Algorithms and Number Systems for Hardware Computer Arithmetic

109/151

Partial Product Generation: Booth-3



A. Tisserand – ISSAC 2005 – Tutorial: Algorithms and Number Systems for Hardware Computer Arithmetic

110/151

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

A. Tisserand – ISSAC 2005 – Tutorial: Algorithms and Number Systems for Hardware Computer Arithmetic

111/151

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

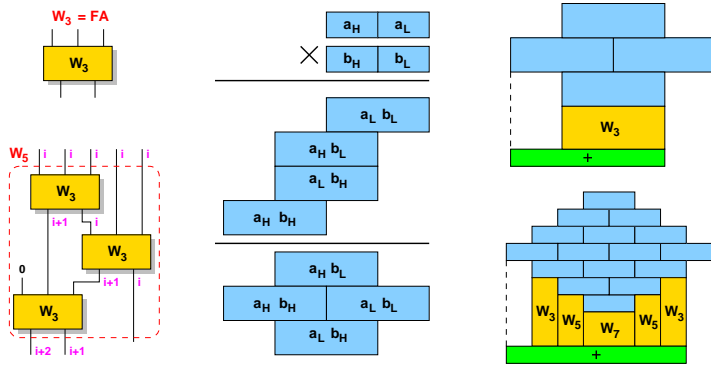
A. Tisserand – ISSAC 2005 – Tutorial: Algorithms and Number Systems for Hardware Computer Arithmetic

112/151

Wallace Trees

Wallace trees³ 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)

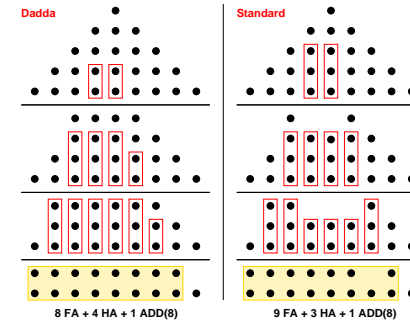


³C.S. Wallace. *A suggestion for a fast multiplier*. IEEE Transactions on Computers, février 1964.

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)

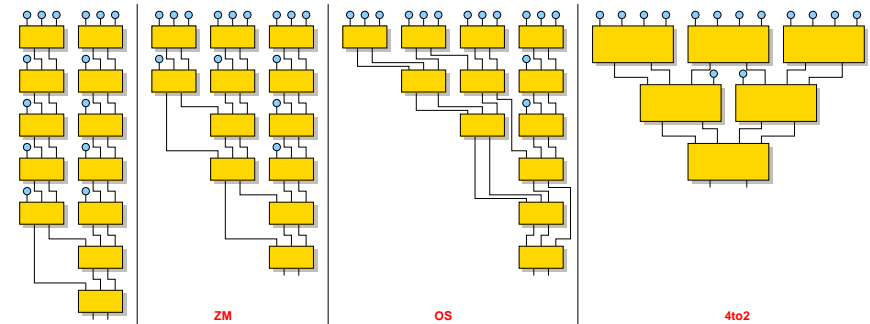
Example: The TDM⁴ method leads to 30% faster and 15% smaller circuits compared to a solution based on "4 to 2" cells

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

⁴V. Oklobdzija, D. Villegier et 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

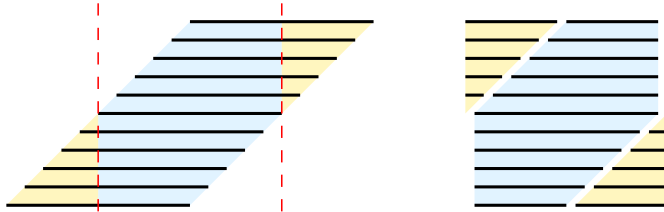
What is the best topology?



Example: 14-bit reduction

Layout Problem

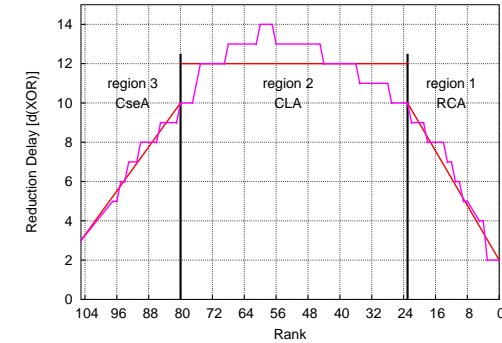
The best circuits have a square shape



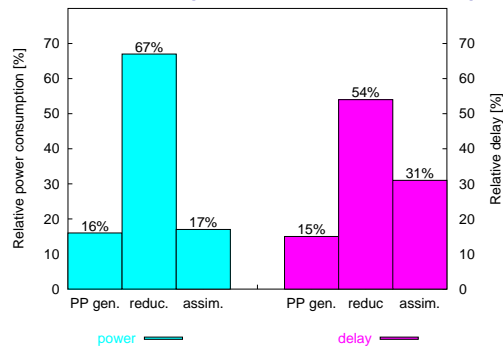
Final Addition in Multipliers

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



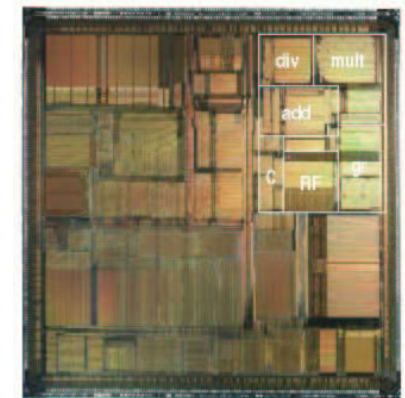
Power Consumption in Fast Multipliers



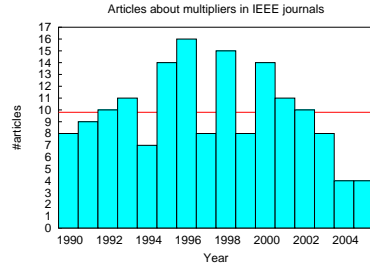
- 30% to 70% of redundant transitions (useless)
- place and route steps based on the internal arrival time
- add a pipeline stage

Example: UltraSPARC FP Multiplier

- 53×53 -bit multiplier
- 3 cycles of latency:
 - 1 PP + reduction
 - 2 assimilation
 - 3 rounding
- throughput = $1 \times$ per cycle
- Booth-2 recoding
- cell “4 to 2”
- assim.: *conditional-sum adder*
- 167 MHz, CMOS 0.5 μm
- source: ARITH12



Research Activities on Multipliers



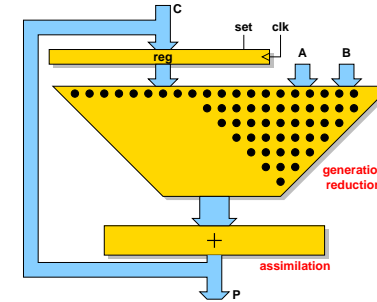
- 1991: Toshiba, CMOS 0.5 μm and 3.3 V multiplier 54×54 bits at 100 MHz, Booth 2, “4 to 2” cells, addition CLA + CSeA
- 2001: Mitsubishi, CMOS 0.18 μm and 1.8 V multiplier 54×54 bits at 600 MHz (2 cycles), Booth 2, “4 to 2” cells, without assimilation (carry-save product).

MAC and FMA

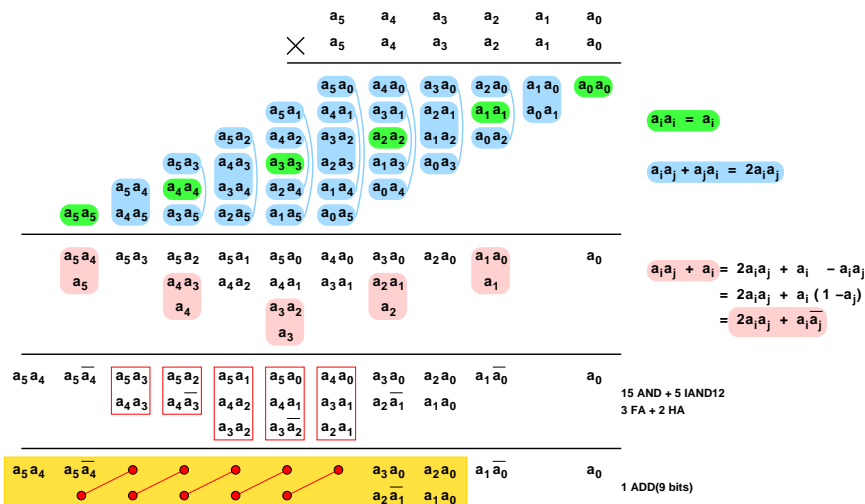
MAC: multiply and accumulate $P(t) = A \times B + P(t-1)$

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

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



Squarer



Multiplication by Constants

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

Example: $p = 111463 \times x$

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

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

Bibliography

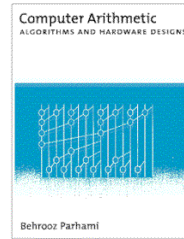
Computer Arithmetic Algorithms and Hardware Designs

Behrooz Parhami

2000

Oxford University Press

ISBN: 0-19-512583-5



Advanced Computer Arithmetic Design

Micheal Flynn and Stuart Obermann

2001

Wiley-Interscience

ISBN: 0-471-41209-0

Part 3

Examples in Cryptography

Contents

- Modular operations: *addition, multiplication*
- Side-channel attacks: *SPA, DPA, EM, timing, countermeasures*
- Residue Number System: *definition, operations, application*
- Bibliography

Modular Addition

Inputs⁵:

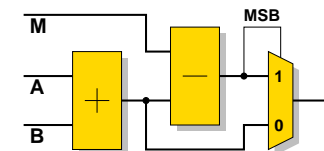
$$A, B \in \{0, 1, 2, 3, \dots, M-1\}$$

Output:

$$(A + B) \bmod M$$

Method:

$$(A + B) \bmod M = \begin{cases} A + B & \text{if } A + B < M \\ A + B - M & \text{if } A + B \geq M \end{cases}$$



⁵ $0 \leq A + B \leq 2M - 2$

Modular Addition $2^n - 1$

Inputs:

$$A, B \in \{0, 1, 2, 3, \dots, 2^n - 2\}$$

Output:

$$(A + B) \bmod (2^n - 1)$$

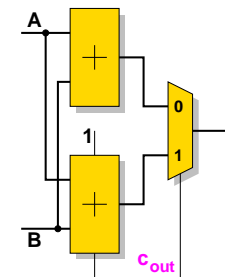
Basic method:

$$(A + B) \bmod (2^n - 1) = \begin{cases} A + B & \text{if } A + B < 2^n - 1 \\ \underbrace{A + B - (2^n - 1)}_{A+B+1} & \text{if } A + B \geq 2^n - 1 \end{cases}$$

But the condition $A + B \geq 2^n - 1$ is not trivial to compute

Modular Addition $2^n - 1$: Modified Method

$$(A + B) \bmod (2^n - 1) = \begin{cases} A + B & \text{if } A + B + 1 < 2^n \\ A + B + 1 & \text{if } A + B + 1 \geq 2^n \end{cases}$$



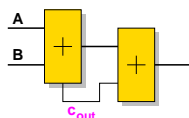
Modular Addition $2^n - 1$: Another Method

Idea: double representation of 0: $00 \dots 00 = 11 \dots 11$

$$(A + B) \bmod (2^n - 1) \in \{0, 1, 2, 3, \dots, 2^n - 2, 2^n - 1\}$$

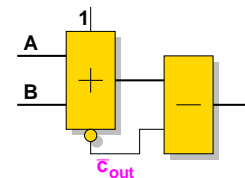
Modified method:

$$(A + B) \bmod (2^n - 1) = \begin{cases} A + B & \text{if } A + B < 2^n \\ A + B + 1 & \text{if } A + B \geq 2^n \end{cases} \\ = A + B + c_{out}$$



Modular Addition $2^n - 1$: Last Method

$$(A + B) \bmod (2^n - 1) = (A + B + 1) \bmod 2^n + \bar{c}_{out}(2^n - 1) \\ = (A + B + 1) \bmod 2^n - \bar{c}_{out}$$



Modular Addition

There are similar “things” for addition modulo $2^n + 1$ or some specific moduli

But:

- it highly depends on the value of M
- difficult to optimize for a set of moduli (M_1, M_2, \dots, M_k)
- it highly depends on the circuit constraints

Useful operations:

- $A + B + 1$ and $A + B - 1$
- compound operations: $A + B$ and $A + B + 1$ simultaneously
- automatic generator (parameters: M , implementation constraints)

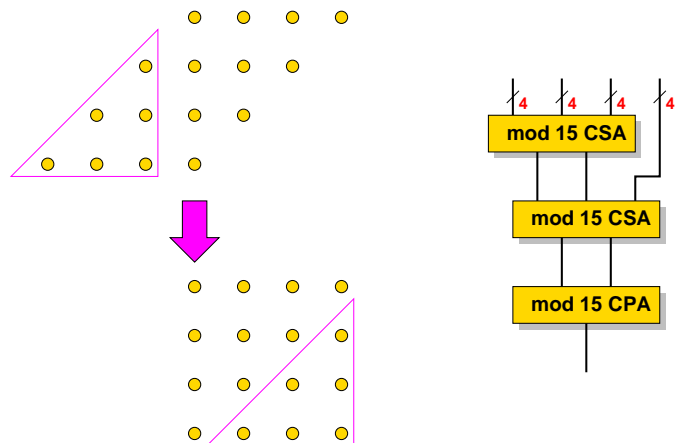
Modular Multiplication

Two main solutions:

- Multiplication **and** correction
 - ▶ first step: product $P = A \times B$
 - ▶ second step: reduction $R = P \bmod M$
 - ▶ possible for some specific M such as $2^n - 1$, $2^n + 1$
- Accumulation of the **reduced partial products**
 - ▶ parallel-parallel
 - ▶ serial-parallel (Most or Least Significant Digit First)

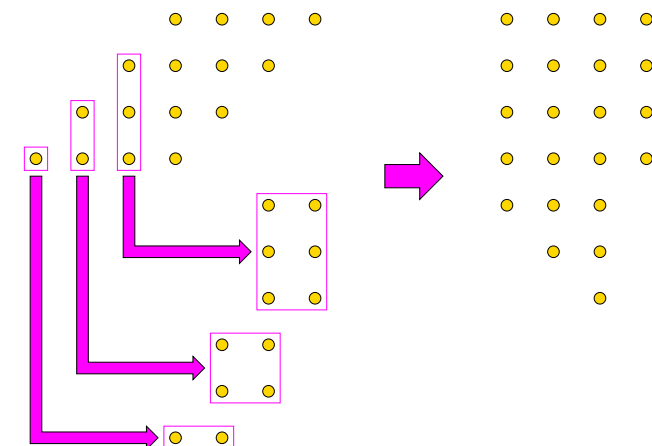
Modular Multiplication: Example $(A \times B) \bmod 15$

Based on: $16 \bmod 15 = 1$

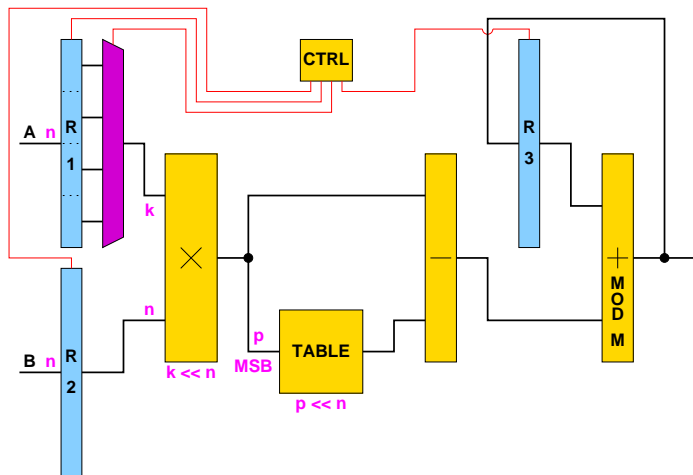


Modular Multiplication: Example $(A \times B) \bmod 13$

Based on: $16 \bmod 13 = 3$, $32 \bmod 13 = 6$, $64 \bmod 13 = 12$



Modular Multiplication: Tradeoff



Cryptosystem Security

- Cryptanalysis:
 - use the theoretical **weaknesses** in the **algorithms**
- Side-channel attacks:
 - use the information **leaked** during the **actual** cryptosystem **execution** (system + operating environment)

Leak sources:

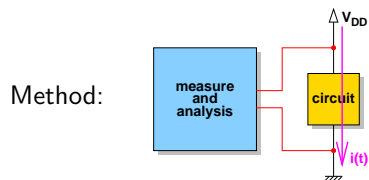
- power consumption
- timing information
- electromagnetic (EM) radiation
- sound
- ...

Power Analysis Attacks

Power dissipation components in CMOS circuits:

- **Static** dissipation due to the structure of the circuit (techno.)
- **Dynamic** dissipation due to:
 - ▶ activity (useful + parasitic)
 - ▶ short-circuit current

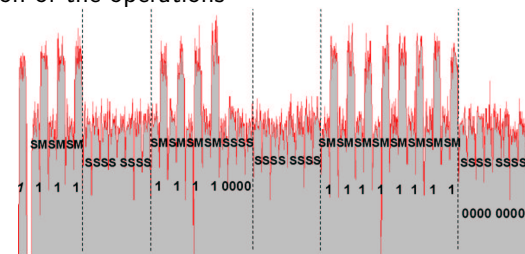
Main problem: the electrical **activity** depends on the **operations** and **data**



Simple/Differential Power Analysis

Simple Power Analysis (SPA):

the secret information is deduced from the **examination** of the **power trace**
 ↳ identification of the operations



Differential Power Analysis (DPA):

exploits the **variations** in power consumption that are **correlated** to the data values being manipulated (statistics)
 ↳ identification of the data

Example: Square and Multiply Algorithm

```

Inputs:  $x, d = (d_{m-1} \dots d_1 d_0)_2$ 
Output:  $y = x^d$ 
1  $R \leftarrow 1$ 
2  $i \leftarrow m - 1$ 
3 while ( $i \geq 0$ ) do
4    $R \leftarrow R^2$  square
5   if ( $d_i = 1$ ) then
6      $R \leftarrow Rx$  multiply
7   endif
8    $i \leftarrow i - 1$ 
9 endwhile
10 return  $R$ 

```

Positional Number System

- Radix β
- Digits a_i from the digit set
- Size n (the number of digits)

$$A = \sum_{i=0}^{n-1} a_i \beta^i$$

Remark: the **order** of the digits in a positional number system gives *information* on the values

Countermeasures

Main ideas:

- add noise \rightarrow make operations and data **not measurable**
- reduce the “differences” \rightarrow make operations (and leak) for different data **indistinguishable**

Levels:

- algorithm
- **arithmetic** (operations and number systems)
- architecture
- circuit

Residue Number System (RNS)

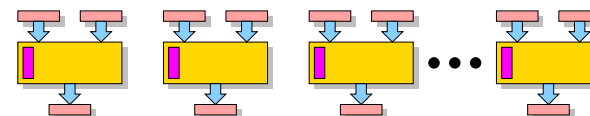
- Base $\mathcal{B} = (m_1, m_2, \dots, m_k)$ of k relatively prime moduli
- Size of the base: k

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

Operations:

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

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



Residue Number System: Example (1/2)

Base: $\mathcal{B} = (8, 7, 5, 3)$

Dynamic range: $M = 8 \times 7 \times 5 \times 3 = 840$, i.e., $0 \leq A < M$

A_{std}	A_{RNS}	A_{std}	A_{RNS}	A_{std}	A_{RNS}
0	[0, 0, 0, 0]	9	[1, 2, 4, 0]	18	[2, 4, 3, 0]
1	[1, 1, 1, 1]	10	[2, 3, 0, 1]	19	[3, 5, 4, 1]
2	[2, 2, 2, 2]	11	[3, 4, 1, 2]	20	[4, 6, 0, 2]
3	[3, 3, 3, 0]	12	[4, 5, 2, 0]	21	[5, 0, 1, 0]
4	[4, 4, 4, 1]	13	[5, 6, 3, 1]	22	[6, 1, 2, 1]
5	[5, 5, 0, 2]	14	[6, 0, 4, 2]	23	[7, 2, 3, 2]
6	[6, 6, 1, 0]	15	[7, 1, 0, 0]	24	[0, 3, 4, 0]
7	[7, 0, 2, 1]	16	[0, 2, 1, 1]	25	[1, 4, 0, 1]
8	[0, 1, 3, 2]	17	[1, 3, 2, 2]	26	[2, 5, 1, 2]

Residue Number System: Conversions

From standard to RNS:

$$\forall i \quad a_i = A \bmod m_i$$

From RNS to standard:

Using a constructing proof of the Chinese Remainder Theorem (CRT)

$$A = \sum_{i=1}^k a_i M_i |M_i^{-1}|_{m_i} \bmod M$$

where

- $M = \prod_{i=1}^k m_i$, $A < M$
- $M_i = M/m_i$
- $|M_i^{-1}|_{m_i}$ is the inverse of M_i modulo m_i

Residue Number System: Example (2/2)

Operands: $A = 6 = [6, 6, 1, 0]$ and $B = 16 = [0, 2, 1, 1]$

Addition:

- $(6 + 0) \bmod 8 = 6$
- $(6 + 2) \bmod 7 = 1$
- $(1 + 1) \bmod 5 = 2$
- $(0 + 1) \bmod 3 = 1$

Verification: $22 = [6, 1, 2, 1]$

Multiplication:

- $(6 \times 0) \bmod 8 = 0$
- $(6 \times 2) \bmod 7 = 5$
- $(1 \times 1) \bmod 5 = 1$
- $(0 \times 1) \bmod 3 = 0$

Verification: $96 = [0, 5, 1, 0]$

Residue Number System: Summary

Advantages:

- fast addition/subtraction and multiplication (parallel)
- no carry propagation (on the whole format)
- natural way to split large numbers (simple scheduling)
- no order in the elements

Disadvantages:

- difficult comparison ($<$ and $>$)
- difficult division
- difficult sign test
- difficult magnitude computation

Leak Resistant Arithmetic (LRA)⁶

Proposed method:

- number system = **RNS**
- operations = **RNS modular operations**
 - ▶ RNS Montgomery multiplication ($AB \bmod N$)
 - ▶ Exponentiation ($X^E \bmod N$)
 - ▶ Reduction ($A \bmod N$)

LRA allows:

- **permutation** of the elements within the base
- **random choice** of the initial base
- **random change** of the base during the exponentiation

⁶J.-C. Bajard, L. Imbert, P.-Y. Liardet and Y. Teglia. LIRMM research report 03-021

The end. . .

Questions ?

Contacts:

- arnaud.tisserand@ens-lyon.fr
- <http://perso.ens-lyon.fr/arnaud.tisserand/>
- LIP, ENS Lyon. 46 allée d'Italie. F-69364 Lyon cedex 07. France.

Thank you.

Bibliography

Guide to Elliptic Curve Cryptography

D. Hankerson, A. Menezes
and S. Vanstone

2004

Springer

ISBN: 0-387-95273-X

