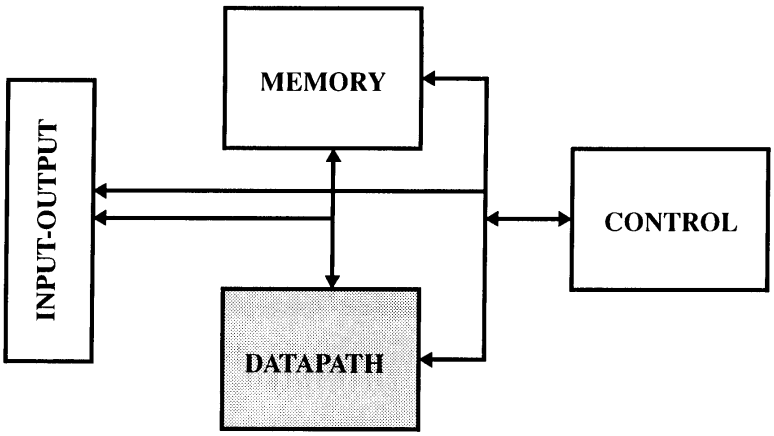
KUNGLTEKNISKA HÖGSKOLAN

CMOS Arithmetic Circuits

Prof. Hannu Tenhunen
Royal Institute of Technology
Department of Electronics
Electronic System Design Laboratory
hannu@ele.kth.se

Arithmetic datapath role



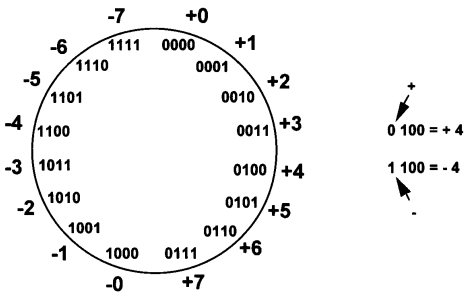
Building blocks for digital architectures

- Arithmetic unit
  - bit-sliced datapath (adder, multiplier, shifter, comparator)
  - bit-serial datapaths (adder, multiplier, shifter, comparator)
- Memory
  - RAM, ROM, buffers, shift registers
- Control
  - finite state machines (PLA, random logic)
  - counters
- Interconnect
  - switches
  - arbiters
  - bus

Binary number representations

Number systems: sign and magnitude

Sign and Magnitude Representation



High order bit is sign: 0 = positive (or zero), 1 = negative

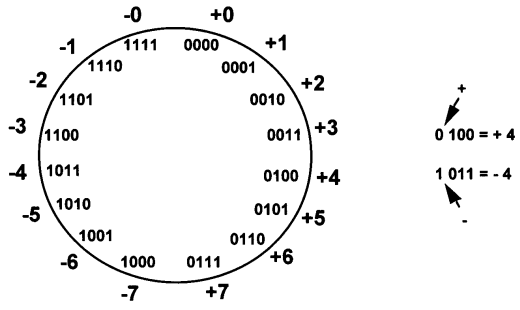
Three low order bits is the magnitude: 0 (000) thru 7 (111)

Number range for n bits = +/-2^{n-1} -1

Representations for 0

Number systems: one’s complement

Ones Complement



Subtraction implemented by addition & 1's complement

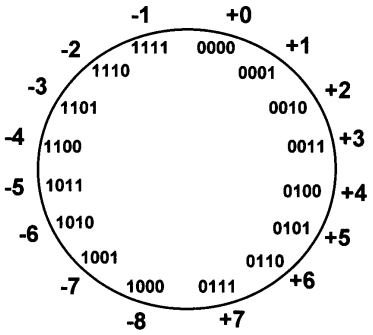
Still two representations of 0! This causes some problems

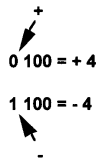
Some complexities in addition

Number systems: two's complement

Twos Complement

like 1's comp except shifted one position clockwise





Only one representation for 0

One more negative number than positive number

Addition and subtraction of numbers

Twos Complement Calculations

	4	0100	-4	1100
	+ 3	0011	+ (-3)	1101
	7	0111	-7	11001

If carry-in to sign = carry-out then ignore carry

if carry-in differs from carry-out then overflow

4	0100	-4	1100
- 3	1101	+ 3	0011
1	10001	-1	1111

Simpler addition scheme makes twos complement the most common choice for integer number systems within digital systems

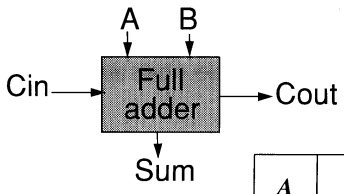
Multiplication of numbers

Partial Product Accumulation

				A3	A2	A1	A0
				B3	B2	B1	B0
				A2 B0	A2 B0	A1 B0	A0 B0
			A3 B1	A2 B1	A1 B1	A0 B1	
		A3 B2	A2 B2	A1 B2	A0 B2		
	A3 B3	A2 B3	A1 B3	A0 B3			
S7	S6	S5	S4	S3	S2	S1	S0

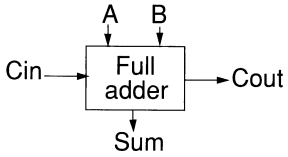
Datapath circuit techniques for adders

Binary adder



A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

Binary adder



Define 3 new variable which ONLY depend on A, B

Generate (G) = AB

Propagate (P) = A ⊕ B

Delete = $\overline{A}\,\overline{B}$

$$S = A \oplus B \oplus C_i$$

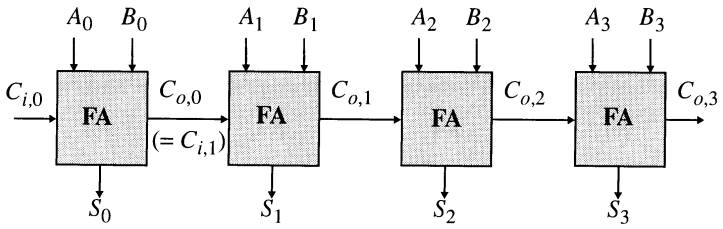
$$= A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i$$

$$C_o = AB + BC_i + AC_i$$

$$C_o(G,P) = G + PC_i$$

$$S(G,P) = P \oplus C_i$$

Ripple carry adder



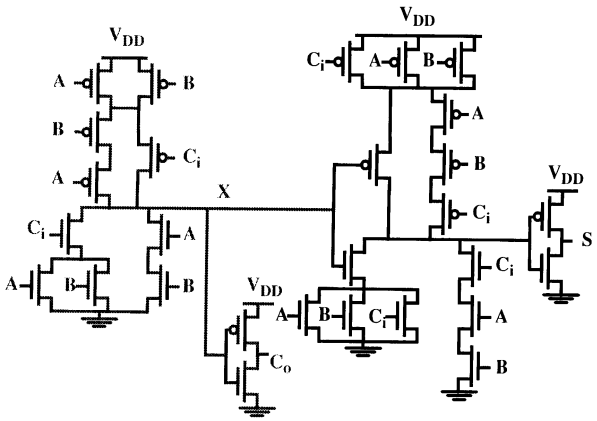
Worst case delay linear with the number of bits

$t_d = O(N)$

$$t_{adder} \approx (N-1)t_{carry} + t_{sum}$$

Goal: Make the fastest possible carry path circuit

CMOS full adder



28 Transistors

Dynamic NP-CMOS full adder

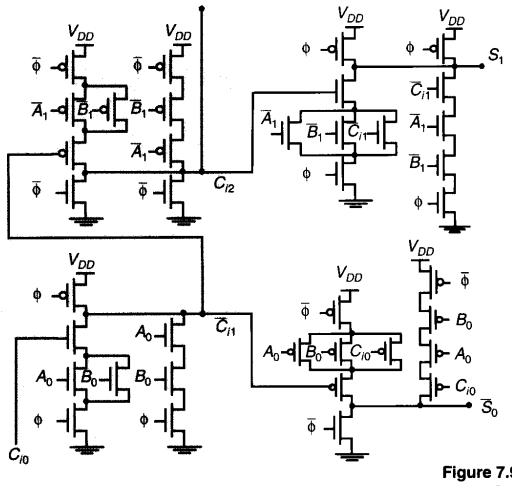


Figure 7.9 Dynamic full adder using the *np* CMOS logic style.

Dynamic NP-CMOS full adder

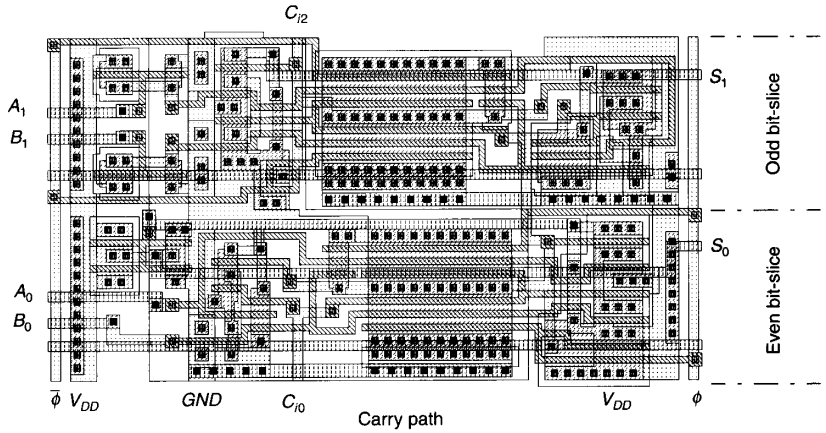


Figure 7.10 Layout of dynamic full adder using the *np*-CMOS logic style (from [Kleinfelder91]).

Pipelined adder

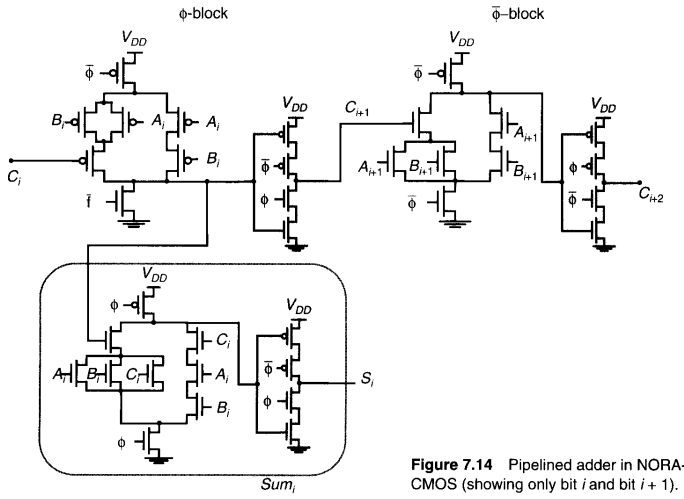
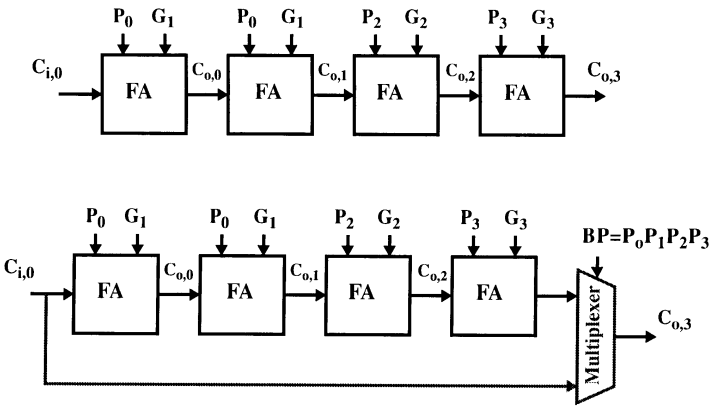


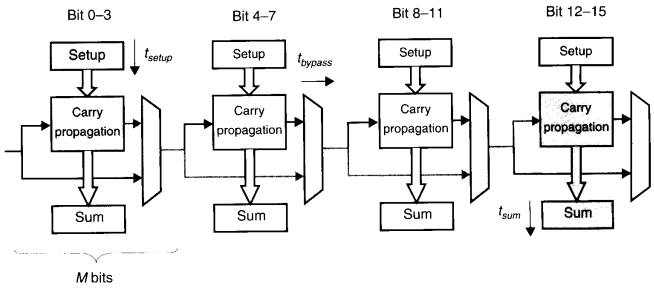
Figure 7.14 Pipelined adder in NORA-CMOS (showing only bit *i* and bit *i* + 1).

Carry bypass adder



Idea: If (P0 and P1 and P2 and P3 = 1)
then $C_{o3} = C_0$, else “kill” or “generate”.

Carry bypass adder



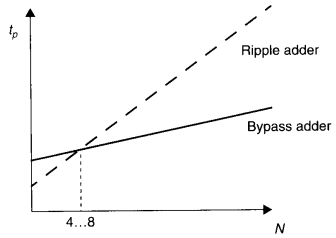
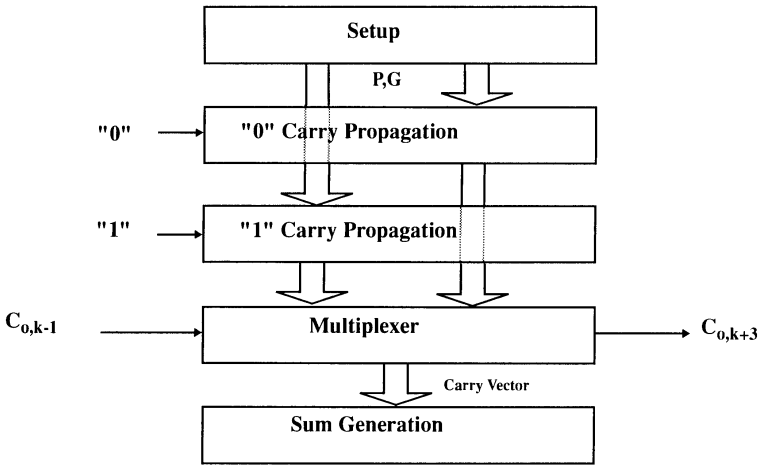
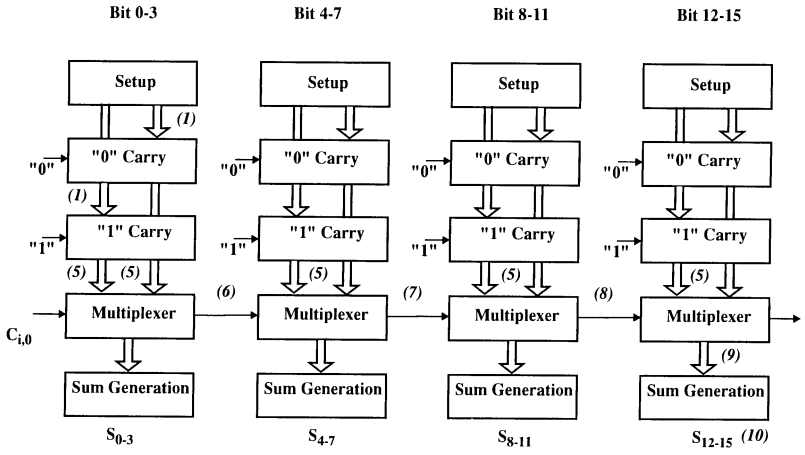


Figure 7.18 Propagation delay of ripple-carry versus carry-bypass adder.

Linear carry select adder



Linear carry select adder: critical path



$$t_{add} = t_{setup} + \left(\frac{N}{M}\right)t_{carry} + Mt_{mux} + t_{sum}$$

Carry look-ahead adder

• THE SIGNALS ASSOCIATED WITH THE ADDER ARE:

GENERATE SIGNAL G_i

$$G_i = A_i \cdot B_i$$

PROPAGATE SIGNAL P_i

$$P_i = A_i + B_i$$

CARRY SIGNAL C_i

$$C_i = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \cdots + P_i \cdots P_1C_0.$$

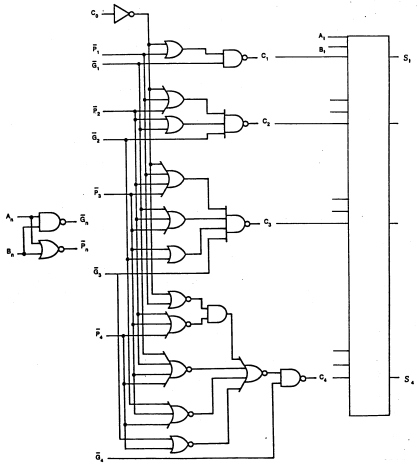
SUM SIGNAL S_i

$$S_i = C_{i-1} \oplus A_i \oplus B_i$$

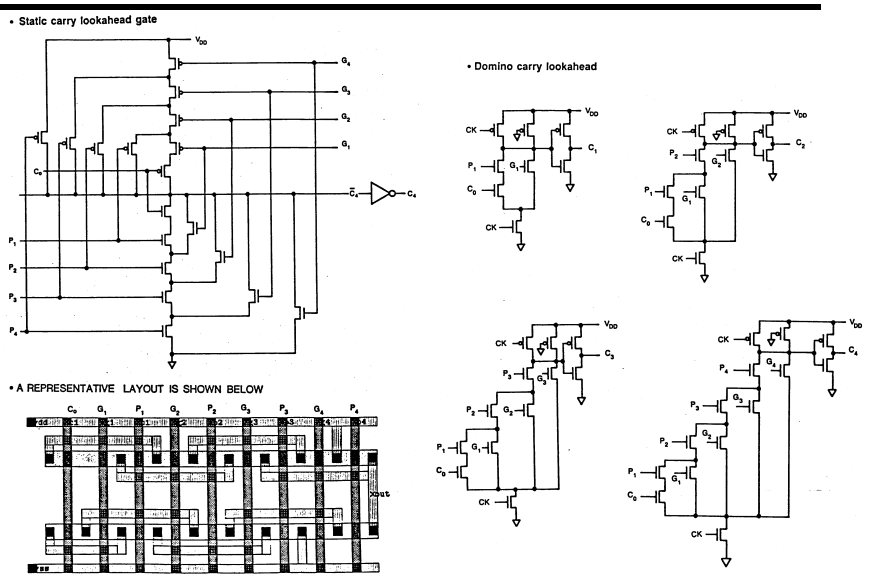
For four stages

$$C_1 = G_1 + P_1C_0$$
$$C_2 = G_2 + P_2G_1 + P_2P_1C_0$$
$$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_0$$
$$C_4 = G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1 + P_4P_3P_2P_1C_0$$
$$= G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot C_0))).$$

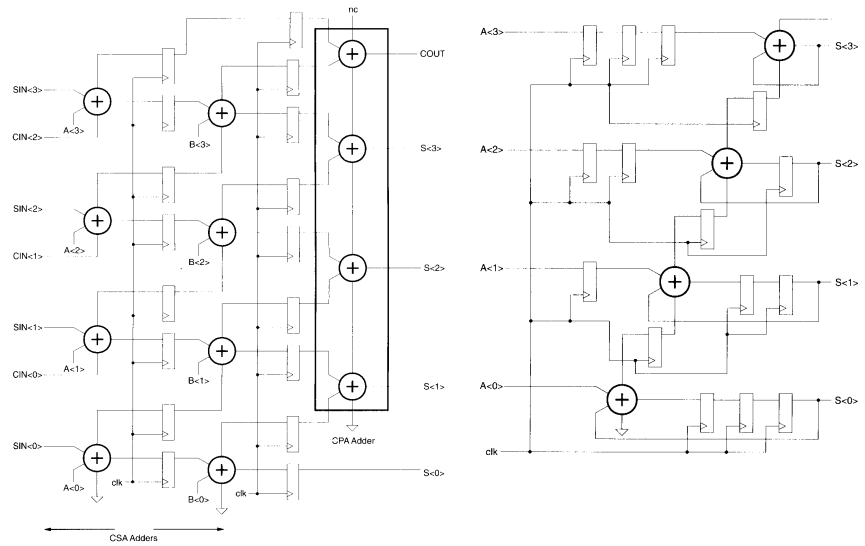
• THE FOLLOWING IS A SCHEMATIC OF A 4-BIT LOOKAHEAD ADDER WITH GATES DECOMPOSED TO YIELD GATES WITH A MAXIMUM OF FOUR TRANSISTORS IN SERIES.



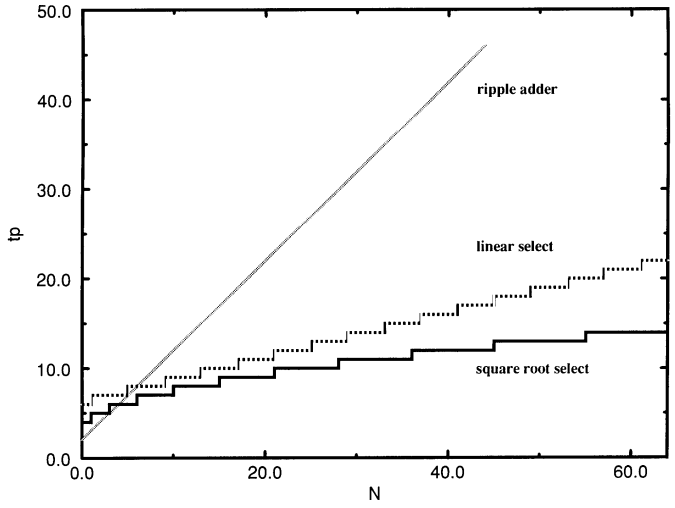
Carry look-ahead circuit structures



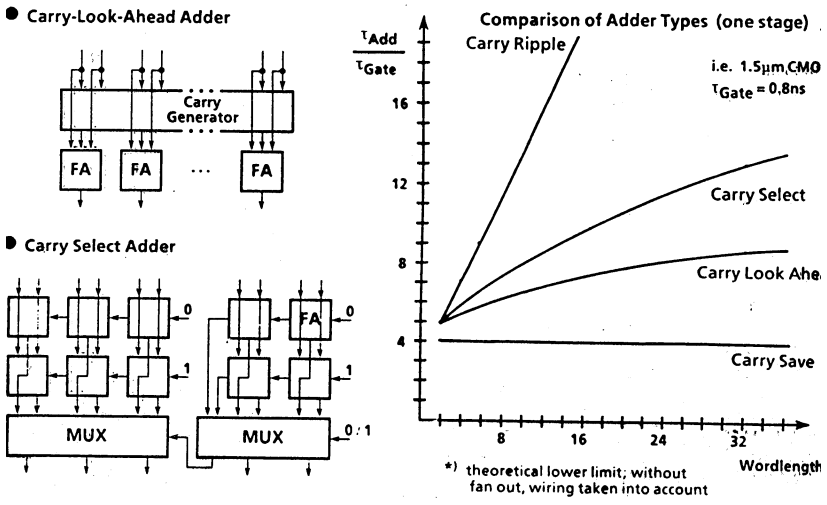
Carry save (CSA) and carry propagate (CPA) adders



Adder delays



Adder delays summary



Datapath circuit techniques for multipliers

Multiplier definition

Consider two *unsigned* binary numbers X and Y that are M and N bits wide respectively. To introduce the multiplication operation, it is useful to express X and Y in a binary representation.

$$X = \sum_{i=0}^{M-1} X_i 2^i \qquad Y = \sum_{j=0}^{N-1} Y_j 2^j \tag{7.23}$$

with $X_i, Y_j \in \{0, 1\}$. The multiplication operation is then defined as follows:

$$\begin{aligned} Z = X \times Y &= \sum_{k=0}^{M+N-1} Z_k 2^k \\ &= \left(\sum_{i=0}^{M-1} X_i 2^i\right)\left(\sum_{j=0}^{N-1} Y_j 2^j\right) = \sum_{i=0}^{M-1}\left(\sum_{j=0}^{N-1} X_i Y_j 2^{i+j}\right) \end{aligned} \tag{7.24}$$

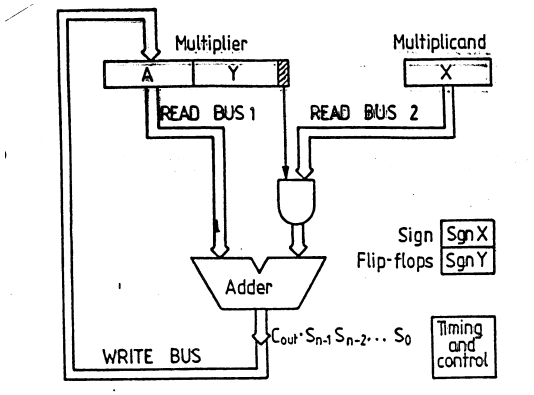
Binary multiplication

```
          1 0 1 0 1 0
×           1 0 1 1
-----------------------
          1 0 1 0 1 0
        1 0 1 0 1 0
      0 0 0 0 0 0
+   1 0 1 0 1 0
-----------------------
  1 1 1 0 0 1 1 1 0
```

AND operation

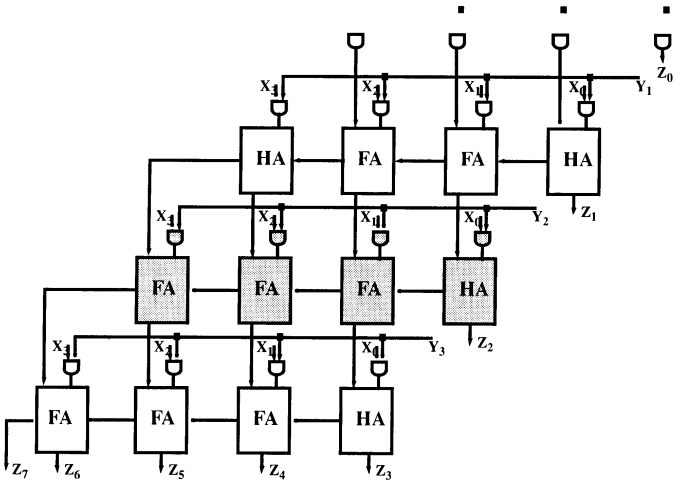
Partial Products

Indirect multiplication

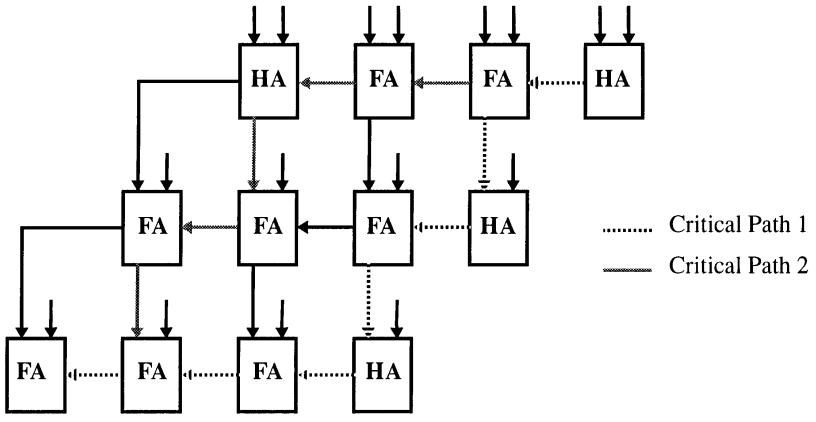


1. Set $A = 0$
2. Read the LSB of Y register
3. AND it with the multiplicand X and add the result to Y
4. Store the sum to A and shift the contents of A and Y one bit right
5. Repeat 2. to 4. until all bits of the multiplier are handled
6. Read the result from A and Y registers

Array multiplier

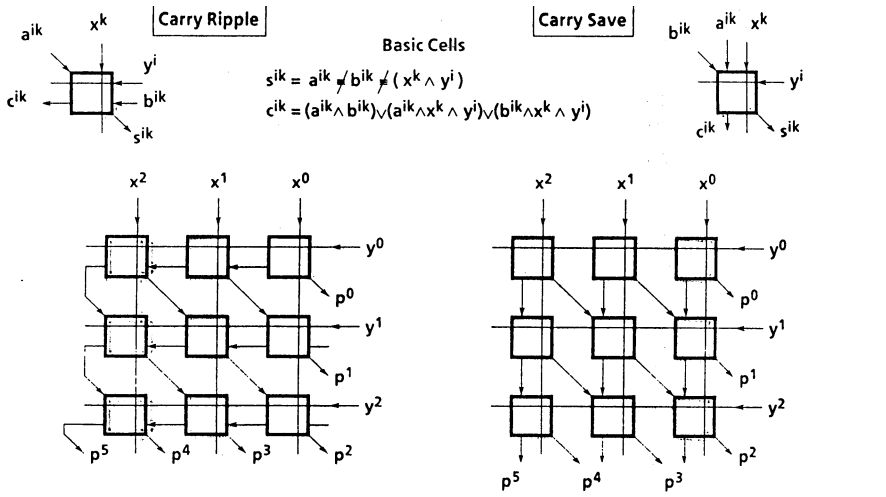


MxN array multiplier critical path

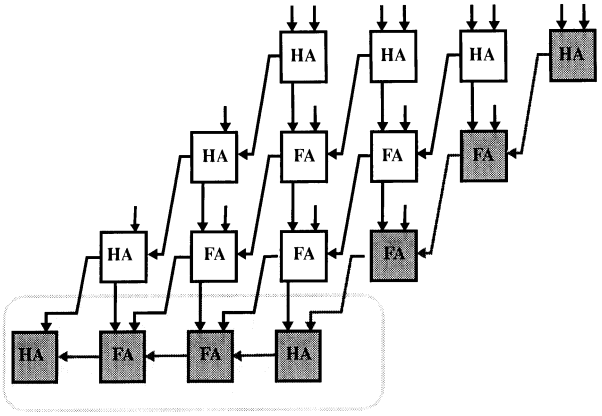


$$t_{mult} \approx [(M-1)+(N-2)]t_{carry} + (N-1)t_{sum} + (N-1)t_{and}$$

Carry ripple vs. carry save array multiplier

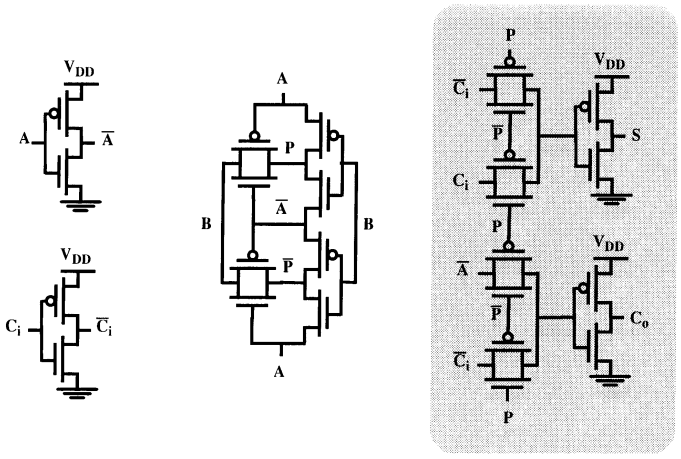


Carry save multiplier



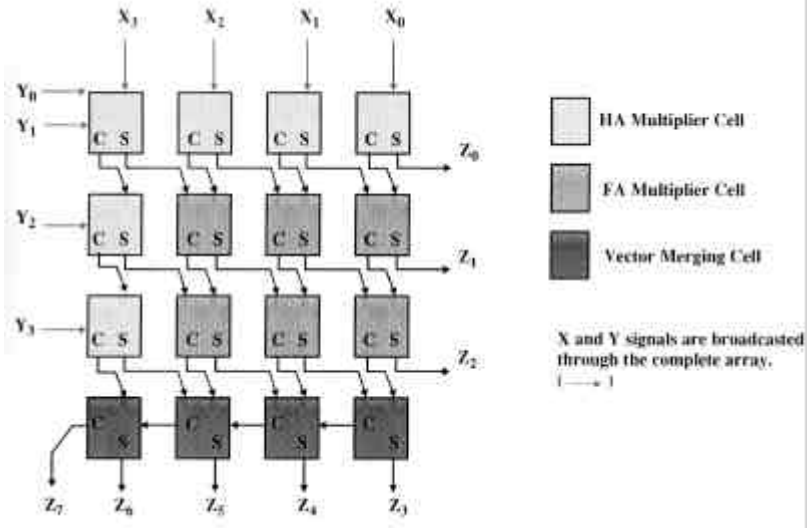
$$t_{mult} = [(N-1)t_{carry} + (N-1)t_{and} + t_{merge}]$$

Adder cells in array multiplier



Identical Delays for Carry and Sum

Array multiplier floorplan



Wallace tree multiplier

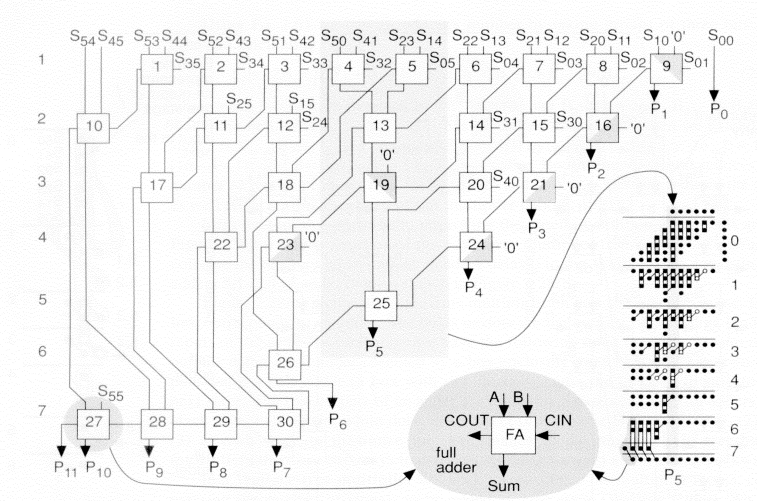
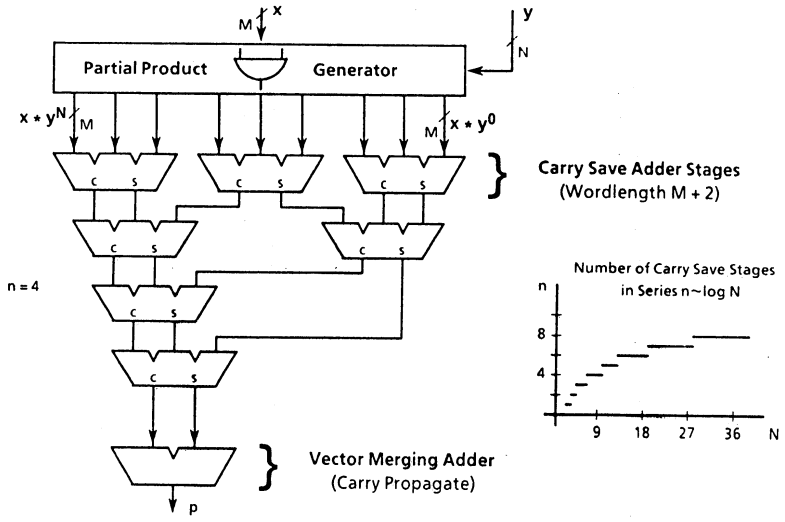
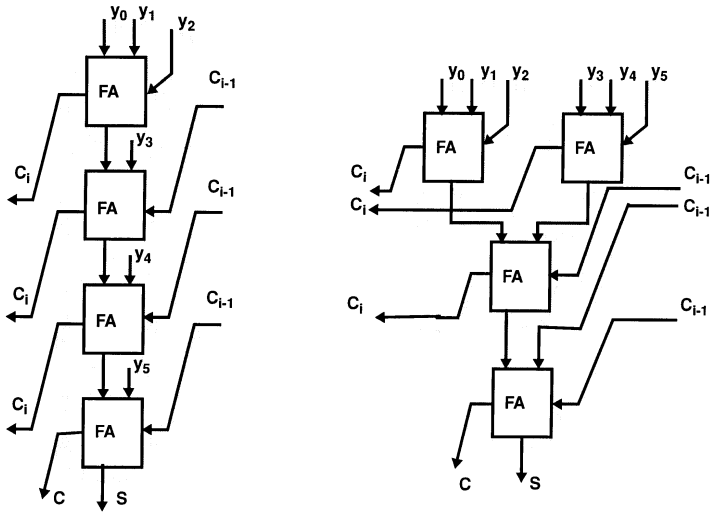


FIGURE 2.29 A 6-bit Wallace-tree multiplier. The carry-save adder (CSA) requires 26 adders (cells 1–26, six are half adders). The final carry-propagate adder (CPA) consists of 4 adder cells (27–30). The delay of the CSA is 6 adders. The delay of the CPA is 4 adders.

Wallace tree multiplier



Wallace tree multiplier



Dadda tree multiplier

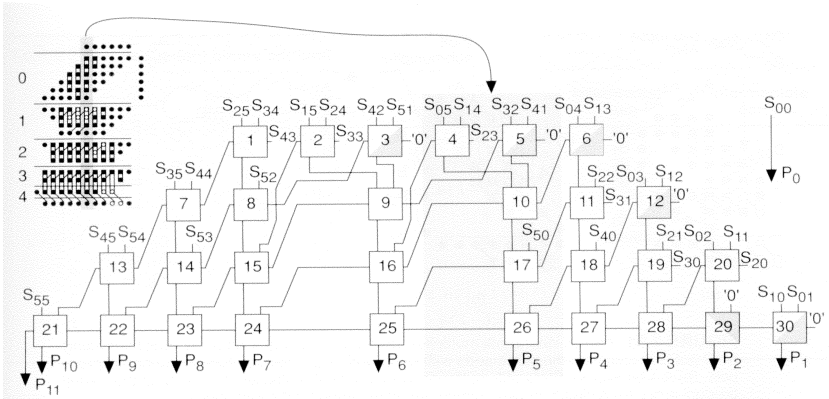
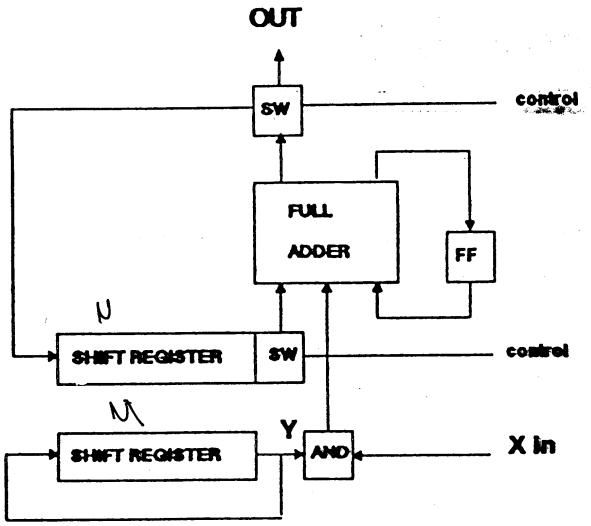
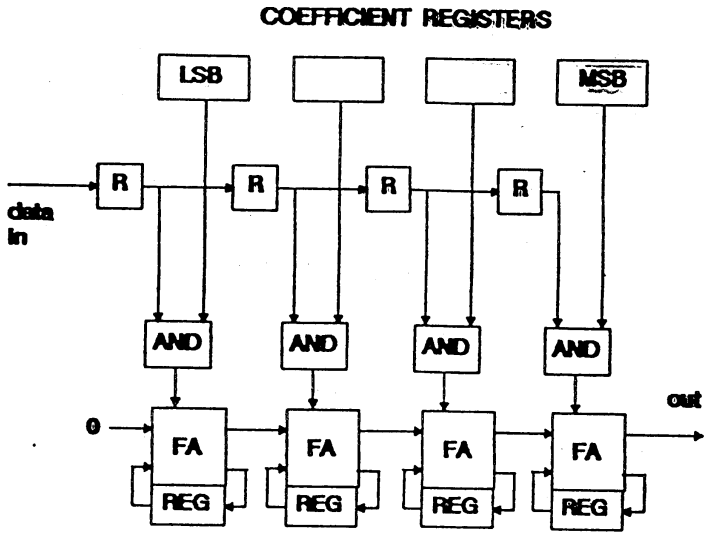


FIGURE 2.30 The 6-bit Dadda multiplier. The carry-save adder (CSA) requires 20 adders (cells 1–20, four are half adders). The carry-propagate adder (CPA, cells 21–30) is a ripple-carry adder (RCA). The CSA is smaller (20 versus 26 adders), faster (3 adder delays versus 6 adder delays), and more regular than the Wallace-tree CSA of Figure 2.29. The overall speed of this implementation is approximately the same as the Wallace-tree multiplier of Figure 2.29; however, the speed may be increased by substituting a faster CPA.

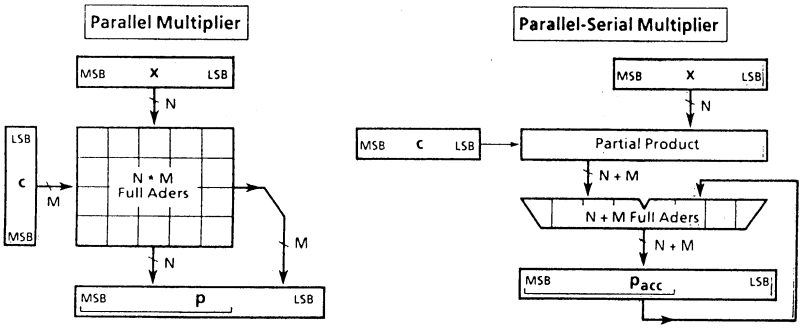
Serial-serial multiplier



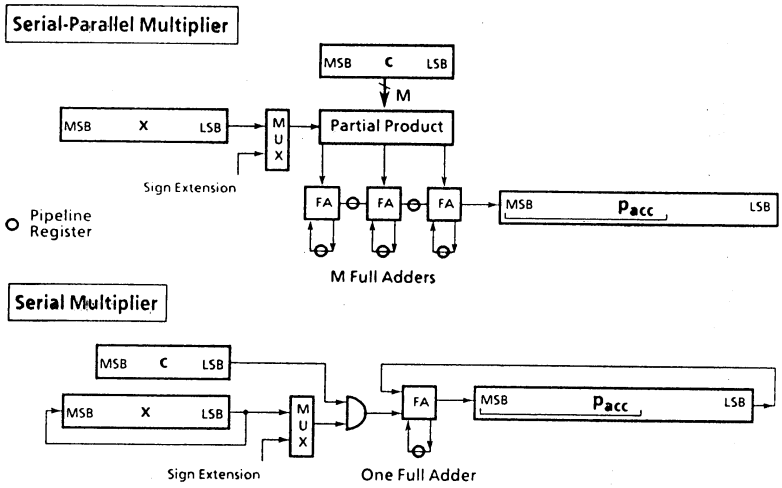
Serial-parallel multiplier



Parallel vs. serial multipliers



Parallel vs. serial multipliers



Multiplier performance

MULTIPLICAND	MULTIPLIER	CLOCK PULSES	FULL ADDERS	AND-GATES
parallel	parallel	1	n^2	n^2
parallel	serial	n	n	n
serial	parallel	$2n$	n	n
serial	serial	$n(n+1)$ or $2n^2$	1	1

Multiplier performance

Multiplier type	N×N	8×8	16×16	32×32
Indirect with RCA	(2N+2)NT	144T	544T	2112T
Linear array with RCA	(4N-2)T	30T	62T	126T
Linear array with CLA	(2N+8)T	24T	40T	72T
Wallace tree with CLA	(2M+8)T	16T	20T	24T
Pipelined *) linear array	> 2T	> 2T	> 2T	> 2T
Logarithmic with CLA	2T(m)+8T	2T(m)+8T	---	---

N is the input word length
M is the number of FA stages in Wallace tree structure
T is the basic gate delay
T(m) is the ROM access time in logarithmic multiplier

Assumed delays in this table:

Full adder delay	2T
CLA delay	8T

*) 1/throughput values are considered in this table

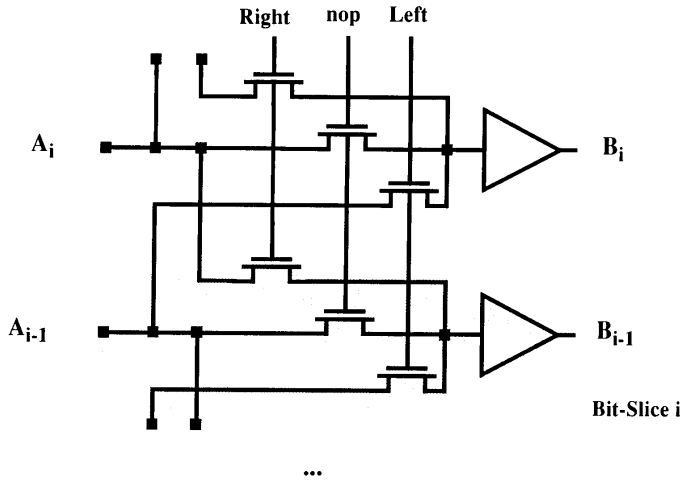
Multiplier summary

- **Optimization Goals Different Vs Binary Adder**
- **Once Again: Identify Critical Path**
- **Other possible techniques**
  - **Logarithmic versus Linear (Wallace Tree Mult)**
  - **Data encoding (Booth)**
  - **Pipelining**

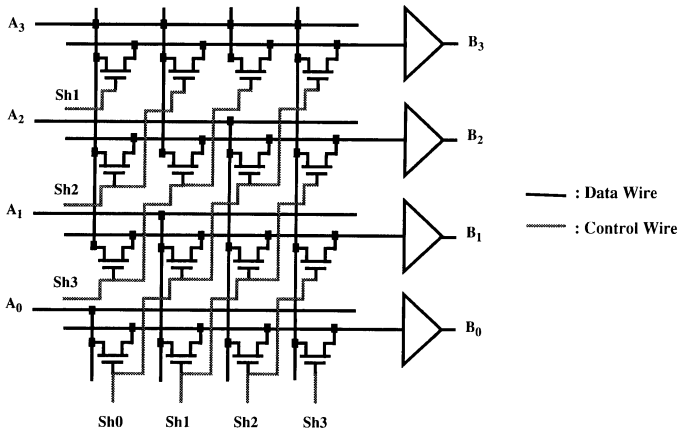
FIRST GLIMPSE AT SYSTEM LEVEL OPTIMIZATION

Other datapath elements

Binary shifter

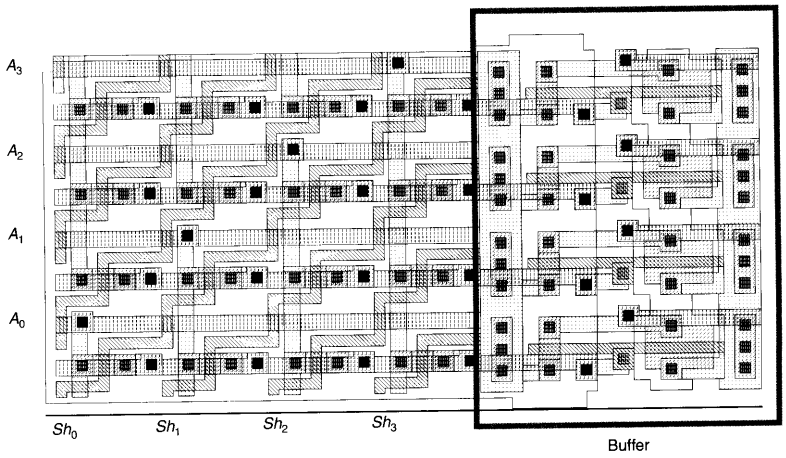


Barrel shifter

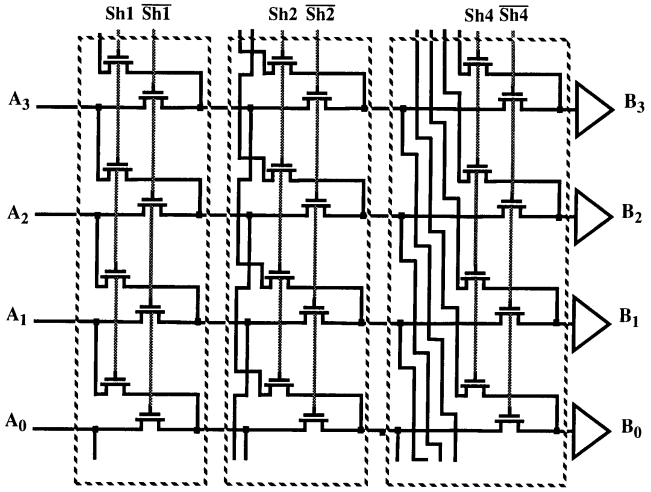


Area dominated by Wiring!

4x4 barrel shifter



Logarithmic shifter



Power considerations in datapath structures

Reducing supply voltage

An architectural approach

- **Operate at lowest supply voltage at lowest possible speed!**
- **Use Architecture optimization to compensate for slower operation.**

  e.g. Concurrency, Pipelining

Trade-off AREA for lower POWER

Reducing supply voltage

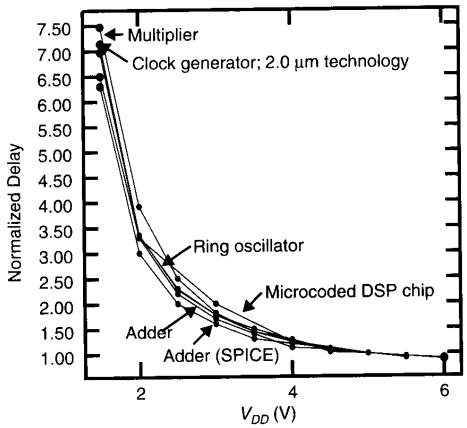
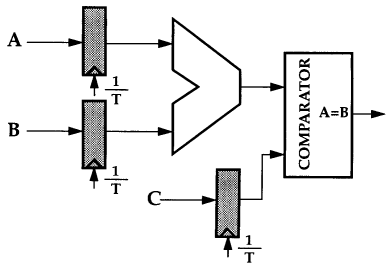
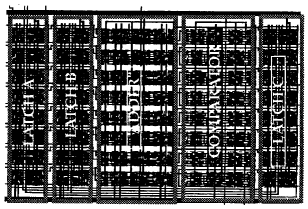


Figure 4.56 Normalized propagation delay as a function of supply voltage for a number of experimental circuits.

Architecture trade-offs: reference datapath

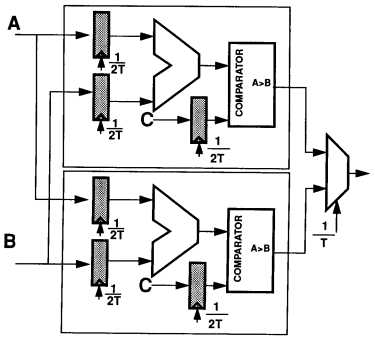


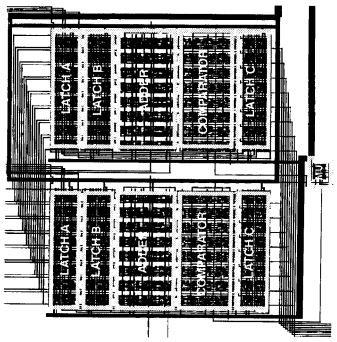


Area = 636 x 833 u^2

- **Critical path delay $\Rightarrow T_{adder} + T_{comparator}$ (= 25ns)**
  $\Rightarrow f_{ref}$ = 40Mhz
- **Total capacitance being switched = C_{ref}**
- **$V_{dd} = V_{ref}$ = 5V**
- **Power for reference datapath = $P_{ref} = C_{ref} V_{ref}^2 f_{ref}$**

Parallel datapath

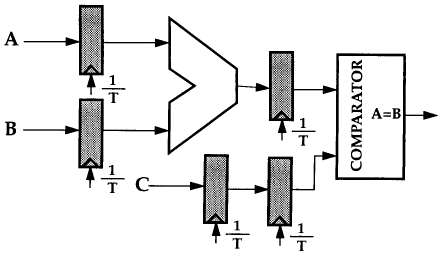


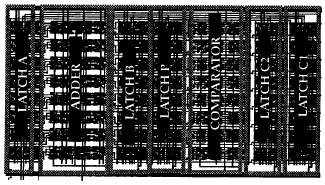


Area = 1476 x 1219 μ^2

- The clock rate can be reduced by half with the same throughput $\Rightarrow f_{par} = f_{ref} / 2$
- $V_{par} = V_{ref} / 1.7$, $C_{par} = 2.15C_{ref}$
- $P_{par} = (2.15C_{ref}) (V_{ref}/1.7)^2 (f_{ref}/2) \approx 0.36\ P_{ref}$

Pipelined datapath





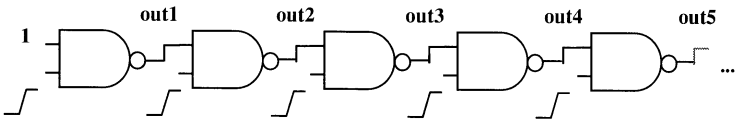
Area = 640 x 1081 u^2

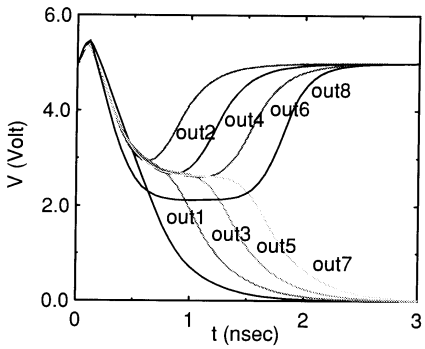
- $f_{pipe} = f_{ref}$
  $C_{pipe} = 1.1C_{ref}$
  $V_{pipe} = V_{ref}/1.7$
- Voltage can be dropped while maintaining the original throughput.
- $P_{pipe} = C_{pipe} V_{pipe}^2 f_{pipe} = (1.1C_{ref}) (V_{ref}/1.7)^2 f_{ref} = 0.37\ P_{ref}$

Datapath architecture summary

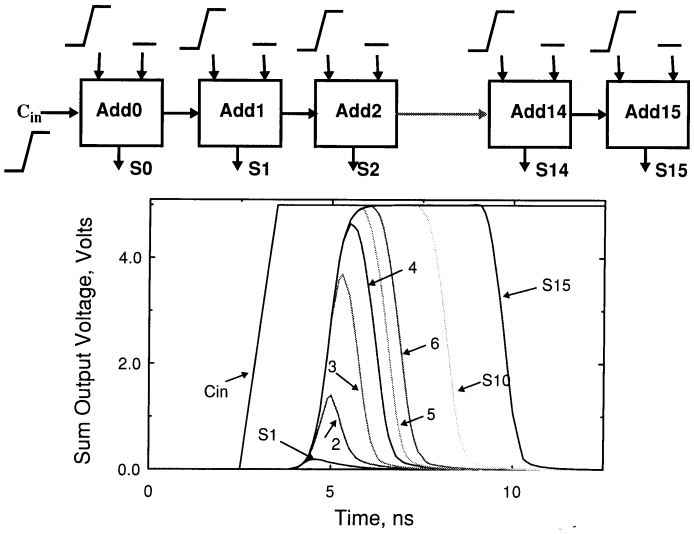
Architecture type	Voltage	Area	Power
Simple datapath (no pipelining or parallelism)	5V	1	**1**
Pipelined datapath	2.9V	1.3	0.37
Parallel datapath	2.9V	3.4	0.34
Pipeline-Parallel	2.0V	3.7	**0.18**

Glitching in NOR chain





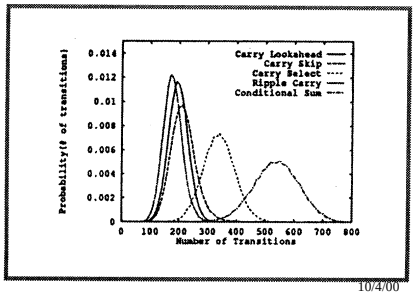
Glitching in RCA



Switching activity in adders

Power-Delay-Product^{-1}

	16 bit	32 bit	64 bit
Ripple Carry	3.09	0.81	0.27
Carry Lookahead	10.0	3.54	1.76
Carry Bypass	5.45	2.39	0.99
Carry Select	4.44	2.08	1.00
Conditional Sum	3.82	1.23	0.42



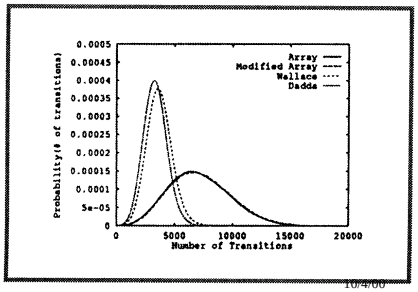
Logic Transition Histogram

from Callaway & Swartzlander

Switching activity in multipliers

Power-Delay-Product^{-1}

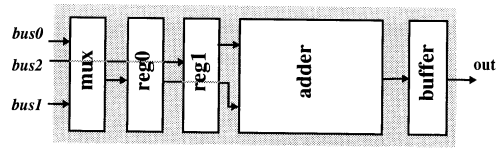
	8 bit	16 bit	32 bit
Array	32.01	1.15	0.04
Wallace	49.86	5.06	0.81
Dadda	51.3	5.79	0.95

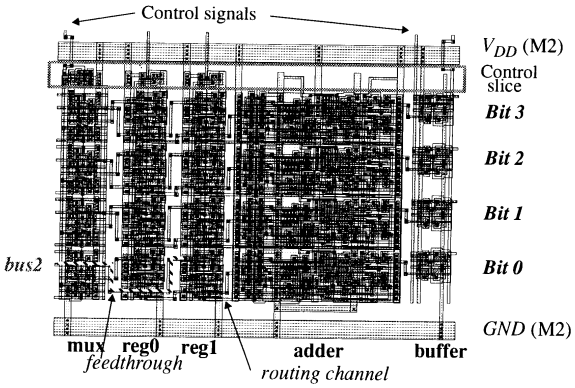


Logic Transition Histogram

from Callaway & Swartzlander

Layout strategy for datapath



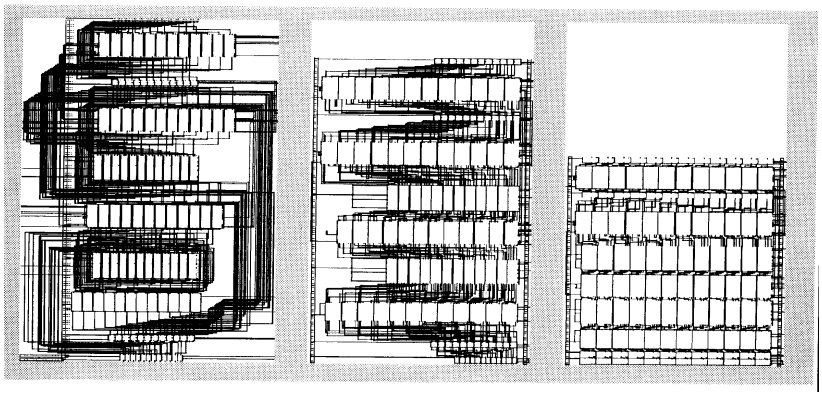


Layout strategy for datapaths

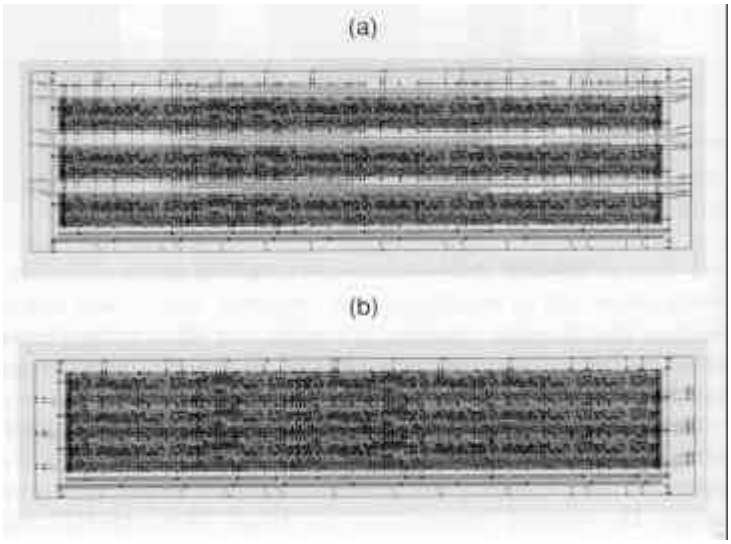
(a) Datapath without feedthroughs and without pitch matching (area = 4.2 mm^2).

(b) Adding feedthroughs (area = 3.2 mm^2)

(c) Equalizing the cell height reduces the area to 2.2 mm^2.



Cell area: 2 vs. 3 metal layer process



Summary

1. The most important message is to select the *right structure* before starting an elaborate circuit optimization. Going for the optimal performance of a complex structure by rigorously optimizing transistor sizes and topologies might not always be the best choice. Optimizations at higher levels of abstraction, such as the logic level, can often generate more dramatic results. Simple first-order calculations can help give a global picture on the pros and cons of a proposed structure.
2. Determine the *critical timing path* through the circuit and focus most of your optimization efforts on that part of the circuit. Computer-aided design tools are available to help determine the critical paths and size the transistors appropriately.
3. Circuit size is not always determined by the number of transistors, but also by other factors such as *wiring and the number of vias and contacts*. These factors are becoming even more important with shrinking dimensions or when extreme performance is a goal, as will become obvious in the next chapter.
4. Although an obscure optimization can sometimes help to get a better result, be wary if this results in an irregular and convoluted topology. *Regularity and modularity* are a designer's best friend.