Multiplication

Multiplication

- Start with long-multiplication approach

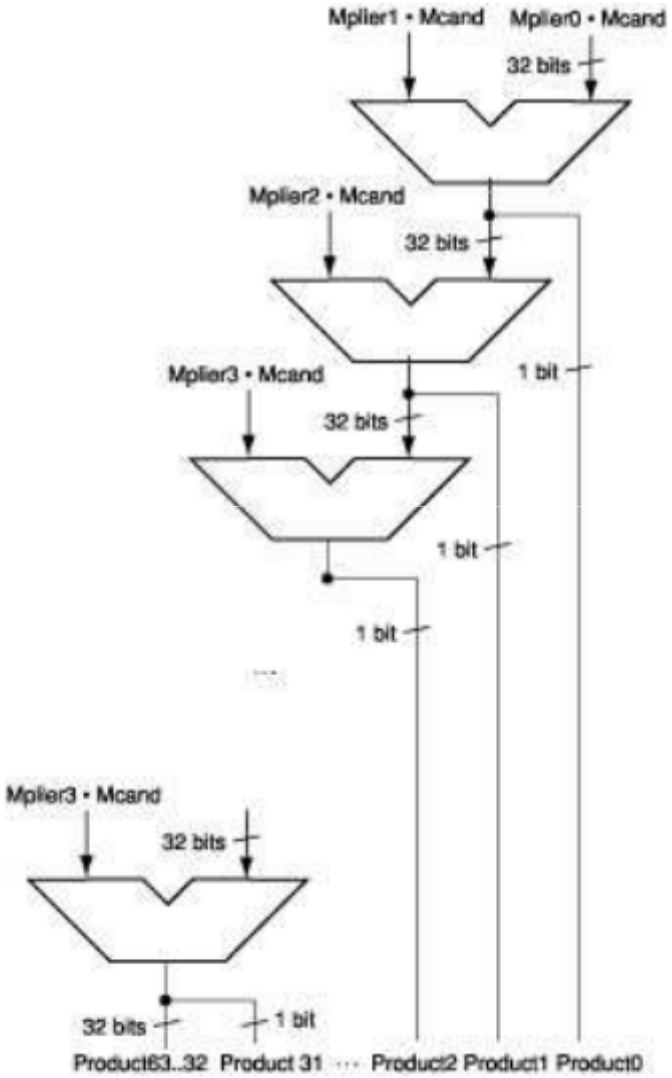
multiplicand → 1000

multiplier → × 1001

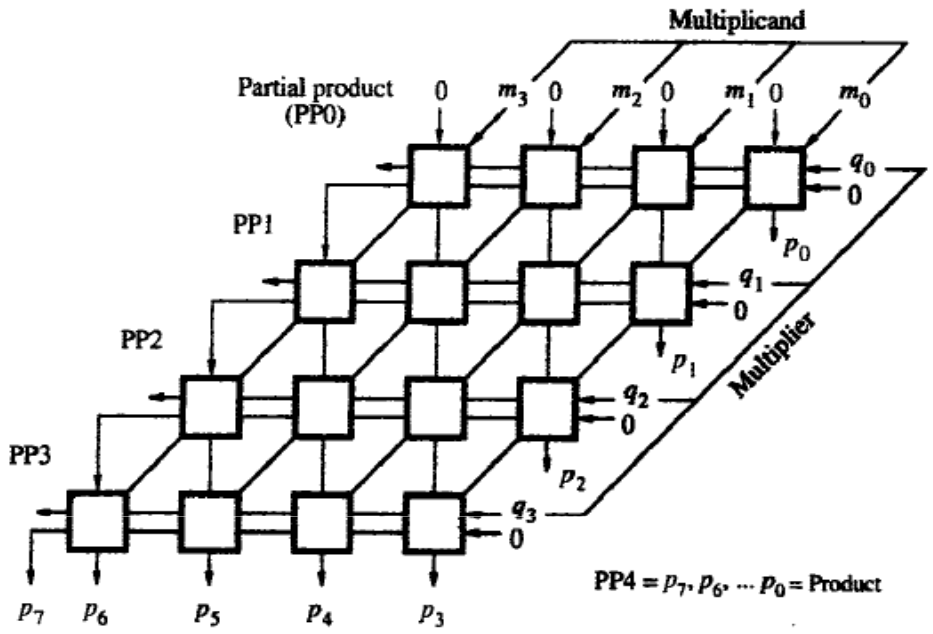
```
      1000
     0000
    0000
   1000
   -------
   1001000
```

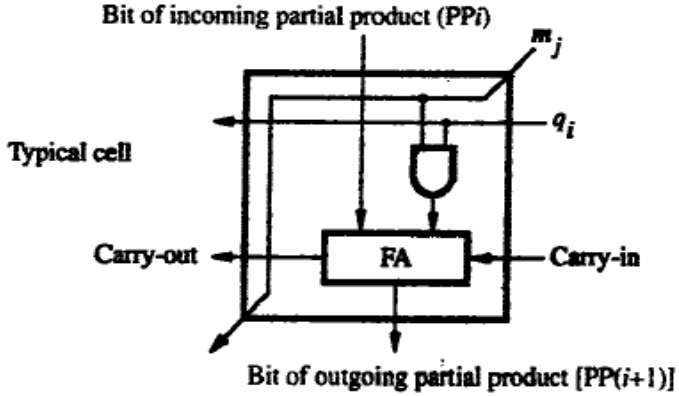
product → 1001000

Length of product is the sum of operand lengths



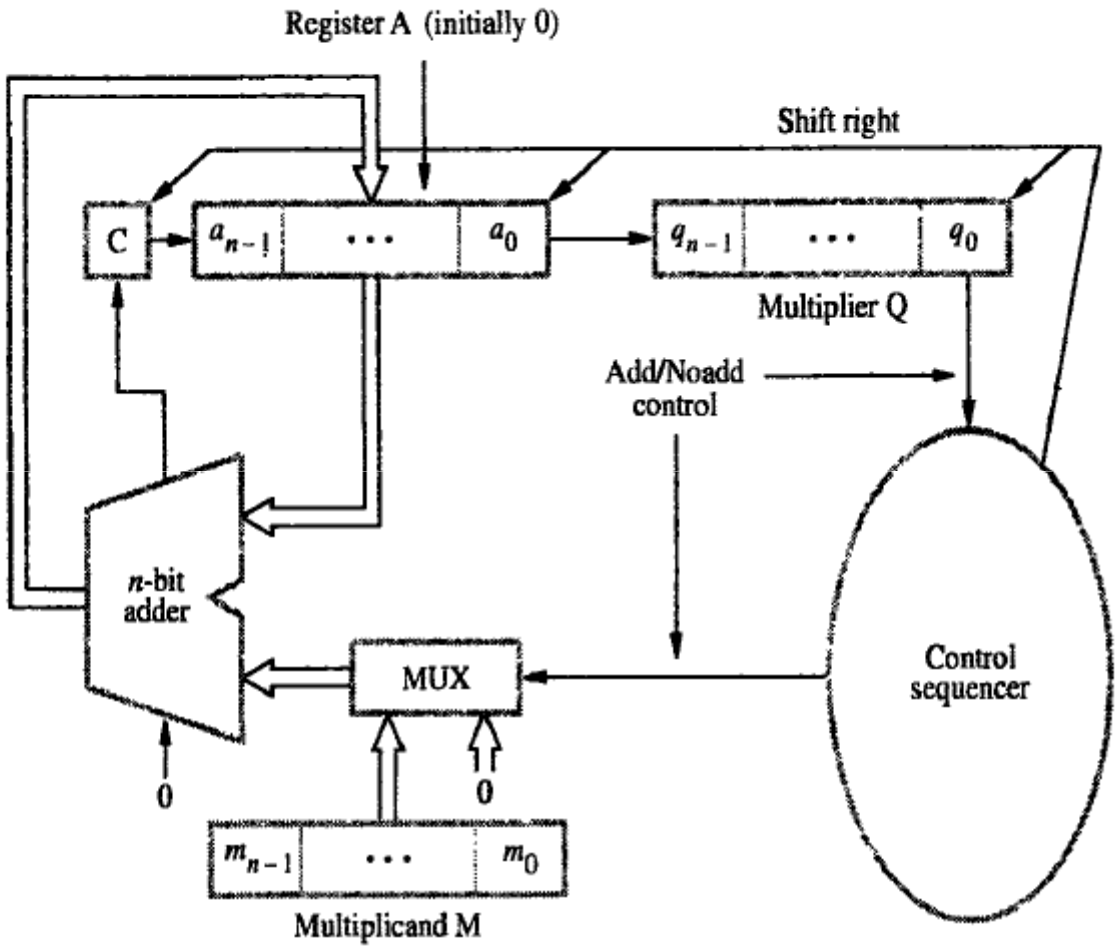
Multiplication of positive numbers





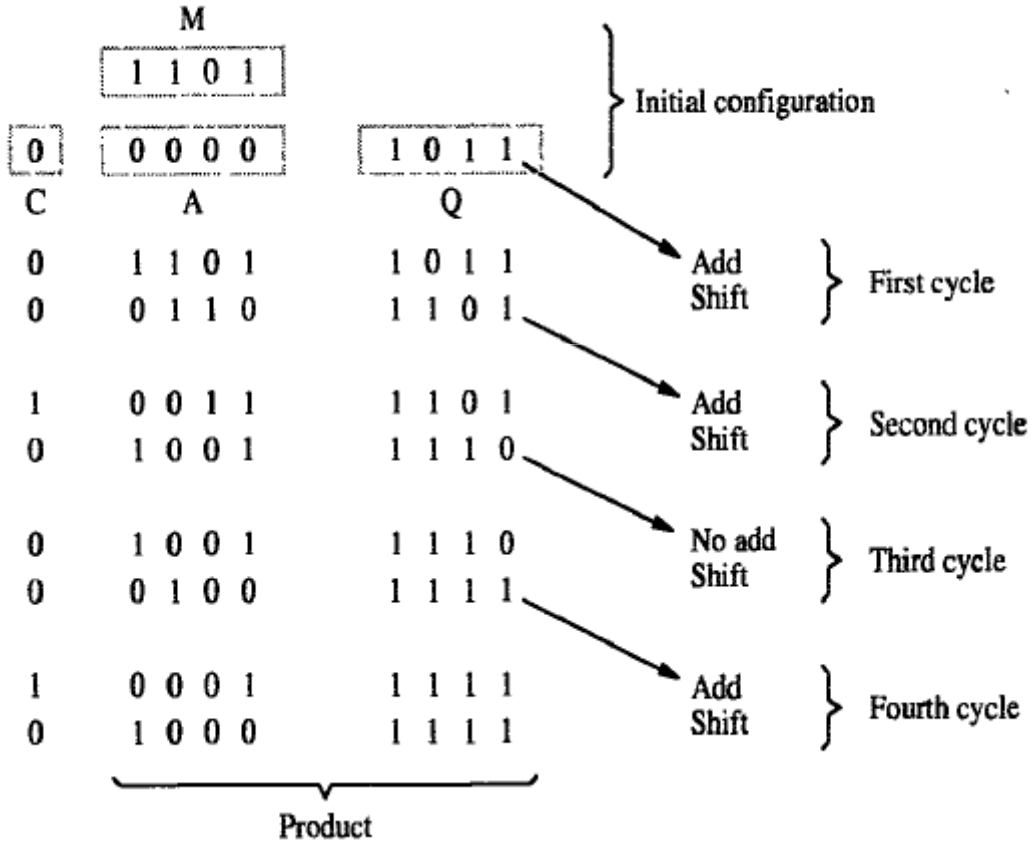
Combinational logic & Array Implementation

Multiplication of positive numbers



Sequential circuit binary multiplier-Register Configuration

Multiplication of positive numbers



Sequential circuit binary multiplier- Multiplication Example

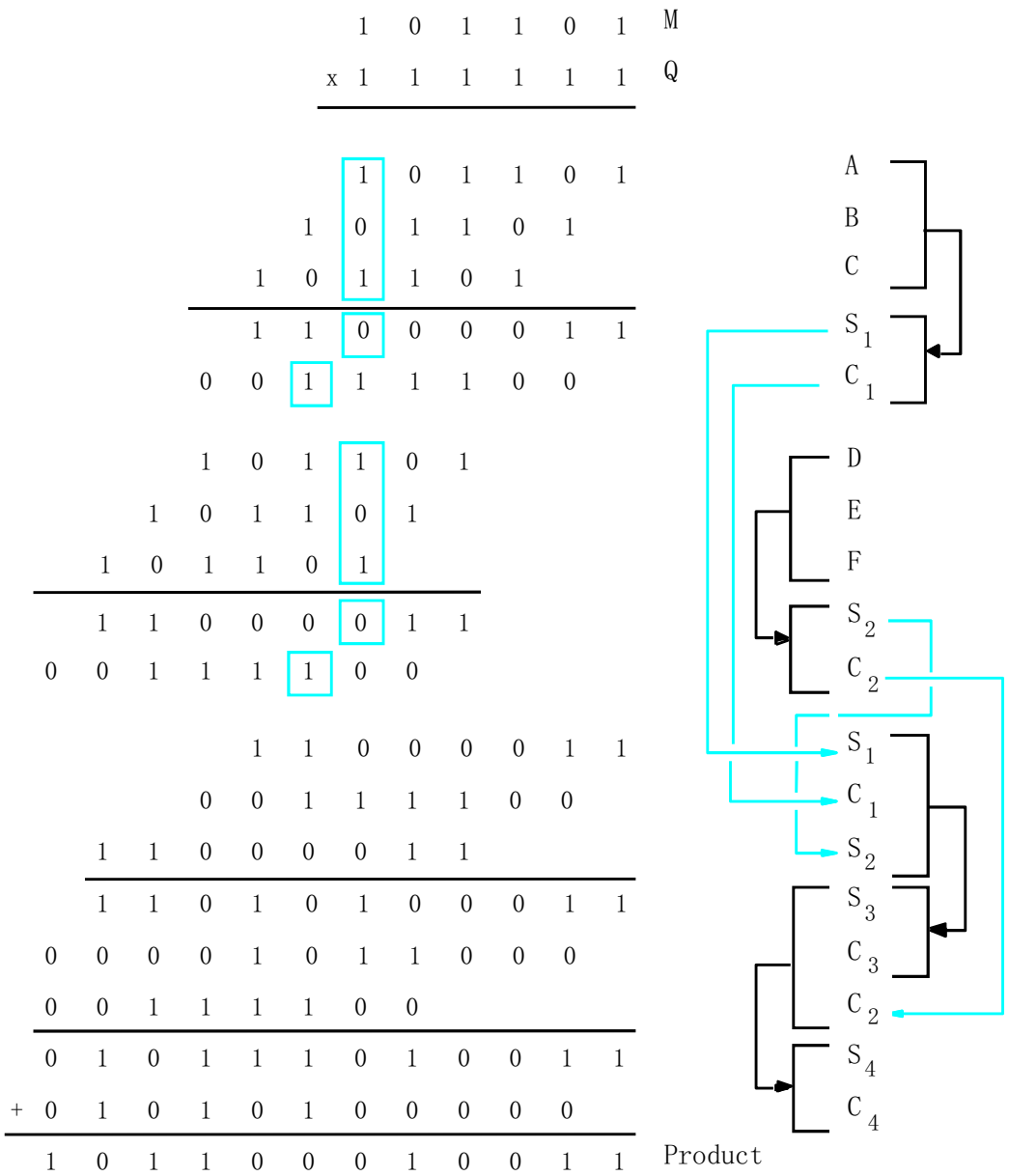


Figure 6.18. The multiplication example from Figure 6.17 performed using carry-save addition.

Carry-Save Addition of Summands

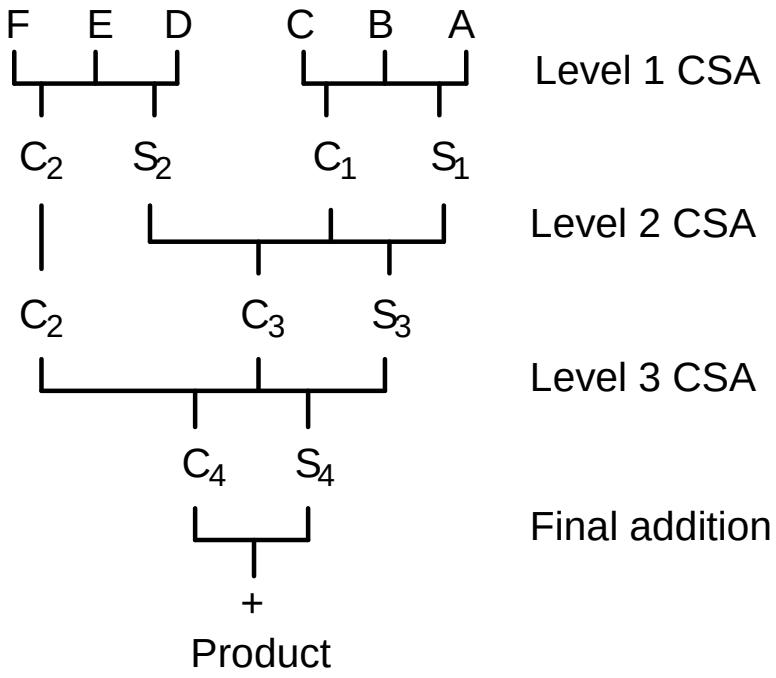


Figure 6.19. Schematic representation of the carry-save addition operations in Figure 6.18.

Multiplication

- Start with long-multiplication approach

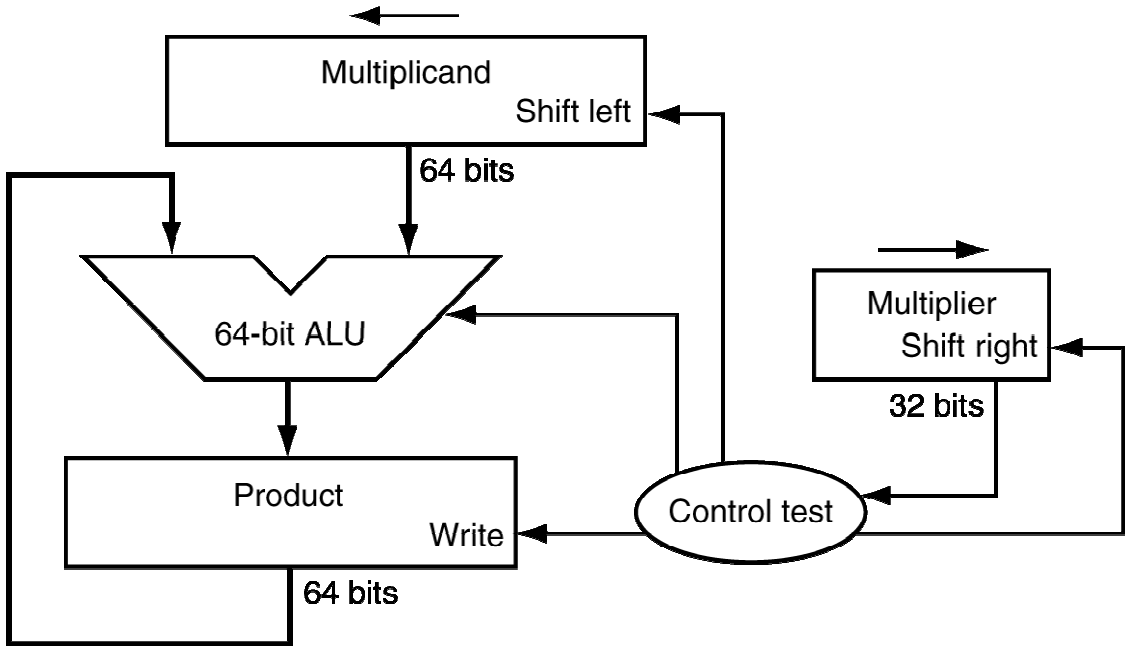
multiplicand

multiplier

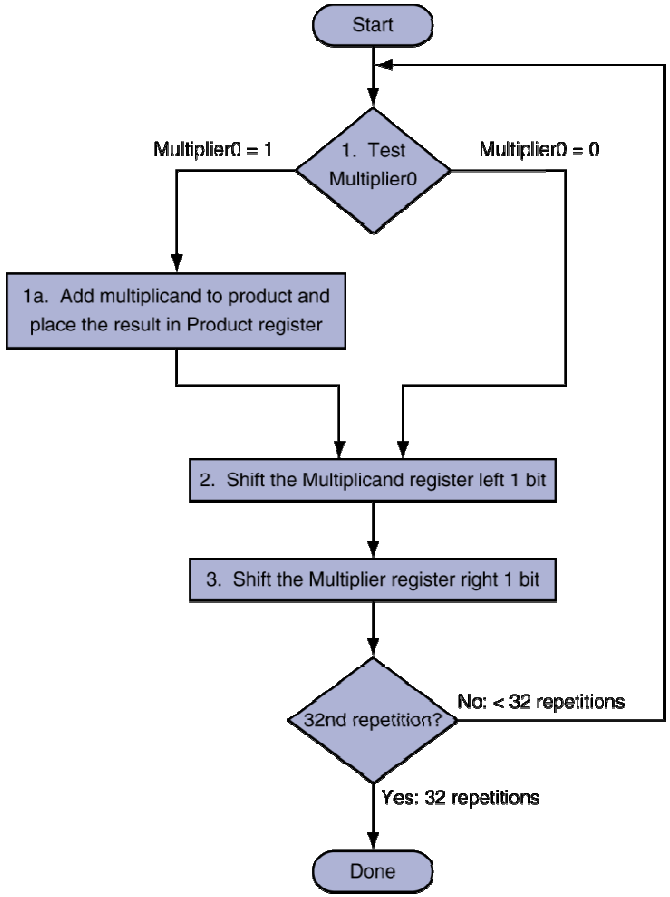
product

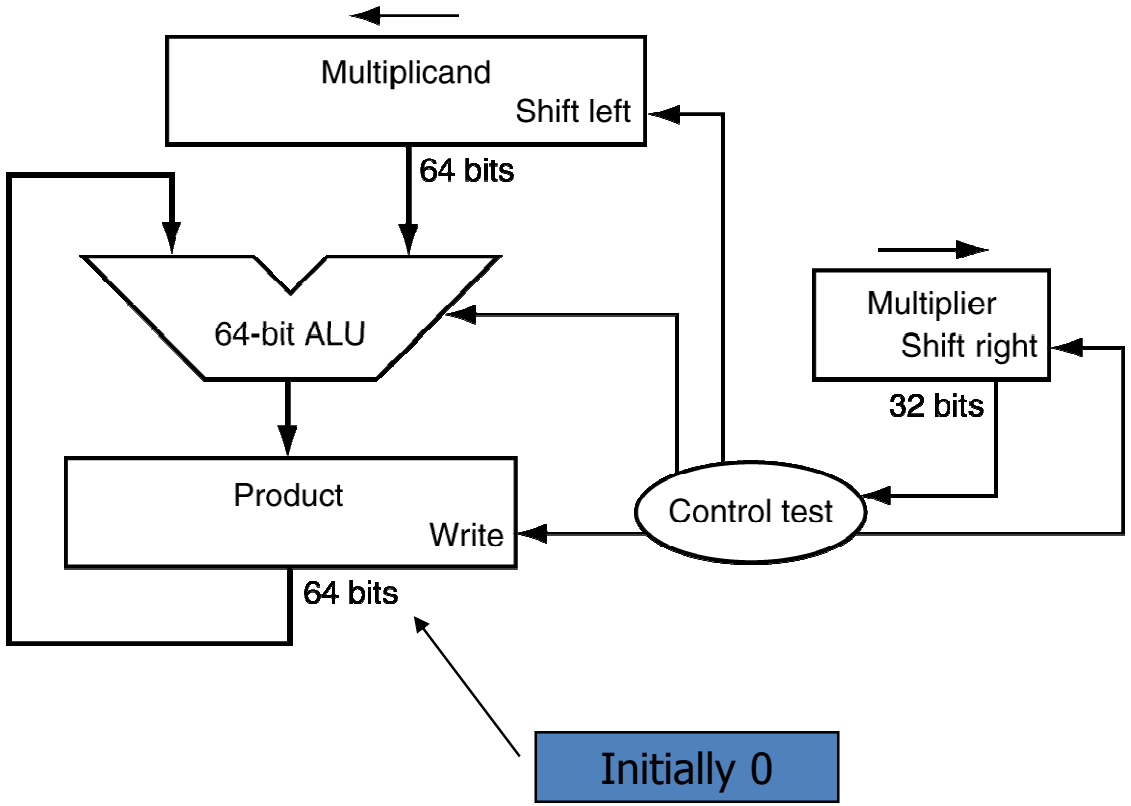
```
      1000
  ×   1001
  --------
      1000
     0000
    0000
   1000
  --------
   1001000
```

Length of product is the sum of operand lengths

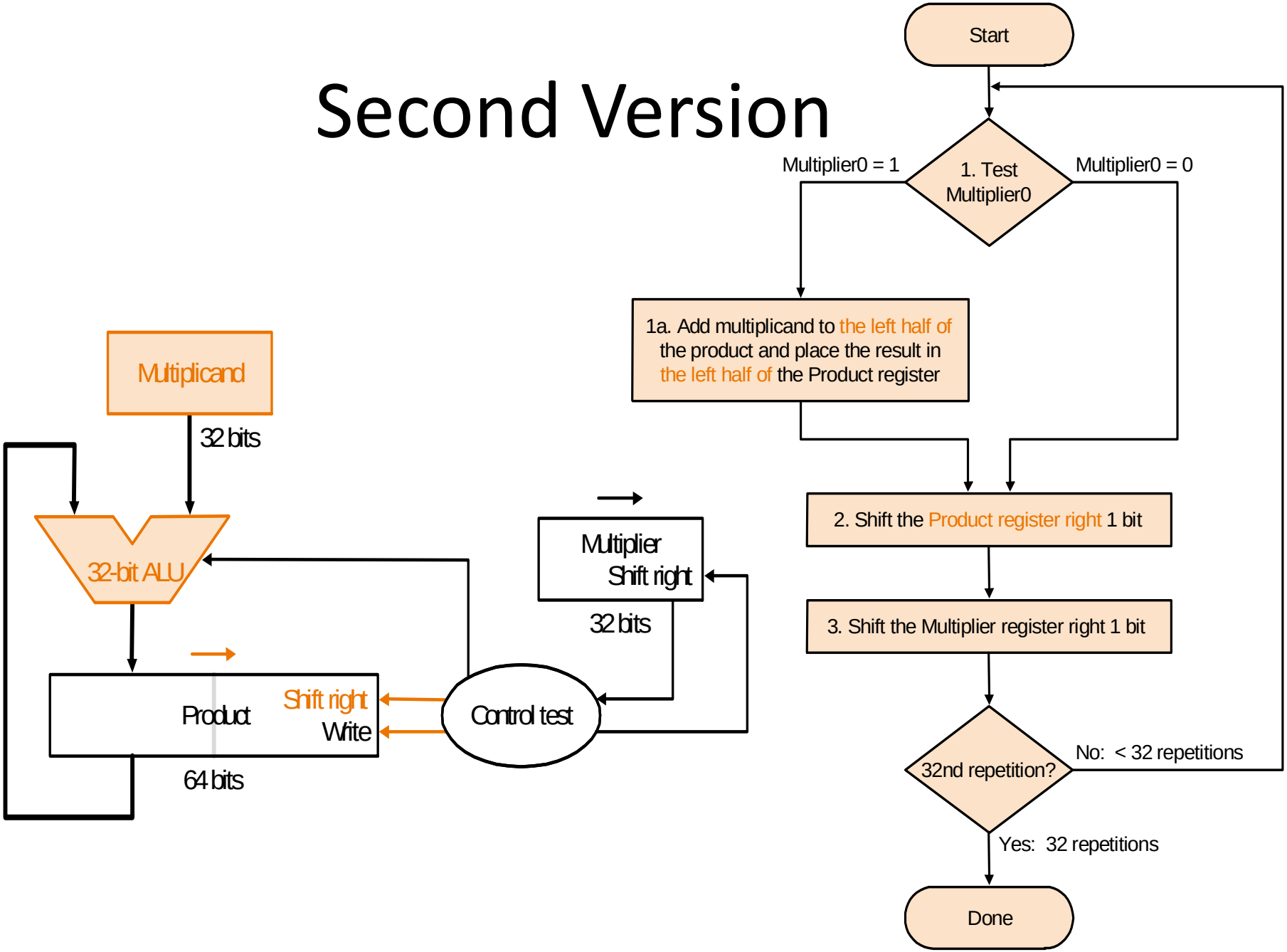


Multiplication Hardware

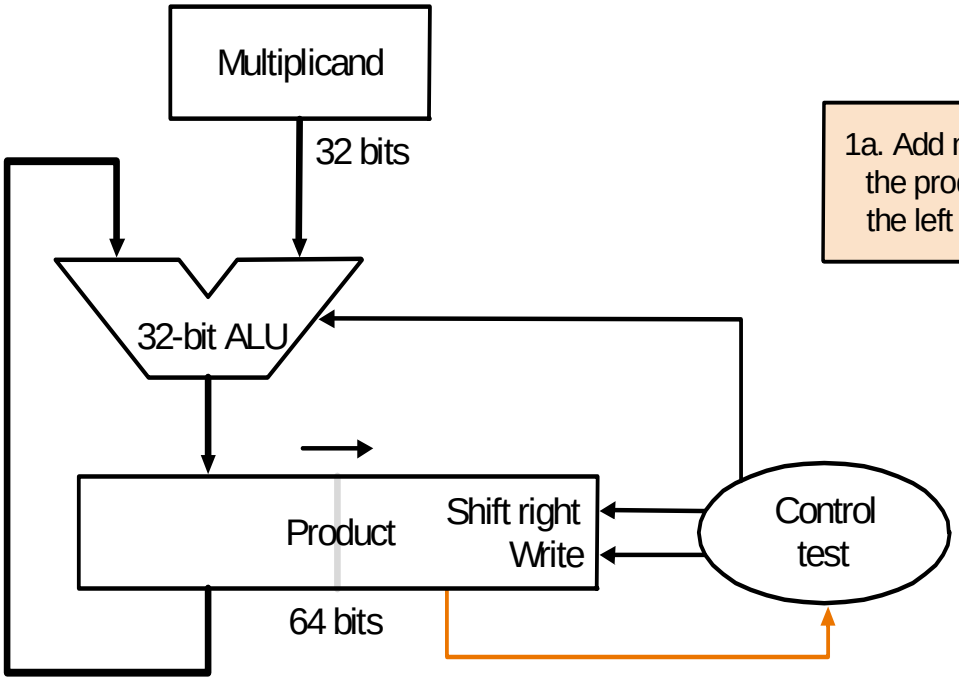


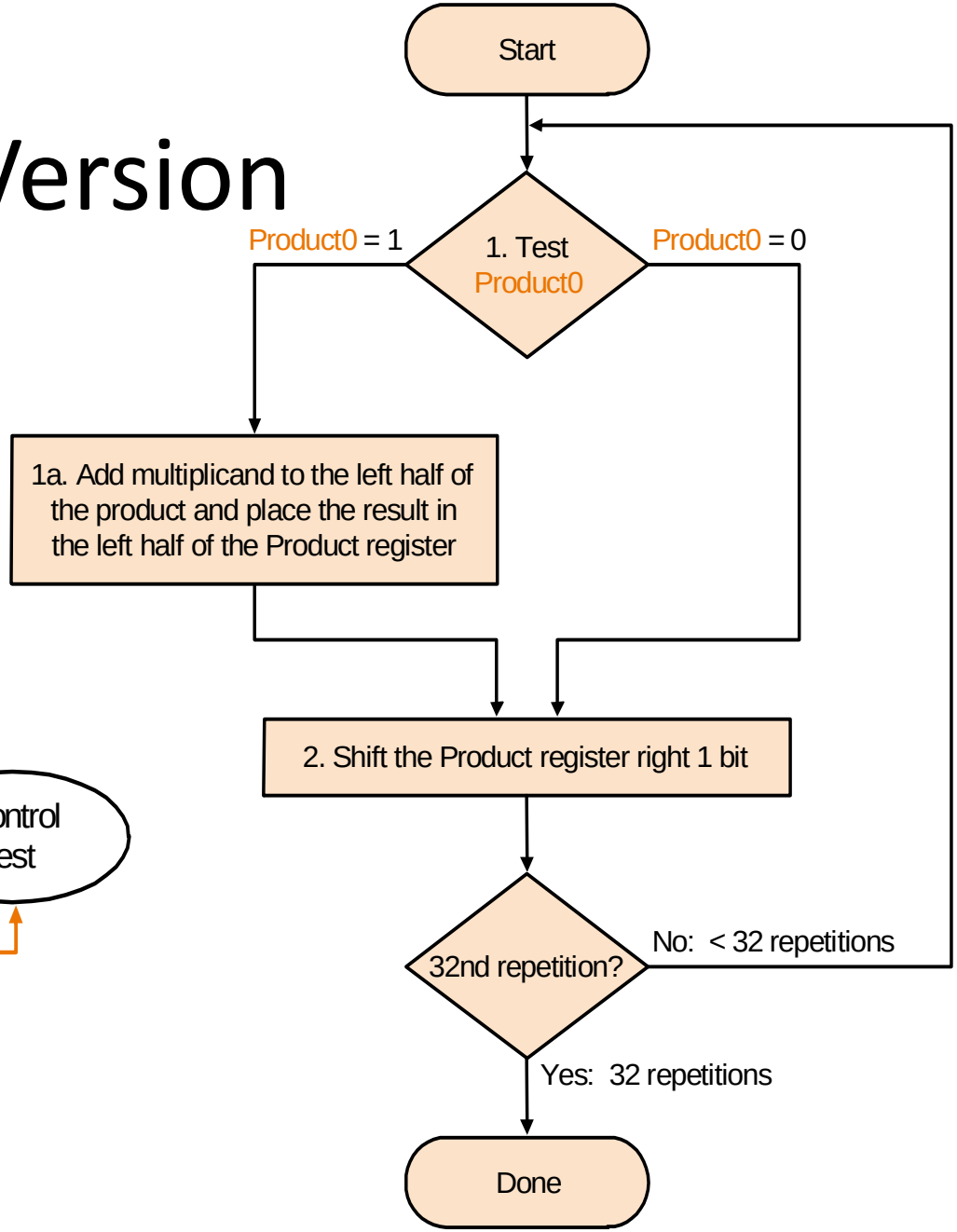


Second Version



Final Version





Signed Multiplication

Signed Multiplication

- Considering 2's-complement signed operands, what will happen to (-13)×(+11) if following the same method of unsigned multiplication?

					1	0	0	1	1		(- 13)
					0	1	0	1	1		(+11)
1	1	1	1	1	1	0	0	1	1		
1	1	1	1	1	0	0	1	1			
0	0	0	0	0	0	0	0				
1	1	1	0	0	1	1					
0	0	0	0	0	0						
1	1	0	1	1	1	0	0	0	1		(- 143)

SIGN EXTENSION IS SHOWN IN BLUE

Figure 6.8. Sign extension of negative multiplicand.

Signed Multiplication

- For a negative multiplier, a straightforward solution is to form the 2's-complement of both the multiplier and the multiplicand and proceed as in the case of a positive multiplier.
- This is possible because complementation of both operands does not change the value or the sign of the product.
- A technique that works equally well for both negative and positive multipliers – Booth algorithm.

Booth Algorithm

- Consider in a multiplication, the multiplier is positive 0011110, how many appropriately shifted versions of the multiplicand are added in a standard procedure?

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

Booth Algorithm

- Since 0011110 = 0100000 – 0000010, if we use the expression to the right, what will happen?

							0	1	0	1	1	0	1
							0	+1	0	0	0	-1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	1	0	0	1	1	← 2's complement of the multiplicand
0	0	0	0	0	0	0	0	0	0	0	0		
0	0	0	0	0	0	0	0	0	0	0			
0	0	0	0	0	0	0	0	0	0				
0	0	0	1	0	1	1	0	1					
0	0	0	0	0	0	0	0						
0	0	0	1	0	1	0	1	0	0	0	1	1	0

Booth Algorithm

- In general, in the Booth scheme, -1 times the shifted multiplicand is selected when moving from 0 to 1, and +1 times the shifted multiplicand is selected when moving from 1 to 0, as the multiplier is scanned from right to left.

0	0	1	0	1	1	0	0	1	1	1	0	1	0	1	1	0	0
								⇓									
0	+1	−1	+1	0	−1	0	+1	0	0	−1	+1	−1	+1	0	−1	0	0

Figure 6.10. Booth recoding of a multiplier.

Booth Algorithm

```
   0 1 1 0 1   (+13)
 × 1 1 0 1 0   (-6)
-------------

                    0  1  1  0  1
                    0 -1 +1 -1  0
                  ----------------
0 0 0 0 0           0  0  0  0  0
1 1 1 1 1           0  0  1  1
0 0 0 0 1           1  0  1
1 1 1 0 0           1  1
0 0 0 0 0           0
-------------------------------------
1 1 1 0 1 1 0 0 1 0   (-78)
```

Figure 6.11. Booth multiplication with a negative multiplier.

Booth Algorithm

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

Figure 6.12. Booth multiplier recoding table.

Booth Algorithm

- Best case – a long string of 1's (skipping over 1s)
- Worst case – 0's and 1's are alternating

	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Worst-case multiplier									⇓							
	+1	−1	+1	−1	+1	−1	+1	−1	+1	−1	+1	−1	+1	−1	+1	−1

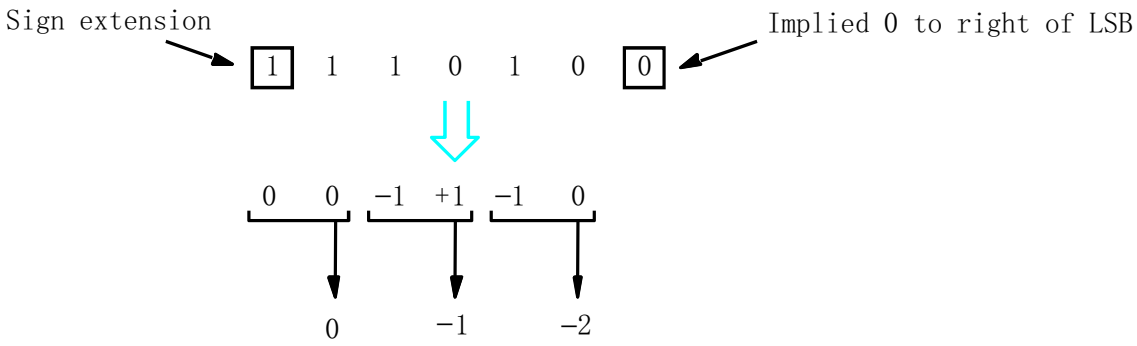
	1	1	0	0	0	1	0	1	1	0	1	1	1	1	0	0
Ordinary multiplier									⇓							
	0	−1	0	0	+1	−1	+1	0	−1	+1	0	0	0	−1	0	0

	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1
Good multiplier									⇓							
	0	0	0	+1	0	0	0	0	−1	0	0	0	+1	0	0	−1

Fast Multiplication

Bit-Pair Recoding of Multipliers

- Bit-pair recoding halves the maximum number of summands (versions of the multiplicand).



(a) Example of bit-pair recoding derived from Booth recoding

Bit-Pair Recoding of Multipliers

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position
$i+1$	i	$i-1$	
0	0	0	0 ×M
0	0	1	+1 ×M
0	1	0	+1 ×M
0	1	1	+2 ×M
1	0	0	−2 ×M
1	0	1	−1 ×M
1	1	0	−1 ×M
1	1	1	0 ×M

(b) Table of multiplicand selection decisions

Bit-Pair Recoding of Multipliers

	0	1	1	0	1	(+13)
×	1	1	0	1	0	(−6)

⇓

					0	1	1	0	1	
					0	−1	+1	−1	0	
0	0	0	0	0	0	0	0	0	0	
1	1	1	1	1	0	0	1	1		
0	0	0	0	1	1	0	1			
1	1	1	0	0	1	1				
0	0	0	0	0	0					
1	1	1	0	1	1	0	0	1	0	(−78)

⇓

					0	1	1	0	1	
					0		−1		−2	
1	1	1	1	1	0	0	1	1	0	
1	1	1	1	0	0	1	1			
0	0	0	0	0	0					
1	1	1	0	1	1	0	0	1	0	

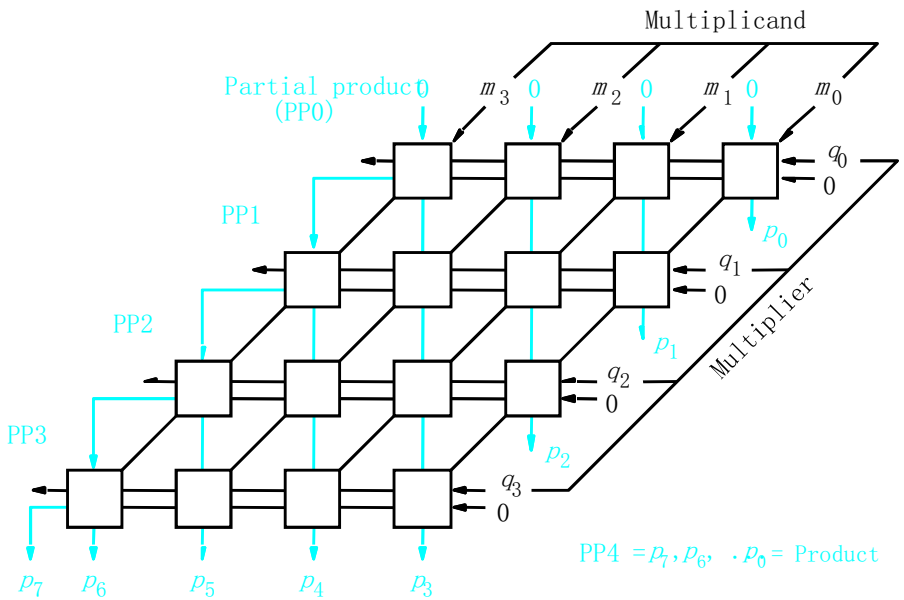
Figure 6.15. Multiplication requiring only $n/2$ summands.

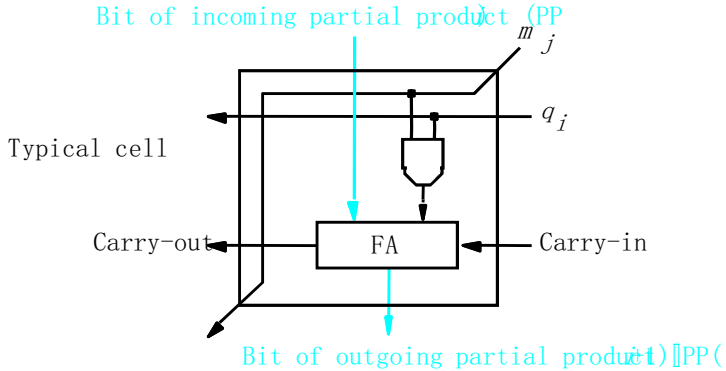
Carry-Save Addition of Summands

```
                1   1   0   1        (13)  Multiplicand M
            ´   1   0   1   1        (11)  Multiplier Q
            ---------------------
                1   1   0   1
            1   1   0   1
        0   0   0   0
    1   1   0   1
-----------------------------------
1   0   0   0   1   1   1   1        (143)  Product P
```

(a) Manual multiplication algorithm

Carry-Save Addition of Summands

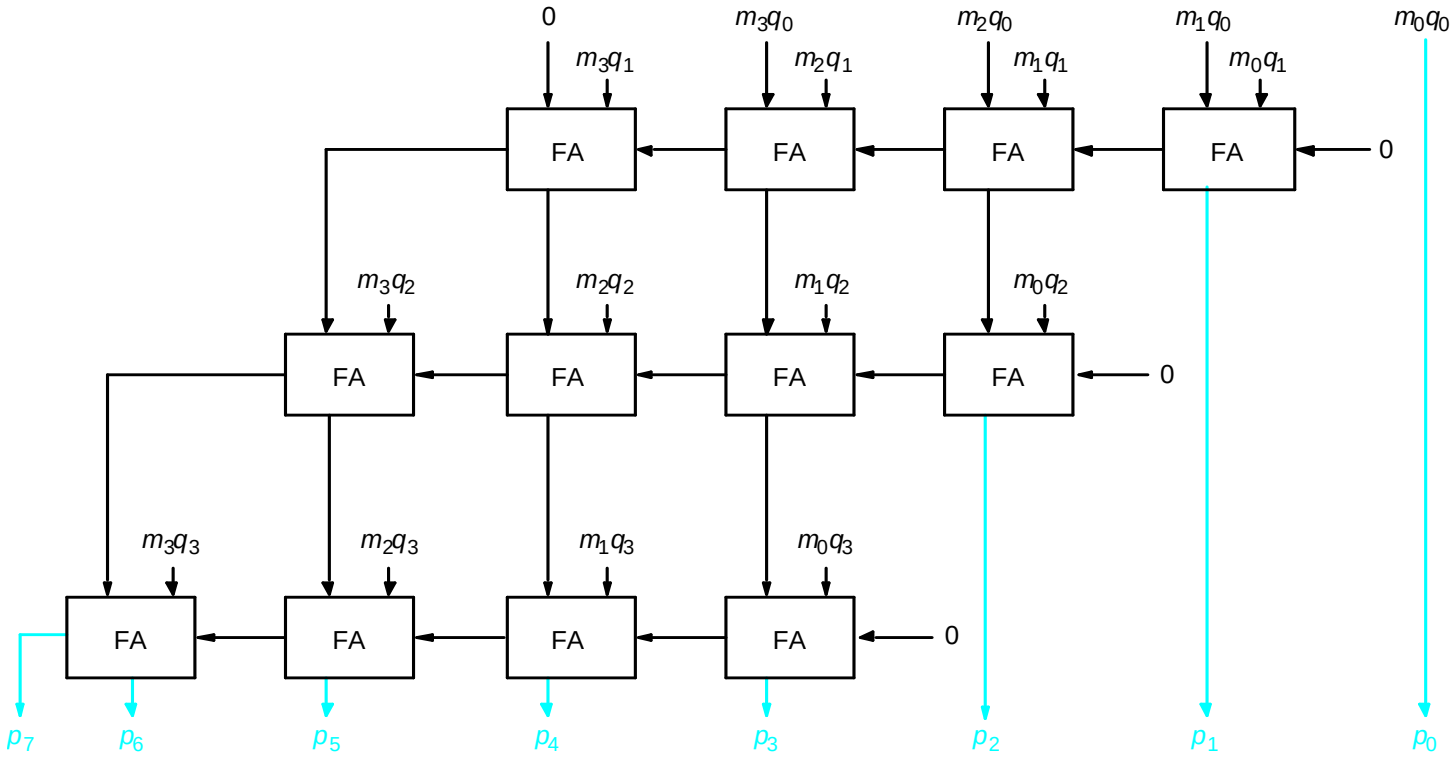




(b) Array implementation

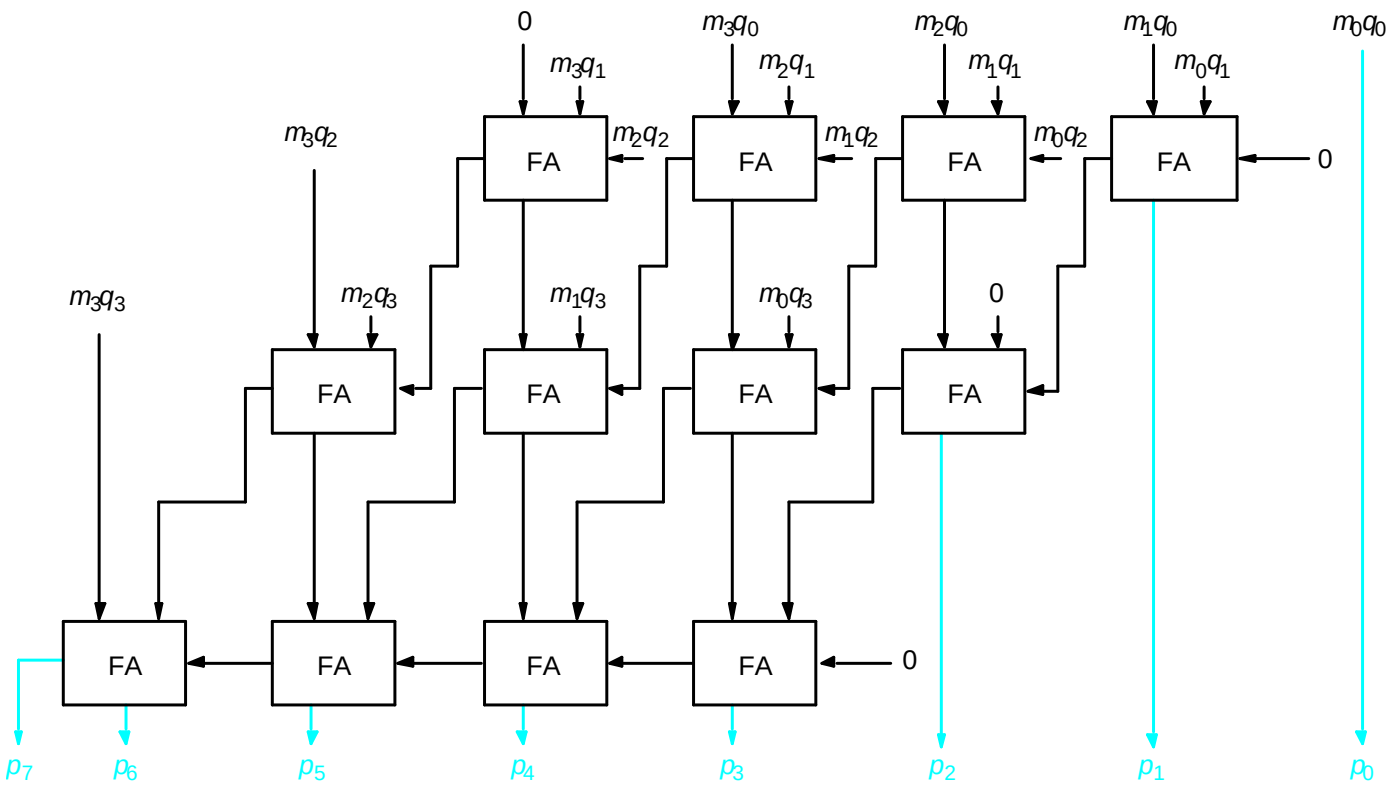
Carry-Save Addition of Summands

- CSA speeds up the addition process.



(a) Ripple-carry array (Figure 6.6 structure)

Carry-Save Addition of Summands



(b) Carry-save array

Figure 6.16. Ripple-carry and carry-save arrays for the multiplication operation M × Q = P for 4-bit operands.

Carry-Save Addition of Summands

- The delay through the carry-save array is somewhat less than delay through the ripple-carry array. This is because the S and C vector outputs from each row are produced in parallel in one full-adder delay.
- Consider the addition of many summands, we can:

➢ Group the summands in threes and perform carry-save addition on each of these groups in parallel to generate a set of S and C vectors in one full-adder delay

➢ Group all of the S and C vectors into threes, and perform carry-save addition on them, generating a further set of S and C vectors in one more full-adder delay

➢ Continue with this process until there are only two vectors remaining

➢ They can be added in a RCA or CLA to produce the desired product

Carry-Save Addition of Summands

						1	0	1	1	0	1	(45)	M
					x	1	1	1	1	1	1	(63)	Q
						1	0	1	1	0	1	A	
					1	0	1	1	0	1		B	
				1	0	1	1	0	1			C	
			1	0	1	1	0	1				D	
		1	0	1	1	0	1					E	
	1	0	1	1	0	1						F	
1	0	1	1	0	0	0	1	0	0	1	1	(2,835)	Product

Figure 6.17. A multiplication example used to illustrate carry-save addition as shown in Figure 6.18.