BINARY MULTIPLICATION

Multipliers

- Multiplies are widely used in digital signal processing, generally more so than in general-purpose workloads
- Major categories of multiplier types
  - *Unsigned* × *Unsigned*
    Also very useful for sign-magnitude data
  - *Signed 2's complement* × *Signed 2's complement*
    Very useful for fixed-point 2's complement data
- Hardware is typically built in a manner broadly similar to how you would do it with paper and pencil
- The naming convention is somewhat unfortunate:

$$\begin{array}{r} \textit{multiplicand} \\ \times \quad \textit{multiplier} \\ \hline \end{array}$$

Multipliers

- Example: 4-bit unsigned *multiplicand "a"* times 4-bit *multiplier "b"*
- *b* could be signed or unsigned
- $p_{xy} = a_x \times b_y$
  $= a_x \text{ AND } b_y$

			a_3	a_2	a_1	a_0	
		$\times$	b_3	b_2	b_1	b_0	
			p_{30}	p_{20}	p_{10}	p_{00}	$\leftarrow b_0$
		p_{31}	p_{21}	p_{11}	p_{01}	0	$\leftarrow b_1$
	p_{32}	p_{22}	p_{12}	p_{02}	0	0	$\leftarrow b_2$
p_{33}	p_{23}	p_{13}	p_{03}	0	0	0	$\leftarrow b_3$

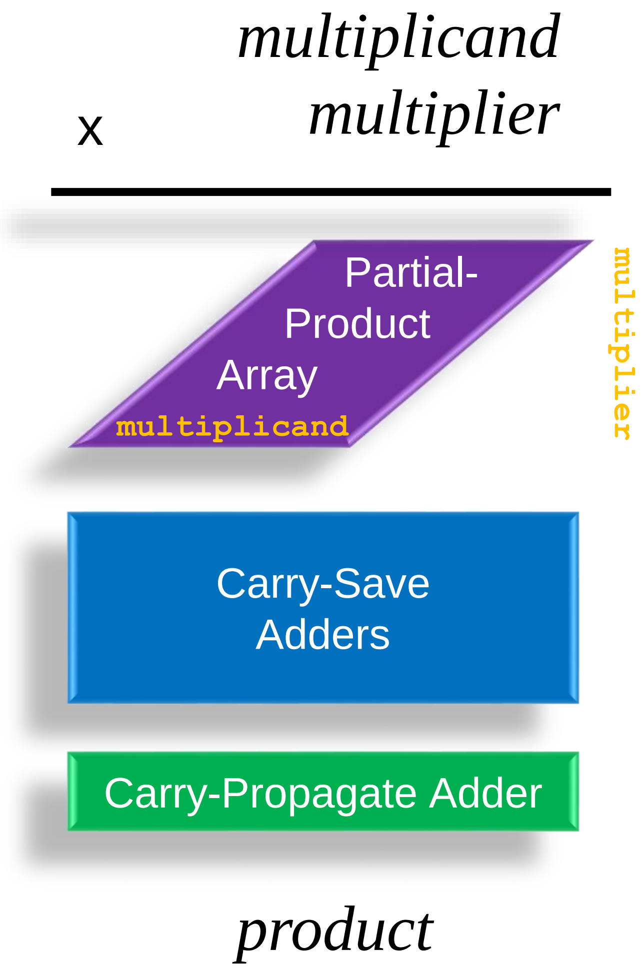
Multipliers

- Example: 4-bit signed 2's complement *multiplicand "a"* times 4-bit *multiplier "b"*
- b could be signed or unsigned
- s = partial product sign extension bits
- $p_{xy} = a_x \times b_y$
  $= a_x$ AND b_y

			a_3	a_2	a_1	a_0	
		$\times$	b_3	b_2	b_1	b_0	
s	s	s	p_{30}	p_{20}	p_{10}	p_{00}	$\leftarrow b_0$
s	s	p_{31}	p_{21}	p_{11}	p_{01}	0	$\leftarrow b_1$
s	p_{32}	p_{22}	p_{12}	p_{02}	0	0	$\leftarrow b_2$
p_{33}	p_{23}	p_{13}	p_{03}	0	0	0	$\leftarrow b_3$

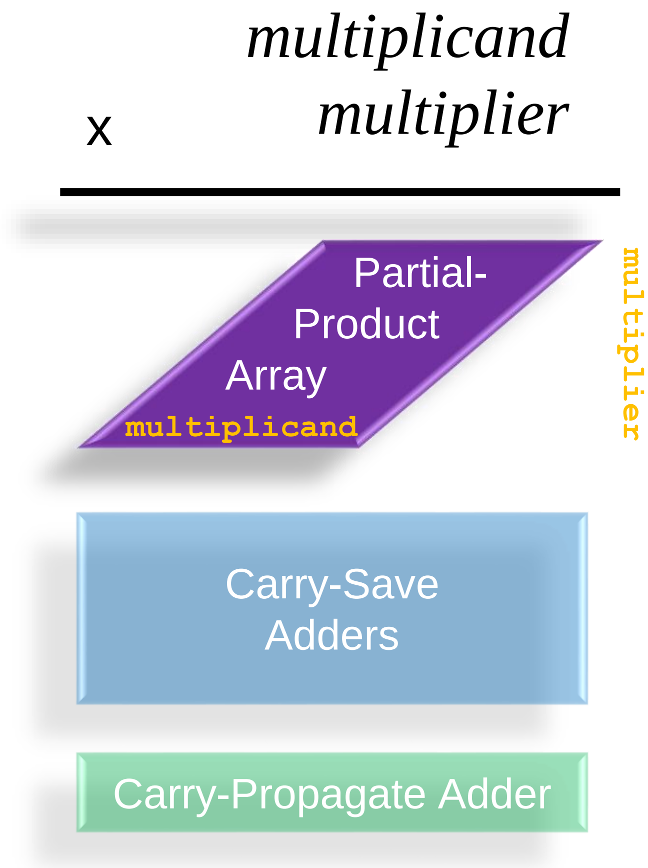
3 Main Steps in Every Multiplier

1) Generation of partial products
2) Reduction or "compression" of the partial product array (normally using carry-save addition) so that the product is composed of two words
   – Linear array addition
   – Tree addition (Wallace tree)
3) Final adder: Carry-propagate adder (CPA)
   – Converts the product in carry-save form into a single word form
   – Any style of CPA is fine though we probably favor faster ones



Straight-forward Partial Product Generation

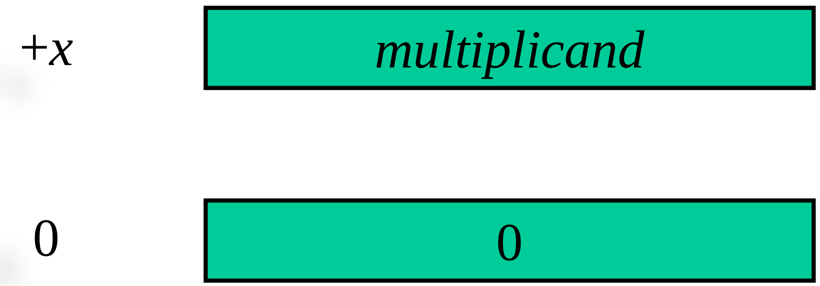
- This is the simplest method to generate partial products
- Hardware looks at one bit of the *multiplier* (Y_i) at a time
- Partial products are copies of the *multiplicand* AND'd by bits of the *multiplier*
- Number of bits in the *multiplier*
  = Number of partial products
  = Number of terms/words/rows that must be added



Y_i	Partial product
0	0
1	$+x$ (= *multiplicand*)

Straight-forward Partial Product Generation

- There are only two possible partial product results
- Two reasonable hardware solutions are:
  - a row of 2:1 muxes with zeros on one input
  - a row of AND gates (this should be more efficient)



Y_i	Partial product
0	0
1	$+x$ (= *multiplicand*)

Example unsigned 4-bit × 4-bit multiplication

- Example: 4-bit unsigned *multiplicand “a”* 1100 times 4-bit *multiplier “b”* 1010
- 1100 × 1010 = 12 × 10 = 120
- 1100 × 1010 = (12 × 8) + (12 × 0) + (12 × 2) + (12 × 0) = 120
- 1111000 = 64 + 32 + 16 + 8 = 120 ☺

			1	1	0	0	
		×	1	0	1	0	
			0	0	0	0	← 0
		1	1	0	0	0	← 1
	0	0	0	0	0	0	← 0
1	1	0	0	0	0	0	← 1
1	1	1	1	0	0	0	