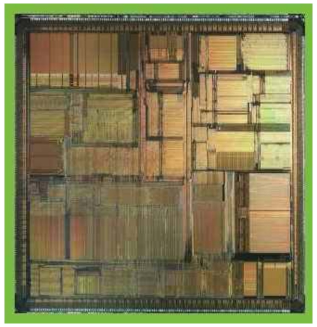
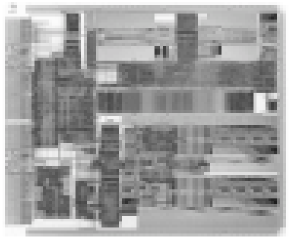
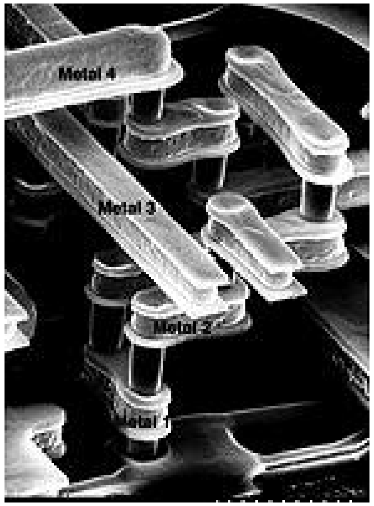




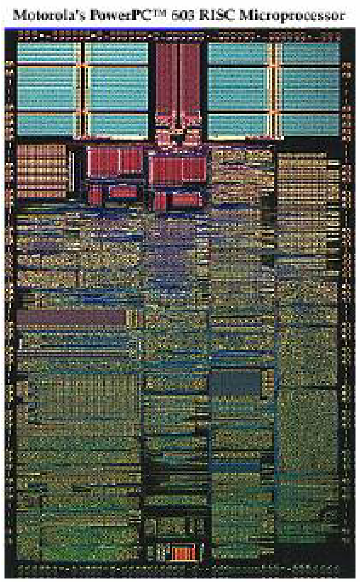
VLSI Arithmetic

Lecture 9: Carry-Save and Multi-Operand Addition

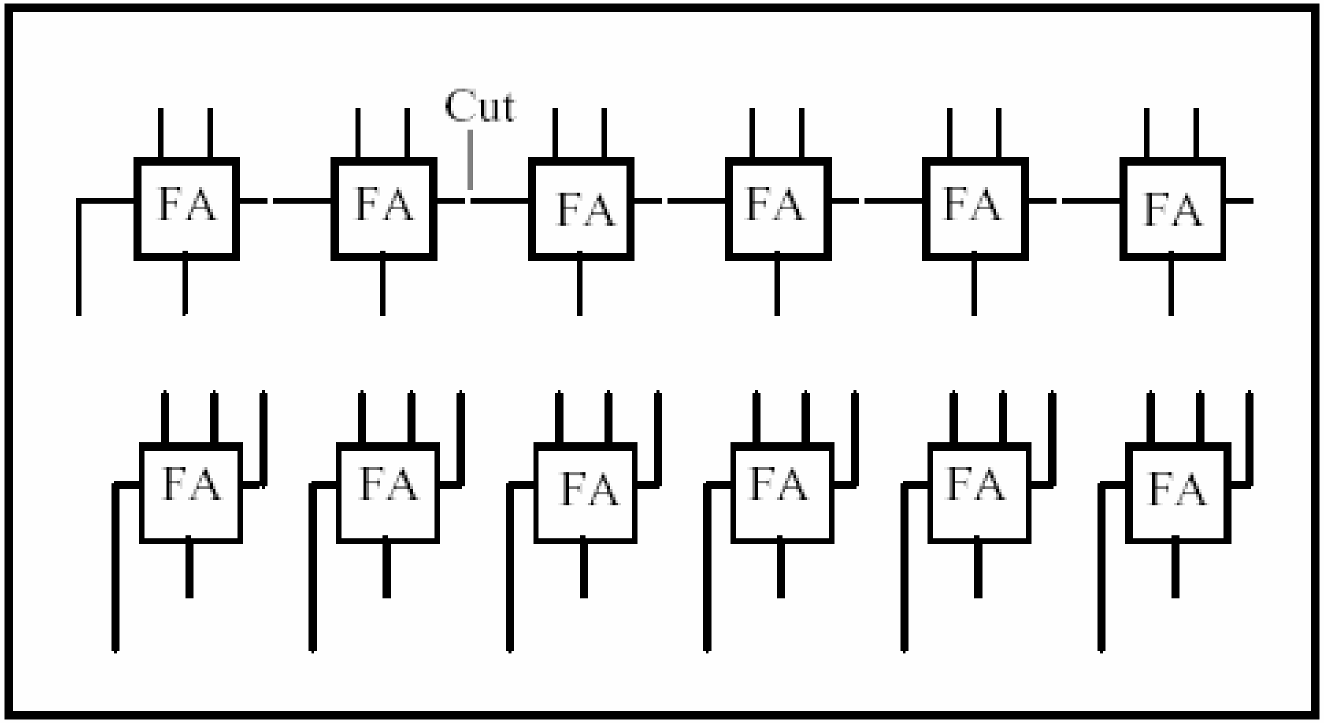


Prof. Vojin G. Oklobdzija
University of California

http://www.ece.ucdavis.edu/acsel



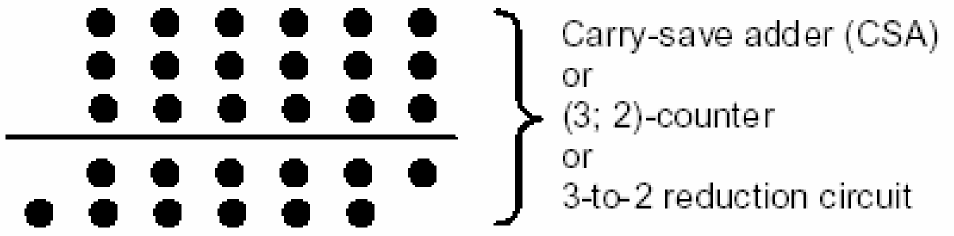
Carry-Save Addition*



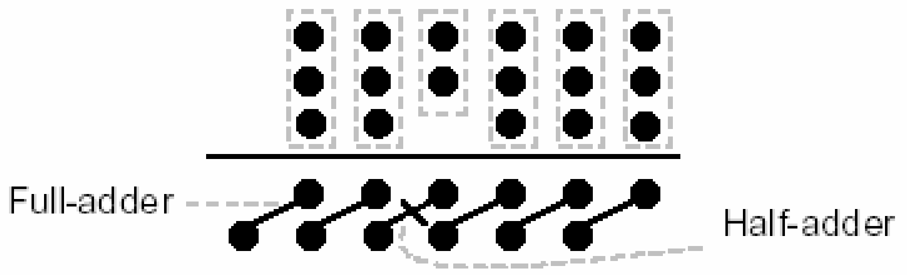
A ripple-carry adder turns into a carry-save adder if the carries are saved (stored) rather than propagated.

**from Parhami*

Carry-Save Addition*



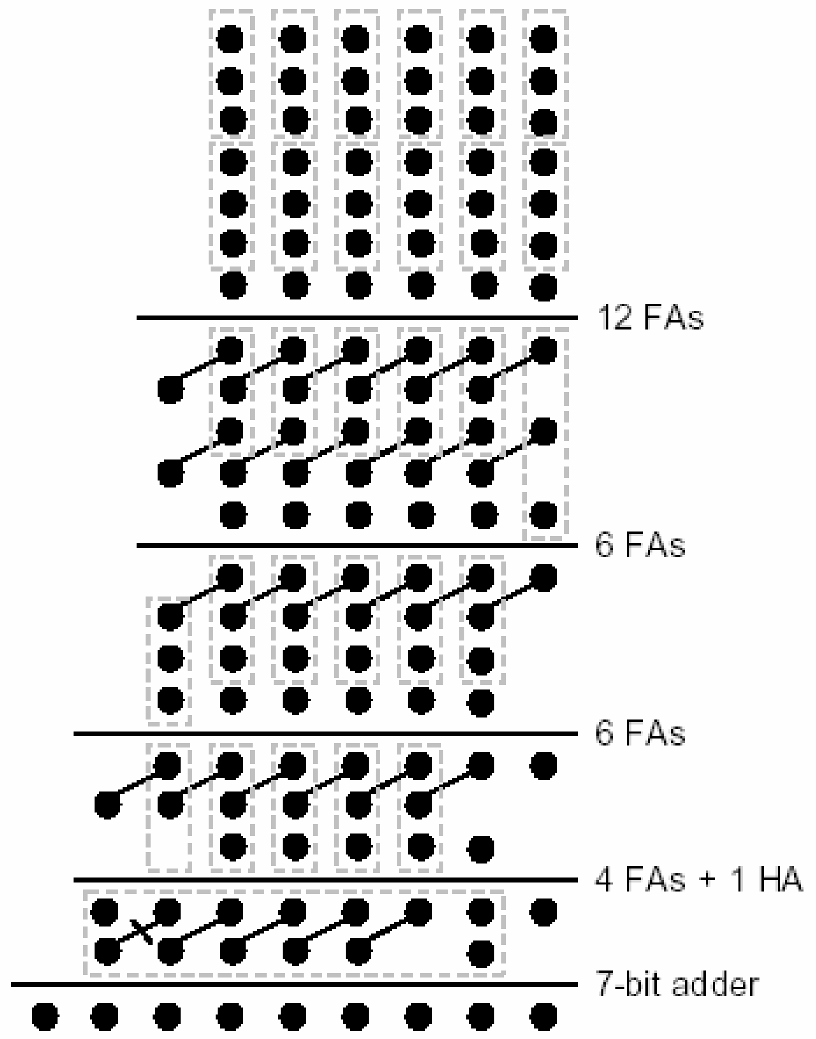
Carry-propagate adder (CPA) and carry-save adder (CSA) functions in dot notation.



Specifying full- and half-adder blocks, with their inputs and outputs, in dot notation.

**from Parhami*

Carry-Save Addition*

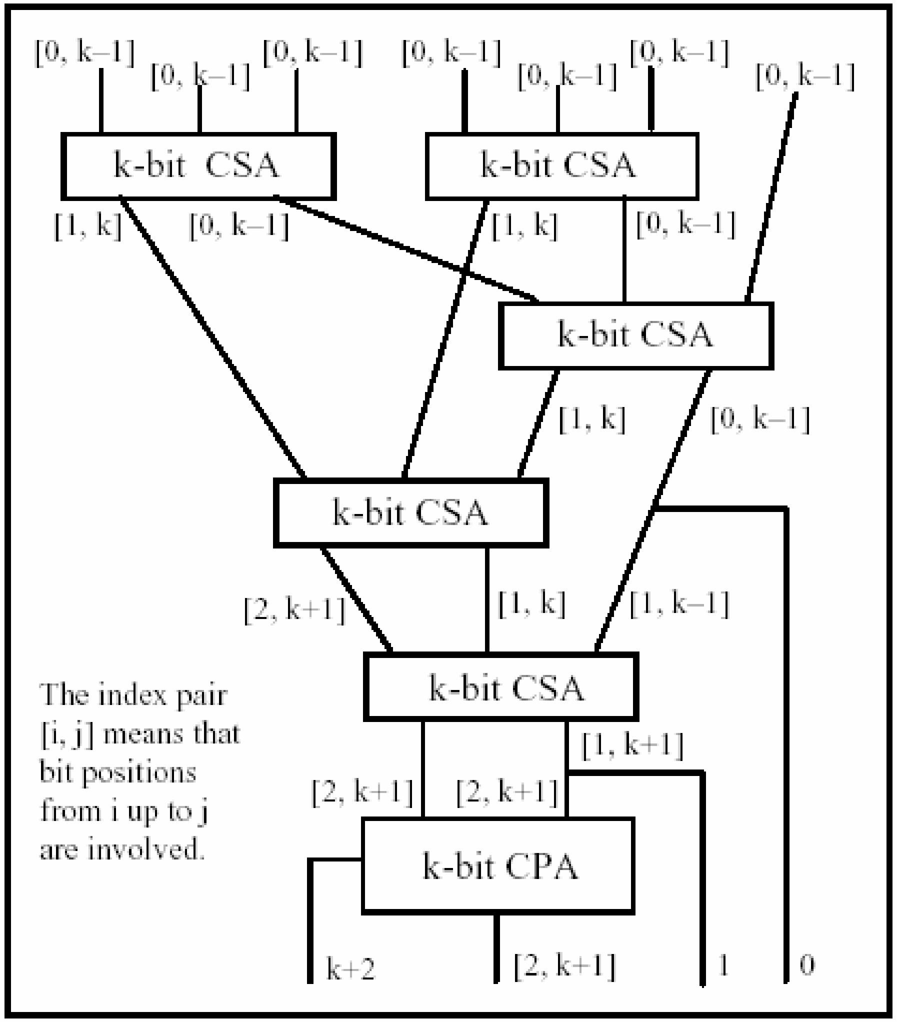


Total cost = 7-bit adder + 28 FAs + 1 HA

*from Parhami

Addition of seven 6-bit numbers in dot notation.

Carry-Save Addition*

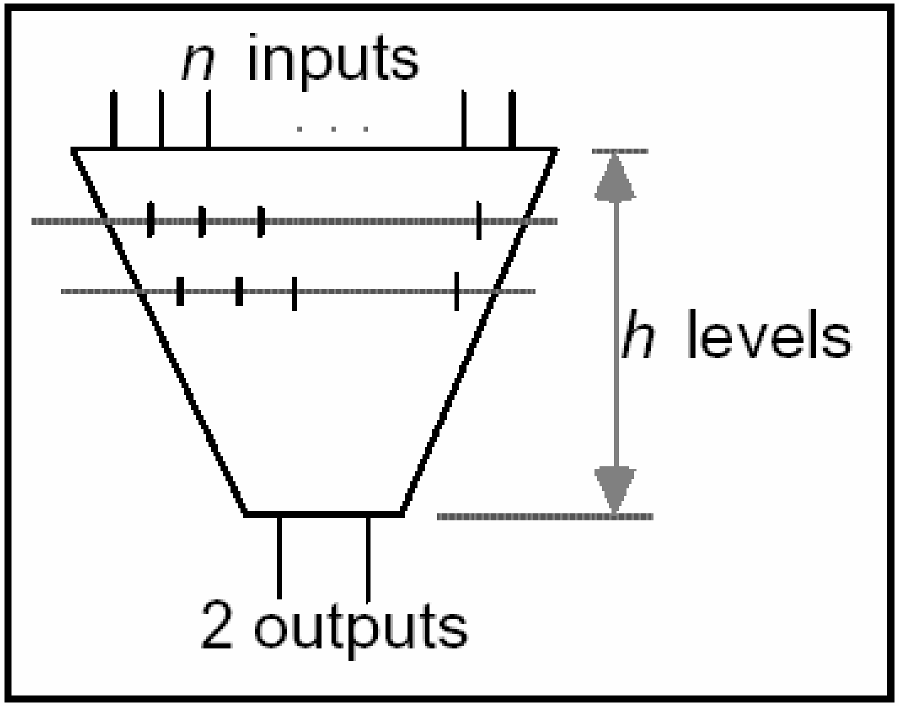


Adding seven *k*-bit numbers and the CSA/CPA widths required.

**from Parhami*

Carry-Save Addition *

Wallace and Dadda Trees



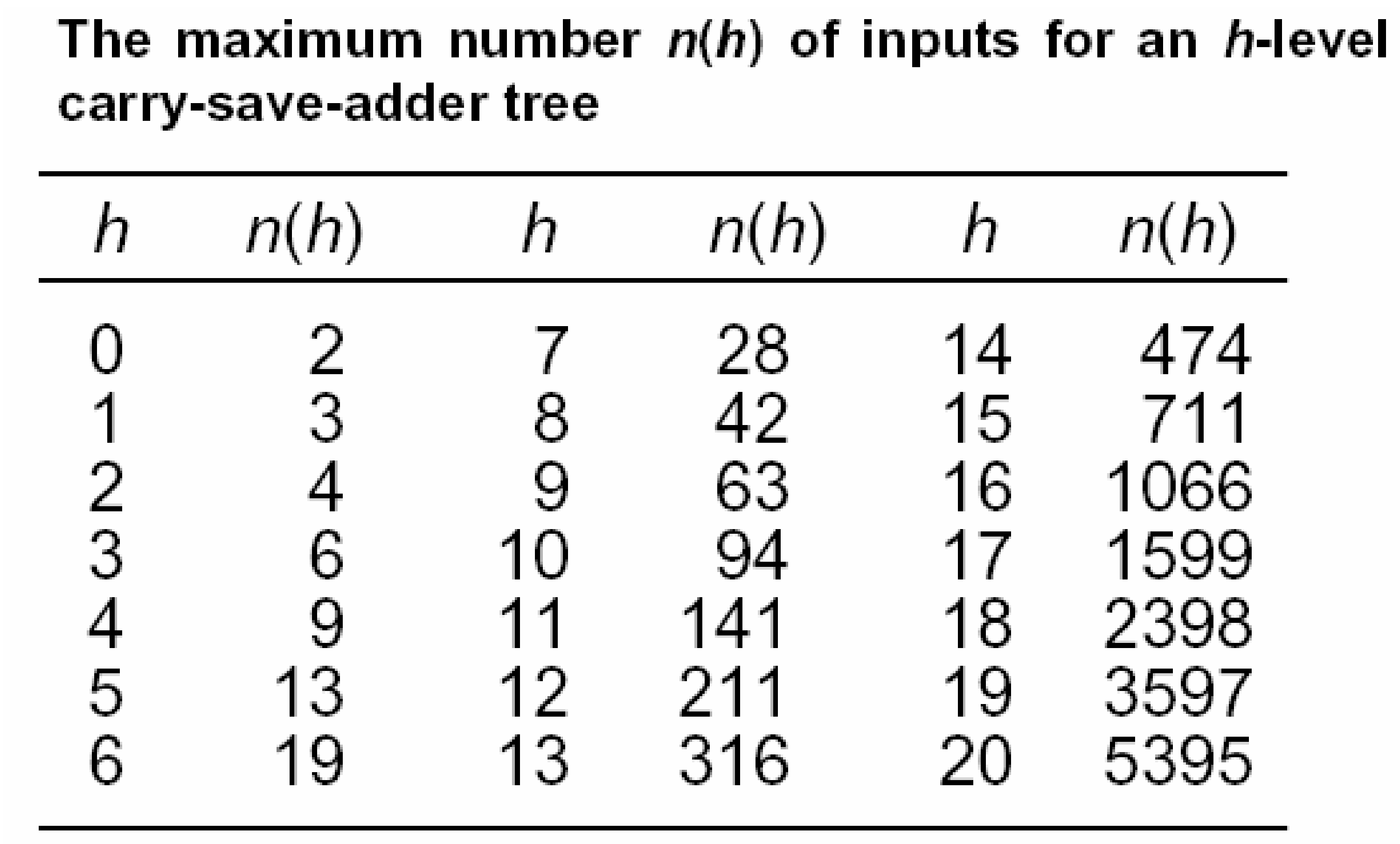
$$h(n) = 1 + h(\lceil 2n/3 \rceil)$$

$$n(h) = \lfloor 3n(h-1)/2 \rfloor$$

$$2 \times 1.5^{h-1} < n(h) \leq 2 \times 1.5^{h}$$

**from Parhami*

Carry-Save Addition *



The maximum number $n(h)$ of inputs for an h-level carry-save-adder tree

h	$n(h)$	h	$n(h)$	h	$n(h)$
0	2	7	28	14	474
1	3	8	42	15	711
2	4	9	63	16	1066
3	6	10	94	17	1599
4	9	11	141	18	2398
5	13	12	211	19	3597
6	19	13	316	20	5395

**from Parhami*

Carry-Save Addition *

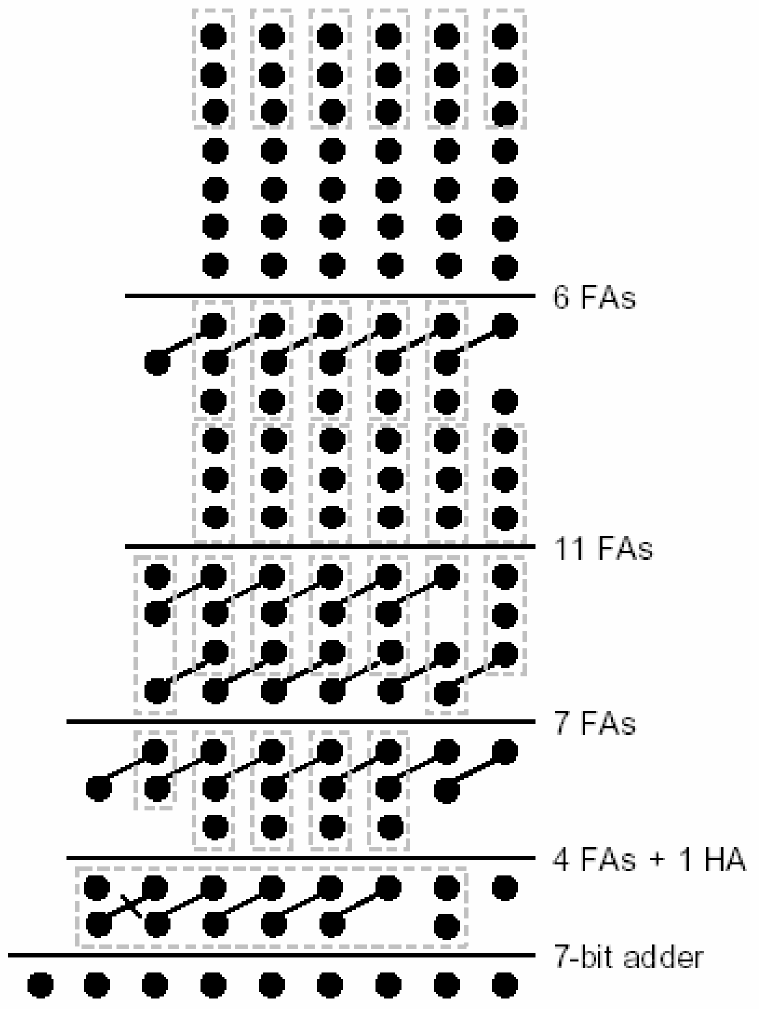
In a Wallace tree, we reduce the number of operands at the earliest possible opportunity

In a Dadda tree, we reduce the number of operands at the latest possible opportunity that leads to no added delay (target the next smaller number in Table 8.1)

h	$n(h)$	h	$n(h)$	h	$n(h)$
0	2	7	28	14	474
1	3	8	42	15	711
2	4	9	63	16	1066
3	6	10	94	17	1599
4	9	11	141	18	2398
5	13	12	211	19	3597
6	19	13	316	20	5395

**from Parhami*

Carry-Save Addition *

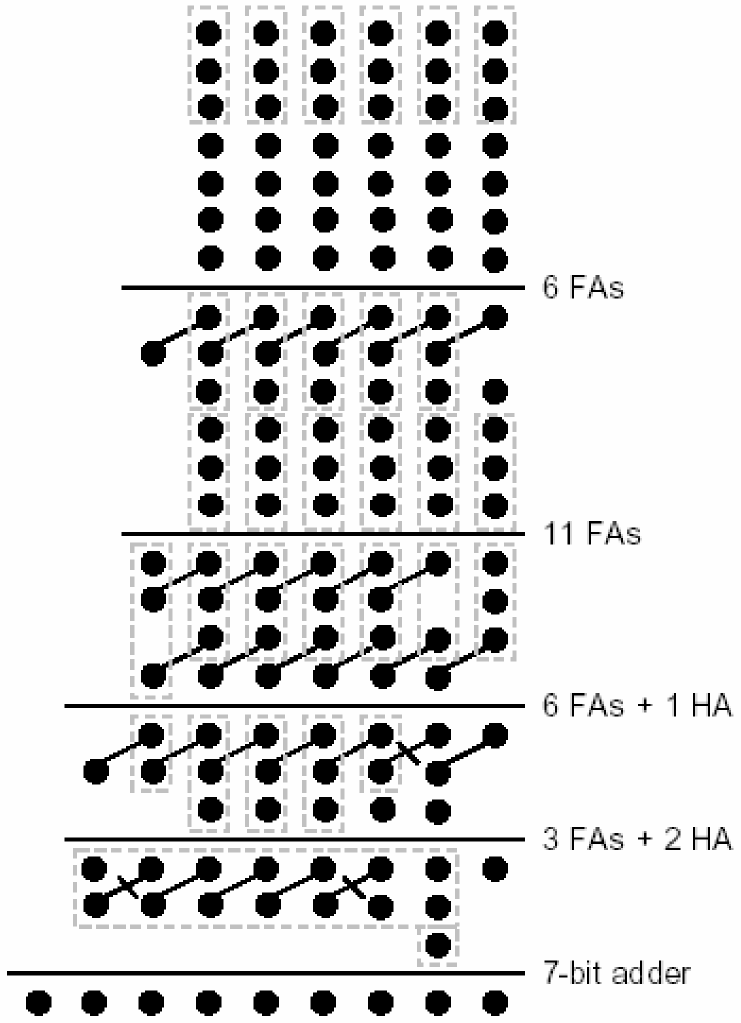


Total cost = 7-bit adder + 28 FAs + 1 HA

**from Parhami*

Adding seven 6-bit numbers using Dadda's strategy.

Carry-Save Addition *



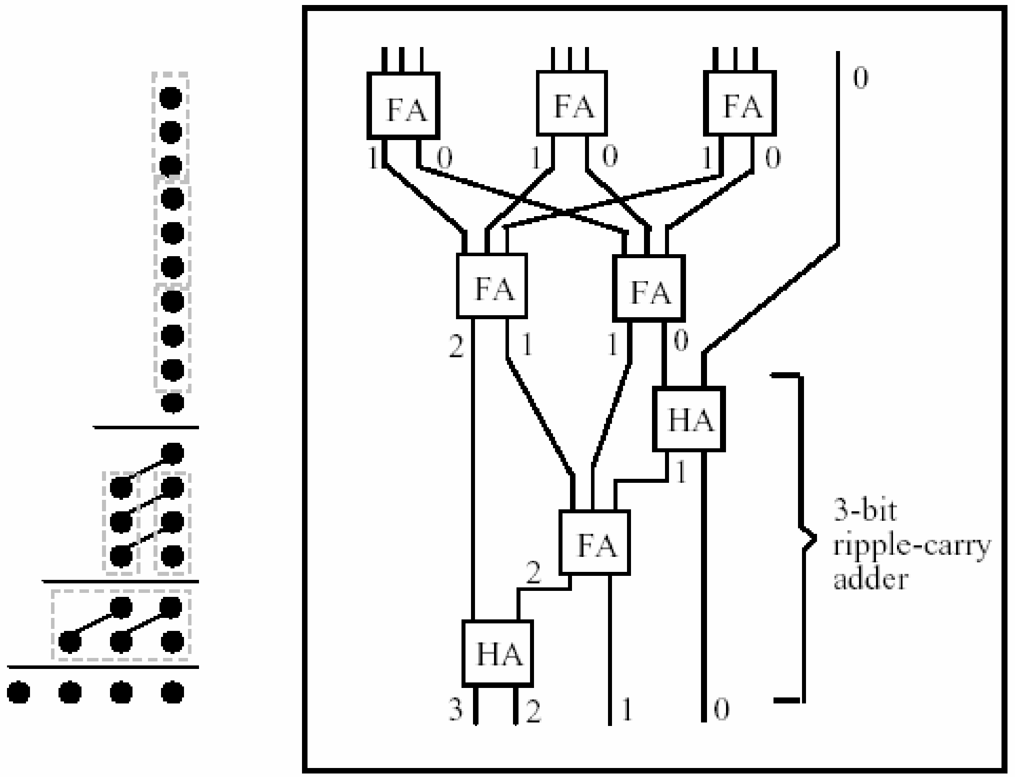
Total cost = 7-bit adder + 26 FAs + 3 HA

**from Parhami*

Adding seven 6-bit numbers by taking advantage of the final adder's carry-in.

Parallel Counters *

Single-bit full-adder = (3; 2)-counter
Circuit reducing 7 bits to their 3-bit sum = (7; 3)-counter
Circuit reducing n bits to their $\lceil \log_2(n + 1) \rceil$-bit sum
= $(n; \lceil \log_2(n + 1) \rceil)$-counter



**from Parhami*

Fig. 8.16 **A 10-input parallel counter also known as a (10; 4)-counter.**

Parallel Counters and Compressors *

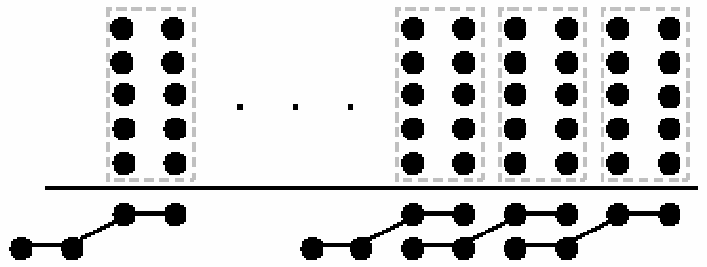
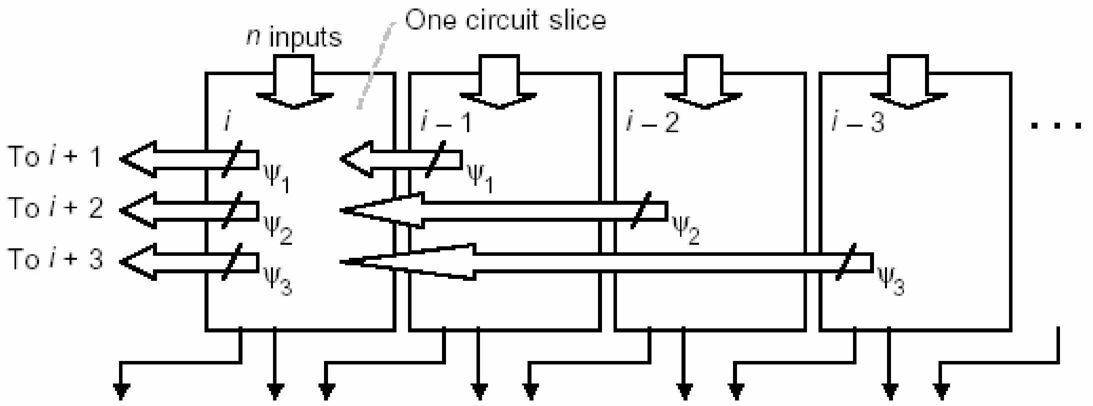


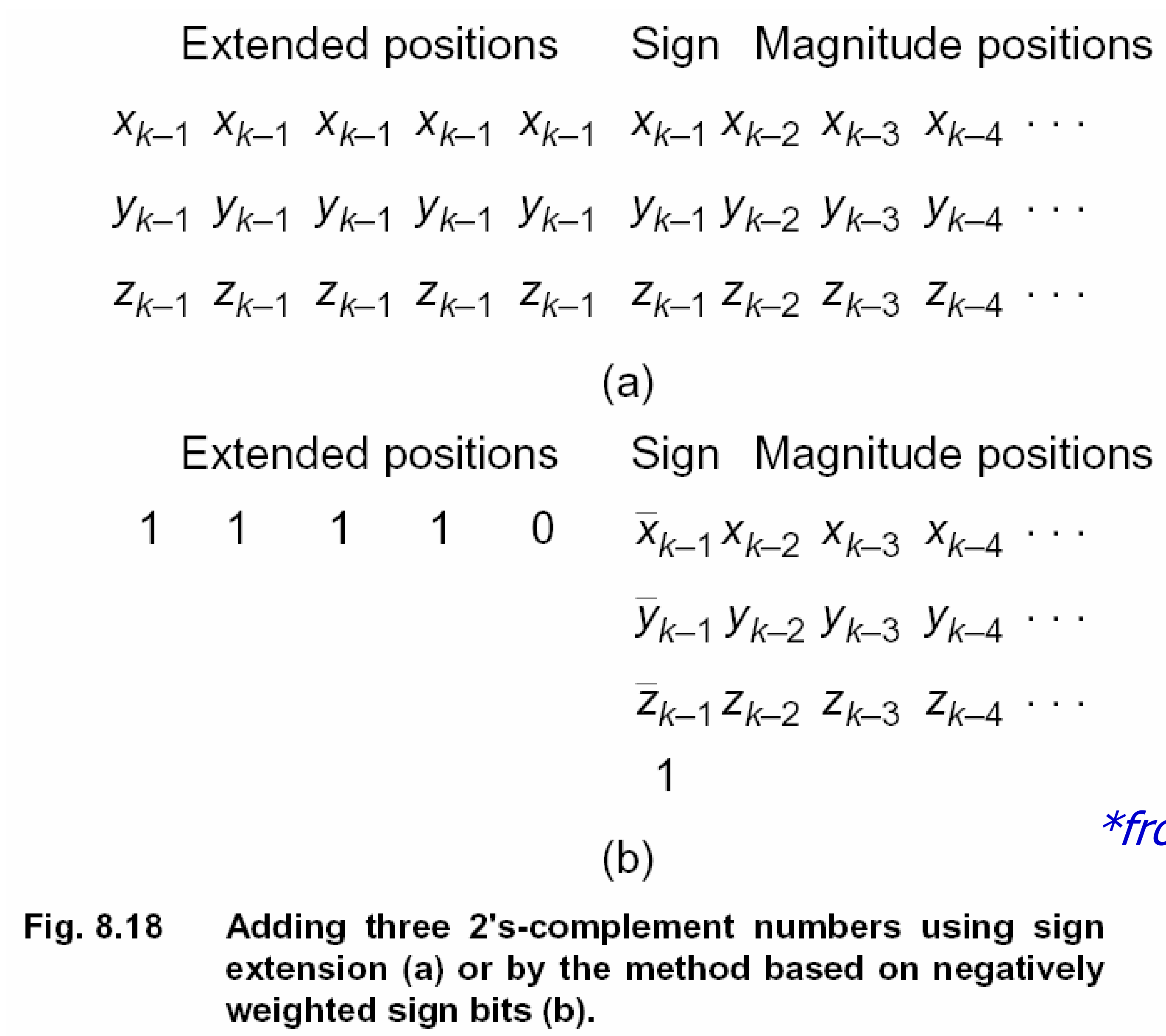
Fig. 8.17 Dot notation for a (5, 5; 4)-counter and the use of such counters for reducing five numbers to two numbers.

(*n*; 2)-counters



**from Parhami*

Adding Multiple Signed Numbers *



$$\begin{array}{ccccc|ccccc}
\multicolumn{5}{c|}{\text{Extended positions}} & \text{Sign} & \multicolumn{4}{l}{\text{Magnitude positions}} \\
x_{k-1} & x_{k-1} & x_{k-1} & x_{k-1} & x_{k-1} & x_{k-1} & x_{k-2} & x_{k-3} & x_{k-4} & \cdots \\
y_{k-1} & y_{k-1} & y_{k-1} & y_{k-1} & y_{k-1} & y_{k-1} & y_{k-2} & y_{k-3} & y_{k-4} & \cdots \\
z_{k-1} & z_{k-1} & z_{k-1} & z_{k-1} & z_{k-1} & z_{k-1} & z_{k-2} & z_{k-3} & z_{k-4} & \cdots
\end{array}$$

(a)

$$\begin{array}{ccccc|ccccc}
\multicolumn{5}{c|}{\text{Extended positions}} & \text{Sign} & \multicolumn{4}{l}{\text{Magnitude positions}} \\
1 & 1 & 1 & 1 & 0 & \bar{x}_{k-1} & x_{k-2} & x_{k-3} & x_{k-4} & \cdots \\
 & & & & & \bar{y}_{k-1} & y_{k-2} & y_{k-3} & y_{k-4} & \cdots \\
 & & & & & \bar{z}_{k-1} & z_{k-2} & z_{k-3} & z_{k-4} & \cdots \\
 & & & & & 1 & & & &
\end{array}$$

(b)

Fig. 8.18 **Adding three 2's-complement numbers using sign extension (a) or by the method based on negatively weighted sign bits (b).**

**from Parhami*

Multi-Operand Addition

<u>from</u>

Digital Arithmetic*, *Morgan-Kauffman Publishers, 2004

Miloš Ercegovac and Tomàs Lang

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MULTI-OPERAND ADDITION

- Bit-arrays for unsigned and signed operands
  - simplification of sign extension
- Reduction by rows and by columns
  - $[p{:}2]$ modules and $[p{:}2]$ adders for reduction by rows
  - $(p{:}q]$ counters and multicolumn counters for reduction by columns
- Sequential implementation
- Combinational implementation
  - Reduction by rows: arrays of adders (linear arrays, adder trees)
  - Reduction by columns: $(p{:}q]$ counters
  - systematic design method for reduction by columns with (3:2] and (2:2] counters
- Pipelined adder arrays
- Partially combinational implementation

BIT ARRAYS FOR UNSIGNED AND SIGNED OPERANDS

$$\begin{array}{l}
a_0\, a_0\, a_0\, a_0 \,.\, a_1\, a_2 \;\ldots\; a_n \\
b_0\, b_0\, b_0\, b_0 \,.\, b_1\, b_2 \;\ldots\; b_n \\
c_0\, c_0\, c_0\, c_0 \,.\, c_1\, c_2 \;\ldots\; c_n \\
d_0\, d_0\, d_0\, d_0 \,.\, d_1\, d_2 \;\ldots\; d_n \\
\underbrace{e_0\, e_0\, e_0}_{\text{sign extension}}\, e_0 \,.\, e_1\, e_2 \;\cdots\; e_n
\end{array}$$

Figure 3.1: SIGN-EXTENDED ARRAY FOR $m = 5$.

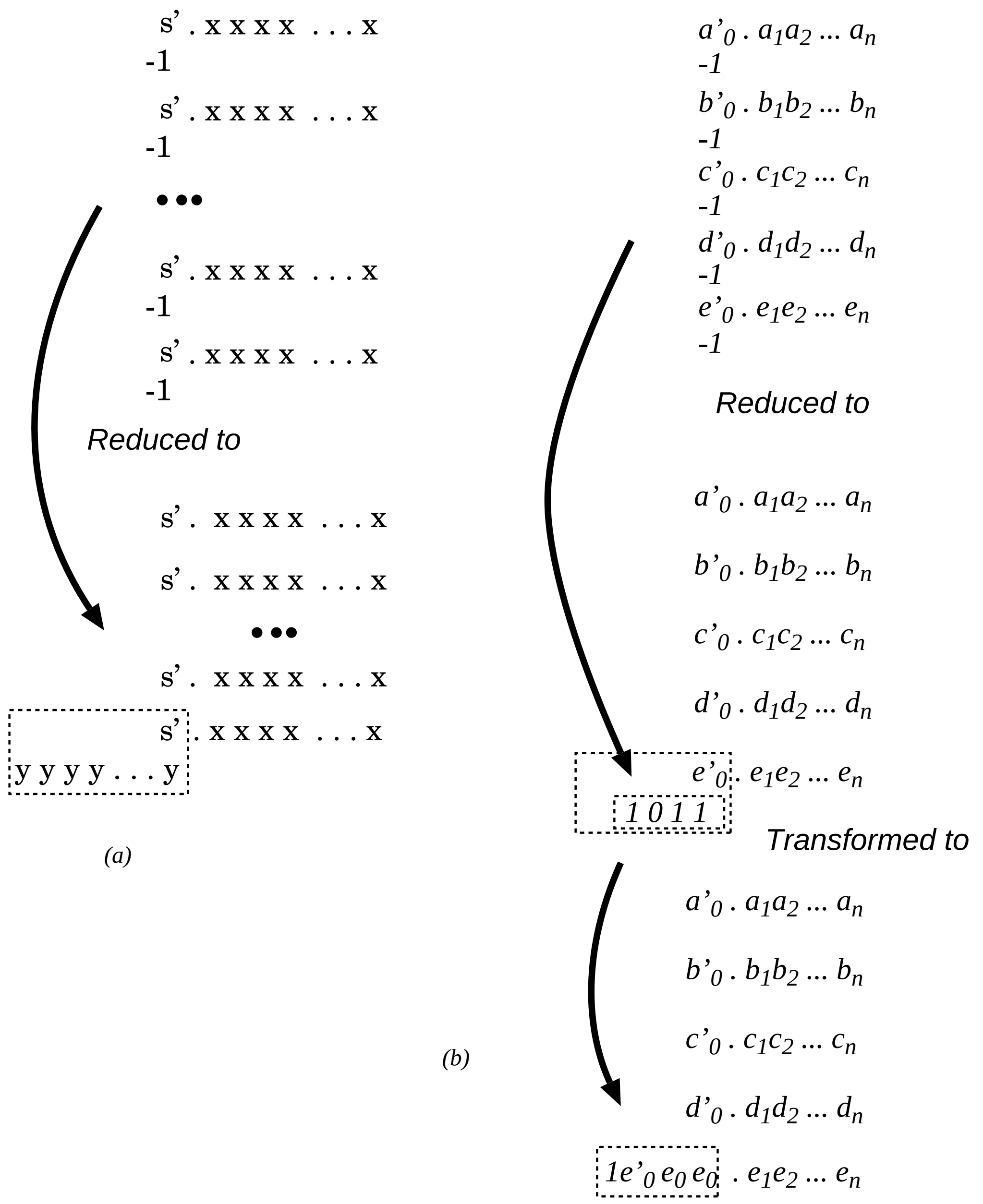


Figure 3.2: SIMPLIFYING SIGN-EXTENSION: (a) GENERAL CASE. (b) EXAMPLE OF SIMPLIFYING ARRAY WITH $m = 5$.

REDUCTION

- By rows
- By columns

[p:2] ADDERS FOR REDUCTION BY ROWS

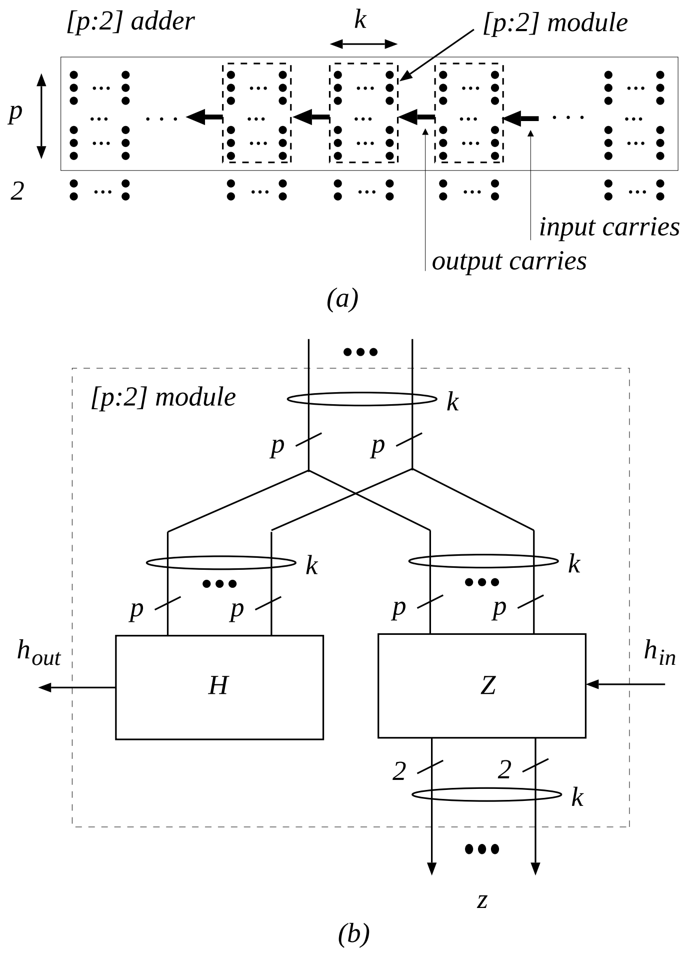


Figure 3.3: A [p:2] adder: (a) Input-output bit-matrix. (b) k-column [p:2] module decomposition.

MODEL OF [p:2] MODULE

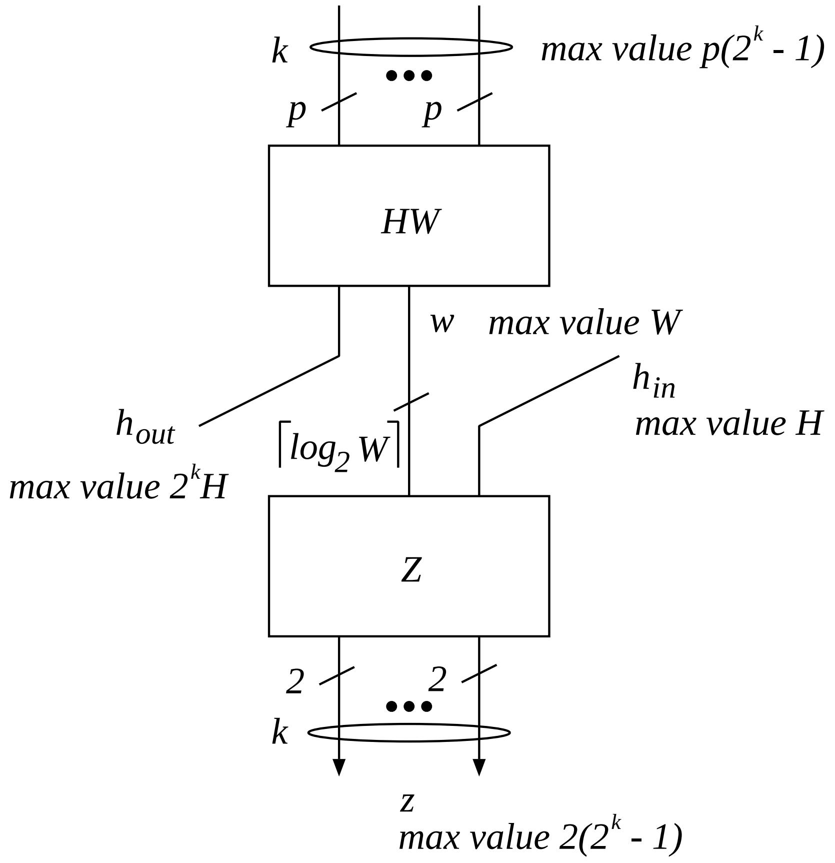


Figure 3.4: A model of a $[p{:}2]$ module.

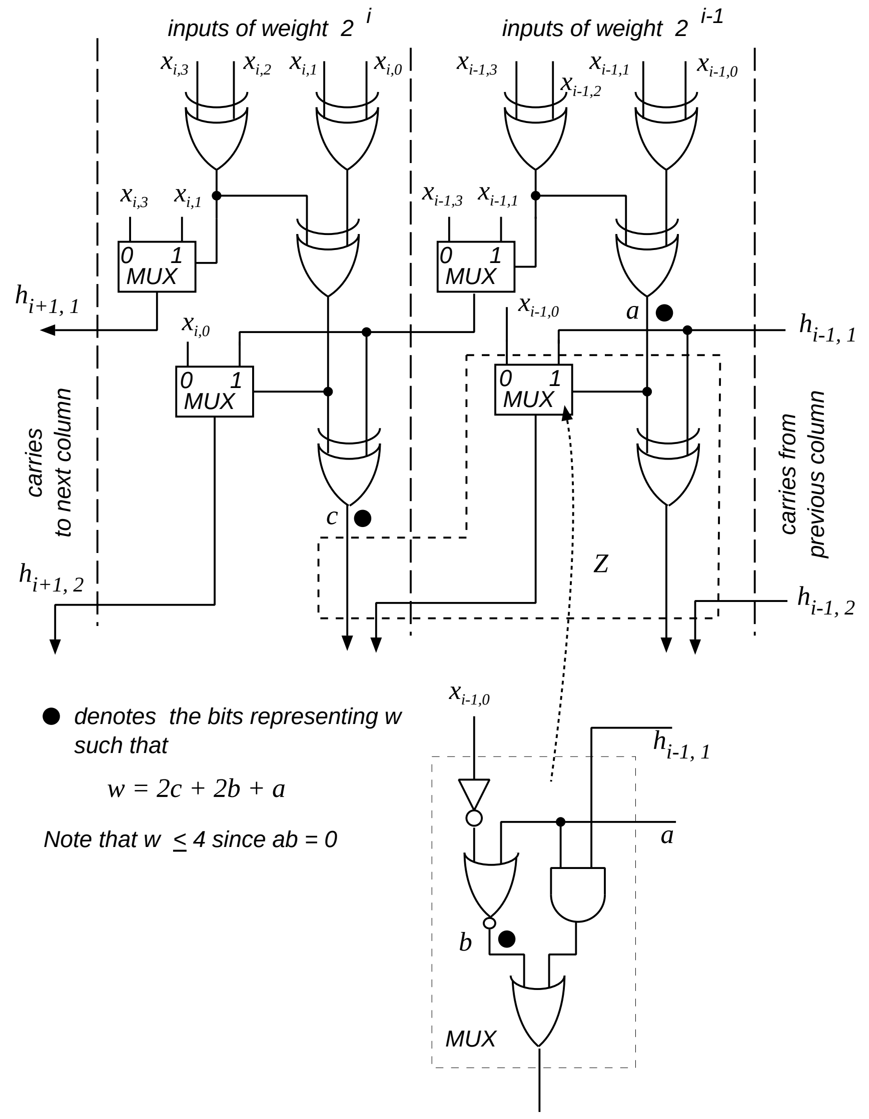


Figure 3.5: Gate network implementation of [4:2] module.

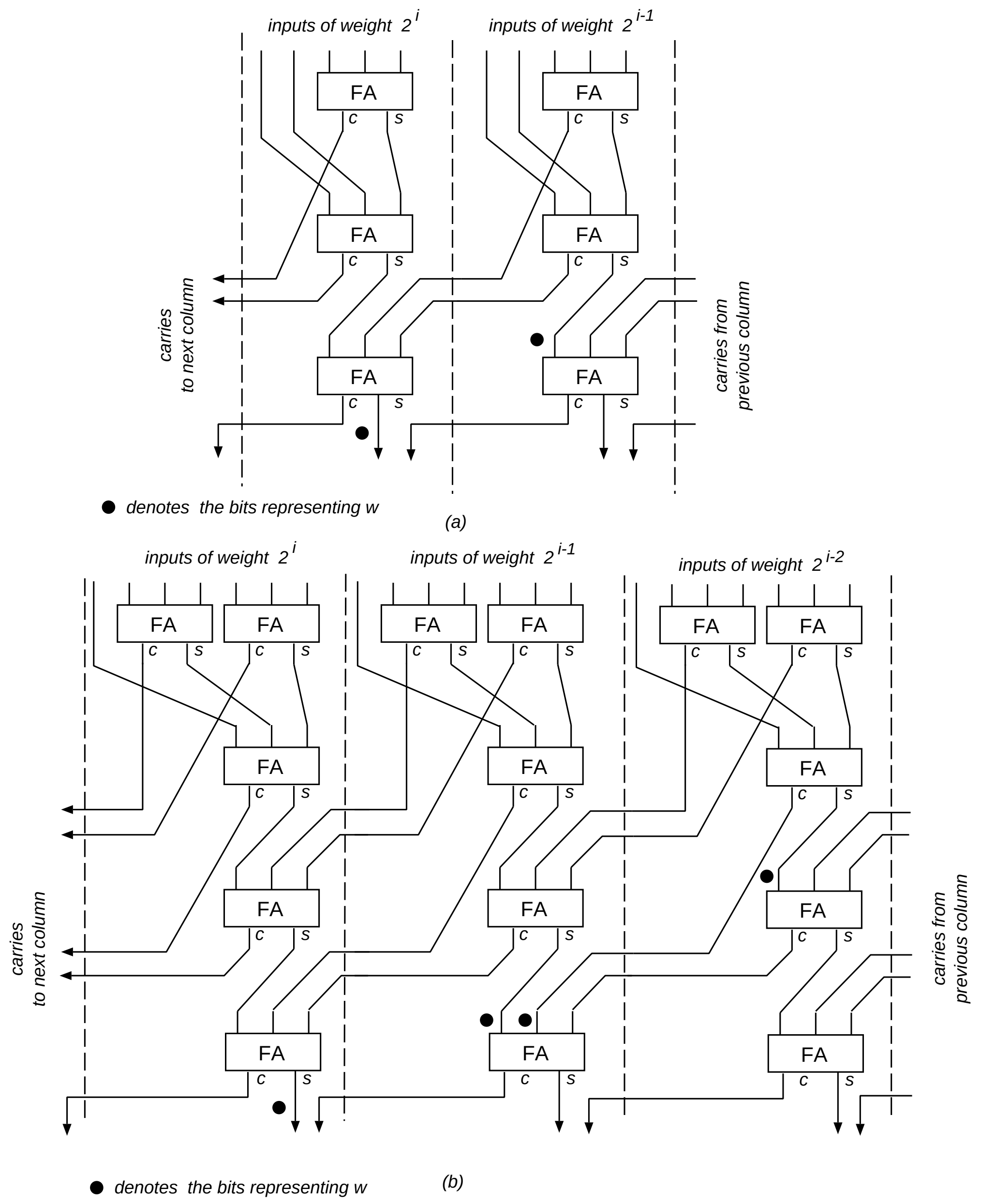

inputs of weight 2^i
inputs of weight 2^{i-1}
FA
c
s
carries to next column
carries from previous column
● denotes the bits representing w
(a)
inputs of weight 2^i
inputs of weight 2^{i-1}
inputs of weight 2^{i-2}
carries to next column
carries from previous column
● denotes the bits representing w
(b)

Figure 3.6: (a) [5:2] module. (b) [7:2] module.

$(p{:}q]$ COUNTERS FOR REDUCTION BY COLUMNS

$$\sum_{i=0}^{p-1} x_i = \sum_{j=0}^{q-1} y_j 2^j$$

$$2^q - 1 \geq p, \text{ i.e., } q = \lceil log_2(p+1) \rceil$$

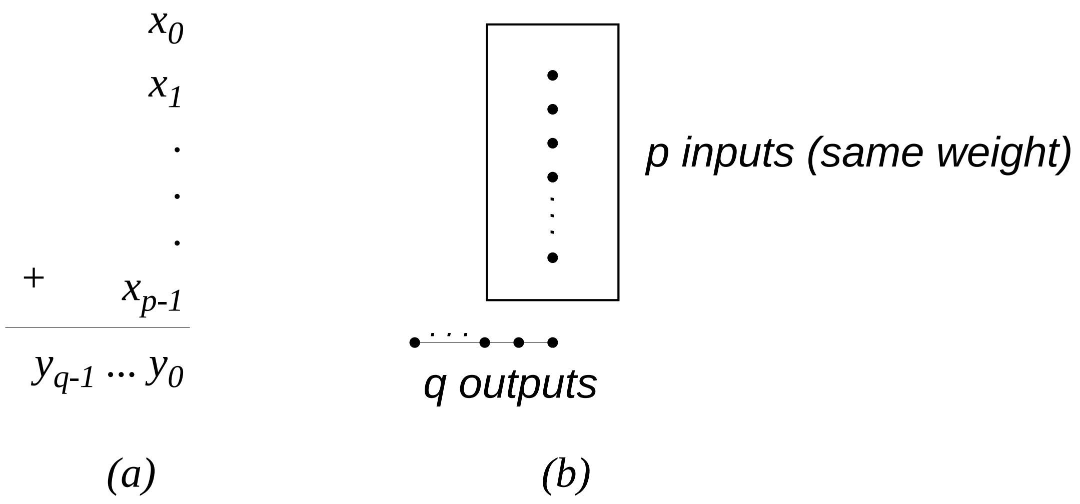


Figure 3.7: (a) $(p{:}q]$ reduction. (b) Counter representation.

IMPLEMENTATION OF $(p{:}q]$ COUNTERS

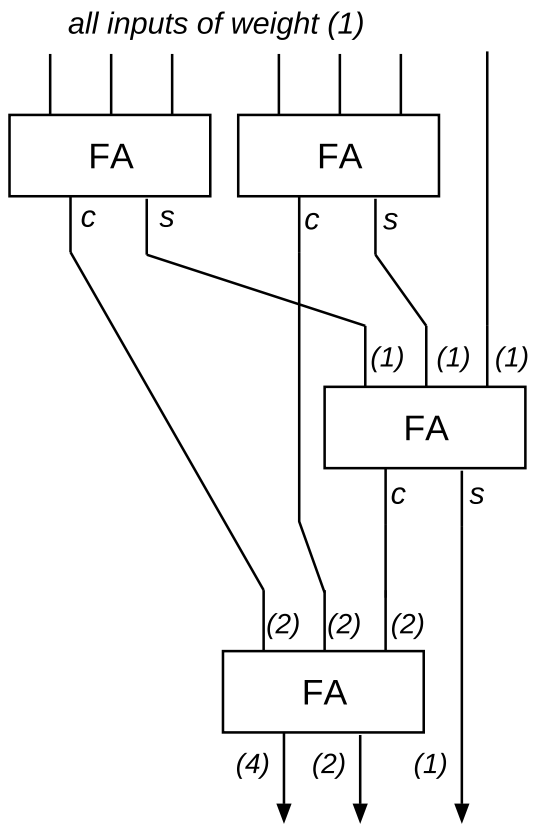


Figure 3.8: Implementation of (7:3] counter by an array of full adders.

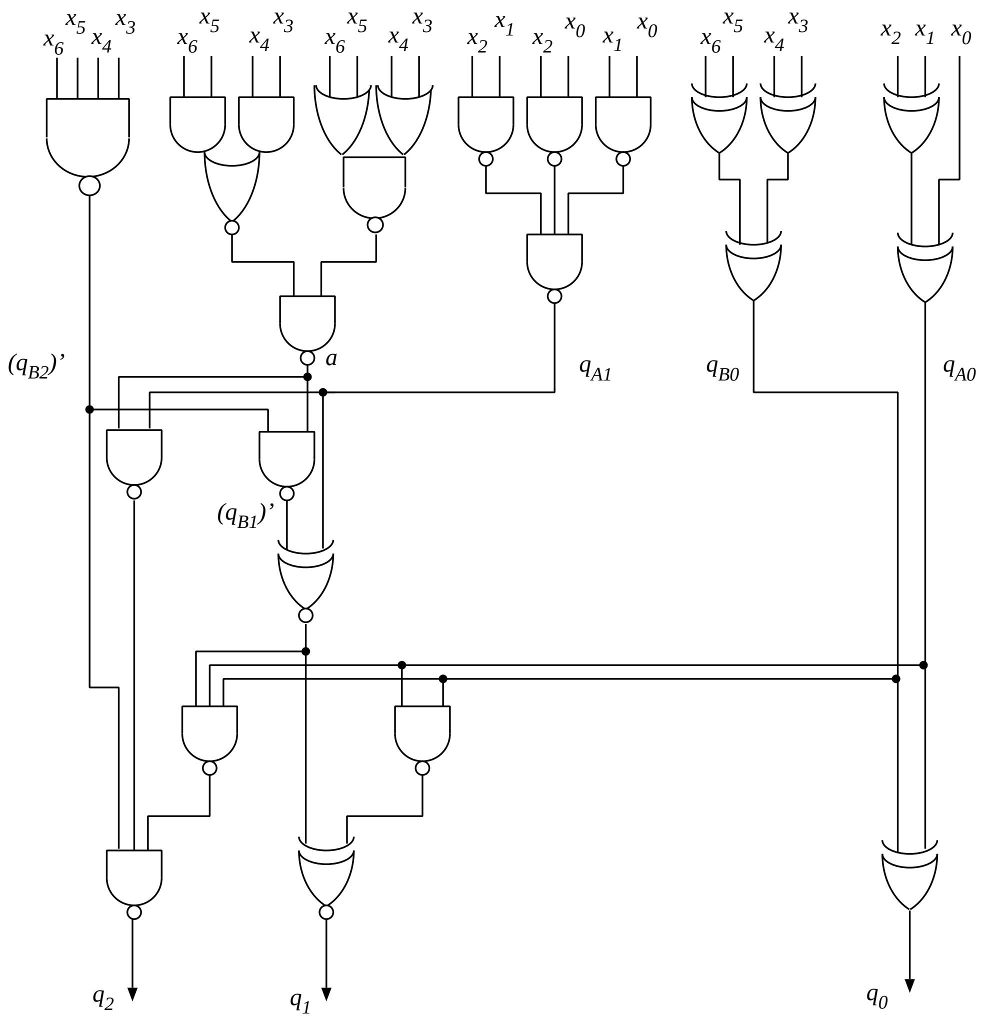


Figure 3.9: Gate network of a (7:3] counter.

MULTICOLUMN COUNTER

$$(p_{k-1}, p_{k-2}, \ldots, p_0 : q]$$

$$v = \sum_{i=0}^{k-1} \sum_{j=1}^{p_i} a_{ij} 2^i \leq 2^q - 1$$

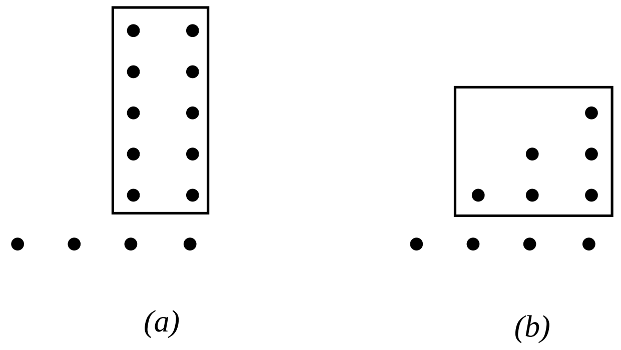


Figure 3.10: (a) (5,5:4] counter. (b) (1,2,3:4] counter.

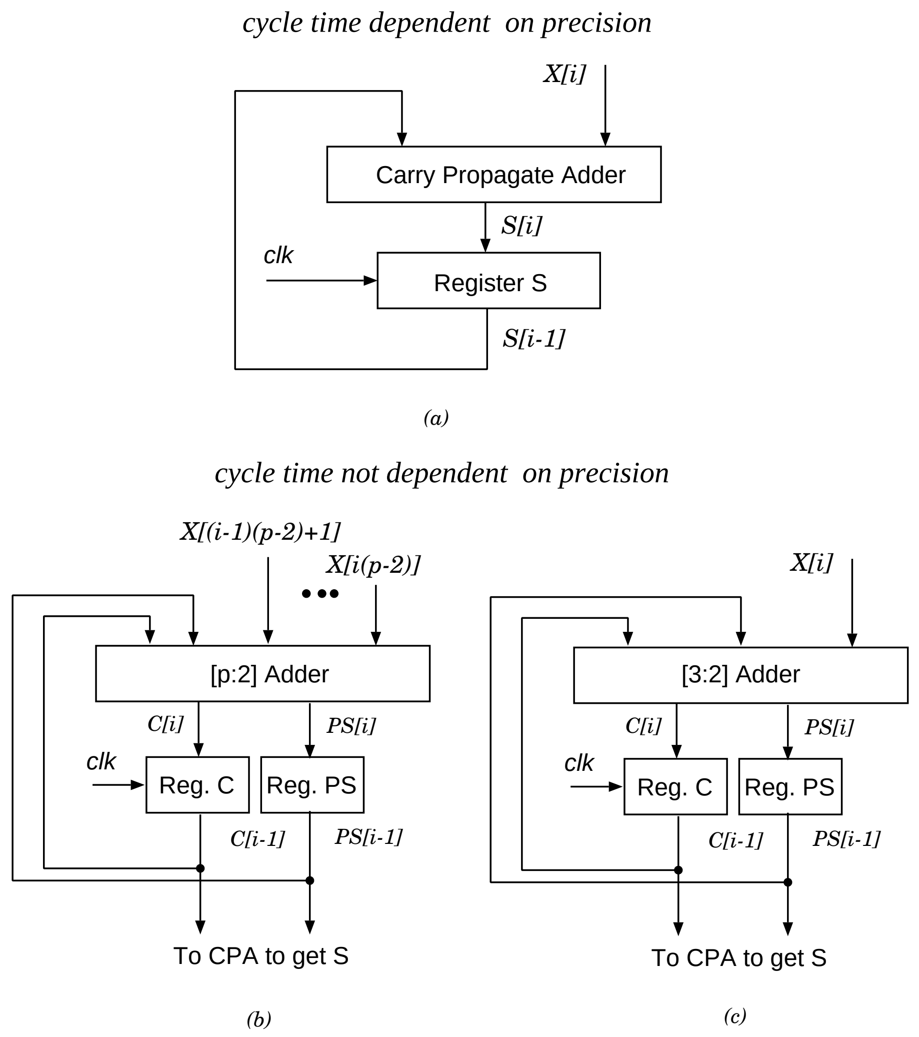


Figure 3.11: SEQUENTIAL MULTIOPERAND ADDITION: a) WITH CONVENTIONAL ADDER. b) WITH [p:2] ADDER. c) WITH [3:2] ADDER.

COMBINATIONAL IMPLEMENTATION

- Reduction by rows: array of adders
  - Linear array
  - Adder tree
- Reduction by columns with (p:q] counters

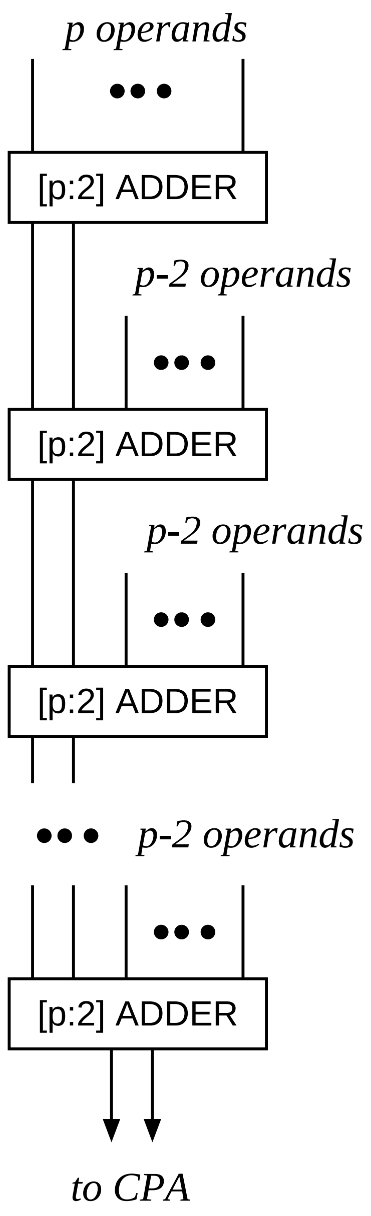


Figure 3.12: LINEAR ARRAY OF [p:2] ADDERS FOR MULTIOPERAND ADDITION.

ADDER TREE

- k - the number of [p:2] CS adders for m operands:

$$pk = m + 2(k-1)$$

$$k = \left\lceil \frac{m-2}{p-2} \right\rceil \qquad \text{[p:2] carry-save adders}$$

- The number of adder levels

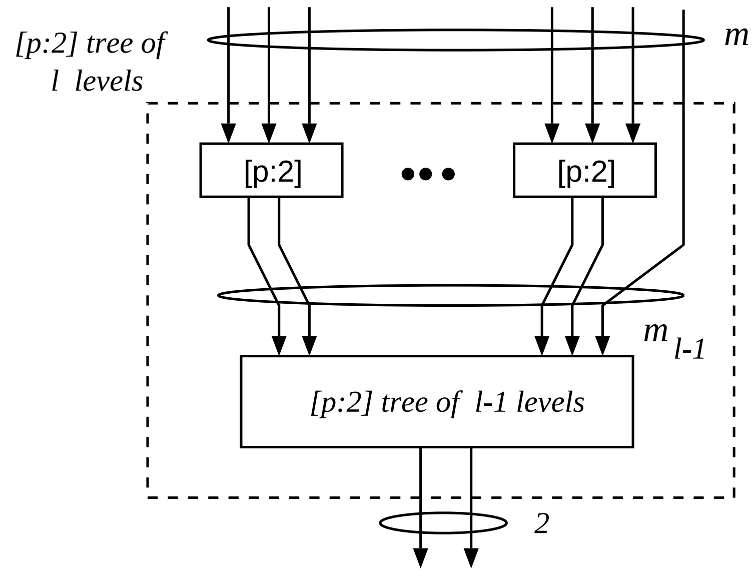


Figure 3.13: Construction of a [p:2] carry-save adder tree.

$$m_l = p \left\lfloor \frac{m_{l-1}}{2} \right\rfloor + m_{l-1} mod\ 2$$

NUMBER OF LEVELS (cont.)

Table 3.1: [3:2] Reduction sequence.

l	1	2	3	4	5	6	7	8	9
m_l	3	4	6	9	13	19	28	42	63

$$m_l \approx \frac{p^l}{2^{l-1}}$$

$$l \approx log_{p/2}(m_l/2)$$

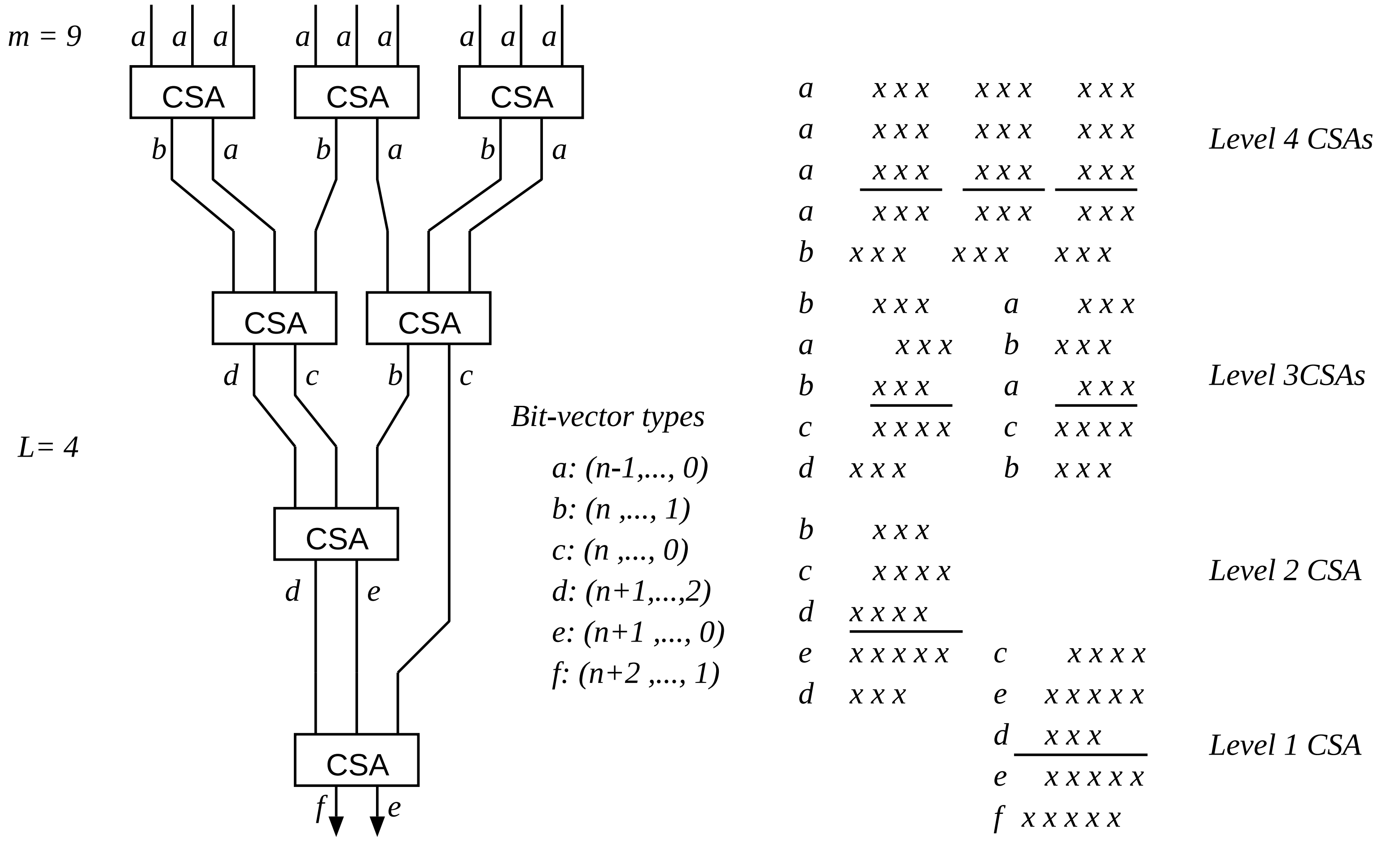


Figure 3.14: [3:2] adder tree for 9 operands (magnitudes with $n = 3$) .

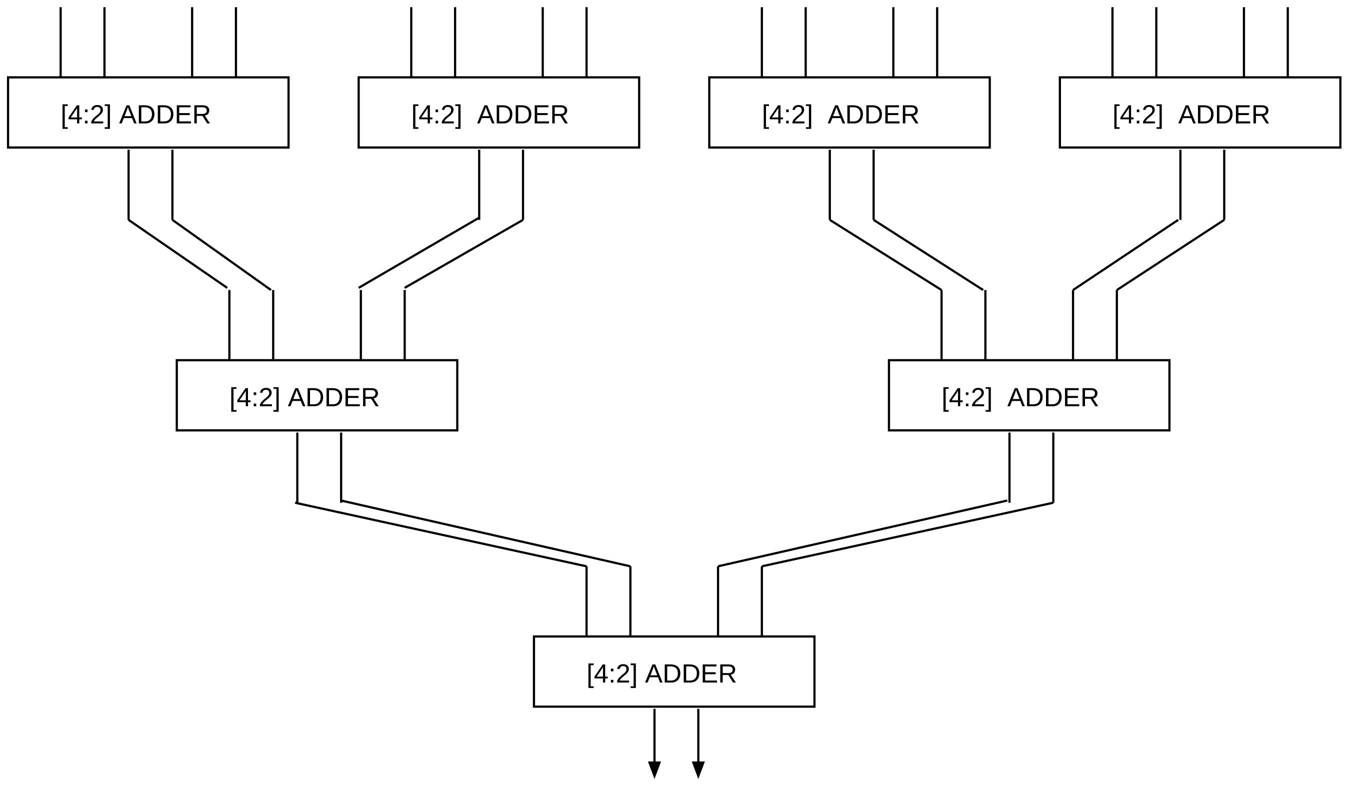


Figure 3.15: Tree of [4:2] adders for $m = 16$.

REDUCTION BY COLUMNS WITH (p:q] COUNTERS

$$
\begin{array}{cccccc}
 & & 1 & \mathbf{0} & 1 & 1 \\
 & & 0 & \mathbf{0} & 1 & 0 \\
 & & 1 & \mathbf{0} & 0 & 1 \\
 & & 0 & \mathbf{1} & 1 & 0 \\
 & & 1 & \mathbf{0} & 1 & 0 \\
 & & 1 & \mathbf{1} & 1 & 1 \\
 & & 0 & \mathbf{1} & 1 & 0 \\
\hline
 & & 0 & \mathbf{1} & 0 & 1 \\
 & 0 & \mathbf{1} & 1 & 1 & \\
1 & \mathbf{0} & 1 & 0 & & \\
\end{array}
$$

Figure 3.16: Example of reduction using (7:3] counters.

NUMBER OF COUNTER LEVELS

$$m_1 = p$$
$$m_l = p\left\lfloor\frac{m_{l-1}}{q}\right\rfloor + m_{l-1}\ mod\ q$$

$$l \approx log_{p/q}(m_l/q)$$

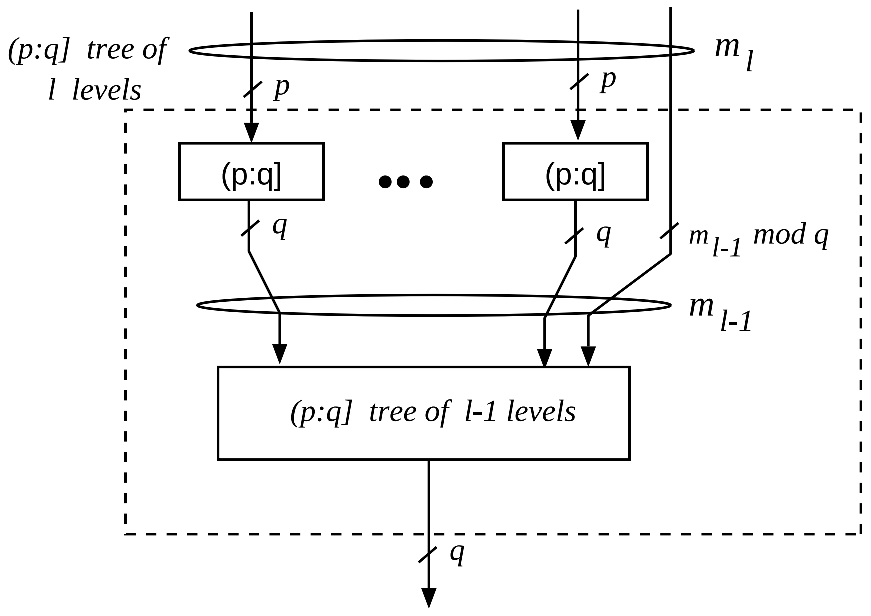


Figure 3.17: Construction of (p:q] reduction tree.

Table 3.2: Sequence for (7:3] counters

Number of levels	1	2	3	4	...
Max. number of rows	7	15	35	79	...

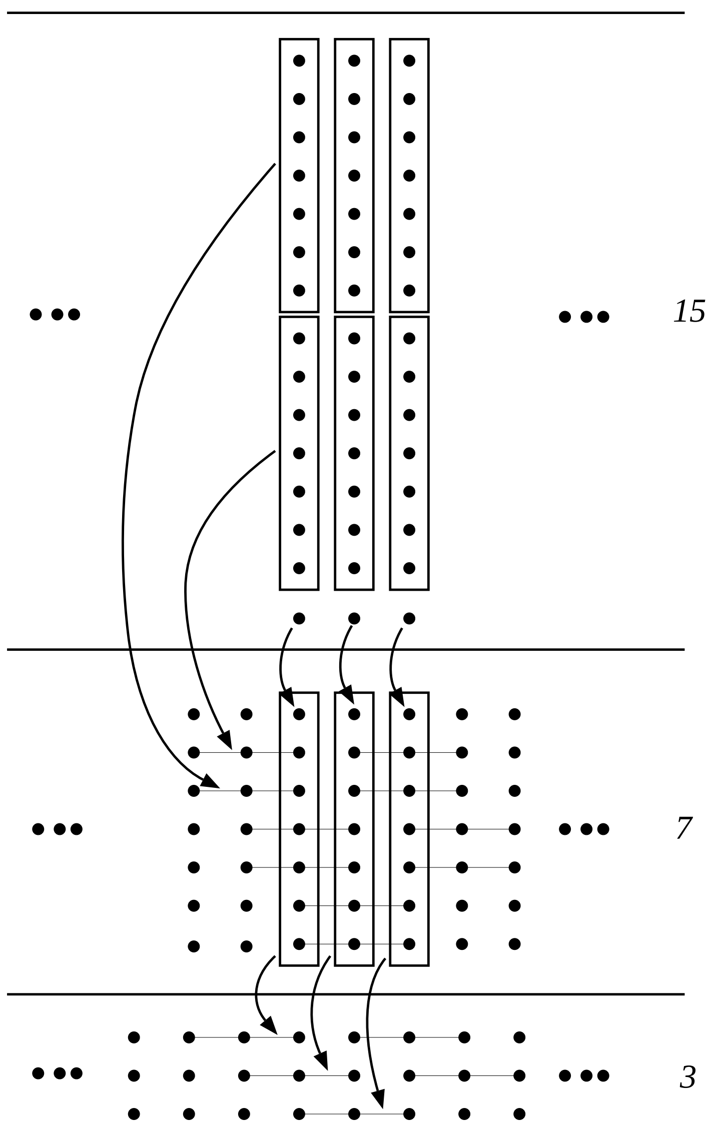


Figure 3.18: Multilevel reduction with (7:3] counters

SYSTEMATIC DESIGN METHOD

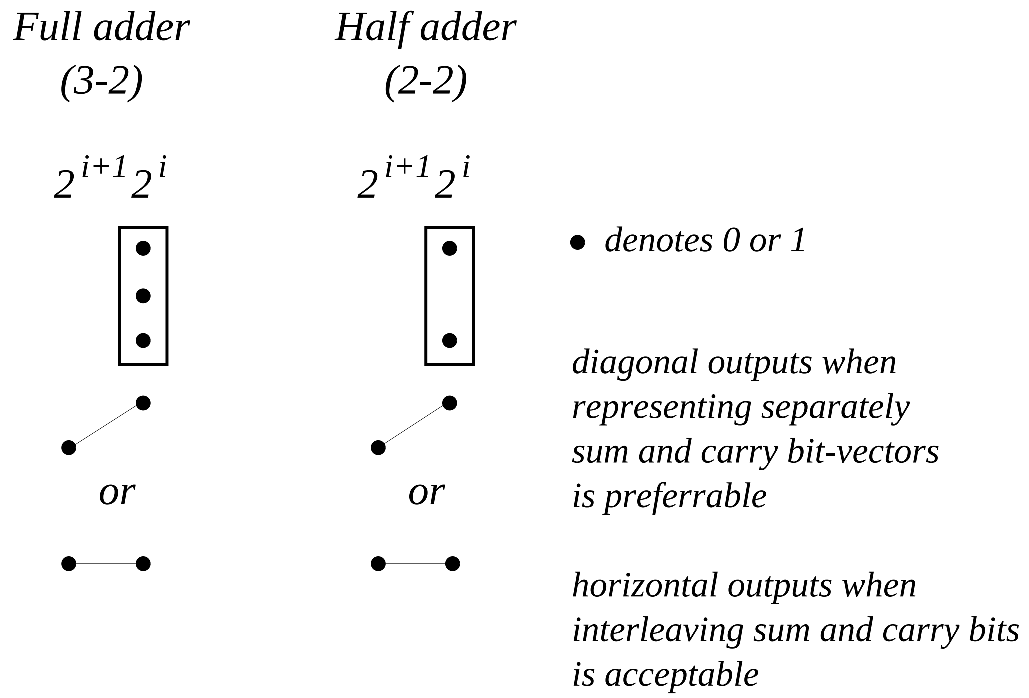

Full adder
(3-2)
Half adder
(2-2)
$2^{i+1}\,2^{i}$
$2^{i+1}\,2^{i}$
or
or
● denotes 0 or 1
diagonal outputs when representing separately sum and carry bit-vectors is preferrable
horizontal outputs when interleaving sum and carry bits is acceptable

Figure 3.19: Full adder and half adder as (3:2] and (2:2] counters.

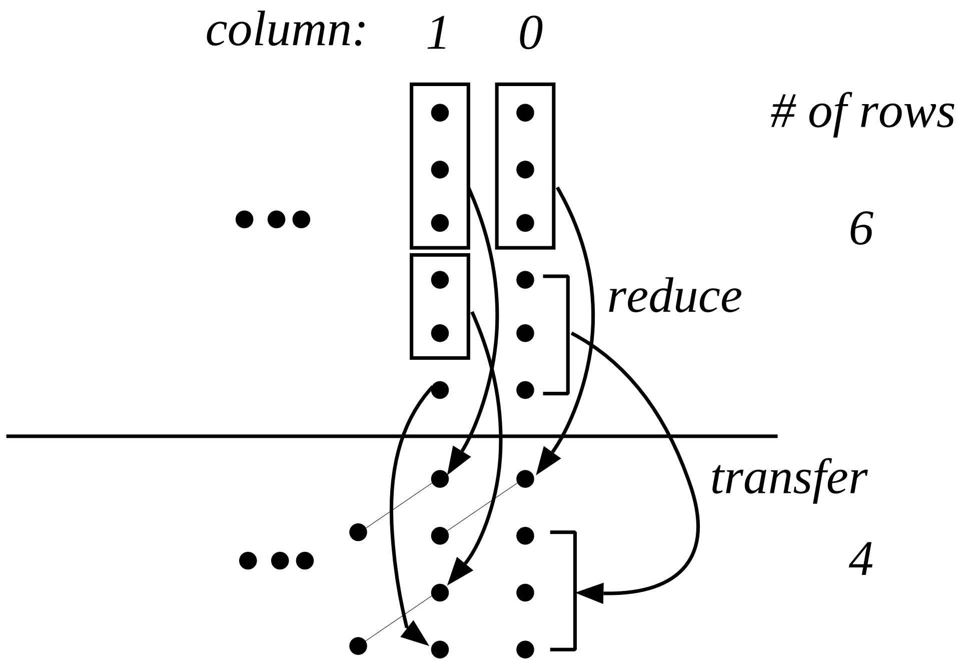


Figure 3.20: Reduction process.

RELATION AT LEVEL l

e_i – number of bits in column i

f_i – number of full adders in column i

h_i – number of half adders in column i

$$e_i - 2f_i - h_i + f_{i-1} + h_{i-1} = m_{l-1}$$

resulting in

$$2f_i + h_i = e_i - m_{l-1} + f_{i-1} + h_{i-1} = p_i$$

Solution producing min number of carries:

$$f_i = \lfloor p_i/2 \rfloor \qquad h_i = p_i \; mod \; 2$$

	i						
	6	5	4	3	2	1	0
$l=4$							
e_i			8	8	8	8	8
m_3			6	6	6	6	6
h_i			0	0	0	1	0
f_i			2	2	2	1	1
$l=3$							
e_i		2	6	6	6	6	6
m_2		4	4	4	4	4	4
h_i		0	0	0	0	1	0
f_i		0	2	2	2	1	1
$l=2$							
e_i		4	4	4	4	4	4
m_1		3	3	3	3	3	3
h_i		0	0	0	0	0	1
f_i		1	1	1	1	1	0
$l=1$							
e_i	1	3	3	3	3	3	3
m_0	2	2	2	2	2	2	2
h_i	0	0	0	0	0	0	1
f_i	0	1	1	1	1	1	0

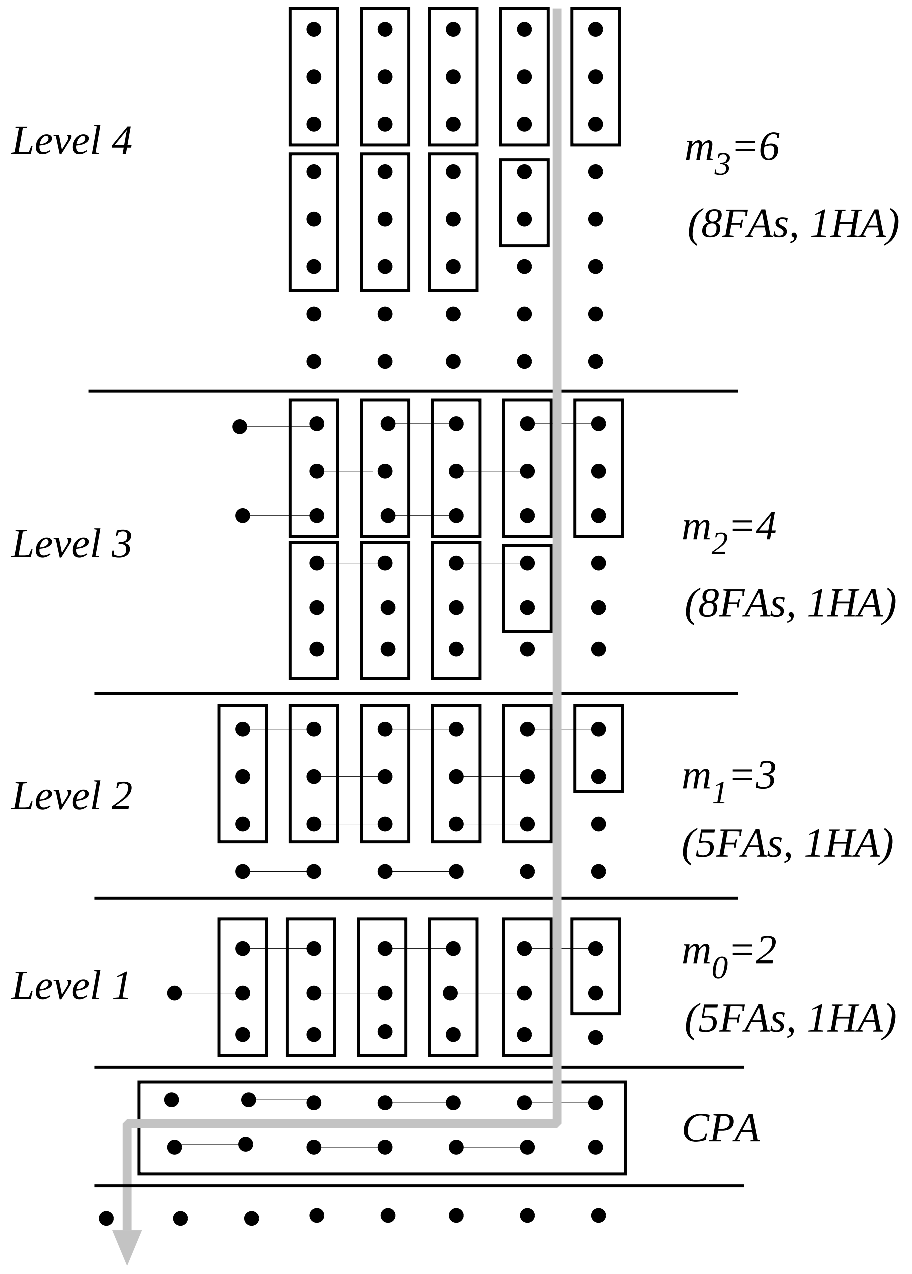


Figure 3.21: Reduction by columns of 8 5-bit magnitudes. Cost of reduction: 26 FAs and 4 HAs.

EXAMPLE: ARRAY FOR $f = a + 3b + 3c + d$

Operands in [-4,3). Result range:

$$-4 + (-12) + (-12) - 4 = -32 \leq f \leq 3 + 9 + 9 + 3 = 24$$

$$\begin{array}{r|cccccc}
a & a_2 & a_2 & a_2 & a_2 & a_1 & a_0 \\
b & b_2 & b_2 & b_2 & b_2 & b_1 & b_0 \\
2b & b_2 & b_2 & b_2 & b_1 & b_0 & 0 \\
c & c_2 & c_2 & c_2 & c_2 & c_1 & c_0 \\
2c & c_2 & c_2 & c_2 & c_1 & c_0 & 0 \\
d & d_2 & d_2 & d_2 & d_2 & d_1 & d_0
\end{array}$$

transformed into

$$\begin{array}{r|cccc}
a & & a_2' & a_1 & a_0 \\
 & & \text{-1} & & \\
b & & b_2' & b_1 & b_0 \\
 & & \text{-1} & & \\
2b & b_2' & b_1 & b_0 & \\
 & \text{-1} & & & \\
c & & c_2' & c_1 & c_0 \\
 & & \text{-1} & & \\
2c & c_2' & c_1 & c_0 & \\
 & \text{-1} & & & \\
d & & d_2' & d_1 & d_0 \\
 & & \text{-1} & &
\end{array}$$

FINAL BIT MATRIX

$$
\begin{array}{|cccccc}
1 & 0 & b'_2 & a'_2 & a_1 & a_0 \\
 & & c'_2 & b'_2 & b_1 & b_0 \\
 & & & b_1 & b_0 & \\
 & & & c'_2 & c_1 & c_0 \\
 & & & c_1 & c_0 & \\
 & & & d'_2 & d_1 & d_0
\end{array}
$$

	i					
	5	4	3	2	1	0
$l=3$						
e_i	1	0	2	6	6	4
m_2	4	4	4	4	4	4
h_i	0	0	0	1	0	0
f_i	0	0	0	1	1	0
$l=2$						
e_i	1	0	4	4	4	4
m_1	3	3	3	3	3	3
h_i	0	0	0	0	0	1
f_i	0	0	1	1	1	0
$l=1$						
e_i	1	1	3	3	3	3
m_0	2	2	2	2	2	1*
h_i	0	0	0	0	0	0
f_i	0	0	1	1	1	1

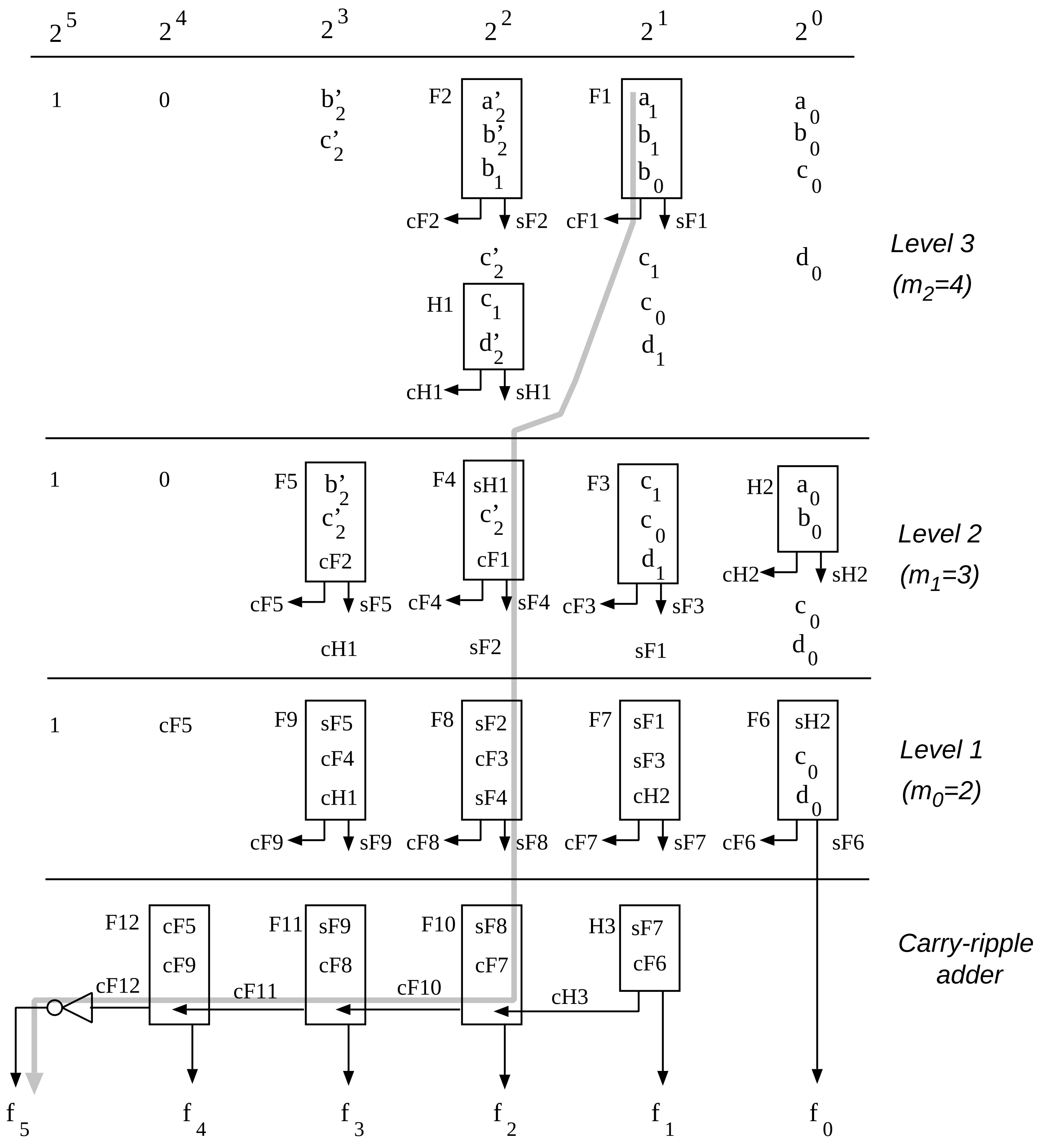
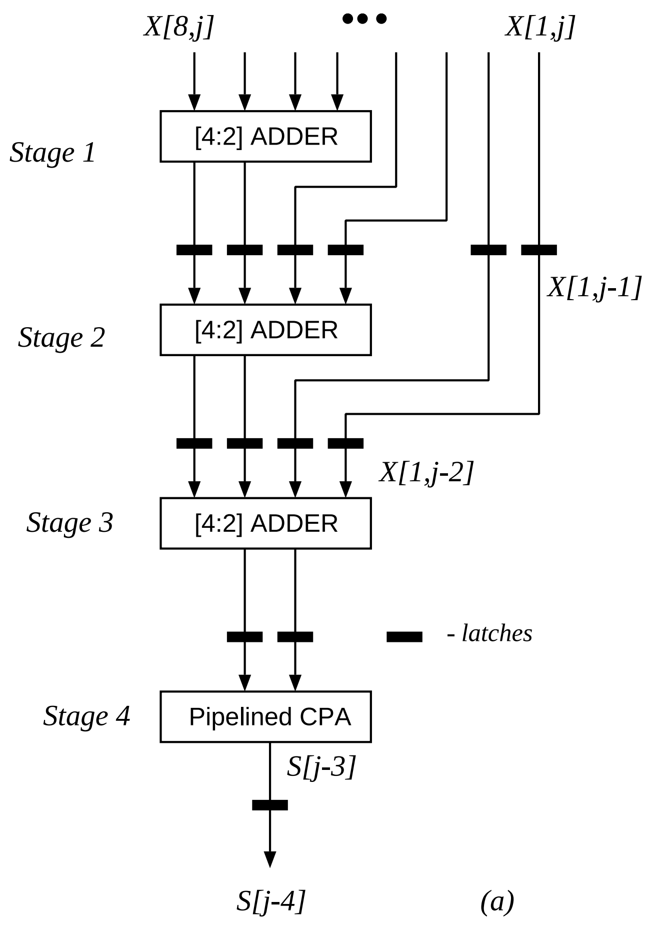


Figure 3.22: Reduction array $f = a + 3b + 3c + d$.

PIPELINED LINEAR ARRAY



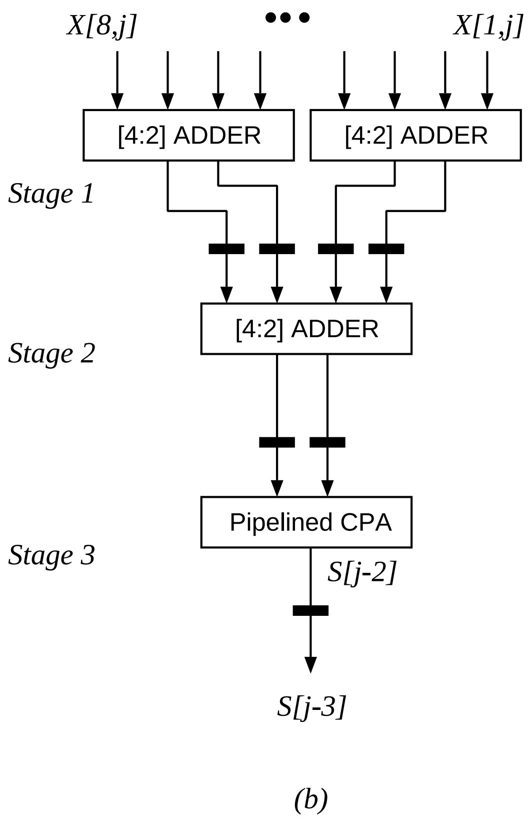


Figure 3.23: Pipelined arrays with [4:2] adders for computing $S[j] = \sum_{i=1}^{8} X[i,j],\ j = 1, \ldots, N$: (a) Linear array. (b) Tree array.

PARTIALLY COMBINATIONAL IMPLEMENTATION

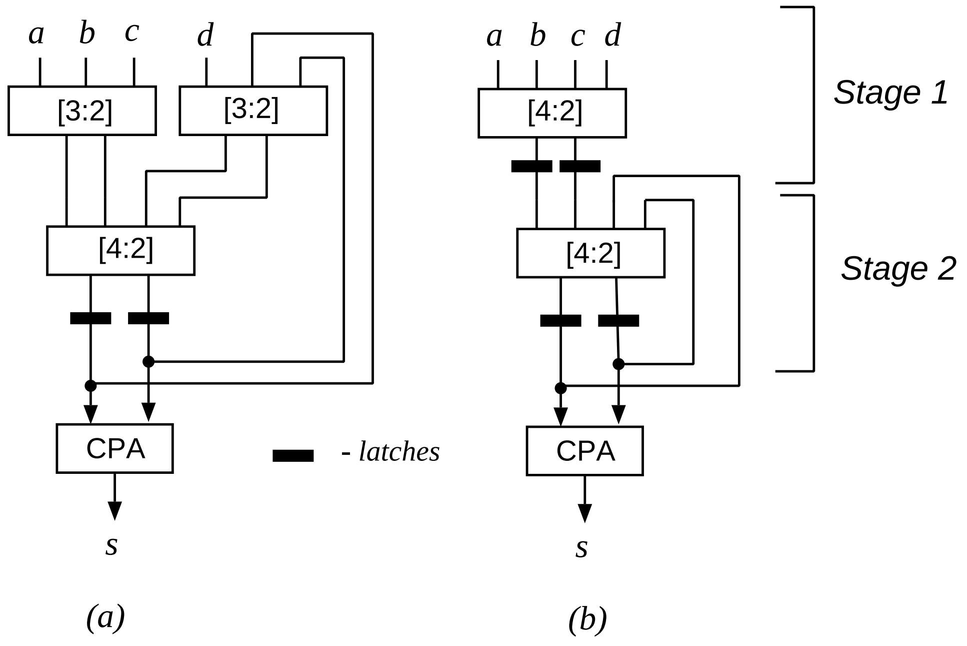


Figure 3.24: Partially combinational scheme for summation of 4 operands per iteration: (a) Nonpipelined. (b) Pipelined.

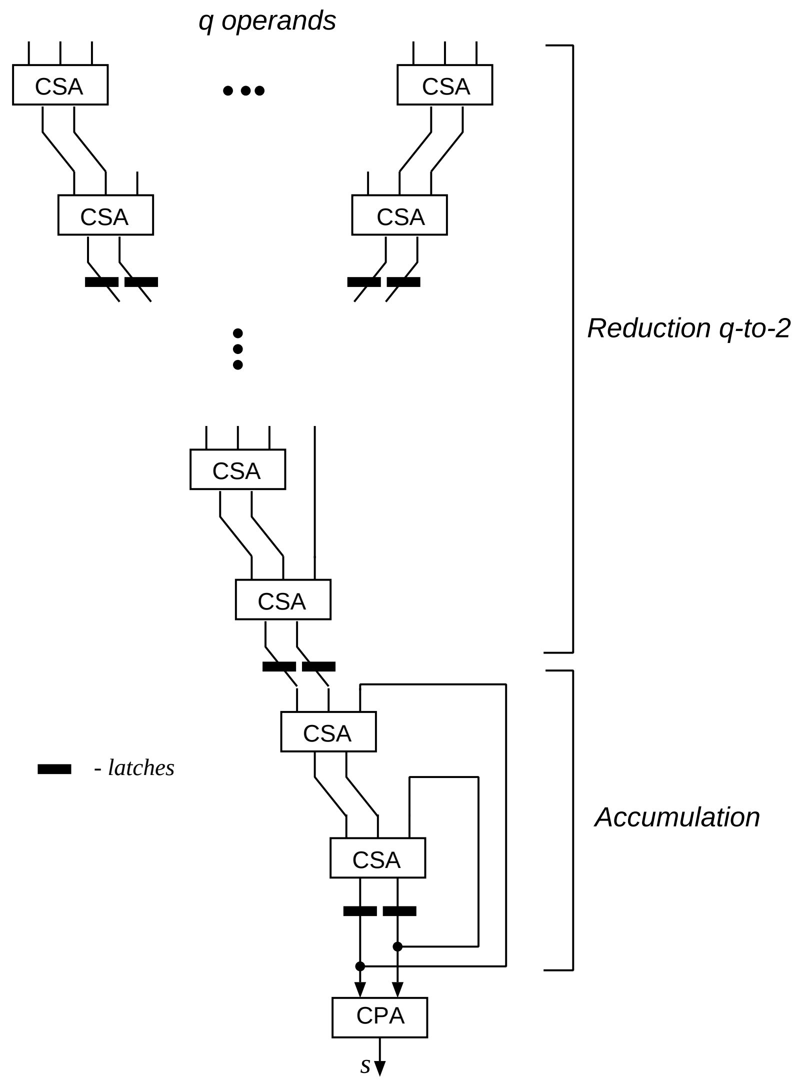
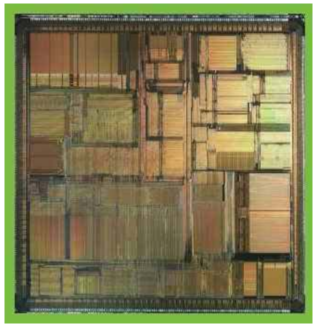
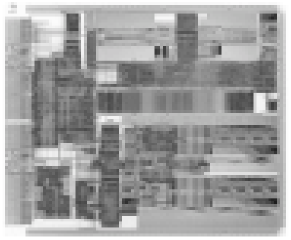
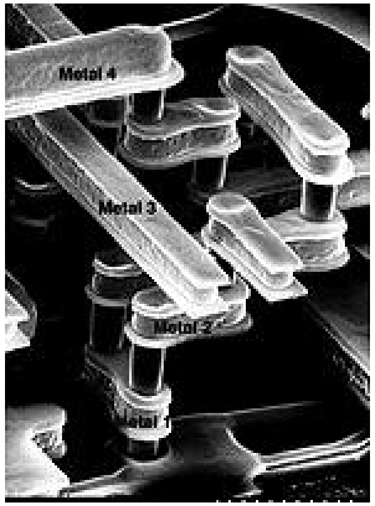


Figure 3.25: Scheme for summation of q operands per iteration.

VLSI Arithmetic

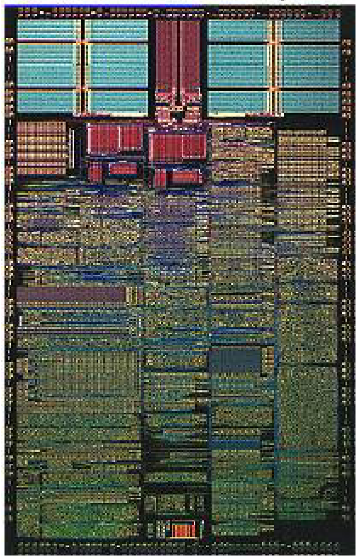


Lecture 9b:

Sign-Digit Arithmetic

Prof. Vojin G. Oklobdzija
University of California

http://www.ece.ucdavis.edu/acsel



ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Sign-Digit Addition

<u>**from**</u>

***Digital Arithmetic**, Morgan-Kauffman Publishers, 2004*

Miloš Ercegovac and Tomàs Lang

SIGNED-DIGIT ADDITION

- Uses signed-digit representation (redundant)

$$x = \sum_{0}^{n-1} x_i r^i$$

with digit set

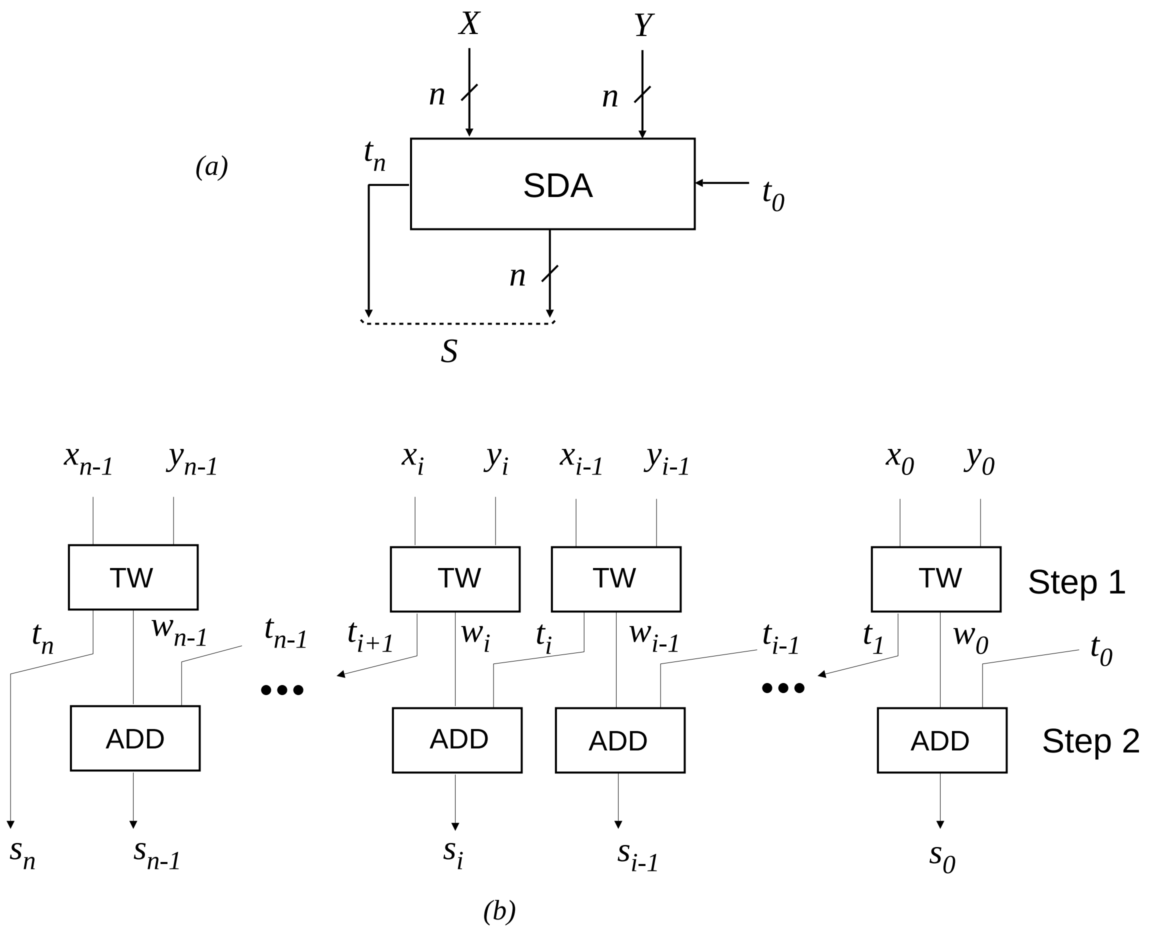
$$D = \{-a, \ldots, -1, 0, 1, \ldots, a\}$$

- Limits carry propagation to next position
- Addition algorithm:

$$\begin{array}{ll} \text{Step 1:} & x + y = w + t \\ & x_i + y_i = w_i + r t_{i+1} \\ & \\ \text{Step 2:} & s = w + t \\ & s_i = w_i + t_i \end{array}$$

- No carry produced in Step 2

SD ADDER



(a)
X
Y
n
n
t_n
SDA
t_0
n
S
x_{n-1} y_{n-1} x_i y_i x_{i-1} y_{i-1} x_0 y_0
TW
Step 1
t_n w_{n-1} t_{n-1} t_{i+1} w_i t_i w_{i-1} t_{i-1} t_1 w_0 t_0
ADD
Step 2
s_n s_{n-1} s_i s_{i-1} s_0
(b)

Figure 2.34: Signed-digit addition.

CASES CONSIDERED

Case A : two SD operands; result SD

Step 1:

$$(t_{i+1},\ w_i) = \begin{cases} (0,\ x_i + y_i) & \textbf{if } -a+1 \leq x_i + y_i \leq a-1 \\ (1,\ x_i + y_i - r) & \textbf{if } x_i + y_i \geq a \\ (-1,\ x_i + y_i + r) & \textbf{if } x_i + y_i \leq -a \end{cases}$$

- algorithm modified for $r = 2$

Case B : two conventional operands; result SD

Case C : one conventional, one SD; result SD

SIGNED-BINARY ADDITION: METHOD 1 (DOUBLE RECODING)

RECODING 1:

$$\begin{aligned} x_i + y_i = 2h_{i+1} + z_i \quad &\in \{-2,-1,0,1,2\} \\ h_i \in \{0,1\}, z_i &\in \{-2,-1,0\} \\ q_i = z_i + h_i \quad &\in \{-2,-1,0,1\} \end{aligned}$$

RECODING 2:

$$\begin{aligned} q_i = z_i + h_i = 2t_{i+1} + w_i \quad &\in \{-2,-1,0,1\} \\ t_i \in \{-1,0\}, \quad w_i &\in \{0,1\} \end{aligned}$$

THE RESULT: $s_i = w_i + t_i \quad \in \{-1,0,1\}$

METHOD 1 SD ADDER

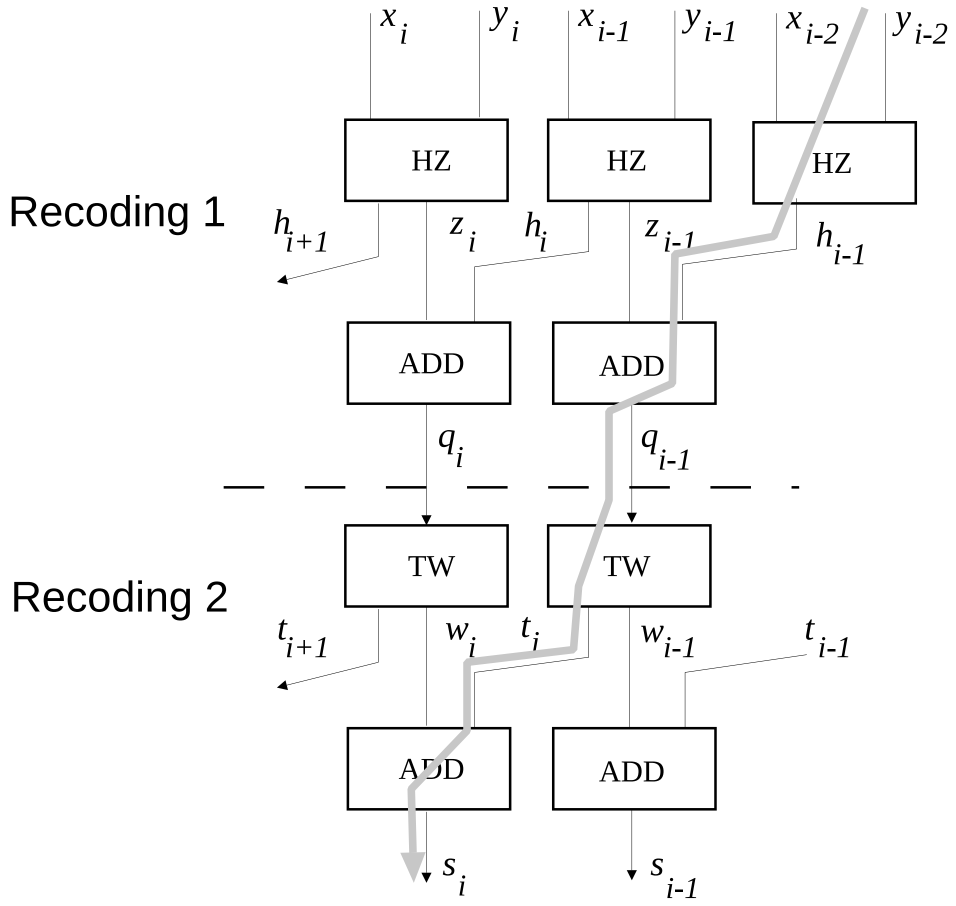


Figure 2.35: Double recoding method for signed-bit addition

SIGNED BINARY ADDITION: METHOD 2 (Using Previous Digit)

$$P_i = \begin{cases} 0 & \text{if } (x_i, y_i) \text{ both nonnegative} \\ & \text{(which implies } t_{i+1} \geq 0) \\ 1 & \text{otherwise } (t_{i+1} \leq 0) \end{cases}$$

$x_i + y_i$	P_{i-1}	t_{i+1}	w_i
2	-	1	0
1	$0(t_i \geq 0)$	1	-1
1	$1(t_i \leq 0)$	0	1
0	-	0	0
-1	$0(t_i \geq 0)$	0	-1
-1	$1(t_i \leq 0)$	-1	1
-2	-	-1	0

METHOD 2 SD ADDER

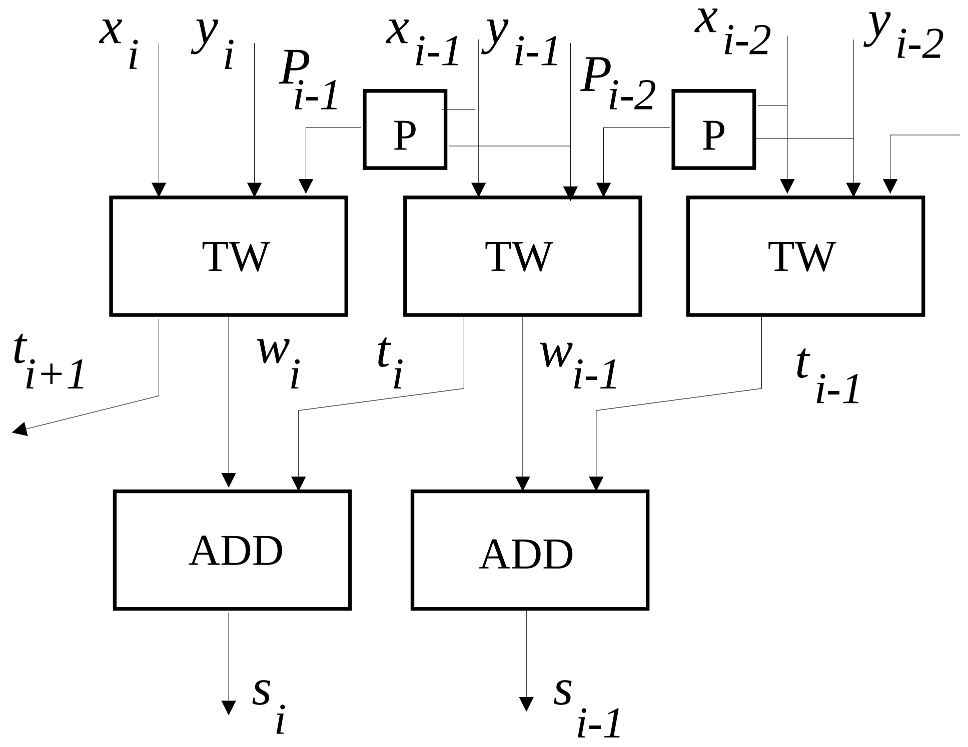


Figure 2.36: Signed-bit addition using the information from previous digit

Example

$$\begin{array}{lcccccccccc}
X & & 0 & 1 & 1 & 1 & \bar{1} & 1 & 0 & \bar{1} & 1 \\
Y & & 0 & 1 & 1 & 0 & \bar{1} & 0 & 1 & 0 & 1 \\
 & & & & & & & & & & \\
P & & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
 & & & & & & & & & & \\
W & & 0 & 0 & 0 & 1 & 0 & \bar{1} & 1 & \bar{1} & 0 \\
T & 0 & 1 & 1 & 0 & \bar{1} & 1 & 0 & 0 & 1 & 0 \\
 & & & & & & & & & & \\
S & & 1 & 1 & 0 & 0 & 1 & \bar{1} & 1 & 0 & 0
\end{array}$$

BIT-LEVEL IMPLEMENTATION OF RADIX-2 ALGORITHMS

- Case C: $x_i \in \{0,1\}, \quad y_i, s_i \in \{-1,0,1\}$
- Code: borrow-save $y_i = y_i^+ - y_i^-$, $y_i^+, y_i^- \in \{0,1\}$, sim. for s_i
- $x_i + y_i \in \{-1,0,1,2\}$: recode to (t_{i+1}, w_i), $t_{i+1} \in \{0,1\}$, $w_i \in \{-1,0\}$

$$x_i + y_i^+ - y_i^- = 2t_{i+1} + w_i$$

$x_i + y_i$	-1	0	1	2
w_i	-1	0	-1	0
t_{i+1}	0	0	1	1

SWITCHING FUNCTIONS FOR t_{i+1} AND w_i

x_i	y_i^+	y_i^-	$x_i + y_i$	t_{i+1}	$-w_i$
0	0	0	0	0	0
0	0	1	-1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	2	1	0
1	1	1	1	1	1

$$\begin{aligned} w_i &= (x_i \oplus y_i^+ \oplus (y_i^-)')' \\ t_{i+1} &= x_i y_i^+ + x_i(y_i^-)' + y_i^+(y_i^-)' \end{aligned}$$

$\Longrightarrow$ implemented using a full-adder and inverters (for variables subtracted)

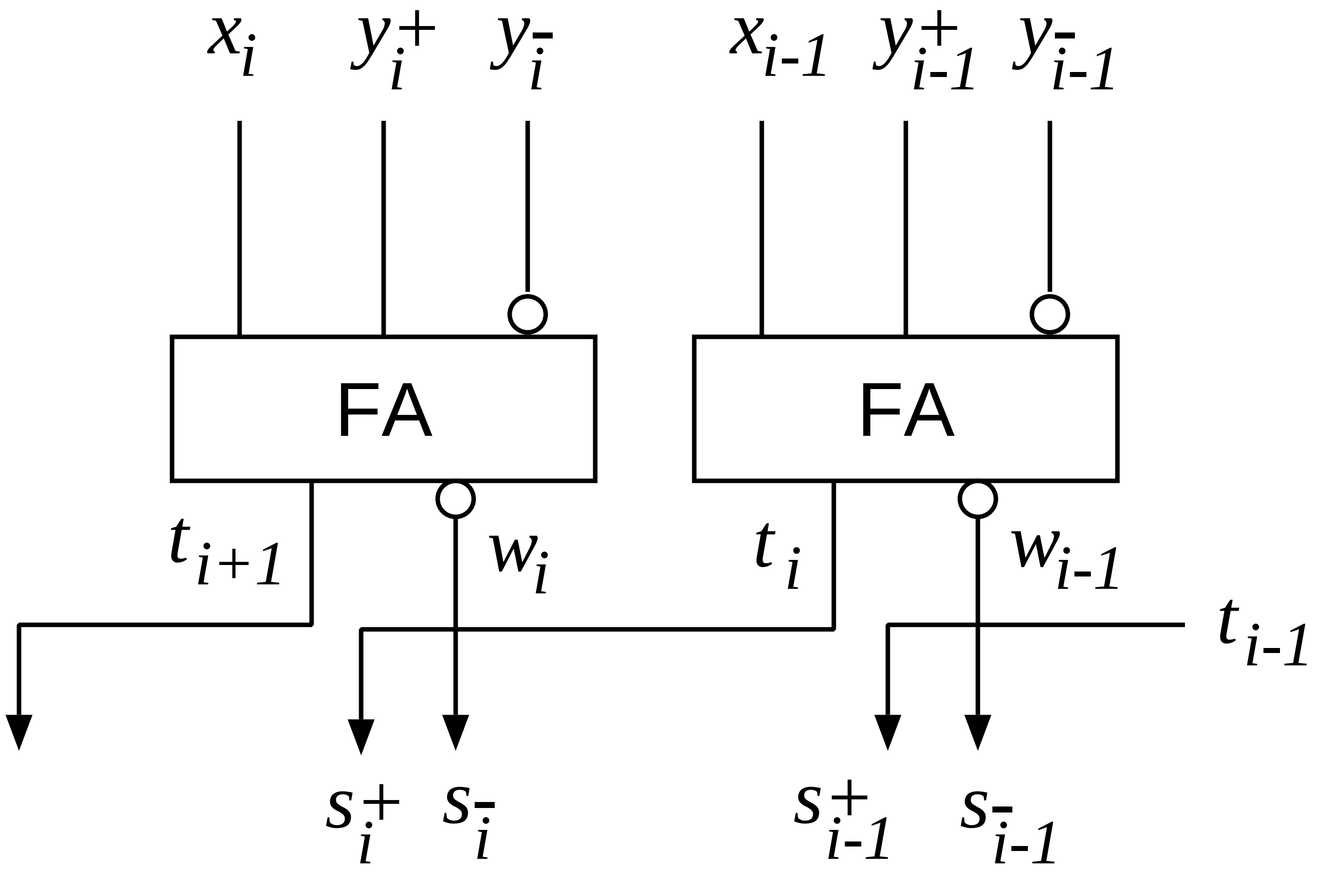


Figure 2.37: Redundant adder: one operand conventional, one operand redundant, result redundant.

BOTH OPERANDS REDUNDANT

- Apply double recoding

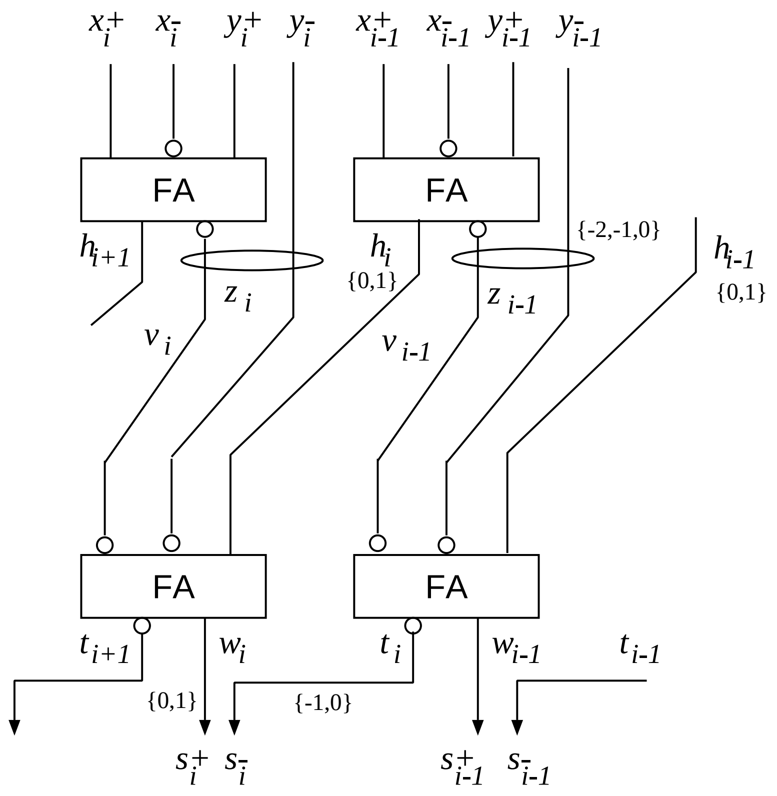
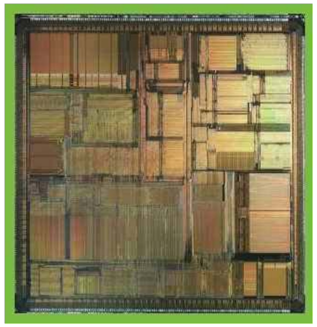
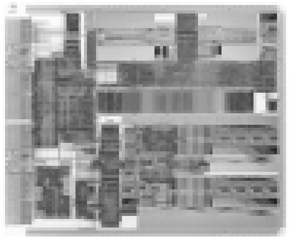
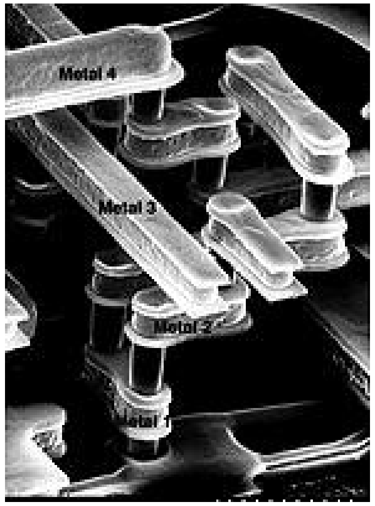


Figure 2.38: Redundant adder: operands and result redundant

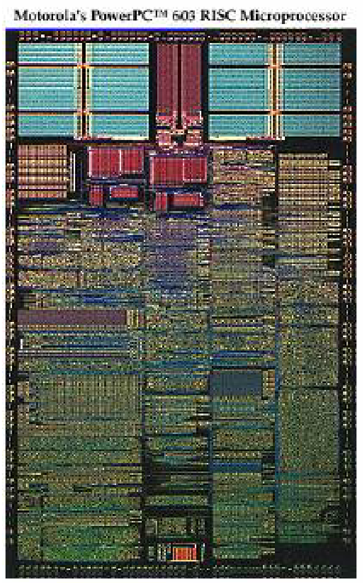



VLSI Arithmetic



Lecture 9c:

Sign-Digit Arithmetic



Prof. Vojin G. Oklobdzija
University of California

http://www.ece.ucdavis.edu/acsel

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Example of Sign-Digit Arithmetic: Viterbi Detector

from

A. K. Yeung, J. Rabaey, “A 210Mb/s Radix-4 Bit-Level Pipelined Viterbi Decoder”, Proceedings of the International Solid-State Circuits Conference, San Francisco, California, 1995.

4-bit Add-Compare-Subtract Unit

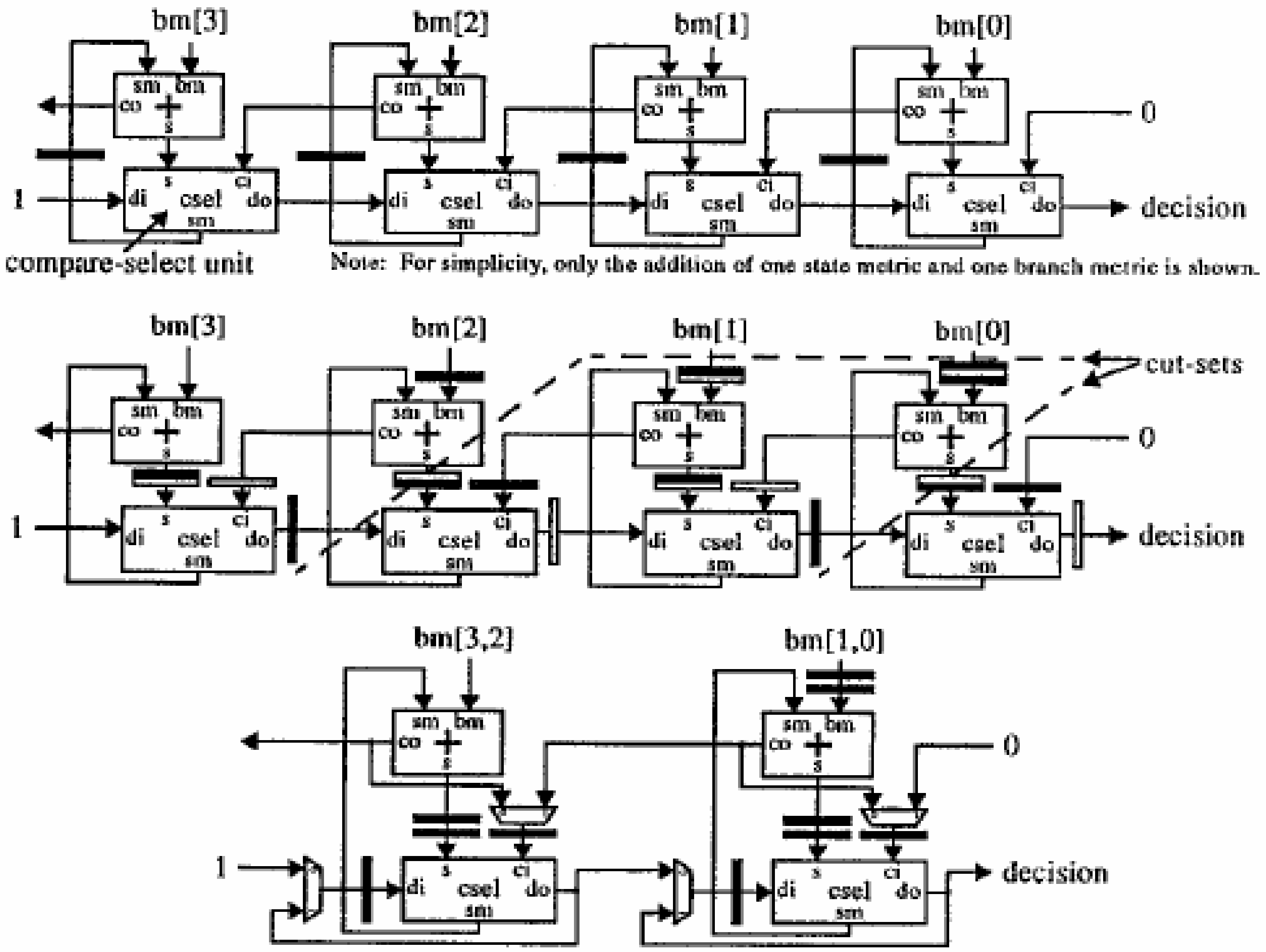
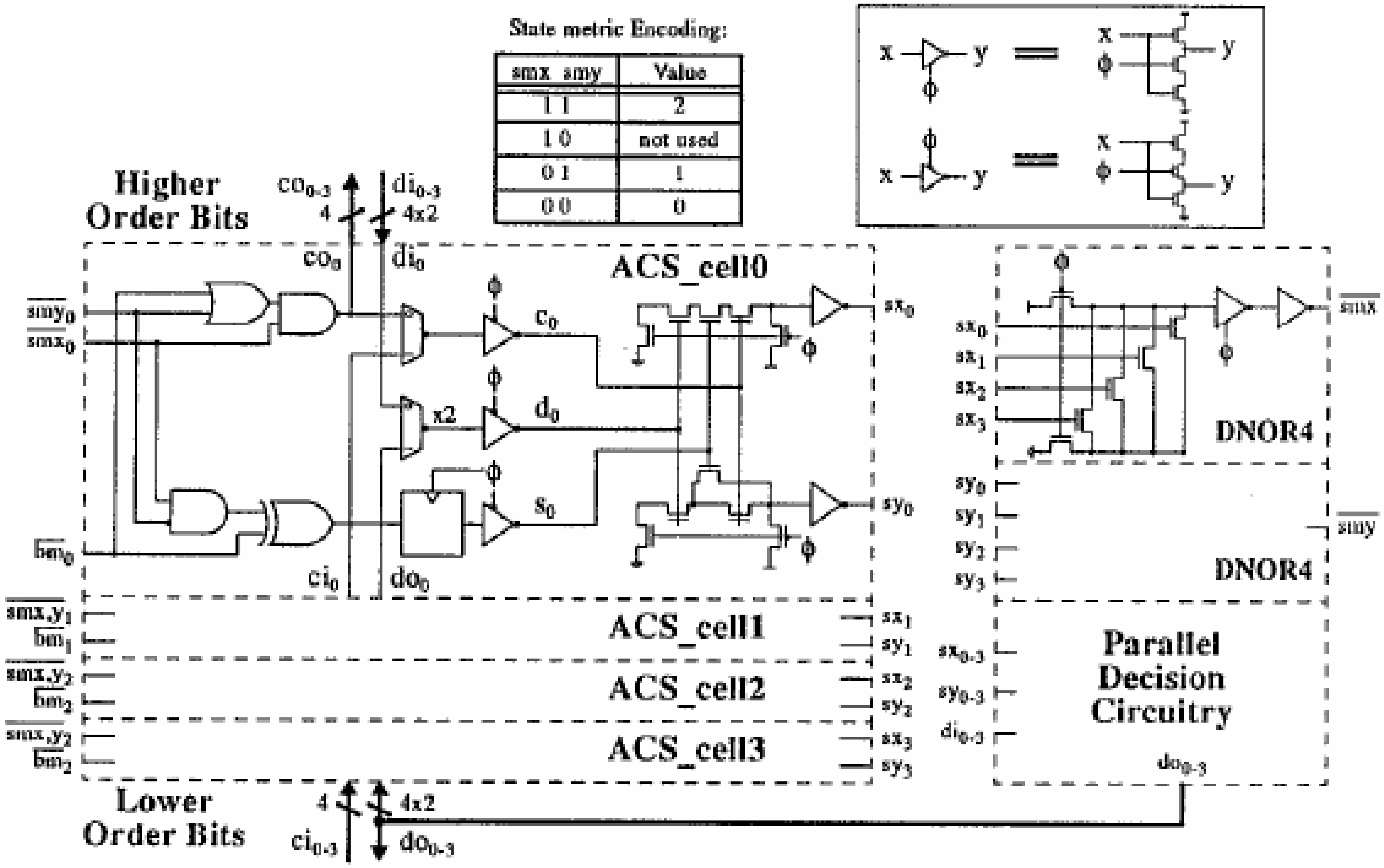

bm[3]
bm[2]
bm[1]
bm[0]
sm bm
co
s
di
csel
ci
do
sm
1
0
decision
compare-select unit
Note: For simplicity, only the addition of one state metric and one branch metric is shown.
cut-sets
bm[3,2]
bm[1,0]

**Figure 1: (a) 4b ACS unit using carry-propagation-free addition.
(b) Bit-level pipelined ACS (clock doubling, retiming).
(c) Bit-level pipelined, time-multiplexed ACS unit.**

Radix-4 ACS (one bit)



State metric Encoding:

smx smy	Value
1 1	2
1 0	not used
0 1	1
0 0	0

Figure 3: Radix-4 ACS bit-slice circuit.