Chapter 03: Computer Arithmetic

Lesson 05:

Arithmetic Multiplication Circuits

Objective

- Learn Booth encoding
- Learn fast multiplication by bit pairing

Multiplication Process By Booth's Encoding Algorithm

Multiplication

- Multiplication of two's-complement numbers more complicated
- Because performing a straightforward unsigned multiplication of the two's-complement representations of the inputs does not give the correct result

Multiplication

- Multipliers could be designed to convert both of their inputs to positive quantities and use the sign bits of the original inputs to determine the sign of the result
- Increases the time required to perform a multiplication

Booth's Algorithm

- A technique called *Booth encoding*
- To quickly convert two's-complement numbers into a format that is easily multiplied

Booth encoding

- Apply encoding to the multiplier bits before the bits are used for getting partial products

1. If i^{th} bit b_i is 0 and $(i-1)^{th}$ bit b_{i-1} is 1, then take b_i as +1
2. If i^{th} bit b_i is 1 and $(i-1)^{th}$ bit b_{i-1} is 0, then take b_i as –1

Booth encoding

3. If i^{th} bit b_i is 0 and $(i-1)^{th}$ bit b_{i-1} is 0, then take b_i as 0
4. If i^{th} bit b_i is 1 and $(i-1)^{th}$ bit b_{i-1} is 1, then take b_i as 0

- When lsb $b_0 = 1$, assume that it had b_{-1} as 0, thus take $b_0 = -1$

Example

Multiplier		After Booth's conversion
0 1 1 1 0 0 0 0	$\longrightarrow$	+1 0 0 –1 0 0 0 0
0 1 1 1 0 1 1 0	$\longrightarrow$	+1 0 0 –1 +1 0 –1 0
0 0 0 0 0 1 1 1	$\longrightarrow$	0 0 0 0 +1 0 0 –1
0 1 0 1 0 1 0 1	$\longrightarrow$	+1–1+1–1+1–1+1–1

Multiplication by Booth's Encoding

- Booth's algorithm permits skipping over 1s and when there are blocks of 1s
- It improves performance significantly

Multiplication using Booth's algorithm

$11101100_{\underline{b}}$	Two's complement	0000000000010100
$\times 00000101_{\underline{b}}$	Two's complement	$\times$1111111111111 0 11
	$\longrightarrow$	00000000 0000 – 1+1 0 – 1
$\times$ – 1		11111111 11101100
		00000000 0000000
$\times$ + 1		00000000 00010100
$\times$ – 1		11111111 11101100
		00000000 0000000
= – 100	$\longrightarrow$	11111111 10011100

Present Case

- Observe— the addition of 00000000 00010100 or its two's complement is done only thrice, in contrast to the addition of 00000000 00010100 done 15 times in earlier described procedures without using Booth's algorithm
- The adder circuit takes longer period to implement than finding –1 and +1 and 0's for multiplier

Worst Case

- The worst case of an implementation using Booth's algorithm is when pairs of 01s or 10s occur very frequently in the multiplier

Fast Multiplication Process

Fast Multiplication

- Fast multiplication by a combination of methods

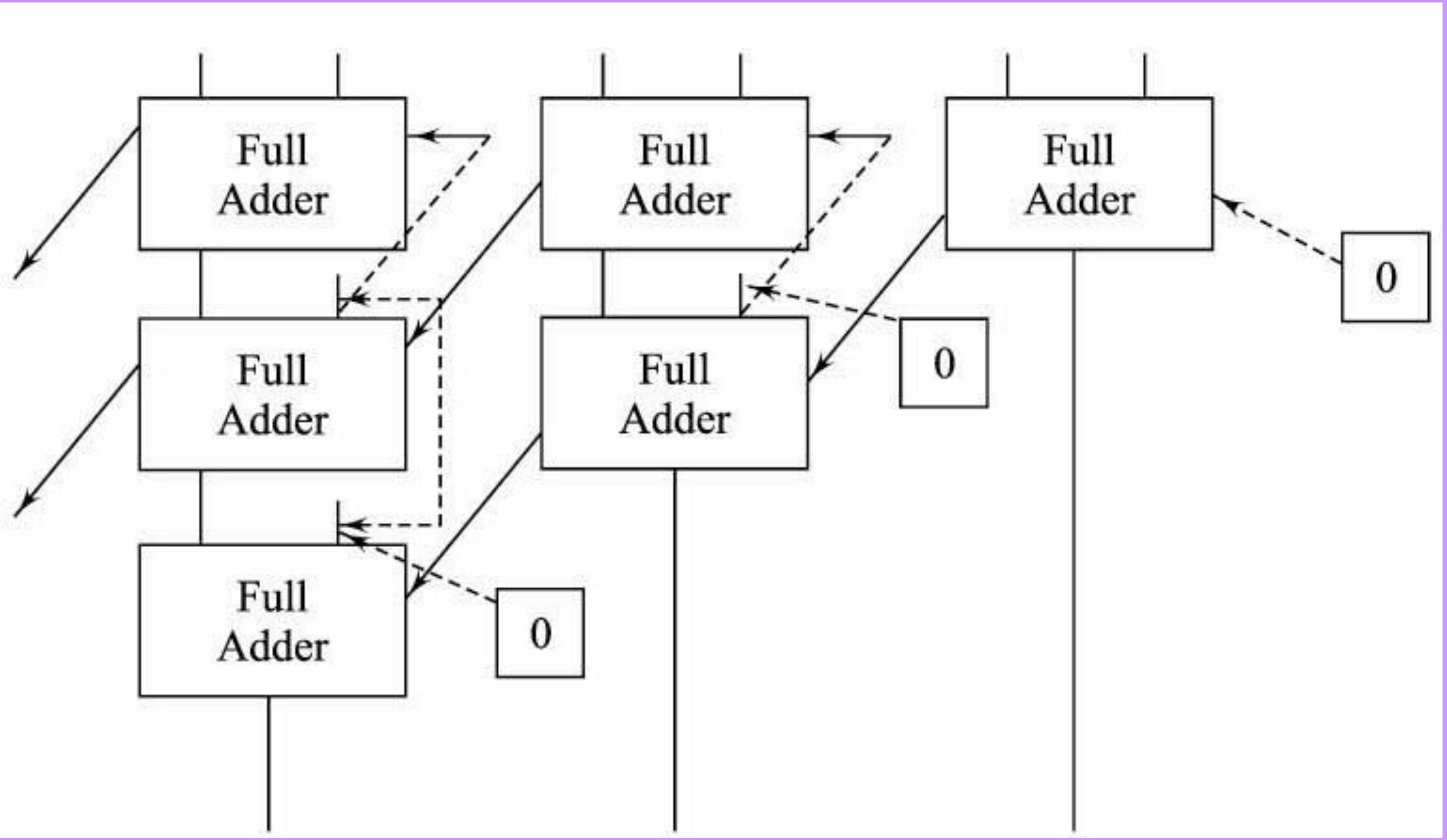
1. Bit Pair Recording of Multipliers and
2. Carry Save Addition of the Sums

Carry Save Addition in the Sums of partial products

Two-dimensional arrays of full adders to get partial products

- The carry of each FA connects the neighboring left side cell in each row
- Each FA in a cell gives the carry out as input to the next row left column FA
- The carry addition method, which reduces the time taken for additions

Carry Save method for faster multiplication



Two-dimensional arrays of full adders to get partial products

- Downward diagonal full arrows as an example
- An FA, instead of getting the ripple carry input from the previous input column of a row is given carry-input from previous column's previous row output
- Refer upward dashed arrows as an example

Two-dimensional arrays of full adders to get partial products

- For example, carry out from first row's right-most column full adder FA is given as input to the second row's right-most FA, carry out from the second row's right- most FA is given as input to the third row's right-most FA, and so on
- Each FA in a cell gives the carry out as input to next row's left column FA
- Delay through carry save adder is less than carry ripple through adder

Bit Pair Recording of Multipliers

Bit Pair Recording of Multipliers

- When Booth's algorithm is applied to the multiplier bits before the bits are used for getting partial products— Get fast multiplication by pairing

1. If pair i^{th} bit and $(i-1)^{th}$ Booth multiplier bit (B_i, B_{i-1}) is $(+1, -1)$, then take $B_{i-1} = +1$ and $B_i = 0$ and pair $(0, +1)$

Bit Pair Recording of Multipliers

2. If pair i^{th} bit and $(i-1)^{th}$ Booth multiplier bit (B_i, B_{i-1}) is $(-1, +1)$, then take $B_{i-1} = -1$ and $B_i = 0$ and make pair $(0, -1)$
3. If pair i^{th} bit and $(i-1)^{th}$ Booth multiplier bit (B_i, B_{i-1}) is $(+1, 0)$, then take $B_{i-1} = 2$ and $B_i = 0$ and make pair $(0, +2)$
4. If pair i^{th} bit and $(i-1)^{th}$ Booth multiplier bit (B_i, B_{i-1}) is $(-1, 0)$, then take $B_{i-1} = -2$ and $B_i = 0$ and make pair $(0, -2)$

Example

Multiplier

0 1 1 1 0 0 0 0

After Booth's conversion

+1 0 0 –1 0 0 0 0

After pairing

0 +2 0 –1 0 0 0 0

Example

Multiplier

0 1 1 1 0 1 1 0

After Booth's conversion

+1 0 0 –1 +1 0 –1 0

After pairing

0 + 2 0 0 –1 0 –1 0

Example

Multiplier

0 0 0 0   0 1 1 1

After Booth's conversion

0 0 0 0 +1 0 0 –1

After pairing

0 0 0 0 0 +2 0 –1

Example

Multiplier

0 1 0 1 0 1 0 1

After Booth’s conversion

+1–1+1–1+1–1+1–1

After pairing

0 +1 0 +1 0 +1 0 +1

Worst case— 0 1 0 1 0 1 0 1

- In the worst case also, the number of additions in an 8-bit multiplier has reduced to 4

Use of triplets

- b_{i+1} 1
- b_i 1 Bi = 0
- b_{i-1} 1

- b_{i+1} 1
- b_i 1 Bi = – 1
- b_{i-1} 0

Use of triplets

- b_{i+1} 1
- b_i 0 $Bi = -2$
- b_{i-1} 0

- b_{i+1} 1
- b_i 0 $Bi = -1$
- b_{i-1} 1

Use of triplets

- b_{i+1} 0
- b_i 0 Bi = 0
- b_{i-1} 0

- b_{i+1} 0
- b_i 0 Bi = + 1
- b_{i-1} 1

Use of triplets

- b_{i+1} 0
- b_i 1 Bi = +1
- b_{i-1} 0

- b_{i+1} 0
- b_i 1 Bi = + 2
- b_{i-1} 1

Summary

We learnt

- Multiplication circuit becomes fast by Booth’s algorithm
- Faster by Bit pair encoding
- Faster by triplets

End of Lesson 5 on

Arithmetic Multiplication Circuits