Digital Integrated Circuits
A Design Perspective

Jan M. Rabaey
Anantha Chandrakasan
Borivoje Nikolic

Arithmetic Circuits
January, 2003
A Generic Digital Processor

- MEMORY
- DATAPATH
- INPUT-OUTPUT
- CONTROL
Building Blocks for Digital Architectures

Arithmetic unit
- Bit-sliced datapath (adder, multiplier, shifter, comparator, etc.)

Memory
- RAM, ROM, Buffers, Shift registers

Control
- Finite state machine (PLA, random logic.)
- Counters

Interconnect
- Switches
- Arbiters
- Bus
An Intel Microprocessor

Itanium has 6 integer execution units like this
Bit-Sliced Design

Tile identical processing elements

© Digital Integrated Circuits 2nd
Arithmetic Circuits
Bit-Sliced Datapath

From register files / Cache / Bypass

Multiplexers

Shifter

Adder stage 1

Wiring

Adder stage 2

Wiring

Adder stage 3

Sum Select

To register files / Cache

© Digital Integrated Circuits 2nd

Arithmetic Circuits
Itanium Integer Datapath
Adders
# Full-Adder

The Full-Adder is a circuit that adds two bits, $A$ and $B$, along with a carry-in, $C_{in}$, to produce a sum, $S$, and a carry-out, $C_{out}$.

![Full-Adder Diagram](image)

## Truth Table

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C_{in}$</th>
<th>$S$</th>
<th>$C_{out}$</th>
<th>Carry Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>delete</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>delete</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>propagate</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>propagate</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>propagate</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>propagate</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>generate</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>generate</td>
</tr>
</tbody>
</table>
The Binary Adder

\[ S = A \oplus B \oplus C_i \]

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

\[ C_o = AB + BC_i + AC_i \]
Express Sum and Carry as a function of $P$, $G$, $D$

Define 3 new variables which ONLY depend on $A$, $B$

- **Generate** ($G$) = $AB$
- **Propagate** ($P$) = $A \oplus B$
- **Delete** = $A \cdot B$

\[
C_o(G, P) = G + PC_i
\]
\[
S(G, P) = P \oplus C_i
\]

Can also derive expressions for $S$ and $C_o$ based on $D$ and $P$

Note that we will be sometimes using an alternate definition for

**Propagate** ($P$) = $A + B$
Complimentary Static CMOS Full Adder

28 Transistors

© Digital Integrated Circuits 2nd Edition
A Better Structure: The Mirror Adder

24 transistors
Mirror Adder

Stick Diagram
The Mirror Adder

• The NMOS and PMOS chains are **completely symmetrical**. A maximum of two series transistors can be observed in the carry-generation circuitry.

• When laying out the cell, the most critical issue is the minimization of the capacitance at node $C_o$. The reduction of the diffusion capacitances is particularly important.

• The capacitance at node $C_o$ is composed of four diffusion capacitances, two internal gate capacitances, and six gate capacitances in the connecting adder cell.

• The transistors connected to $C_i$ are placed closest to the output.

• Only the transistors in the carry stage have to be optimized for optimal speed. All transistors in the sum stage can be minimal size.
Transmission Gate Full Adder

Setup

Sum Generation

Carry Generation
One-phase dynamic CMOS adder
One-phase dynamic CMOS adder
One-phase dynamic CMOS adder
The Ripple-Carry Adder

Worst case delay linear with the number of bits

\[ t_d = O(N) \]

\[ t_{adder} = (N-1)t_{carry} + t_{sum} \]

Goal: Make the fastest possible carry path circuit
Inversion Property

\[ \bar{S}(A, B, C_i) = S(\bar{A}, \bar{B}, \bar{C_i}) \]

\[ \bar{C_o}(A, B, C_i) = C_o(\bar{A}, \bar{B}, \bar{C_i}) \]
Minimize Critical Path by Reducing Inverting Stages

Exploit Inversion Property
Carry Look-Ahead Adders

Carry boolean equations

\[ C_i = G_i + P_i \overline{C}_{i-1} = K_i + P_i \overline{C}_{i-1} \]
\[ \overline{C}_i = K_i + P_i \overline{C}_{i-1} = G_i + P_i C_{i-1} \]

Having defined:

\[ G_i = A_i \overline{B}_i \]
\[ P_i = A_i + B_i \]
\[ K_i = \overline{A}_i + \overline{B}_i = \overline{A}_i \overline{B}_i \]
Carry-Lookahead Adders

Therefore:

\[ C_1 = G_1 + P_1 C_0 \]
\[ C_2 = G_2 + P_2 (G_1 + P_1 C_0) \]
\[ C_3 = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 C_0)) \]
\[ C_4 = G_4 + P_4 (G_3 + P_3 (G_2 + P_2 (G_1 + P_1 C_0))) \]
Carry-Lookahead Adders

Two-level carry equations:

\[ C_1 = G_1 + P_1 C_0 \]
\[ C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0 \]
\[ C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0 \]
\[ C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0 \]
Expanding Lookahead equations:

\[ C_{o,k} = G_k + P_k (G_{k-1} + P_{k-1} C_{o,k-2}) \]

All the way:

\[ C_{o,k} = G_k + P_k (G_{k-1} + P_{k-1} (\ldots + P_1 (G_0 + P_0 C_{i,0}))) \]
Manchester Carry Chain
Manchester Carry Chain

\[ \overline{C}_{i,0}, P_0, \phi \]

\[ G_0, G_1, G_2, G_3 \]

\[ C_0, C_1, C_2, C_3 \]

\[ V_{DD} \]
Manchester Carry Chain

Stick Diagram

Propagate/Generate Row

Inverter/Sum Row

© Digital Integrated Circuits 2nd
Arithmetic Circuits
Carry-Bypass Adder

Also called Carry-Skip

Idea: If \((P_0 \text{ and } P_1 \text{ and } P_2 \text{ and } P_3 = 1)\) then \(C_{o,3} = C_0\), else “kill” or “generate”.
Carry-Bypass Adder (cont.)

$t_{adder} = t_{setup} + M t_{carry} + (N/M - 1) t_{bypass} + (M - 1) t_{carry} + t_{sum}$
Carry Ripple versus Carry Bypass

\[ t_p \]

\( N \)

ripple adder

bypass adder

4.8
Carry-Select Adder

Setup

P,G

"0" Carry Propagation

"0"

"1" Carry Propagation

"1"

Multiplexer

$C_{0,k-1}$

$C_{0,k+3}$

Sum Generation

Carry Vector
Carry Select Adder: Critical Path

Bit 0–3

- Setup
- 0-Carry
- 1-Carry
- Multiplexer
- Sum Generation
- $S_{0–3}$

Bit 4–7

- Setup
- 0-Carry
- 1-Carry
- Multiplexer
- Sum Generation
- $S_{4–7}$

Bit 8–11

- Setup
- 0-Carry
- 1-Carry
- Multiplexer
- Sum Generation
- $S_{8–11}$

Bit 12–15

- Setup
- 0-Carry
- 1-Carry
- Multiplexer
- Sum Generation
- $S_{12–15}$
Linear Carry Select

\[ t_{\text{add}} = t_{\text{setup}} + \left( \frac{N}{M} \right) t_{\text{carry}} + M t_{\text{mux}} + t_{\text{sum}} \]
Square Root Carry Select

\[ t_{\text{add}} = t_{\text{setup}} + P \cdot t_{\text{carry}} + (\sqrt{2N})t_{\text{mux}} + t_{\text{sum}} \]
Adder Delays - Comparison

![Diagram showing comparison of adder delays]

- Ripple adder
- Linear select
- Square root select

$t_p$ (in unit delays) vs. $N$

© Digital Integrated Circuits 2nd
Arithmetic Circuits
“O” Operator

- Definizione

\[(G_1^2, P_1^2) = (g_2, p_2) \circ (g_1, p_1) = (g_2 + p_2 g_1, p_2 p_1)\]

\[G_{2:1} = G_1^2 = g_2 + p_2 g_1\]

\[P_{2:1} = P_1^2 = p_2 p_1\]
Properties of the “O” operator

Proprietà associativa

\[ ((g_3, p_3) \circ (g_2, p_2)) \circ (g_1, p_1) = (g_3, p_3) \circ ((g_2, p_2) \circ (g_1, p_1)) \]

Dimostrazione

\[ ((g_3, p_3) \circ (g_2, p_2)) \circ (g_1, p_1) = (g_3 + p_3 g_2, p_3 p_2) \circ (g_1, p_1) = (g_3 + p_3 g_2 + p_3 p_2 g_1, p_3 p_2 p_1) \]

\[ (g_3, p_3) \circ ((g_2, p_2) \circ (g_1, p_1)) = (g_3, p_3) \circ (g_2 + p_2 g_1, p_2 p_1) = (g_3 + p_3(g_2 + p_2 g_1), p_3 p_2 p_1) = (g_3 + p_3 g_2 + p_3 p_2 g_1, p_3 p_2 p_1) \]
Properties of the “O” operator

Idempotenza

\[(g, p) \circ (g, p) = (g, p)\]

Elemento neutro: \((0,1)\)

\[(g, p) \circ (0, 1) = (g, p)\]

N.B.: Non gode della proprietà commutativa

\[(g_2, p_2) \circ (g_1, p_1) \neq (g_1, p_1) \circ (g_2, p_2)\]
Definizione:

\[(G_k^{0}, P_k^{0}) = (g_0, p_0) \text{ per } k = 0\]
\[(G_k^{0}, P_k^{0}) = (g_k, p_k) \circ (g_{k-1}, p_{k-1}) \circ \cdots \circ (g_0, p_0) \text{ per } k > 0\]

Inoltre, se \(i < k\):

\[(G_i^{k}, P_i^{k}) = (g_k, p_k) \circ (g_{k-1}, p_{k-1}) \circ \cdots \circ (g_i, p_i)\]
**Group Generate and Propagate**

**Teorema:**

Se $C_{in} = 0$, allora $G^k_0 = C_{out,k}$

_Dimostrazione per induzione:_

Posto $k = 0$, $C_{in} = 0$,

$G^0_0 = g_0$

$C_{out,0} = g_0 + p_0 C_{in} = g_0$

$G^0_0 = C_{out,0}$
Group Generate and Propagate

\[ \text{Posto } k = 1, \ C_{in,1} = C_{out,0} = g_0, \]

\[ G^1_0 = g_1 + p_1 g_0 \]

\[ C_{out,1} = g_1 + p_1 C_{in,1} = g_1 + p_1 g_0 \]

\[ G^1_0 = C_{out,1} \]

\[ \text{Posto } k > 1, \ C_{in,k} = C_{out,k-1} = G^{k-1}_0 \]

\[ (G^k_0, P^k_0) = (g_k, p_k) \circ (G^{k-1}_0, P^{k-1}_0) = (g_k + p_k G^{k-1}_0, p_k P^{k-1}_0) \]

\[ C_{out,k} = g_k + p_k C_{in,k} = g_k + p_k G^{k-1}_0 \]

\[ G^k_0 = C_{out,k} \]
Look-Ahead - Basic Idea

\[ C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1} \]
Logarithmic Look-Ahead Adder

\[ t_p \sim N \]

\[ t_p \sim \log_2(N) \]
Brent-Kung BLC adder

\[(g_{16}, p_{16}) \quad \rightarrow \quad (g_{15}, p_{15}) \quad \rightarrow \quad (g_{14}, p_{14}) \quad \rightarrow \quad (g_{13}, p_{13}) \quad \rightarrow \quad (g_{12}, p_{12}) \quad \rightarrow \quad (g_{11}, p_{11}) \quad \rightarrow \quad (g_{10}, p_{10}) \quad \rightarrow \quad (g_{9}, p_{9}) \quad \rightarrow \quad (g_{8}, p_{8}) \quad \rightarrow \quad (g_{7}, p_{7}) \quad \rightarrow \quad (g_{6}, p_{6}) \quad \rightarrow \quad (g_{5}, p_{5}) \quad \rightarrow \quad (g_{4}, p_{4}) \quad \rightarrow \quad (g_{3}, p_{3}) \quad \rightarrow \quad (g_{2}, p_{2}) \quad \rightarrow \quad (g_{1}, p_{1}) \quad \rightarrow \quad c_{16} \quad \rightarrow \quad c_{15} \quad \rightarrow \quad c_{14} \quad \rightarrow \quad c_{13} \quad \rightarrow \quad c_{12} \quad \rightarrow \quad c_{11} \quad \rightarrow \quad c_{10} \quad \rightarrow \quad c_{9} \quad \rightarrow \quad c_{8} \quad \rightarrow \quad c_{7} \quad \rightarrow \quad c_{6} \quad \rightarrow \quad c_{5} \quad \rightarrow \quad c_{4} \quad \rightarrow \quad c_{3} \quad \rightarrow \quad c_{2} \quad \rightarrow \quad c_{1}\]
Folding of the inverse tree
Folding the inverse tree
Dense tree with minimum fan-out
Dense tree with simple connections
Carry Lookahead Trees

\[
C_{o,0} = G_0 + P_0 C_{i,0}
\]

\[
C_{o,1} = G_1 + P_1 G_0 + P_1 P_0 C_{i,0}
\]

\[
C_{o,2} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0}
\]

\[
= (G_2 + P_2 G_1) + (P_2 P_1)(G_0 + P_0 C_{i,0}) = G_{2:1} + P_{2:1} C_{o,0}
\]

Can continue building the tree hierarchically.
Tree Adders

16-bit radix-2 Kogge-Stone tree
Tree Adders

16-bit radix-4 Kogge-Stone Tree
Sparse Trees

16-bit radix-2 sparse tree with sparseness of 2
Tree Adders

Brent-Kung Tree

© Digital Integrated Circuits 2nd
Arithmetic Circuits
Example: Domino Adder

- $P_i = a_i + b_i$
- $G_i = a_i b_i$

**Propagate**
- $V_{DD}$
- $Clk$
- $a_i$
- $b_i$

**Generate**
- $V_{DD}$
- $Clk$
- $G_i = a_i b_i$
- $a_i$
- $b_i$
Example: Domino Adder

Propagate

Generate
Example: Domino Sum

![Diagonal Sum Circuit Diagram]
## Adders – Summary

<table>
<thead>
<tr>
<th>Adder</th>
<th>Complexity</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripple carry</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>Carry select [1]</td>
<td>$2N$</td>
<td>$(2\sqrt{N} - 1)$</td>
</tr>
<tr>
<td>Carry select [2]</td>
<td>$2N$</td>
<td>$\sqrt{(2N - 7/4)} + 1/2$</td>
</tr>
<tr>
<td>Carry skip</td>
<td>$N$</td>
<td>$2(\sqrt{2N} - 1)$</td>
</tr>
<tr>
<td>Bin. look. (tree)</td>
<td>$2(N - 1) - \log N$</td>
<td>$2 \log N$</td>
</tr>
<tr>
<td>Bin. look. (array)</td>
<td>$N \log N$</td>
<td>$2 + \log N$</td>
</tr>
</tbody>
</table>

[1] Uniform blocks sized $K = \sqrt{N}$

[2] Scaled blocks with $K_{min} = 1$ or $K_{min} = 2$
Multipliers
The Binary Multiplication

\[ X = \sum_{n=0}^{N-1} x_n 2^n, \quad Y = \sum_{n=0}^{N-1} y_n 2^n \]

\[ Z = XY = \left( \sum_{n=0}^{N-1} x_n 2^n \right) \left( \sum_{m=0}^{N-1} x_m 2^m \right) \]

\[ = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x_n y_m 2^{n+m} = \sum_{n=0}^{2N-1} z_n 2^n \]
# The Binary Multiplication

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>$y_3$</th>
<th>$y_2$</th>
<th>$y_1$</th>
<th>$y_0$</th>
<th>$\times$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>$x_3$</td>
<td>$x_2$</td>
<td>$x_1$</td>
<td>$x_0$</td>
<td>$=$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partial Products</th>
<th>$x_0y_3$</th>
<th>$x_0y_2$</th>
<th>$x_0y_1$</th>
<th>$x_0y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x_1y_3$</td>
<td>$x_1y_2$</td>
<td>$x_1y_1$</td>
<td>$x_1y_0$</td>
</tr>
<tr>
<td></td>
<td>$x_2y_3$</td>
<td>$x_2y_2$</td>
<td>$x_2y_1$</td>
<td>$x_2y_0$</td>
</tr>
<tr>
<td></td>
<td>$x_3y_3$</td>
<td>$x_3y_2$</td>
<td>$x_3y_1$</td>
<td>$x_3y_0$</td>
</tr>
<tr>
<td></td>
<td>$z_7$</td>
<td>$z_6$</td>
<td>$z_5$</td>
<td>$z_4$</td>
</tr>
<tr>
<td></td>
<td>$z_3$</td>
<td>$z_2$</td>
<td>$z_1$</td>
<td>$z_0$</td>
</tr>
</tbody>
</table>

© Digital Integrated Circuits 2nd Arithmetic Circuits
The Binary Multiplication

\[
\begin{array}{cccccc}
1 & 0 & 1 & 0 & 1 & 0 \\
\times & 1 & 0 & 1 & 1 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
\hline
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
\end{array}
\]

- Multiplicand
- Multiplier
- Partial products
- Result
The Array Multiplier

© Digital Integrated Circuits 2nd

Arithmetic Circuits
The MxN Array Multiplier — Critical Path

\[ t_{mult} = [(M - 1) + (N - 2)] t_{carry} + (N - 1) t_{sum} + t_{and} \]
Carry-Save Multiplier

\[ t_{\text{mult}} = (N-1)t_{\text{carry}} + (N-1)t_{\text{and}} + t_{\text{merge}} \]
X and Y signals are broadcasted through the complete array.
Wallace-Tree Multiplier

Partial products

First stage

Second stage

Final adder
Wallace-Tree Multiplier

Partial products

First stage

Second stage

Final adder

© Digital Integrated Circuits 2nd
Arithmetic Circuits
Wallace-Tree Multiplier
Wallace Tree Mult. Performance

\[
\left( \frac{3}{2} \right)^n = \frac{N}{2}
\]

\[
n \log_2 \left( \frac{3}{2} \right) = \log_2 \left( \frac{N}{2} \right)
\]

\[
n = \frac{\log_2 \left( \frac{N}{2} \right)}{\log_2 \left( \frac{3}{2} \right)} = 1.71 \left( \log_2 N - 1 \right)
\]

\[
t_{\text{mult}} = 1.71 \left( \log_2 N - 1 \right) t_{\text{sum}} + t_{\text{adder}}
\]
Wallace Tree Multiplier Complexity

\[ N_{CSA} = \frac{N}{3} + \left( \frac{2}{3} \right) \frac{N}{3} + \left( \frac{2}{3} \right)^2 \frac{N}{3} + \ldots + \left( \frac{2}{3} \right)^{n-1} \frac{N}{3} \]

\[ = \frac{N}{3} \sum_{n=0}^{n} \left( \frac{2}{3} \right)^{n-1} \]

\[ = \frac{N}{3} \frac{1 - (2/3)^n}{1 - 2/3} = N - 2 \]

\[ A_{WTM} \sim N (N - 2) A_{adder} \]
4:2 Adder

![4:2 Adder Diagram](image)

<table>
<thead>
<tr>
<th>n</th>
<th>Cin</th>
<th>Cout</th>
<th>Carry</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Eight-input Tree
Architectural comparison of multiplier solutions
SPIM Architecture

Diagram of SPIM architecture showing the flow of data through the Booth Encoders, A Block Booth Muxes, B Block Booth Muxes, A Block 4:2, B Block 4:2, C Block 4:2, D Block Accumulator, and To Carry Propagate Adder.
## SPIM Pipe Timing

<table>
<thead>
<tr>
<th>Action</th>
<th>Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Booth Encode startup 0-15</td>
<td></td>
<td>16-31</td>
<td>32-47</td>
<td>48-63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A and B block Booth Muxs</td>
<td>0-15</td>
<td></td>
<td>16-31</td>
<td>32-47</td>
<td>48-63</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A Block CSA's</td>
<td></td>
<td>0-7</td>
<td></td>
<td>16-23</td>
<td>32-39</td>
<td>48-55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B Block CSA's</td>
<td></td>
<td>8-15</td>
<td></td>
<td>24-31</td>
<td>40-47</td>
<td>56-63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C Block</td>
<td></td>
<td>0-15</td>
<td></td>
<td>16-31</td>
<td>32-47</td>
<td>48-63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D Block</td>
<td></td>
<td></td>
<td></td>
<td>Clear 0-15</td>
<td></td>
<td>16-31</td>
<td>32-47</td>
<td>48-63</td>
<td></td>
</tr>
</tbody>
</table>
SPIM Microphotograph

A Block

B Block

C Block

D Block

CPAdder and Output bus drivers
SPIM clock generator circuit
Binary Tree Multiplier Performance

\[ 2^n = \frac{N}{2} \]

\[ n = \log_2 \left( \frac{N}{2} \right) \]

\[ t_{\text{mult}} = 2 \left( \log_2 N - 1 \right) t_{\text{sum}} + t_{\text{adder}} \]
Binary Tree Multiplier Complexity

\[ N_{CSA} = \frac{N}{4} + \left( \frac{1}{2} \right) \frac{N}{4} + \left( \frac{1}{2} \right)^2 \frac{N}{4} + \ldots + \left( \frac{1}{2} \right)^{n-1} \frac{N}{4} \]

\[ = \frac{N}{4} \sum_{k=0}^{n-1} \left( \frac{1}{2} \right)^k \]

\[ = \frac{N}{4} \frac{1 - (1/2)^n}{1 - 1/2} = \frac{N}{2} - 1 \]

\[ A_{BTM} \approx N (N - 2) A_{adder} \]
**Multipliers – 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**
# Multipliers – Summary

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Complexity</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial multiplier</td>
<td>$N$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>Serial carry save</td>
<td>$N$</td>
<td>$2N$</td>
</tr>
<tr>
<td>Array multiplier</td>
<td>$N(N-2)$</td>
<td>$N-2$</td>
</tr>
<tr>
<td>Wallace tree</td>
<td>$N(N-2)$</td>
<td>$1.71 \log N - 1$</td>
</tr>
<tr>
<td>Binary tree</td>
<td>$N(N-2)$</td>
<td>$2 \log N - 1$</td>
</tr>
</tbody>
</table>
Booth encoding

\[
X = \sum_{n=0}^{N-2} x_n 2^n - x_{N-1} 2^{N-1}
\]

\[
= \sum_{n=0}^{N/2-1} x_{2n} 2^{2n} + \sum_{n=1}^{N/2-1} x_{2n-1} 2^{2n-1} - x_{N-1} 2^{N-1}
\]

\[
= \sum_{n=0}^{N/2-1} (x_{2n-1} + x_{2n} - 2x_{2n+1}) 2^{2n}
\]
### Booth encoding

<table>
<thead>
<tr>
<th>$x_{2n+1}$</th>
<th>$x_{2n}$</th>
<th>$x_{2n-1}$</th>
<th>$f(2n)$</th>
<th>$f(2n)Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$Y$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$Y$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>$2Y$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$-2$</td>
<td>$-2Y$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$-1$</td>
<td>$-Y$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$-1$</td>
<td>$-Y$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Tree Multiplier with Booth Encoding

Y
Booth Encoder
f(2n)
Booth MUX
CSA
CSA
CSA
CSA
CSA
CSA
CPA

© Digital Integrated Circuits 2nd
Arithmetic Circuits
The f.p. addition algorithm

- Exponent comparison and swap (if needed)
- Mantissas’ alignment
- Addition
- Normalization of result
- Rounding of result
The f.p. multiplication algorithm

- Mantissas’ multiplication
- Exponent addition
- Mantissa normalization and exponent adjusting (if needed)
- Rounding of result
Dividers
Iterative Division (Newton-Raphson)

\[
\frac{a}{b} = \frac{1}{b} \times a
\]

\[
f(x) = \frac{1}{x} - b = 0 \quad \rightarrow \quad x^* = \frac{1}{b}
\]

\[
y(x) \approx f(x_0) + f'(x_0) (x - x_0)
\]
Iterative Division (Newton-Raphson)

\[ x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 - \frac{1/x_0 - b}{-1/x_0^2} = x_0(2 - b x_0) \]

\[ x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = x_1 - \frac{1/x_1 - b}{-1/x_1^2} = x_1(2 - b x_1) \]

\[ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{1/x_n - b}{-1/x_n^2} = x_n(2 - b x_n) \]
Quadratic Convergence of the Newton Method

\[ \varepsilon_n = \left| \frac{x_n - 1/b}{1/b} \right| = |1 - b \, x_n| \]

\[ \varepsilon_{n+1} = |1 - b \, x_{n+1}| = |1 - b \, x_n (2 - b \, x_n)| \]

\[ = |(1 - b \, x_n)^2| = \varepsilon_n^2 \]
Properties of the Newton Method

- Asymptotically quadratic convergence
- Correction of round-off errors
- Final multiplication by $a$ generates a round-off problem incompatible with the Standard IEEE-754
Iterative Division (Goldschmidt)

Define two sequences $x_n$ and $y_n$ such that

$$x_0 = a \quad \text{and} \quad y_0 = b$$

$$\frac{a}{b} = \frac{x_0}{y_0} = \frac{x_1}{y_1} = \frac{x_2}{y_2} = \ldots = \frac{x_n}{y_n} = \ldots$$

$$\delta = 1 - y_0 \quad \rightarrow \quad |\delta| < 1$$

$$y_1 = y_0 (1 + \delta) = 1 - \delta^2$$
Iterative division (Goldschmidt)

\[
x_1 = x_0 (1 + \delta)
\]

\[
y_1 = y_0 (1 + \delta) = 1 - \delta^2
\]

\[
x_2 = x_0 (1 + \delta)(1 + \delta^2)
\]

\[
y_2 = y_0 (1 + \delta)(1 + \delta^2) = 1 - \delta^4
\]

\[
x_n = x_0 (1 + \delta)(1 + \delta^2) \ldots (1 + \delta^{2^{n-1}})
\]

\[
y_n = y_0 (1 + \delta)(1 + \delta^2) \ldots (1 + \delta^{2^{n-1}}) = 1 - \delta^{2^n}
\]
Iterative Division (Goldschmidt)

- The sequence of $x_n$ tends to $a/b$
- The convergence is quadratic
- In its present form, this method is affected by round-off errors
Modified Goldschmidt Algorithm (correction of round-off errors)

\[
x_1 = x_0 (1 + \delta_0)
\]

\[
y_1 = y_0 (1 + \delta_0) = 1 - \delta_0^2
\]

\[
x_2 = x_0 (1 + \delta_0)(1 + \delta_1)
\]

\[
y_2 = y_0 (1 + \delta_0)(1 + \delta_1)
\]

\[
x_n = x_0 (1 + \delta_0)(1 + \delta_1).....(1 + \delta_{n-1})
\]

\[
y_n = y_0 (1 + \delta_0)(1 + \delta_1).....(1 + \delta_{n-1})
\]
The Link between the Newton-Raphson and Goldschmidt Methods

By setting $\delta_n = 1 - b x_n$, the Newton method provides

\[
x_1 = x_0(1 + \delta_0)
\]
\[
x_2 = x_0(1 + \delta_0)(1 + \delta_1)
\]
\[
... = ............
\]
\[
x_n = x_0(1 + \delta_0)(1 + \delta_1) ...... (1 + \delta_{n-1})
\]
The Link between the Newton-Raphson and Goldschmidt Methods

By expressing $\delta_n$ vs $\delta_0$, the Newton method provides

\[
x_1 = x_0(1 + \delta_0)
\]
\[
x_2 = x_0(1 + \delta_0)(1 + \delta_0^2)
\]
\[
\text{....} = \text{...........}
\]
\[
x_n = x_0(1 + \delta_0)(1 + \delta_0^2) \ldots \ldots (1 + \delta_0^{2^n-1})
\]
Comparison between Newton and Goldschmidt methods

- The Newton and Goldschmidt methods are essentially equivalent;
- Both methods exhibit an asymptotically quadratic convergence;
- Both methods are able to correct round-off errors;
- The Goldschmidt methods directly compute the $a/b$ ratio.
Shifters
The Binary Shifter

Right  nop  Left

Bit-Slice i

A_i  A_{i-1}

B_i  B_{i-1}
The Barrel Shifter

Area Dominated by Wiring
4x4 barrel shifter

\[ \text{Width}_{\text{barrel}} \sim 2 \ p_m \ M \]
Logarithmic Shifter
0-7 bit Logarithmic Shifter

\[ \text{width}_{\log} \approx p_m \left( 2^K + \left( 1 + 2 + \ldots + 2^{K-1} \right) \right) = p_m \left( 2^K + 2^K - 1 \right) \]
ALUs
Two-bit MUX

\[ F(A, B) = G_0 \cdot \overline{A} \overline{B} + G_1 \cdot \overline{A} B + G_2 \cdot A \overline{B} + G_3 \cdot A B \]
# Two-bit MUX Truth Table

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$G_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$G_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$G_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$G_3$</td>
</tr>
</tbody>
</table>

\[
F(A, B) = G_0 \cdot \overline{A} \overline{B} + G_1 \cdot \overline{A} B + G_2 \cdot A \overline{B} + G_3 \cdot A B
\]
# Two-bit Selector Truth Table

<table>
<thead>
<tr>
<th>$G_3$</th>
<th>$G_2$</th>
<th>$G_1$</th>
<th>$G_0$</th>
<th>$F(A, B)$</th>
<th>$G_3$</th>
<th>$G_2$</th>
<th>$G_1$</th>
<th>$G_0$</th>
<th>$F(A, B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$A \cdot B = \overline{A} + \overline{B}$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$A \cdot \overline{B} = A + B$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$A \cdot \overline{B} + A \cdot B$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$\overline{A} \cdot B = A + \overline{B}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$B$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$\overline{A}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$A \cdot \overline{B} = \overline{A} + B$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$A \cdot \overline{B} = \overline{A} + B$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$A$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$\overline{B}$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$A \cdot \overline{B} = A + \overline{B}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$A \cdot \overline{B} + \overline{A} \cdot B$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$\overline{A} \cdot \overline{B} = A + B$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$A \cdot \overline{B} = A + B$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$$F(A, B) = G_0 \cdot \overline{A} \overline{B} + G_1 \cdot \overline{A} B + G_2 \cdot A \overline{B} + G_3 \cdot A B$$

© Digital Integrated Circuits

Arithmetic Circuits
### Carry-chain Truth Table

<table>
<thead>
<tr>
<th>$C_{in}$</th>
<th>$KILL$</th>
<th>$PROP$</th>
<th>$C_{out}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>N.A.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

\[
C_{out} = K + P \overline{C_{in}} = \overline{K} (\overline{P} + C_{in})
\]
ALU block diagram (Mead-Conway)

\[ A_3 \rightarrow \text{KILL} \rightarrow P \rightarrow \text{PROP} \rightarrow P \rightarrow \text{Carry} \rightarrow P \rightarrow \text{RES} \rightarrow O_3 \]

\[ A_2 \rightarrow \text{KILL} \rightarrow P \rightarrow \text{PROP} \rightarrow P \rightarrow \text{Carry} \rightarrow P \rightarrow \text{RES} \rightarrow O_2 \]

\[ A_1 \rightarrow \text{KILL} \rightarrow P \rightarrow \text{PROP} \rightarrow P \rightarrow \text{Carry} \rightarrow P \rightarrow \text{RES} \rightarrow O_1 \]

\[ A_0 \rightarrow \text{KILL} \rightarrow P \rightarrow \text{PROP} \rightarrow P \rightarrow \text{Carry} \rightarrow P \rightarrow \text{RES} \rightarrow O_0 \]

\[ K_3 K_2 K_1 K_0 \rightarrow \text{KILL} \rightarrow P \rightarrow \text{PROP} \rightarrow P \rightarrow \text{Carry} \rightarrow P \rightarrow \text{RES} \]

\[ R_3 R_2 R_1 R_0 \rightarrow \text{RES} \rightarrow O_3 \]

\[ C_{out} \rightarrow \text{Carry} \rightarrow P \rightarrow \text{RES} \rightarrow O_3 \]

\[ C_{in} \rightarrow \text{RES} \rightarrow O_3 \]

\[ C_{in} \rightarrow \text{RES} \rightarrow O_3 \]
# ALU Operations

<table>
<thead>
<tr>
<th>Function</th>
<th>$KILL$</th>
<th>$PROP$</th>
<th>$RES$</th>
<th>$C_{in}$</th>
<th>$CF$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A + B$</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A + B + C_{in}$</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$A - B$</td>
<td>2</td>
<td>9</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$B - A$</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$A - B - C_{in}$</td>
<td>2</td>
<td>9</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$B - A - C_{in}$</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
## ALU Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>( KILL )</th>
<th>( PROP )</th>
<th>( RES )</th>
<th>( C_{in} )</th>
<th>( CS )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A + 1 )</td>
<td>3</td>
<td>12</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>( B + 1 )</td>
<td>5</td>
<td>10</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>( A - 1 )</td>
<td>12</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>( B - 1 )</td>
<td>10</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>( -A )</td>
<td>12</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>( -B )</td>
<td>10</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
### ALU Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>KILL</th>
<th>PROP</th>
<th>RES</th>
<th>C&lt;sub&gt;in&lt;/sub&gt;</th>
<th>C&lt;sub&gt;S&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.and.B</td>
<td>0</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A.or.B</td>
<td>0</td>
<td>14</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A.xor.B</td>
<td>0</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\overline{A}$</td>
<td>0</td>
<td>3</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\overline{B}$</td>
<td>0</td>
<td>5</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>10</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## ALU Operations

<table>
<thead>
<tr>
<th>Function</th>
<th>$KILL$</th>
<th>$PROP$</th>
<th>$RES$</th>
<th>$C_{in}$</th>
<th>$CF$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiply</td>
<td>1</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Divide</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$A/O$</td>
<td>0</td>
<td>14</td>
<td>12</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Mask</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>SHL $A$</td>
<td>3</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Zero</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
# ALU Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Select</th>
<th>Flag bit</th>
<th>( KILL )</th>
<th>( PROP )</th>
<th>( RES )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncond.</td>
<td>0</td>
<td>×</td>
<td>- - - -</td>
<td>- - - -</td>
<td>- - - -</td>
</tr>
<tr>
<td>Multiply</td>
<td>1</td>
<td>0</td>
<td>- - - 0</td>
<td>- 0 0 -</td>
<td>- 0 0 -</td>
</tr>
<tr>
<td>Multiply</td>
<td>1</td>
<td>1</td>
<td>- - - -</td>
<td>0 - - -</td>
<td>0 - - -</td>
</tr>
<tr>
<td>Divide</td>
<td>2</td>
<td>0</td>
<td>- - - 0</td>
<td>0 0 0 -</td>
<td>0 - - 0</td>
</tr>
<tr>
<td>Divide</td>
<td>2</td>
<td>1</td>
<td>- 0 - 0</td>
<td>0 - 0 0</td>
<td>0 - 0 0</td>
</tr>
<tr>
<td>AND/OR</td>
<td>3</td>
<td>0</td>
<td>- - - -</td>
<td>- - - -</td>
<td>- - - -</td>
</tr>
<tr>
<td>AND/OR</td>
<td>3</td>
<td>1</td>
<td>- - - -</td>
<td>0 0 0 -</td>
<td>- - - -</td>
</tr>
</tbody>
</table>
MIPS-X Instruction Format

- **Memory**
  - `1 0` | OP | Src1 | Dest | Offset (17)

- **Branch**
  - `0 0` | Cond | Src1 | Src2 |elo | Disp (16)

- **Compute**
  - `1 0` | OP | Src1 | Src2 | Dest | Comp Func (12)

- **Compute Immediate**
  - `1 1` | OP | Src1 | Dest | Immed (17)
Pipeline dependencies in MIPS-X
Die Photo of MIPS-X
MIPS-X Architecture
MIPS-X Instruction Cache-miss timing
MIPS-X Tag Memory
MIPS-X Valid Store Array
RAM Sense Amplifier
CMOS Dual-port Register Cell
Self-timed bit-line write circuit
Register bypass logic
Schematic of comparator circuit
Squash FSM
Cache-miss FSM