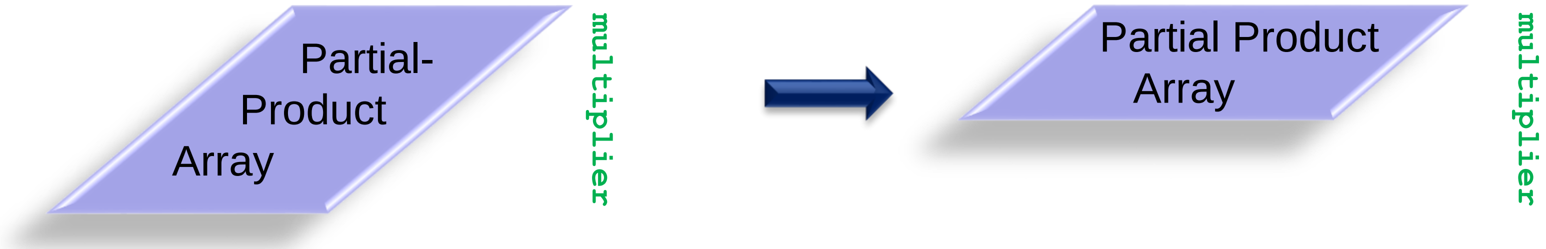
BOOTH ENCODING OF THE "MULTIPLIER" INPUT

Booth Encoding

- Method to reduce the number of partial products
- Named after Andrew Booth (1918-2009) who published the algorithm in 1951 while at Birkbeck College, London
- Booth-*n*
  - Examines *n*+1 bits of the *multiplier*
  - Encodes *n* bits
  - *n* × reduction in the number of partial products
- But partial products must then be more complex than simply 0 or +*multiplicand*



Booth Encoding: Booth-2 or "Modified Booth"

- Can view the *multiplier* as being built of strings of 1's
  - Examine multiplier bits Y_{i+1}, Y_i, and Y_{i-1}
  - Perspective of moving right to left towards the MSB
- There are $\left\lfloor \frac{N+2}{2} \right\rfloor = \left\lfloor \frac{N}{2} + 1 \right\rfloor$ partial products in the worst case

Y_{i+1}	Y_i	Y_{i-1}	Partial product	Comment
0	0	0	0	no string of 1's
0	0	1	$+x$	end of string of 1's
0	1	0	$+x$	a string of 1's
0	1	1	$+2x$	end of string of 1's
1	0	0	$-2x$	beginning of string of 1's
1	0	1	$-x$	$-2x + x$
1	1	0	$-x$	beginning of string of 1's
1	1	1	0	center of string of 1's

[Waser and Flynn]

Booth Encoding: Booth-2 or "Modified Booth"

- There are five possible partial products compared to two with non-Booth encoding

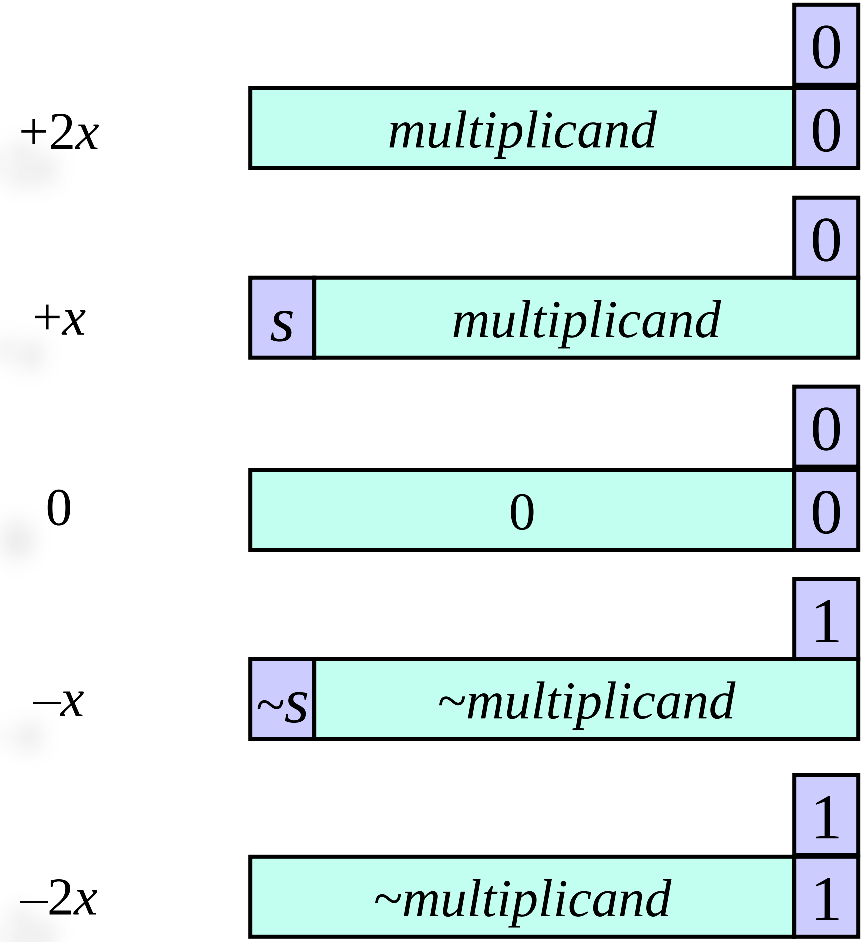
$+2x$
$+x$
0
$-x$
$-2x$

Y_{i+1}	Y_i	Y_{i-1}	Partial product	Comment
0	0	0	0	no string of 1's
0	0	1	$+x$	end of string of 1's
0	1	0	$+x$	a string of 1's
0	1	1	$+2x$	end of string of 1's
1	0	0	$-2x$	beginning of string of 1's
1	0	1	$-x$	$-2x + x$
1	1	0	$-x$	beginning of string of 1's
1	1	1	0	center of string of 1's

[Waser and Flynn]

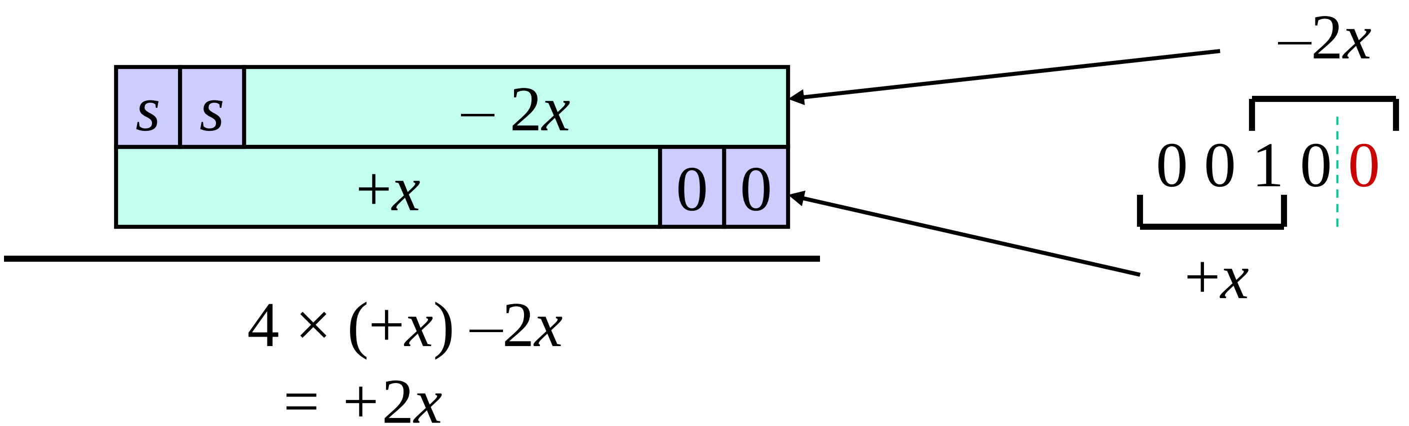
Booth Encoding: Booth-2 or "Modified Booth"

- Fortunately, these five possible partial products are very easy to generate
- Correctly generating the $-x$ and $-2x$ PPs requires a little care
  - The key issue is to not separate the 1) negation and 2) adding "1" LSB operations during the inversion process



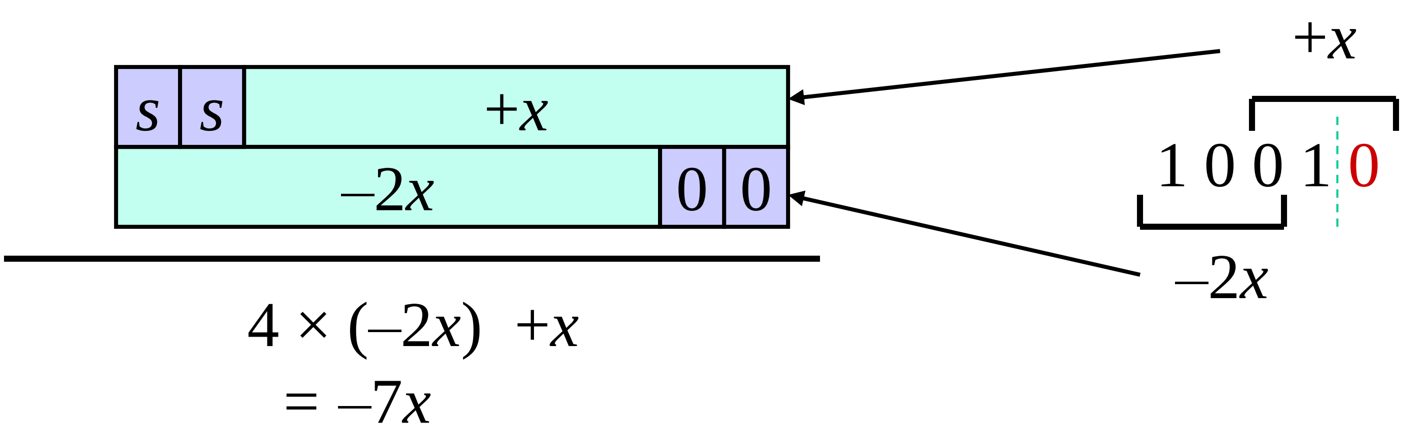
Booth Encoding: Booth-2 or "Modified Booth"

- Example: *multiplier* = 0010 = 2
  - Add 0 to the right of the LSB since the first group has no group with which to overlap
  - Examine 3 bits at a time
  - Encode 2 bits at a time
    → Overlap one bit between partial products



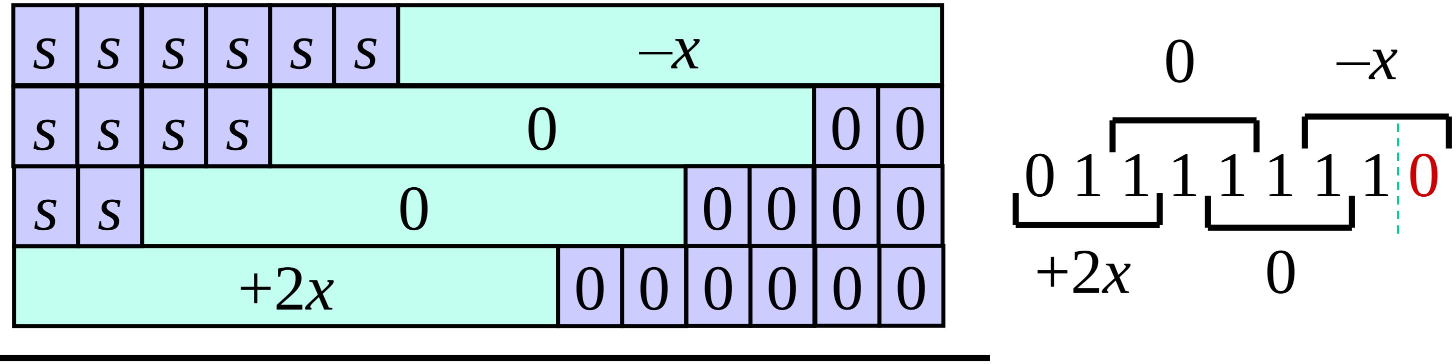
Booth Encoding: Booth-2 or "Modified Booth"

- Example: *multiplier* = 1001 = –7
  - Add 0 to the right of the LSB since the first group has no group with which to overlap
  - Examine 3 bits at a time
  - Encode 2 bits at a time
    - → Overlap one bit between partial products



Booth Encoding: Booth-2 or "Modified Booth"

- Example: *multiplier* = 01111111 = +127
  - Nice example of encoding a long string of 1's
  - Examine 3 bits at a time
  - Encode 2 bits at a time

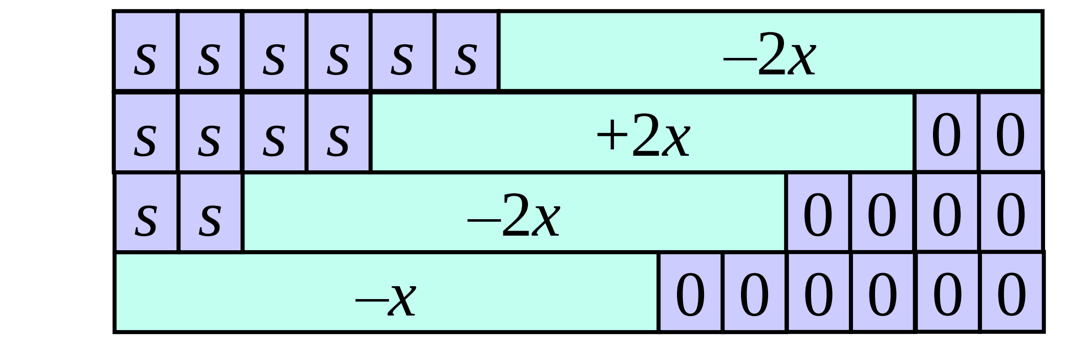


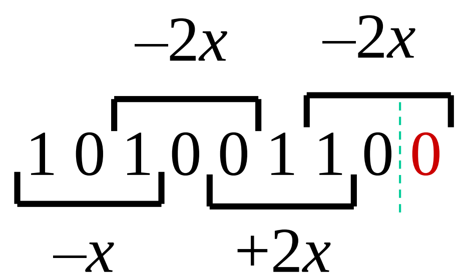
$64 \times (+2x) + 16 \times (0) + 4 \times (0) - x$

$= +127x$

Booth Encoding: Booth-2 or "Modified Booth"

- Example: *multiplier* = 10100110 = –90
  - Examine 3 bits at a time
  - Encode 2 bits at a time



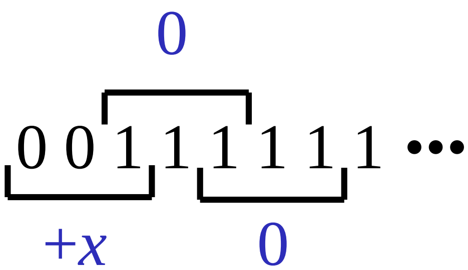


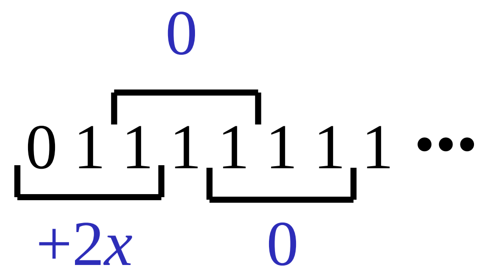
$64 \times (–x) + 16 \times (–2x) + 4 \times (+2x) – 2x$

$= – 90x$

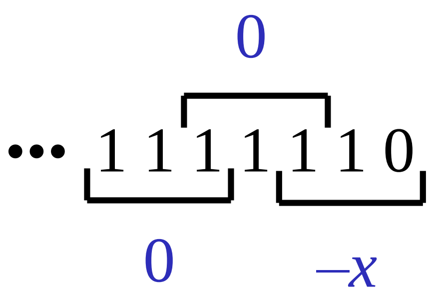
Booth Encoding: Booth-2 or "Modified Booth"

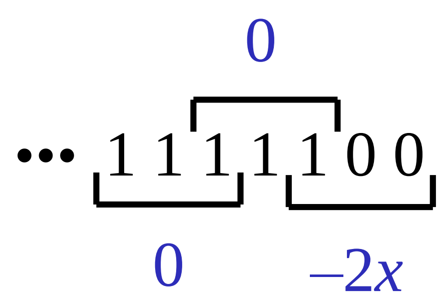
- (Left side) *End* of a string of 1's





- (Right side) *Beginning* of a string of 1's





Booth Encoding: Booth-3

Y_{i+2}	Y_{i+1}	Y_i	Y_{i-1}	Partial product
0	0	0	0	0
0	0	0	1	$+x$
0	0	1	0	$+x$
0	0	1	1	$+2x$
0	1	0	0	$+2x$
0	1	0	1	$+3x$
0	1	1	0	$+3x$
0	1	1	1	$+4x$
1	0	0	0	$-4x$
1	0	0	1	$-3x$
1	0	1	0	$-3x$
1	0	1	1	$-2x$
1	1	0	0	$-2x$
1	1	0	1	$-x$
1	1	1	0	$-x$
1	1	1	1	0

[Waser and Flynn]