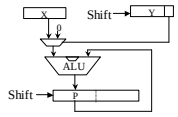
EE 486 lecture 7: Integer Multiplication

M. J. Flynn

slides prepared by Albert Liddicoat and Hossam Fahmy

Multiplication Add-and-Shift Algorithm

Multiplicand	"X"	1 1 0	6
Multiplier	"Y"	x 1 0 1	x 5
Partial Products	"PP"	1 1 0 0 0 0 1 1 0	
Result	"S"	1 1 1 1 0	30

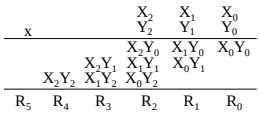


Parallel Multiplication

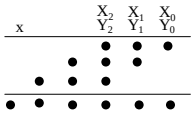
1. **Simultaneous generation of partial products**
2. **Parallel reduction of partial products**
3. **Carry Propagate Addition (CPA)**

Parallel generation of partial products

Using AND gate as 1x1-bit multiplier

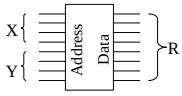


DOT representation

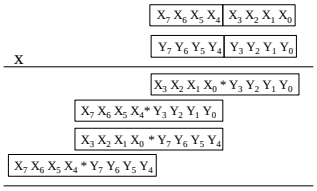


Generation of PP's Using ROMs

4b x 4b Multiply Using 256x8-bit ROM

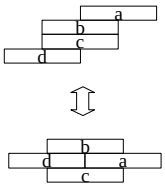


8b x 8b Multiply Using 256x8-bit ROMs

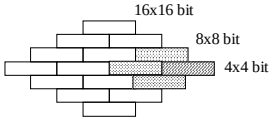


Generation of PP's Using ROMs

8 x 8 bit Multiply using 256x8b ROMs



4, 8, and 16 bit Multiply using 256x8bit ROMs



n	h	CSA Delay
4	1	0
8	3	1
16	7	4
32	15	6
64	31	8

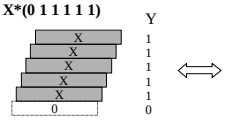
Booth's Algorithm

Observation: We can replace a string of 1's in the multiplier by +1 and -1.

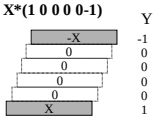
Example:

. . . 0 1 1 1 1 1 1 0 . . .
. . . 0 1 1 1 1 1 1 0 . . .
+1
- 1
. . 1 0 0 0 0 0 0 . . .
- 1
. . . 1 0 0 0 0 -1 0 . . .

X*(0 1 1 1 1 1)



X*(1 0 0 0 0-1)



Booth's Algorithm

+ Reduces the number of partial products which in turn reduces the hardware and delay required to sum the partial products.
- Adds Delay into the formation of the Partial Products.

- Works well for serial multiplication that can tolerate variable latency operations by reducing the number of serial additions required for the multiplication.
- The number of serial additions depends on the data (multiplicand).
- Worst case 8-bit multiplicand requires 8 additions
  01010101 <=> 1 -1 1 -1 1 -1 1 -1
- Parallel systems generally are designed for worst case hardware and latency requirements. Standard booth 2 does not significantly reduce the worst case number of partial products.

Modified Booth 2

- Booth 2 modified to produce at most $^{n}/_{2}+1$ partial products.

Algorithm: (for unsigned numbers)

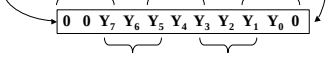
1) Pad the LSB with one zero.
2) Pad the MSB with 2 zeros if n is even and 1 zero if n is odd.
3) Divide the multiplier into overlapping groups of 3-bits.
4) Determine partial product scale factor from modified booth 2 encoding table.
5) Compute the Multiplicand Multiples
6) Sum Partial Products

Modified Booth 2

Example: (n=8-bits unsigned)

1) Pad LSB with 1 zero

2) n is even so Pad MSB with 2 zeros



3) Form 3-bit overlapping groups for n=8 we have 5 groups

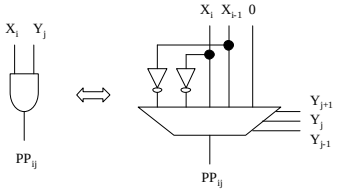
4) Determine action from table for each 3-bit group

Y_{i+1}	Y_i	Y_{i-1}	Scale Factor
0	0	0	+0
0	0	1	+X
0	1	0	+X
0	1	1	+2X
1	0	0	-2X
1	0	1	-X
1	1	0	-X
1	1	1	-0

Modified Booth 2

5) Compute Multiplicand Scale Factor (~4 gate delays)

Direct Multiplier | **Modified Booth 2 Multiplier** | **Mux Control Logic**



Scale Factor	Action
+0	Mux 0
+X	Mux X_i
+2X	Mux X_{i-1}
-X	Mux X_i'
-2X	Mux X_{i-1}'

Modified Booth 2

6) Sum Partial Products

- **Sign extend partial products to the full width of the final result**
- **Logic may replace the A_9, B_9, C_9, D_9, and E_9 sign extention bits.**
- **Y_{i+1} bit determines if the multiple needs to be complemented**

Partial Products	Multiplicand $Y_{i+1}\,Y_i\,Y_{I-1}$
$A_9 \ldots A_9\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$	$(Y_1\ Y_0\ 0)$
$B_9 \ldots B_8\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0\ \ Y_1$	$(Y_3\ Y_2\ Y_1)$
$C_9 \ldots C_6\ C_5\ C_4\ C_3\ C_2\ C_1\ C_0\ \ Y_3$	$(Y_5\ Y_4\ Y_3)$
$D_9 \ldots D_4\ D_3\ D_2\ D_1\ D_0\ \ Y_5$	$(Y_7\ Y_6\ Y_5)$
$E_9 \ldots E_2\ E_1\ E_0\ \ Y_7$	$(0\ \ 0\ Y_7)$
$S_{15} \ldots S_{10}\ S_9\ S_8\ S_7\ S_6\ S_5\ S_4\ S_3\ S_2\ S_1\ S_0$	

Modified Booth 2

Algorithm Extension: (for signed multiplier)

1) Pad the LSB with one zero.
2) ***If n is even don't pad the MSB ($^{n}/_{2}$ PP's) and if n is odd sign extend the MSB by 1 bit ($^{n+1}/_{2}$ PP's).***
3) Divide the multiplier into overlapping groups of 3-bits.
4) Determine partial product factor from table.
5) Compute the Multiplicand Multiples
6) Sum Partial Products

	n Even:			n Odd:		
	X_{n-1}	X_{n-2}	X_{n-2}	Ø	X_{n-1}	X_{n-2}
Positive:	0	x	x	0 ⇦ 0		x
Negative:	1	x	x	1 ⇦ 1		x

Modified Booth 2

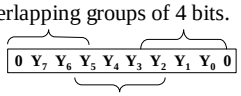
Algorithm Extension: (for signed multiplicand)

Nothing the algorithm works fine !

- The multiplicand may be represented in 2's complement code.
- The scale factors (0, +X, +2X, -X, and -2X) are handled correctly.
  - Shift left for 2 times weighting.
  - 2's complement the multiplicand for subtraction.

Booth 3

- Reduces the partial products to $\sim n/3$
- Form overlapping groups of 4 bits.

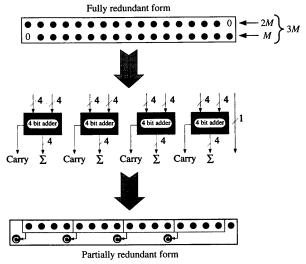


- Booth 3 Encoding

Note: 3x is a hard multiple that must be precomputed

Y_{i+2}	Y_{i+1}	Y_i	Y_{i-1}	Scale Factor	Y_{i+2}	Y_{i+1}	Y_i	Y_{i-1}	Scale Factor
0	0	0	0	+0	1	0	0	0	-4X
0	0	0	1	+X	1	0	0	1	-3X
0	0	1	0	+X	1	0	1	0	-3X
0	0	1	1	+2X	1	0	1	1	-2X
0	1	0	0	+2X	1	1	0	0	-2X
0	1	0	1	+3X	1	1	0	1	-X
0	1	1	0	+3X	1	1	1	0	-X
0	1	1	1	+4X	1	1	1	1	-0

Partially Redundant Booth 3



Partially Redundant Booth 3