EE5311- Digital IC Design

Module 6 - Adders and Multipliers

Janakiraman V

Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology Madras
Chennai

November 23, 2018

Learning Objectives

- Design a full adder with least PMOS stack size using self duality principle
- Construct adder architectures to reduce delay from $O(N)$ to $O(\sqrt{N})$ - $O(log_2(N))$
- Draw timing diagrams to show the signal propagation of various adders
- Design an array multiplier for both signed and unsigned multiplication
- Optimize the arrary multiplier using the inverting property of a Full Adder
- Derive the Modified Booth Encoding to reduce the number of partial products
- Design and implement a multipler based on the Modified Booth Encoding algorithm

Outline

- Adders
  - Basic terminology
  - Full adder circuit design
  - Inverting Adder
  - Carry Save Adder
  - Carry Select Adder
  - Carry Look Ahead Adder
- Multipliers
  - Basic Terminology
  - Booth and Modified Booth Encoding
  - 2s Complement Airthmetic
  - Array Multiplier
  - Carry Save Multipler
  - Signed multiplication and carry save implementation
  - Final Addition

Basic Adder Terminology

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

Table: Truth Table of a Full Adder

$$S = A \oplus B \oplus C_i$$
$$C_o = AB + BC_i + C_iA$$

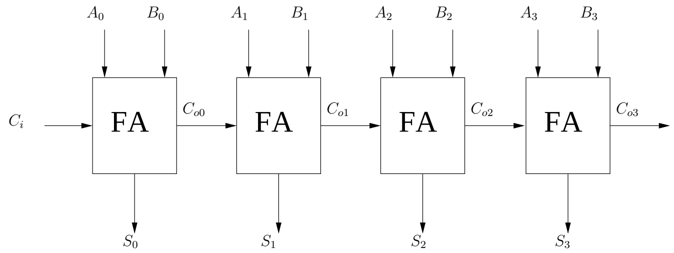
Basic Adder Terminology

- $G = AB$ - Generate Carry
- $D = \overline{A}.\overline{B}$ - Delete Carry
- $P = A \oplus B$ - Propagate Carry

G, D, P are independent of C_i

$$S = P \oplus C_i$$
$$C_o = G + PC_i$$

Ripple Adder



$$t_{ripple-adder} = (N-1)t_{carry} + t_{sum}$$

- Worst case delay is linear in N i.e. $t_{ripple-adder} \propto N$
- Carry optimization is more important than Sum

Properties of a Full Adder

Logic functions are written in SoP form (Sum of minterms)

$$\begin{aligned} S &= \sum m(1,2,4,7) \\ \overline{S} &= \sum m(0,3,5,6) \\ C_o &= \sum m(3,5,6,7) \\ \overline{C_o} &= \sum m(0,1,2,4) \end{aligned}$$

Inverting Property

Full Adder - Inverting Property

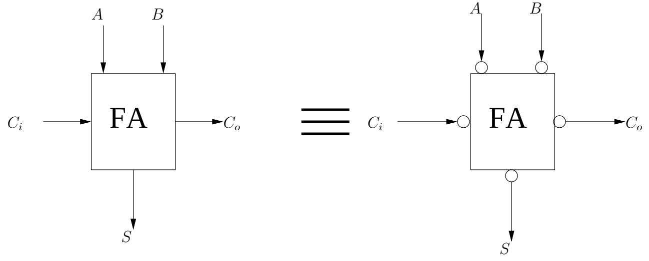


Figure: Inverting Property

$$\overline{S} = S(\overline{A}, \overline{B}, \overline{C})$$
$$\overline{C_o} = C_o(\overline{A}, \overline{B}, \overline{C})$$

Full Adder

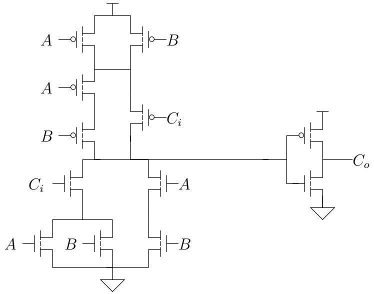


Figure: Full Adder Carry Circuit

$$C_o = AB + BC_i + C_iA$$

$$S = ABC_i + \overline{C_o}(A + B + C_i)$$

Full Adder - Problems

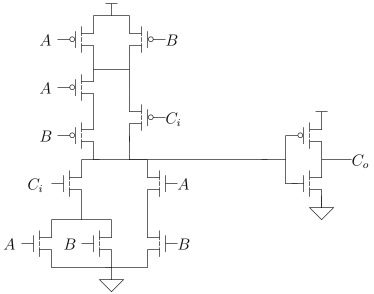


Figure: Full Adder Carry Circuit

- Tall PMOS stacks in both Sum and Carry
- Load capacitance of C_o is very large - $6C_G + 2C_{Diff}$
- Carry goes through extra inverter - Critical path
- Sum optimization is not as important

Full Adder - Observations

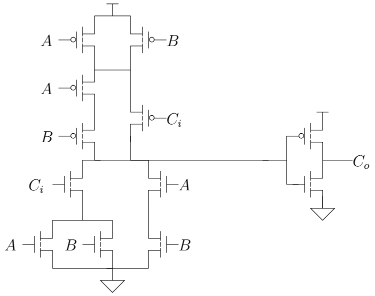


Figure: Full Adder Carry Circuit

- C_i is connected to smaller PMOS stack - Lower logical effort
- C_i is connected to the transistor closest to the output.

Full Adder - Optimization

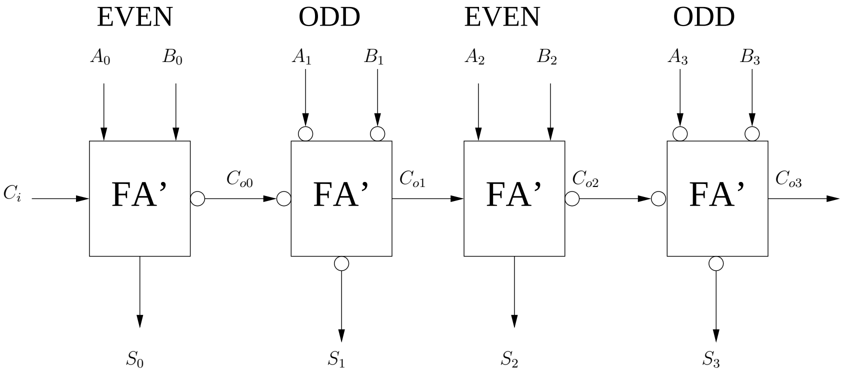


Figure: FA' is a Full Adder w/o inverter in the carry path

Full Adder - Optimization

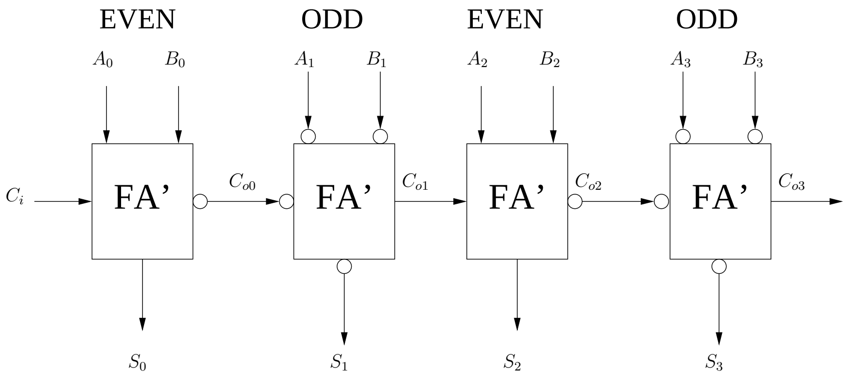
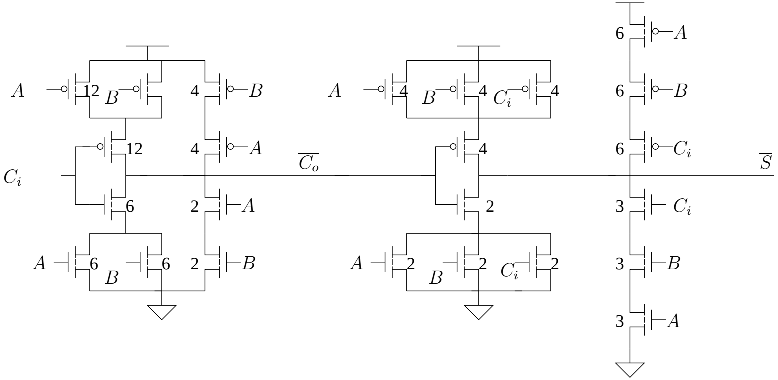


Figure: FA' is a Full Adder w/o inverter in the carry path

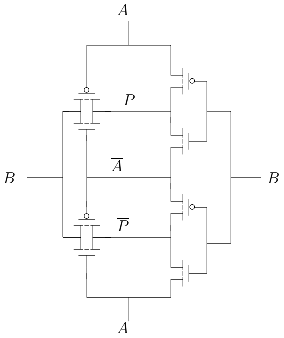
Mirror Adder

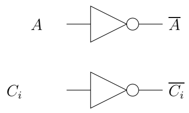


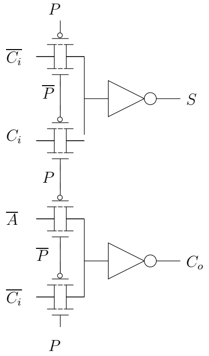
Mirror Adder - Features

- Requires 24 transistors as opposed to 28 transistors
- PUN and PDN as *identical* - Self duality
- C_i is connected to the transistors closest to the output
- Transistors in the carry stage need to be optimized (Upsized) for speed
- Invereter not available to size for delay optimization
- Upsize carry stage to 3-4X

Transmission Gate Based Adder



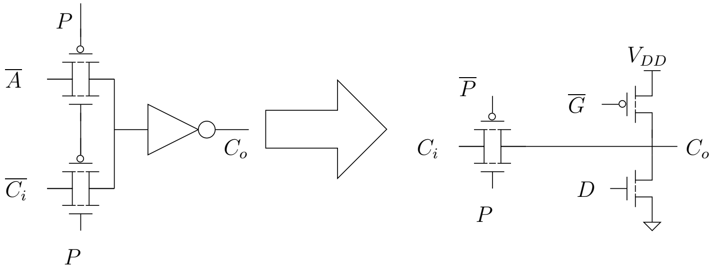




$$S = P \oplus C_i$$

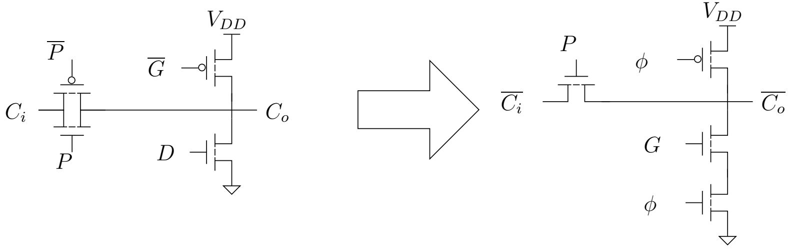
$$C_o = G + PC_i$$

Manchester Carry Gate - Static Implementation



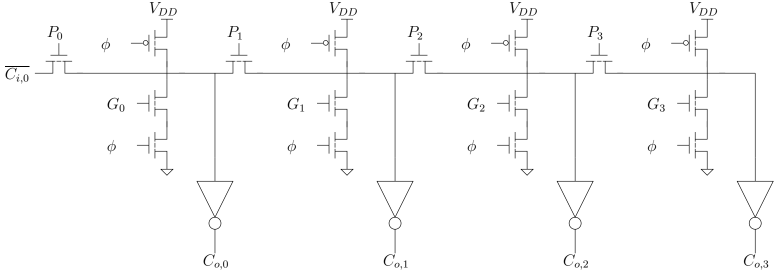
$$S = P \oplus C_i$$
$$C_o = G + PC_i$$
$$P = A \oplus B$$
$$G = AB$$
$$D = \overline{A}.\overline{B}$$

Manchester Carry Gate - Dynamic Implementation

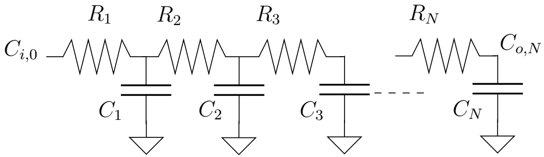


$$S = P \oplus C_i$$
$$C_o = G + PC_i$$
$$P = A \oplus B$$
$$G = AB$$
$$D = \overline{A}.\overline{B}$$

Manchester Carry Chain Adder



Manchester Carry Chain - Delay



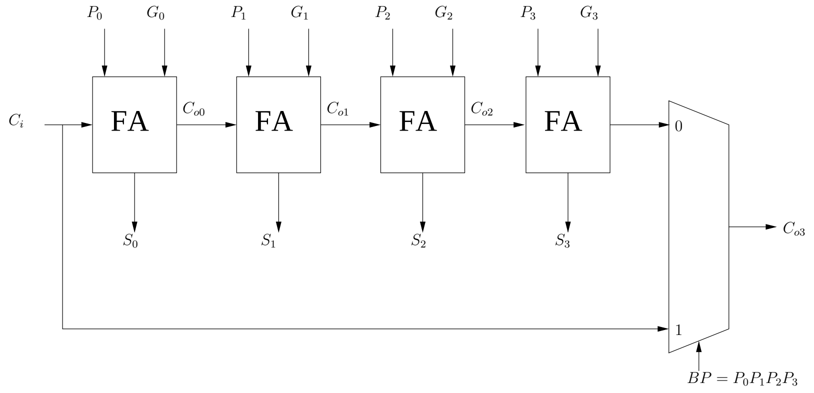
$$\tau = \frac{N(N+1)}{2} RC$$

- Worst case delay - Carry propagates from input to output
- Delays scales as N^2 - Need to buffer between wires

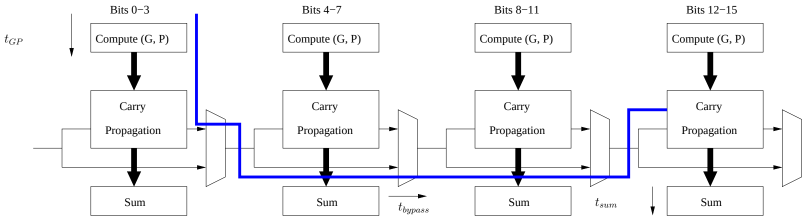
Adder Optimization

- Ripple adder delay $O(N)$
- Practical only for few bits
- Desktops use 32 bits
- Servers use 64 Bits
- GPUs require 128 Bits
- Adders are citial to their performance
- Goal : Propagation delay should be $< O(N)$

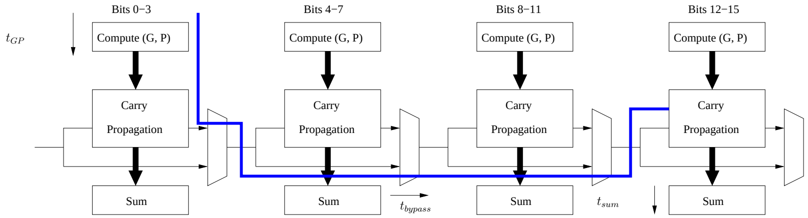
Carry Skip Adder



Carry Skip Chain



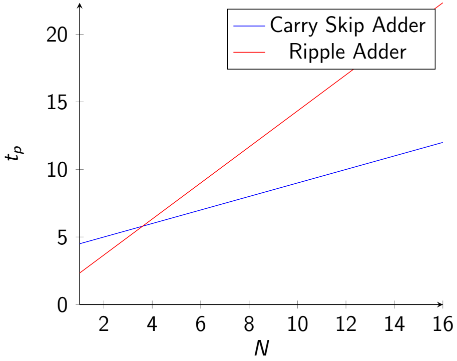
Carry Skip Chain



$$t_{carry-skip} = t_{GP} + Mt_{carry} + (\frac{N}{M} - 1)t_{bypass} + (M-1)t_{carry} + t_{sum}$$

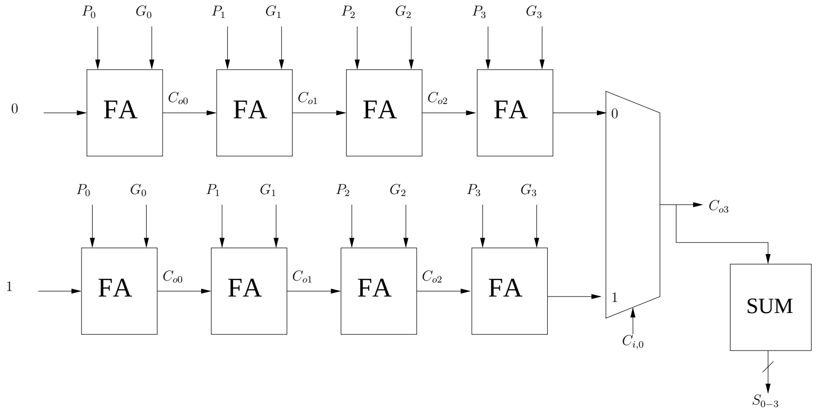
- Still proportional to N, however linear with N/M
- Still better than ripple adder.
- Carry skip starts showing lesser delay for $N > 4 - 8$

Carry Skip vs Ripple Adder

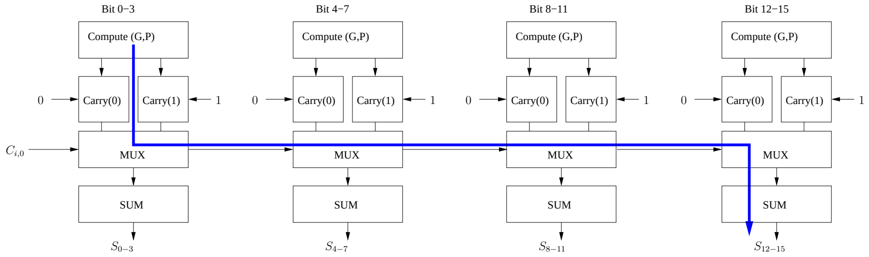


- The Carry-Skip adder is beneficial for $N > 4 \dots 8$
- The bypass overhead is high for small N

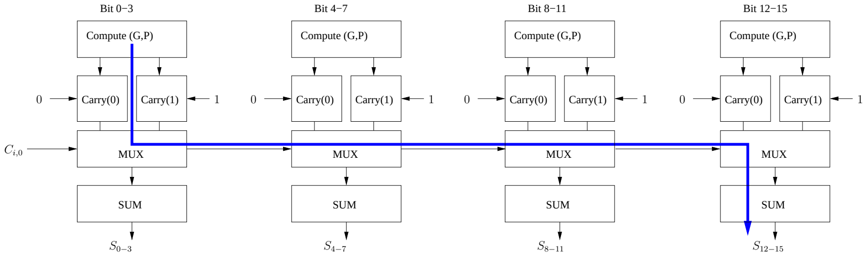
Linear Carry Select Adder



Linear Carry Select Chain

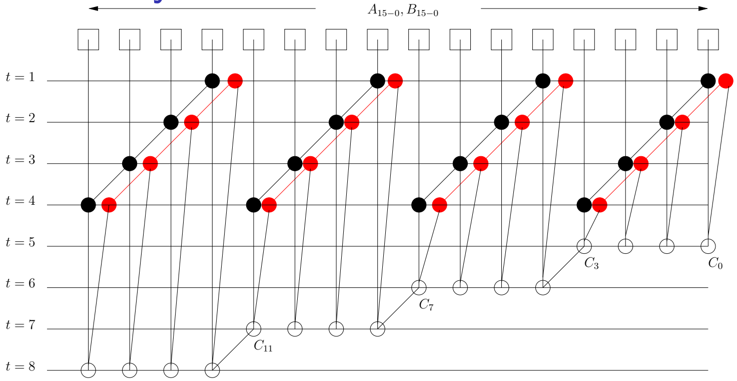


Linear Carry Select Chain



$$t_{carry-select} = t_{GP} + Mt_{carry} + \frac{N}{M}t_{mux} + t_{sum}$$

Linear Carry Select Chain

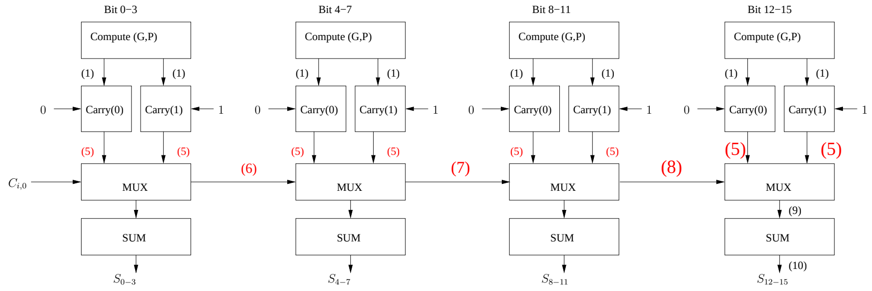


CARRY-GEN-0
CARRY-GEN-1
GEN PROP
MUX

$$t_{carry-select} = t_{GP} + Mt_{carry} + \frac{N}{M}t_{mux} + t_{sum}$$

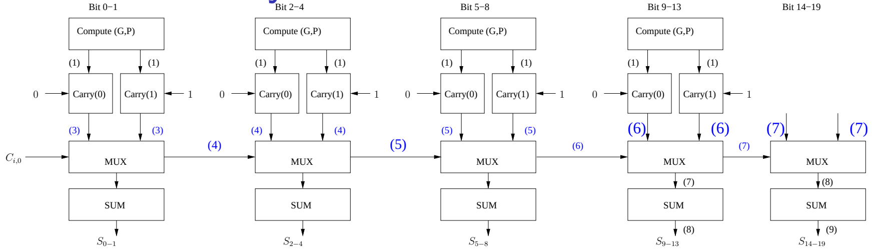
Reference: Prof. Vinita's Lecture notes [3]

Linear Carry Select Chain



Increasing mismatch in arrival times at the mux

Square Root Carry Select Chain

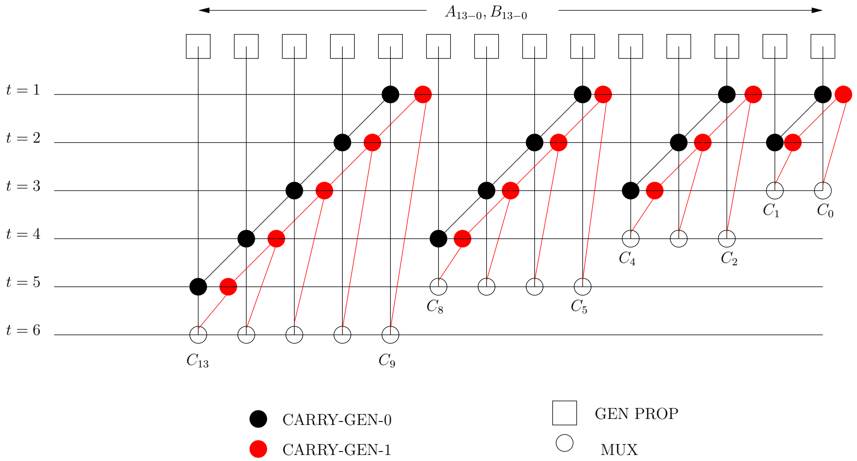


- Equalize the arrival times at the multiplexer
- Let the first stage add M bits
- Subsequent stages progressively add one bit more
- There are P such stages

$$N = M + (M+1) + (M+2) + (M+3) + \ldots + (M+P-1)$$

$$N = MP + \frac{P(P-1)}{2} = \frac{P^2}{2} + P(M - \frac{1}{2})$$

Square Root Carry Select Chain



Reference: Prof. Vinita's Lecture notes [3]

Square Root Carry Select Chain

Propagatation delay, for large N and small M

$$t_{add} = t_{GP} + Mt_{carry} + Pt_{mux} + t_{sum}$$
$$N \approx \frac{P^2}{2}$$
$$t_{add} = t_{GP} + Mt_{carry} + (\sqrt{2N})t_{mux} + t_{sum}$$
$$t_{add} = O(\sqrt{N})$$

Still limited by carry ripple

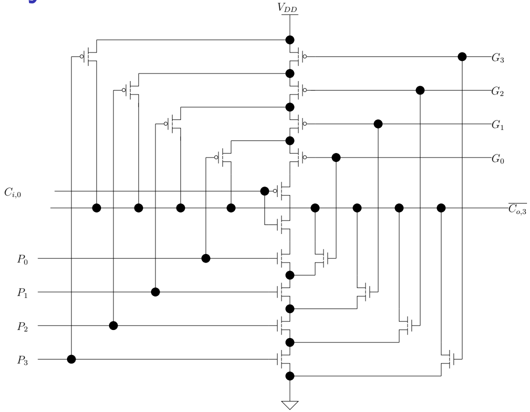
Carry Look Ahead Adder

Can we do away with the carry ripple?

$$C_{o,k} = f(A_k, B_k, C_{o,k-1})$$
$$C_{o,k} = G_k + P_k C_{o,k-1}$$
$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1} C_{o,k-2})$$
$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(G_{k-2} + P_{k-2}(\dots P_1(P_0 + G_0 C_{i,0}))))$$

- Each bit now is independent of the carry ripple
- Extra hardware required
- Delay is indepenent of number of bits?

Carry Look Ahead Adder



- Not scalable for large N
- G_0, P_0 drives all carry out cicuits. Needs strong driver
- N bits has $N+1$ parallel branches - Large area required
- N bit implementaion has up to $N+1$ transistor stacks - Slow performance

Block Propagate and Generate

Define a sub-block from bits $i - j$ and define the block propagate and generate as follows

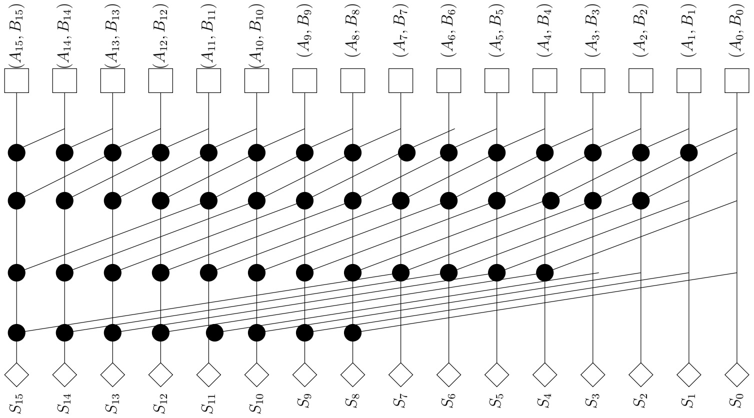
- $G_{i:j}$ - Block Generates a carry out independent of carry in - $G_{3:2} = G_3 + P_3 G_2$
- $P_{i:j}$ - Block Propagates a carry - $P_{3:2} = P_3 P_2$

Define a pair $(G_{i:j}, P_{i:j})$ and a new DOT operator as

$$(G, P).(G', P') = (G + PG', PP')$$
$$(G_{3:2}, P_{3:2}) = (G_3, P_3).(G_2, P_2)$$

DOT operator is associative but not commutative

Carry Look Ahead Log Adder



● – DOT operator

◇ – Compute Sum

□ – Compute Propagate and Generate

Multipliers - Definitions

$$X = \sum_{i=0}^{N-1} X_i 2^i$$

$$Y = \sum_{j=0}^{M-1} Y_j 2^j$$

$$Z = X \times Y$$

$$Z = \sum_{k=0}^{M+N-1} Z_k 2^k = \sum_{i=0}^{N-1} \left(\sum_{j=0}^{M-1} X_i Y_j 2^{i+j} \right)$$

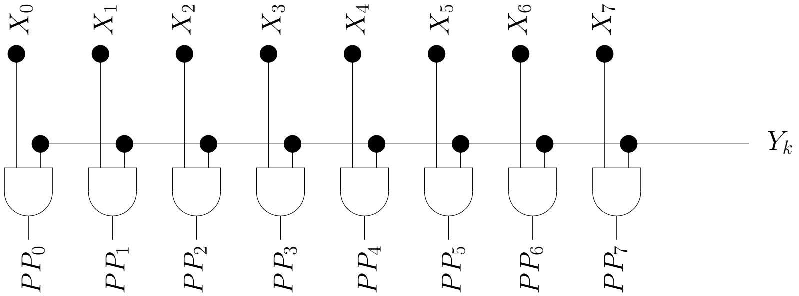
Multipliers - Definitions

			1	0	1	1	Multiplicand
			$\times$	1	1	0	Multiplier
			0	0	0	0	PP0
		1	0	1	1		PP1
	1	0	1	1			PP2
1	0	0	0	0	1	0	Result

Multipliers need to perform three main tasks:

- Partial Product (PP) Generation
- Partial Product Accumulation
- Final addition

Partial Product Generation



- AND of the multiplicand ($X - N$ bits) with each bit of the multiplier ($Y - M$ bits)
- Generates M partial products which are N bits each
- Each PP is either ZERO or the Multiplicand itself
- Large number of additions
- How do you reduce the number of PP?

Booth Encoding

- If multiplier = 0111-1110, No. of non-zero PP = 6
- Can be encoded as follows : $0111 - 1110 = 1000 - 00\bar{1}0$
- $\bar{1} = -1$
- Equivalent to treating the multiplier as a radix 4 number

$$Y = \sum_{j=0}^{(M-1)/2} Y_j 4^j \ldots (Y_j \in \{-1, 0, 1, 2\})$$

- Number of partial products reduced to at most half
- Multiplying by $\{0, 1\}$ is a simple AND operation
- Multiplying by $\{-1, 0, 1, 2\}$ requires inversion and shiting

Modified Booth Encoding

$$Z = X \times (Y_{M-1}2^{M-1} + \ldots Y_4 2^4 + Y_3 2^3 + Y_2 2^2 + Y_1 2^1 + Y_0)2^0$$
$$Z = X \times (\ldots Y_4 2^4 + Y_3(2^4 - 2^3) + Y_2 2^2 + Y_1(2^2 - 2^1) + Y_0))$$
$$Z = X \times (\ldots + [-2Y_3 + Y_2 + Y_1]2^2 + [-2Y_1 + Y_0]2^0))$$

- Partition the miltiplier into sets of 3 with 1 overlapping bit $(b_{2i+1}, b_{2i}, b_{2i-1})$
- Multiplicand is operated on based on $-2b_{2i+1} + b_{2i} + b_{2i-1}$

Reference: Prof. Vinita's Lecture notes [3]

Modified Booth Encoding

b_{2i+1}	b_{2i}	b_{2i-1}	Recoded Bits
0	0	0	0
0	0	1	$+$Multiplicand
0	1	0	$+$Multiplicand
0	1	1	$+2\times$Multiplicand
1	0	0	$-2\times$Multiplicand
1	0	1	-Multiplicand
1	1	0	-Multiplicand
1	1	1	0

Example

				1	1	0	1	-3	
				1	0	1	1	-5	
0	0	0	0	0	0	1	1	3	$Y_1 Y_0 Y_{-1} = 110[-X]$
0	0	0	0	1	1			12	$Y_3 Y_2 Y_1 = 101[(-X) << 2]$
0	0	0	0	1	1	1	1	15	

- Sign extension is critical.

Partial Product Accumulation

- Generated PP have to be summed
- Requires a large nmumber of adders
- Simplest is an array of adders - Array Multiplier
- PP Shifting to the left is achieved through routing

2s Complement Numbers

Needed to represent negative numbers. MSB used to represent the sign

$$X = \sum_{i=0}^{N-1} X_i 2^i$$

$$X = -2^{N-1} X_{N-1} + \sum_{i=0}^{N-2} X_i 2^i$$

$$X = -([2^{N-1} X_{N-1} - 1] - \sum_{i=0}^{N-2} X_i 2^i + 1)$$

- If $X_{N-1} = 1$, invert the least $N-1$ bits and add 1

2s Complement Numbers - Sign Extension

Extend the sign bit (MSB) to additonal K bits to form a $N + K$ bit number

$$X = \sum_{i=0}^{N-1} X_i 2^i$$

$$X = -2^{N-1} X_{N-1} + \sum_{i=0}^{N-2} X_i 2^i$$

$$X = -2^{N-1} X_{N-1}(2^K - 2^{K-1} - 2^{K-2} \ldots - 2 - 1) + \sum_{i=0}^{N-2} X_i 2^i$$

$$(2^K - 2^{K-1} - 2^{K-2} \ldots - 2 - 1) = 2^K - (2^K - 1) = 1$$

2s Complement Multiplication

$$Z = X \times (-2^{M-1} Y_{M-1} + \sum_{i=0}^{M-2} Y_i 2^i)$$

$$Z = X \times (-2^{M-1} Y_{M-1}) + X \times \sum_{i=0}^{M-2} Y_i 2^i$$

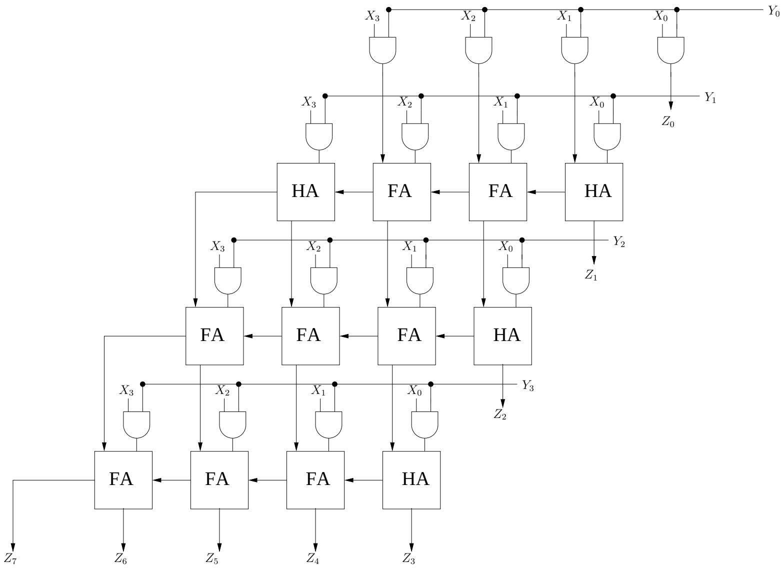
- If $b_{M-1} = 1$, the final partial product should be Twos Complement of X
- All intermedaite partial products should be signed extensions of X

Example

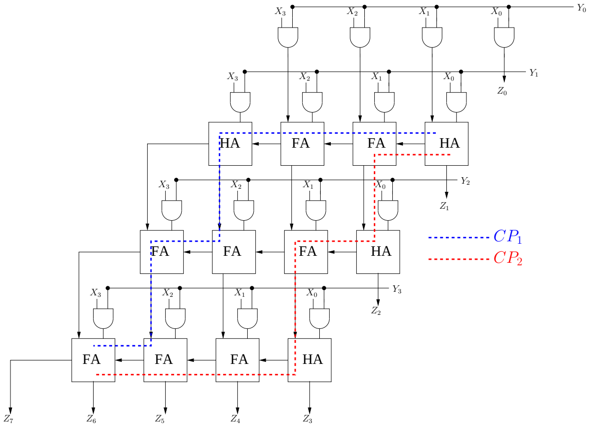
				1	1	0	1	-3	
				1	0	1	1	-5	
1	1	1	1	1	1	0	1	-3	-3
1	1	1	1	1	0	1		-6	$-3 << 1$
0	0	0	0	0	0			0	$0 << 2$
0	0	0	1	1				24	2s complement of $-3 << 3$
0	0	0	0	1	1	1	1	15	

- Sign extension is critical.

Array Multiplier



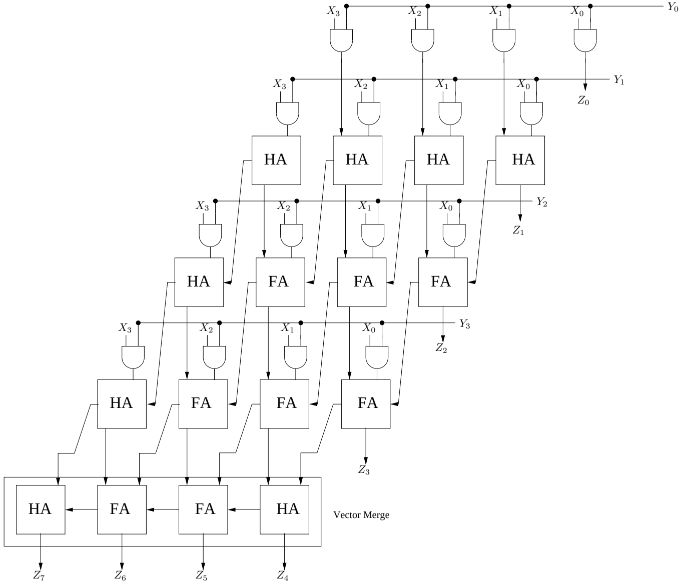
Array Multiplier Critical Path



- All path delays are almost equal
- Cannot optimize for one critical path

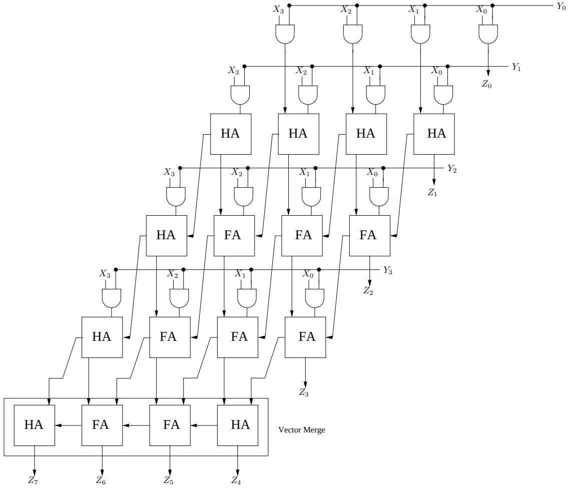
$$t_{mult} = [(N-1) + (M-2)]t_{carry} + (M-1)t_{sum} + t_{and}$$

Carry Save Multiplier Critical Path



X_3 X_2 X_1 X_0 Y_0
X_3 X_2 X_1 X_0 Y_1
Z_0
HA HA HA HA
X_3 X_2 X_1 X_0 Y_2
Z_1
HA FA FA FA
X_3 X_2 X_1 X_0 Y_3
Z_2
HA FA FA FA
Z_3
HA FA FA HA
Vector Merge
Z_7 Z_6 Z_5 Z_4

Carry Save Multiplier Critical Path



X_3 X_2 X_1 X_0 Y_0
X_3 X_2 X_1 X_0 Y_1
Z_0
HA HA HA HA
X_3 X_2 X_1 X_0 Y_2
Z_1
HA FA FA FA
X_3 X_2 X_1 X_0 Y_3
Z_2
HA FA FA FA
Z_3
HA FA FA HA
Vector Merge
Z_7 Z_6 Z_5 Z_4

$$t_{mult-cs} = t_{and} + (M-1)t_{carry} + t_{merge}$$

Optimized 2s Complement Multiplication

$$Z = (-2^{N-1}X_{N-1} + \sum_{i=0}^{N-2} X_i 2^i) \times (-2^{M-1}Y_{M-1} + \sum_{j=0}^{M-2} Y_j 2^j)$$

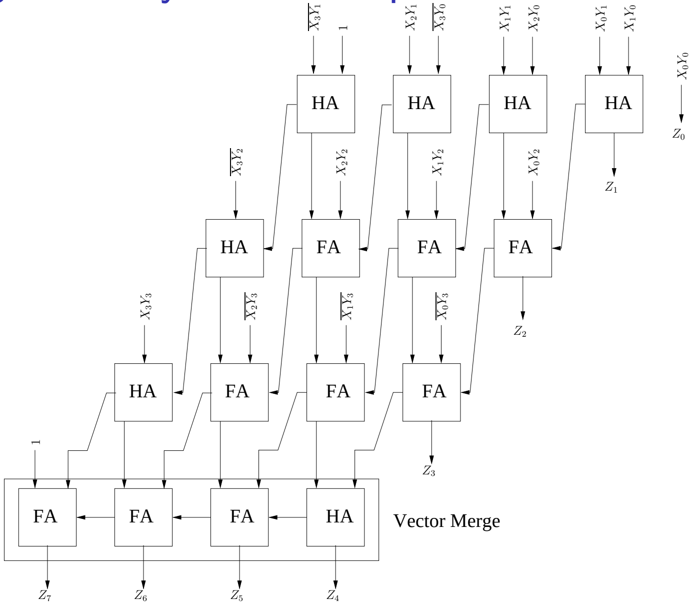
$$Z = 2^{M+N-2}X_{N-1}Y_{M-1} - 2^{N-1}X_{N-1}\sum_{j=0}^{M-2} Y_j 2^j) - 2^{M-1}Y_{M-1}\sum_{i=0}^{N-2} X_i 2^i + \sum_{i=0}^{N-2}\sum_{j=0}^{M-2} X_i Y_j 2^{i+j}$$

				X_3	X_2	X_1	X_0	$X = -2^{N-1}X_{N-1} + \sum_{i=0}^{N-2} X_i 2^i$
				Y_3	Y_2	Y_1	Y_0	$Y = -2^{M-1}Y_{M-1} + \sum_{j=0}^{M-2} Y_j 2^j$
					Y_0X_2	Y_0X_1	Y_0X_0	$\sum_{i=0}^{N-1} Y_0 X_i 2^i$
				Y_1X_2	Y_1X_1	Y_1X_0		$\sum_{i=0}^{N-1} Y_1 X_i 2^{i+1}$
			Y_2X_2	Y_2X_1	Y_2X_0			$\sum_{i=0}^{N-1} Y_2 X_i 2^{i+2}$
	Y_3X_3							$X_{N-1}Y_{M-1}2^{N+M-2}$
1	1	$\overline{Y_3X_2}$	$\overline{Y_3X_1}$	$\overline{Y_3X_0}$	1	1	1	$\sum_{i=0}^{N-2} Y_{M-1}X_i 2^{i+M-1}$
							1	
1	1	$\overline{X_3Y_2}$	$\overline{X_3Y_1}$	$\overline{X_3Y_0}$	1	1	1	$\sum_{j=0}^{M-2} X_{N-1}Y_j 2^{j+N-1}$
							1	
Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0	

Optimized 2s Complement Multiplication

				X_3	X_2	X_1	X_0	
				Y_3	Y_2	Y_1	Y_0	
			1	$\overline{X_3Y_0}$	Y_0X_2	Y_0X_1	Y_0X_0	
			$\overline{X_3Y_1}$	Y_1X_2	Y_1X_1	Y_1X_0		
		$\overline{X_3Y_2}$	Y_2X_2	Y_2X_1	Y_2X_0			
1	Y_3X_3	$\overline{Y_3X_2}$	$\overline{Y_3X_1}$	$\overline{Y_3X_0}$				
Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0	

Signed Carry Save Multiplier



Final Additon

- Vector merginig (t_{merge}) is in the critical path
- Use a suitable fast adder to reduce the delay
  - Carry save adder
  - Carry select adder
  - Carry Look Ahead Adder
- Make sure it fits well with the array multiplier layout

References

The material presented here is based on the following books/ lecture notes

1. Digital Integrated Circuits Jan M. Rabaey, Anantha Chandrakasan and Borivoje Nikolic 2nd Edition, Prentice Hall India
2. CMOS VLSI Design, Neil H.E. Weste, David Harris and Ayan Banerjee, 3rd Edition, Pearson Education
3. Prof. Vinita Vasudevan's lecture notes on adders and multipliers [http://www.ee.iitm.ac.in/vinita/digic.html]