Digital Integrated Circuits
A Design Perspective

Jan M. Rabaey
Anantha Chandrakasan
Borivoje Nikolic

Arithmetic Circuits

January, 2003
A Generic Digital Processor
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
Itanium has 6 integer execution units like this
Bit-Sliced Design

Tile identical processing elements
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
Adders
Full-Adder

\[ \begin{array}{cccccc}
A & B & C_i & S & C_o & \text{Carry status} \\
0 & 0 & 0 & 0 & 0 & \text{delete} \\
0 & 0 & 1 & 1 & 0 & \text{delete} \\
0 & 1 & 0 & 1 & 0 & \text{propagate} \\
0 & 1 & 1 & 0 & 1 & \text{propagate} \\
1 & 0 & 0 & 1 & 0 & \text{propagate} \\
1 & 0 & 1 & 0 & 1 & \text{propagate} \\
1 & 1 & 0 & 0 & 1 & \text{generate} \\
1 & 1 & 1 & 1 & 1 & \text{generate} \\
\end{array} \]
The Binary Adder

\[ S = A \oplus B \oplus C_i \]
\[ = A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_j \]
\[ C_o = AB + BC_i + AC_i \]
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* = $A \odot B$

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

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

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

28 Transistors
Inversion Property

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

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

Exploit Inversion Property
A Better Structure: The Mirror Adder

24 transistors
Mirror Adder

Stick Diagram

V_{DD}

A  B  C_i  B  A  C_i  A  B

C_o

S

GND

© Digital Integrated Circuits 2nd

Arithmetic Circuits
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

© Digital Integrated Circuits 2nd

Arithmetic Circuits
Manchester Carry Chain

\[ C_i \rightarrow \overline{P_i} \rightarrow \overline{G_i} \rightarrow V_{DD} \rightarrow \overline{C_i} \rightarrow \overline{C_o} \]

\[ P_i \rightarrow D_i \rightarrow \overline{G_i} \rightarrow V_{DD} \rightarrow \overline{C_i} \rightarrow \overline{C_o} \]
Manchester Carry Chain

Stick Diagram

Propagate/Generate Row

Inverter/Sum Row
Carry-Bypass Adder

Also called Carry-Skip

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

© Digital Integrated Circuits 2nd
Arithmetic Circuits
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

![Graph showing comparison between ripple and bypass adders.](image)
Carry-Select Adder

Setup

"0"

"0" Carry Propagation

"1"

"1" Carry Propagation

C_{0,k-1}

Multiplexer

C_{0,k+3}

Sum Generation

P,G

Carry Vector

EE141

© Digital Integrated Circuits 2nd

Arithmetic Circuits
Carry Select Adder: Critical Path

Bit 0–3
- Setup
- 0-Carry
- 1-Carry
- Multiplexer
- Sum Generation

Bit 4–7
- Setup
- 0-Carry
- 1-Carry
- Multiplexer
- Sum Generation

Bit 8–11
- Setup
- 0-Carry
- 1-Carry
- Multiplexer
- Sum Generation

Bit 12–15
- Setup
- 0-Carry
- 1-Carry
- Multiplexer
- Sum Generation
Linear Carry Select

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

![Graph showing comparison of adder delays for Ripple adder, Linear select, and Square root select.](image)
LookAhead - Basic Idea

\[ C_{0,k} = f(A_k, B_k, C_{0,k-1}) = G_k + P_k C_{0,k-1} \]
Expanding Lookahead equations:

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

All the way:

\[ C_{o,k} = G_k + P_k (G_{k-1} + P_{k-1} (\ldots + P_1 (G_0 + P_0 C_{i,0}))) \]
Logarithmic Look-Ahead Adder

\[ t_p \sim N \]

\[ t_p \sim \log_2(N) \]
Carry Lookahead Trees

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

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

© Digital Integrated Circuits 2nd Arithmetic Circuits
Tree Adders

Brent-Kung Tree
Example: Domino Adder

\[ P_i = a_i + b_i \]

Propagate

Generate

\( G_i = a_i b_i \)
Example: Domino Adder

Propagate

Generate
Example: Domino Sum
Multipliers
The Binary Multiplication

\[ Z = \bar{X} \times Y = \sum_{k=0}^{M+N-1} Z_k 2^k \]

\[ = \left( \sum_{i=0}^{M-1} X_i 2^i \right) \left( \sum_{j=0}^{N-1} Y_j 2^j \right) \]

\[ = \sum_{i=0}^{M-1} \left( \sum_{j=0}^{N-1} X_i Y_j 2^{i+j} \right) \]

with

\[ X = \sum_{i=0}^{M-1} X_i 2^i \]

\[ Y = \sum_{j=0}^{N-1} Y_j 2^j \]
The Binary Multiplication

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

- **Multiplicand**: 101010
- **Multiplier**: 1011
- **Partial products**:
  - 101010
  - 000000
  - 101010
- **Result**: 111001110
The Array Multiplier
The MxN Array Multiplier

— Critical Path

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

\[ t_{\text{mult}} = (N-1)t_{\text{carry}} + (N-1)t_{\text{and}} + t_{\text{merge}} \]
Multiplier Floorplan

X and Y signals are broadcasted through the complete array.

( — — — )
Wallace-Tree Multiplier

Partial products

\[
\begin{array}{cccccc}
6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}
\]

First stage

\[
\begin{array}{cccccc}
6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}
\]

Bit position

\[
\begin{array}{cccccc}
\circ & \circ & \circ & \circ & \circ & \circ \\
\end{array}
\]

Second stage

\[
\begin{array}{cccccc}
6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}
\]

Final adder

\[
\begin{array}{cccccc}
6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
\circ & \circ & \circ & \circ & \circ & \circ \\
\end{array}
\]

(c) FA

(d) HA
Wallace-Tree Multiplier

Partial products

First stage

Second stage

Final adder
Wallace-Tree Multiplier

© Digital Integrated Circuits 2nd
Arithmetic Circuits
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
Shifters
The Binary Shifter

[Diagram of a binary shifter with labels for Ai, Bi, Ai-1, Bi-1, and Bit-Slice i.]

© Digital Integrated Circuits 2nd
Arithmetic Circuits
The Barrel Shifter

Area Dominated by Wiring
4x4 barrel shifter

\[ \text{Width}_{\text{barrel}} \sim 2 \ p_m \ M \]
Logarithmic Shifter

Sh1 Sh1
Sh2 Sh2
Sh4 Sh4

A3
A2
A1
A0

B3
B2
B1
B0

© Digital Integrated Circuits 2nd
Arithmetic Circuits
0-7 bit Logarithmic Shifter

\[ \text{width}_{\log} \cdot p_m \left( 2^K + \left( 1 + 2 + \ldots + 2^{K-1} \right) \right) = p_m \left( 2^K + 2^K - 1 \right) \]