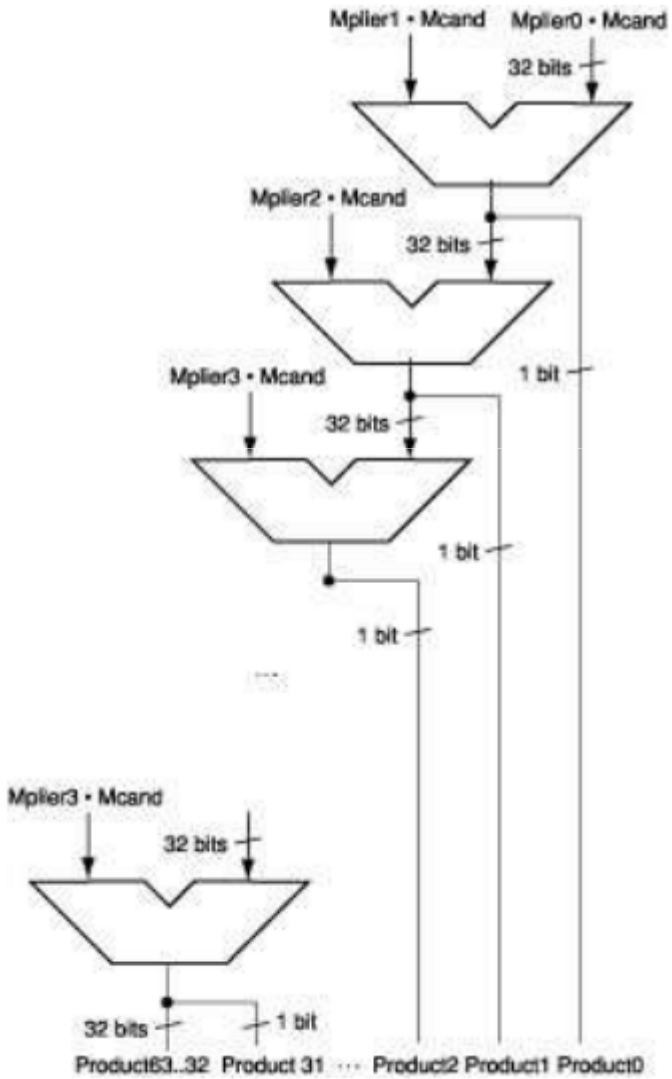
Unsigned Multiplication

Multiplication of Unsigned Numbers

				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



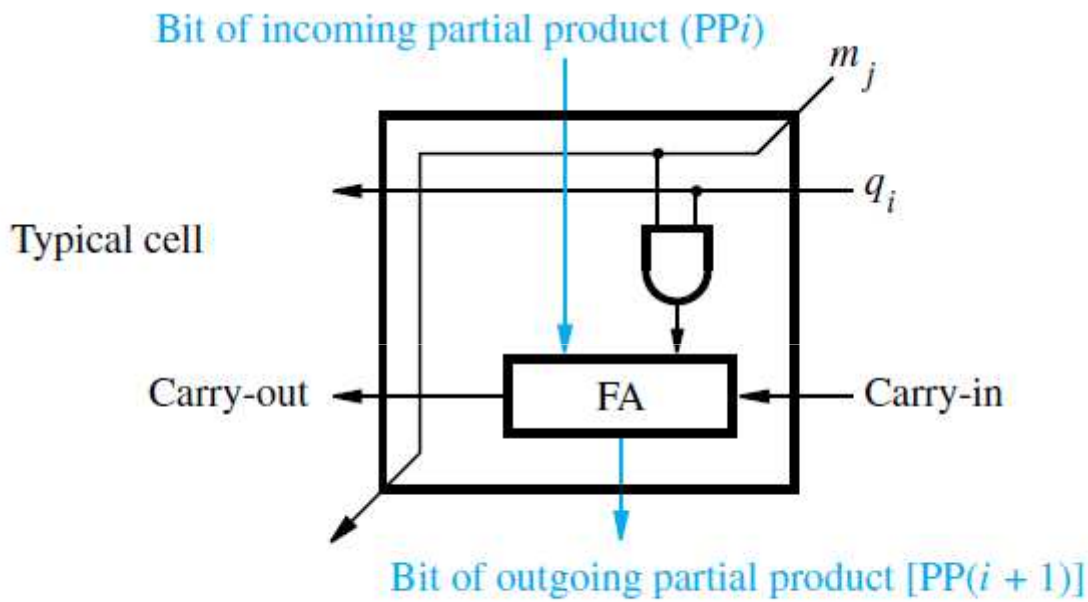
Multiplication of two fixed-point binary numbers in signed magnitude representations is done by a process of successive shift and add operations

Multiplication of Unsigned Numbers

- The process consists of looking at successive bits of the multiplier, LSB first.
- If the multiplier bit is a 1 → multiplicand is copied down

  0 → zeroes are copied down.
- The number copied down in successive lines are shifted one position to the left from the previous number finally the numbers are added and their sum forms a product.
- Sign of the product is determine from the signs of the multiplicand and multiplier.
  - If they are alike the sign of the product is positive.
  - It they are unlike the sign of the product is negative

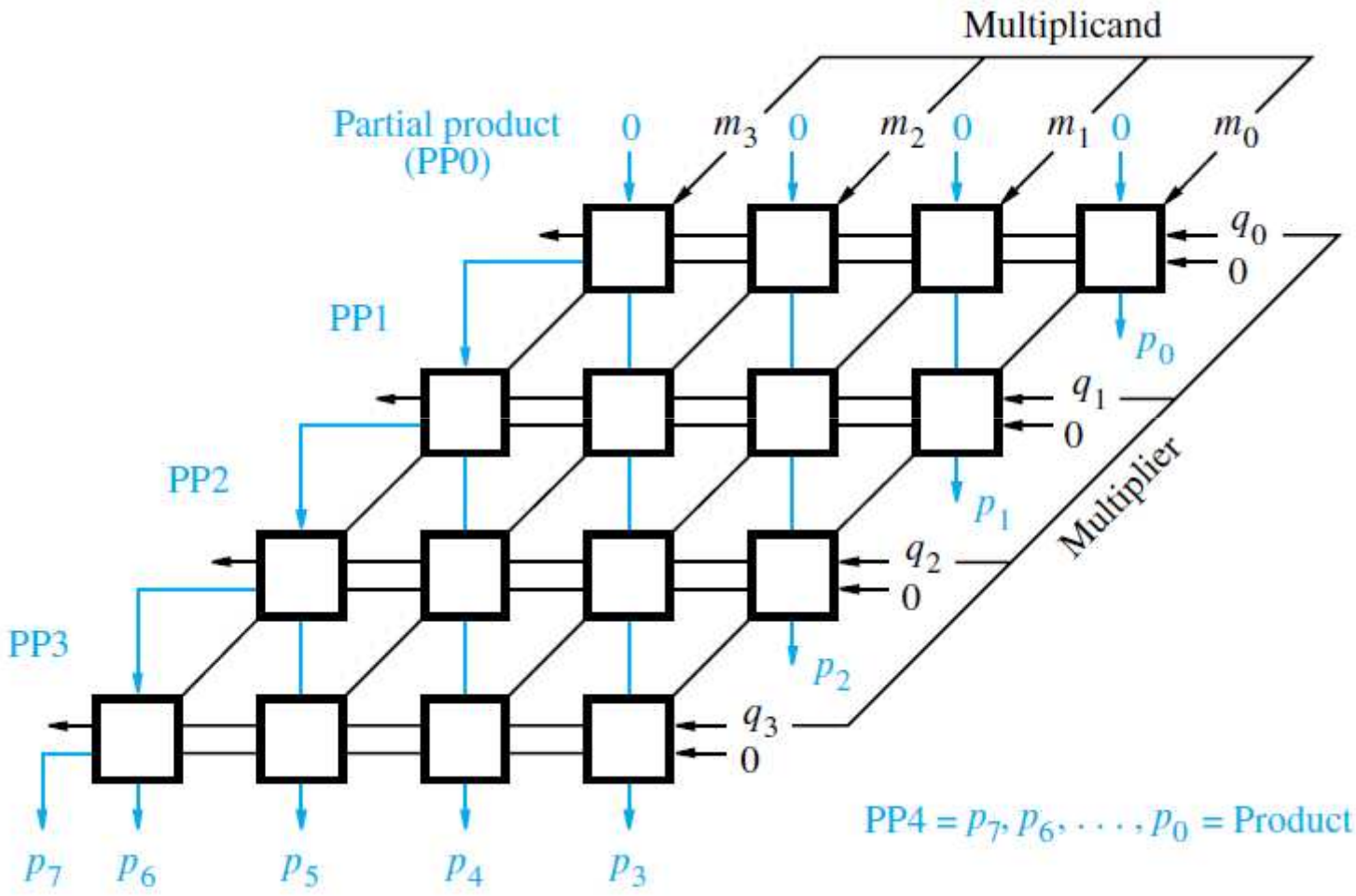
Array Multiplier



(b) Array implementation

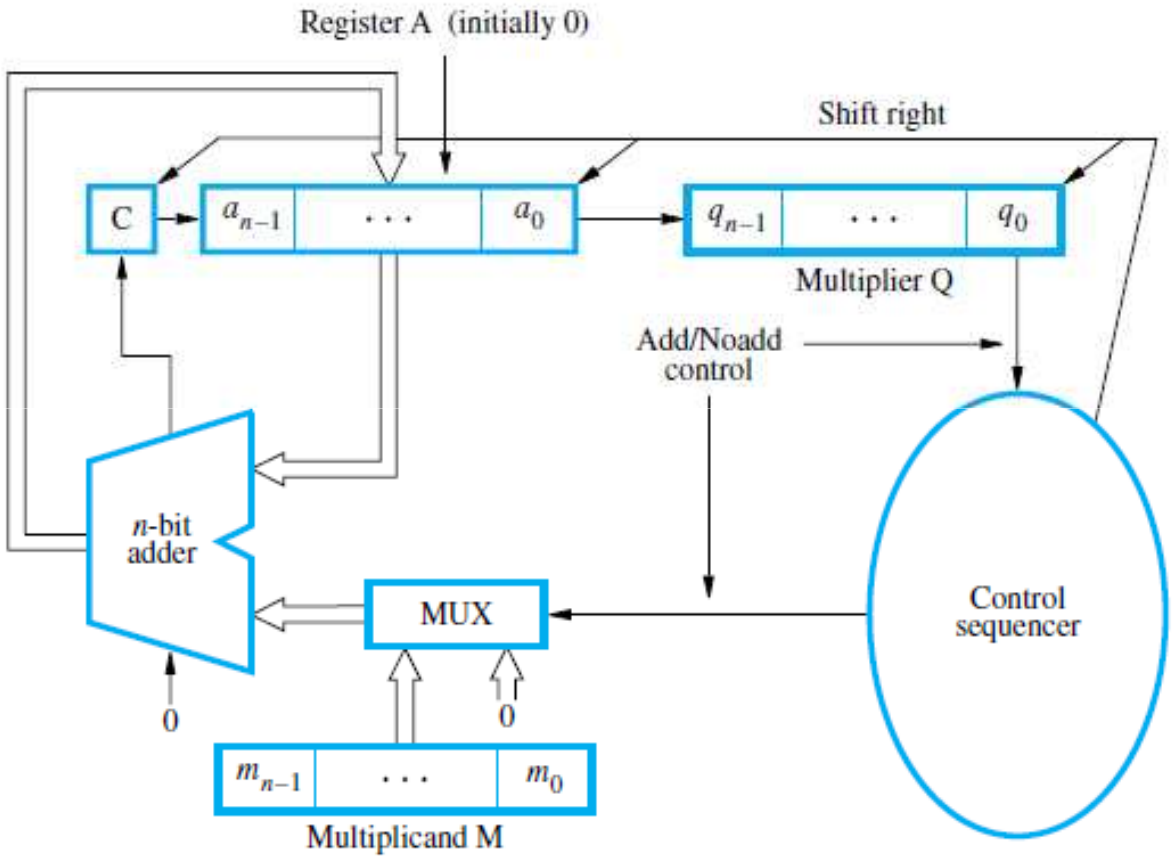
Combinational logic & Array Implementation

Array Multiplier



Combinational logic & Array Implementation

Sequential Circuit Multiplier



(a) Register configuration

Final Version

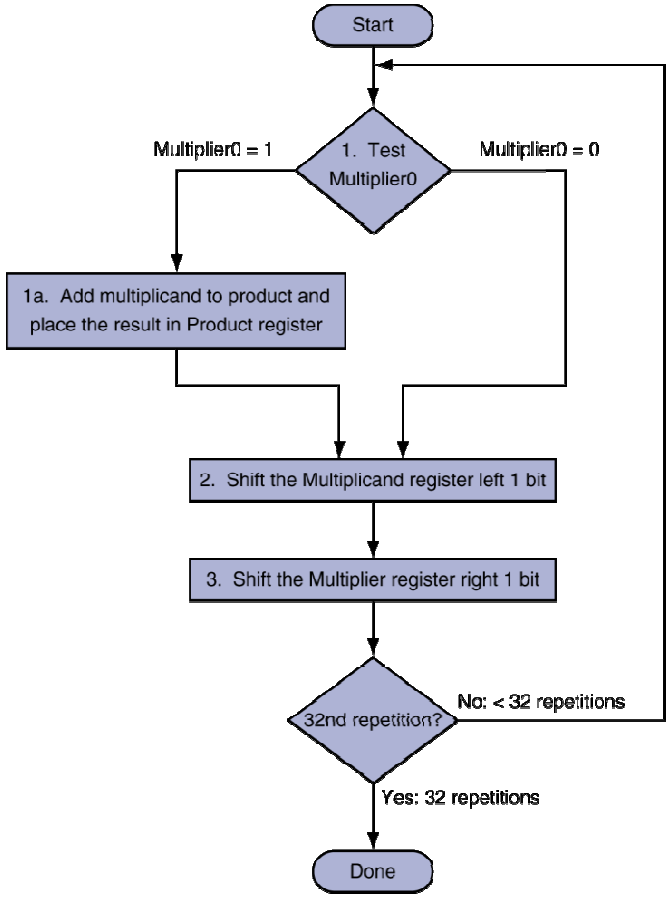
Sequential Circuit Multiplier

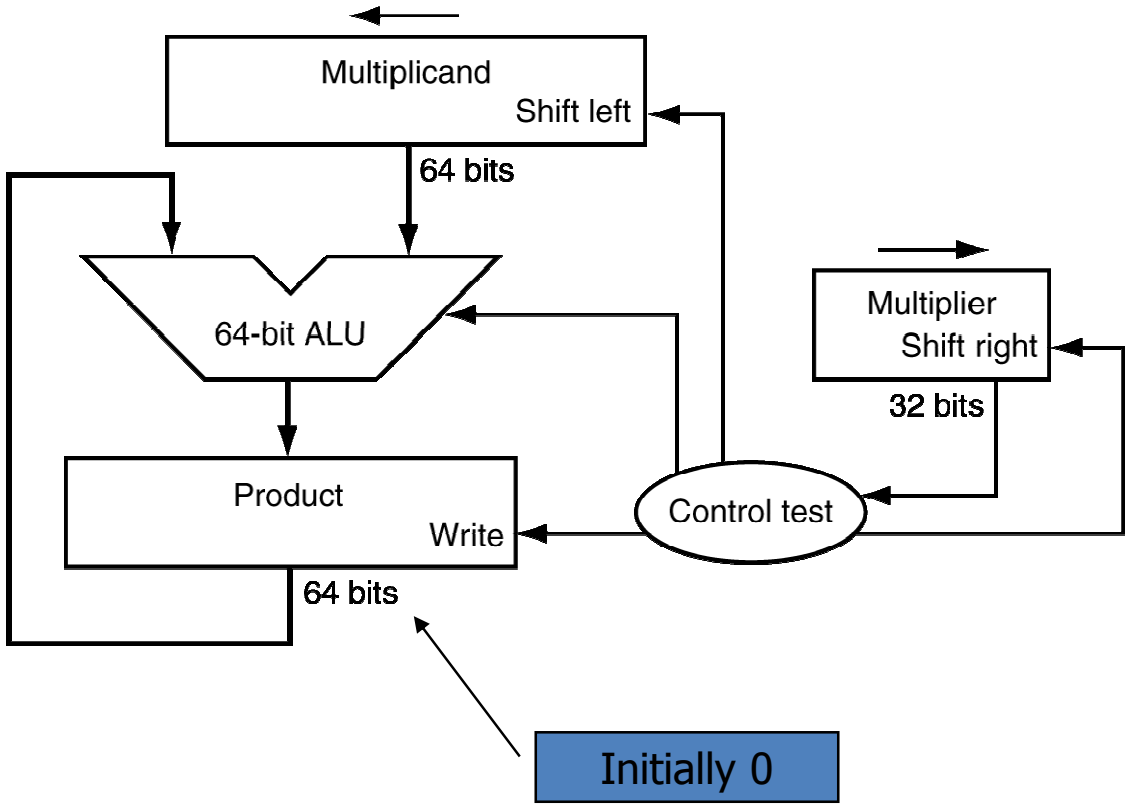
	M		
	1 1 0 1		Initial configuration
0	0 0 0 0	1 0 1 1	
C	A	Q	
0	1 1 0 1	1 0 1 1	Add
0	0 1 1 0	1 1 0 1	Shift — First cycle
1	0 0 1 1	1 1 0 1	Add
0	1 0 0 1	1 1 1 0	Shift — Second cycle
0	1 0 0 1	1 1 1 0	No add
0	0 1 0 0	1 1 1 1	Shift — Third cycle
1	0 0 0 1	1 1 1 1	Add
0	1 0 0 0	1 1 1 1	Shift — Fourth cycle
	Product		

(b) Multiplication example

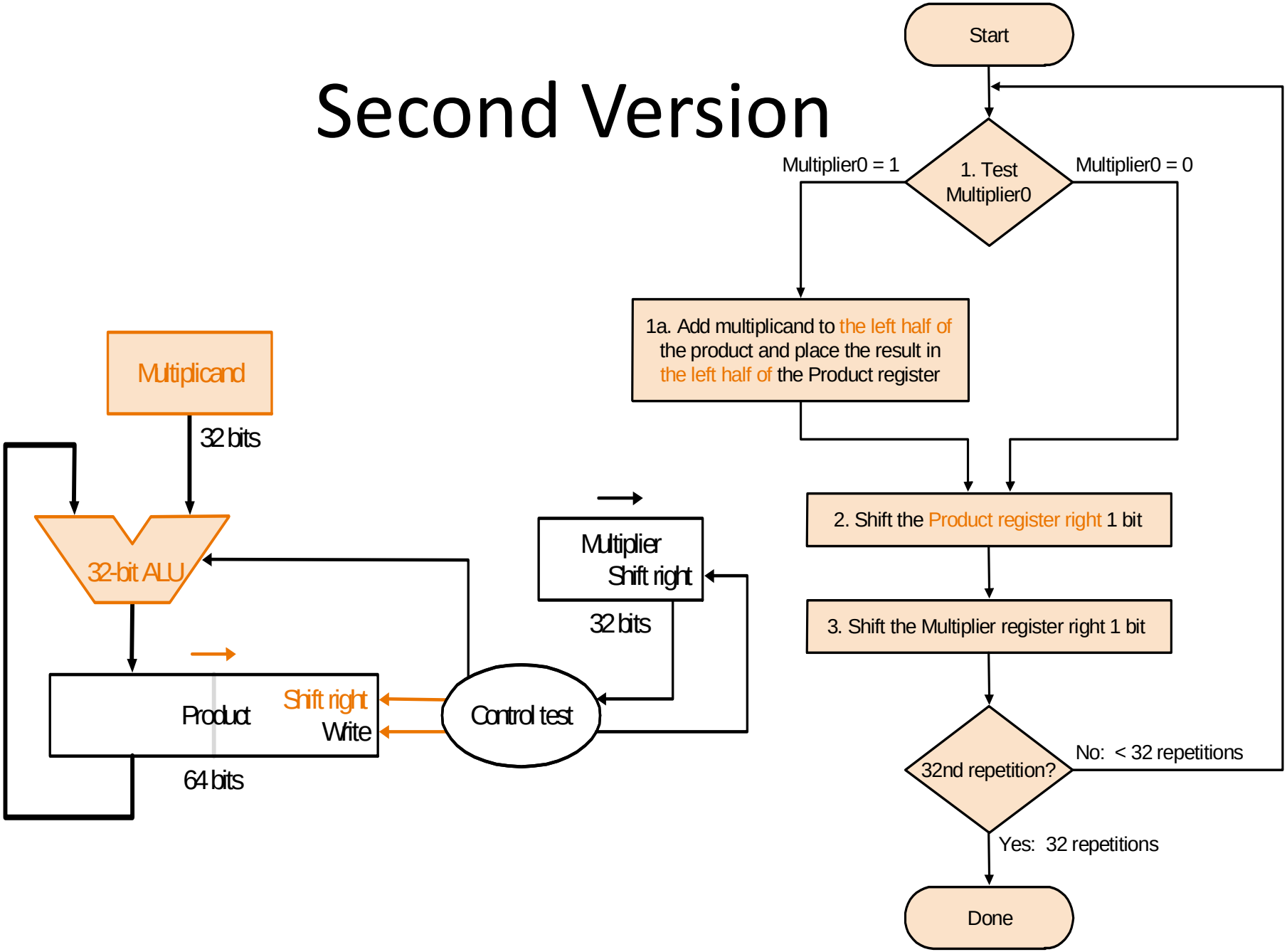
Sequential circuit binary multiplier.

Multiplication Hardware First version

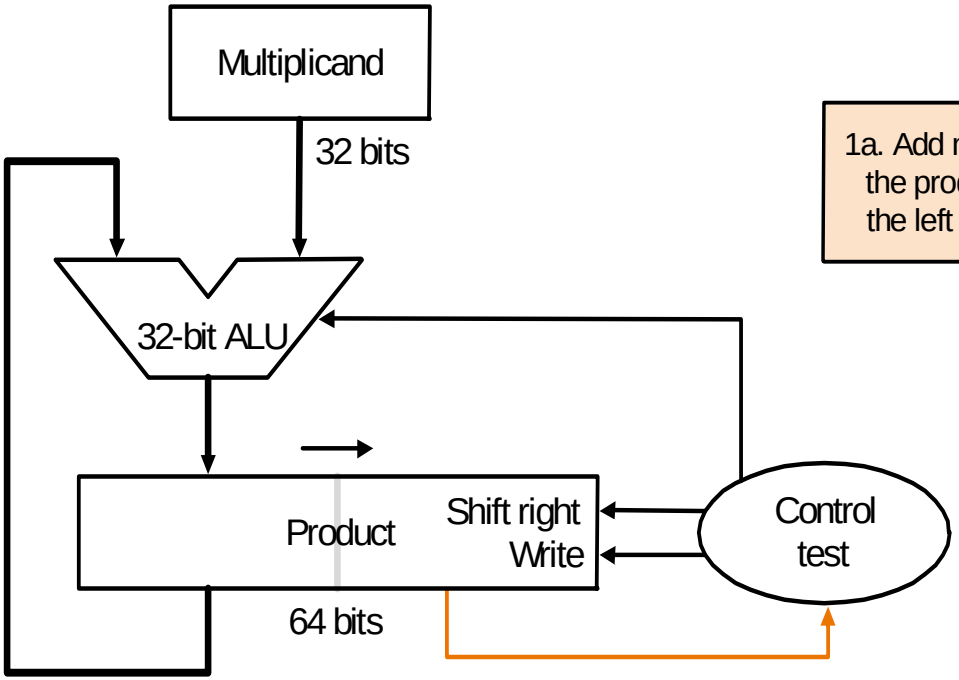


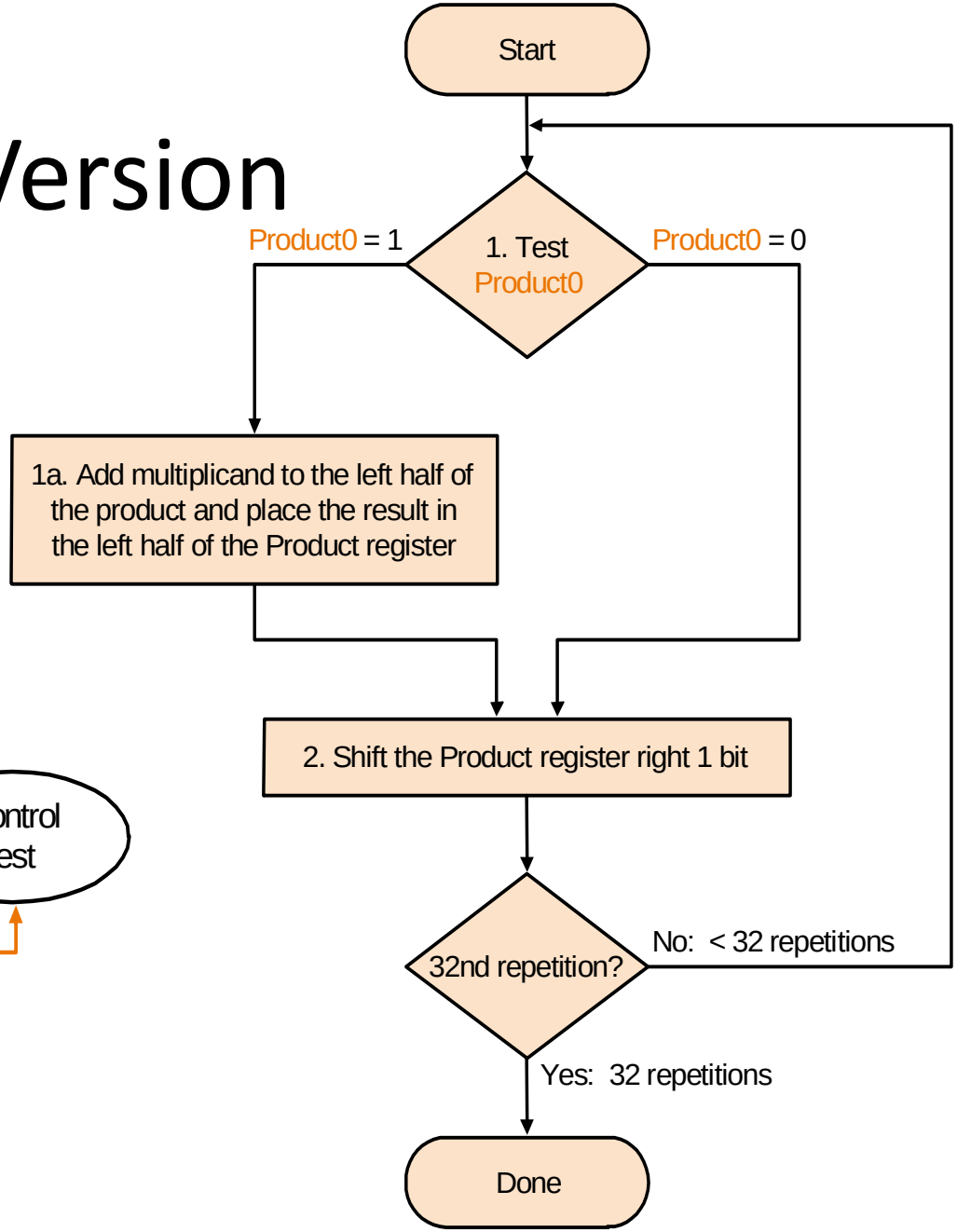


Second Version



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?

Sign extension is shown in blue

					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 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

Normal multiplication schemes

Booth Algorithm

- Handles both positive and negative multipliers uniformly
- it achieves some efficiency in the number of additions required when the multiplier has a few large blocks of 1s.

Booth Algorithm

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

	0100000	(32)
–	0000010	(2)
	0011110	(30)

							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 multiplication schemes.

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	$0 \times M$

Booth multiplier recoding table.

The Booth technique for recoding multipliers is summarized in Figure

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

Booth recoding of a multiplier.

The transformation *is called skipping over 1s.*

Booth Algorithm

Worst-case multiplier	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	+1	–1	+1	–1	+1	–1	+1	–1	+1	–1	+1	–1	+1	–1	+1	–1
Ordinary multiplier	1	1	0	0	0	1	0	1	1	0	1	1	1	1	0	0
	0	–1	0	0	+1	–1	+1	0	–1	+1	0	0	0	–1	0	0
Good multiplier	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1
	0	0	0	+1	0	0	0	0	–1	0	0	0	+1	0	0	–1

Booth recoded multipliers.

Booth Algorithm: Example

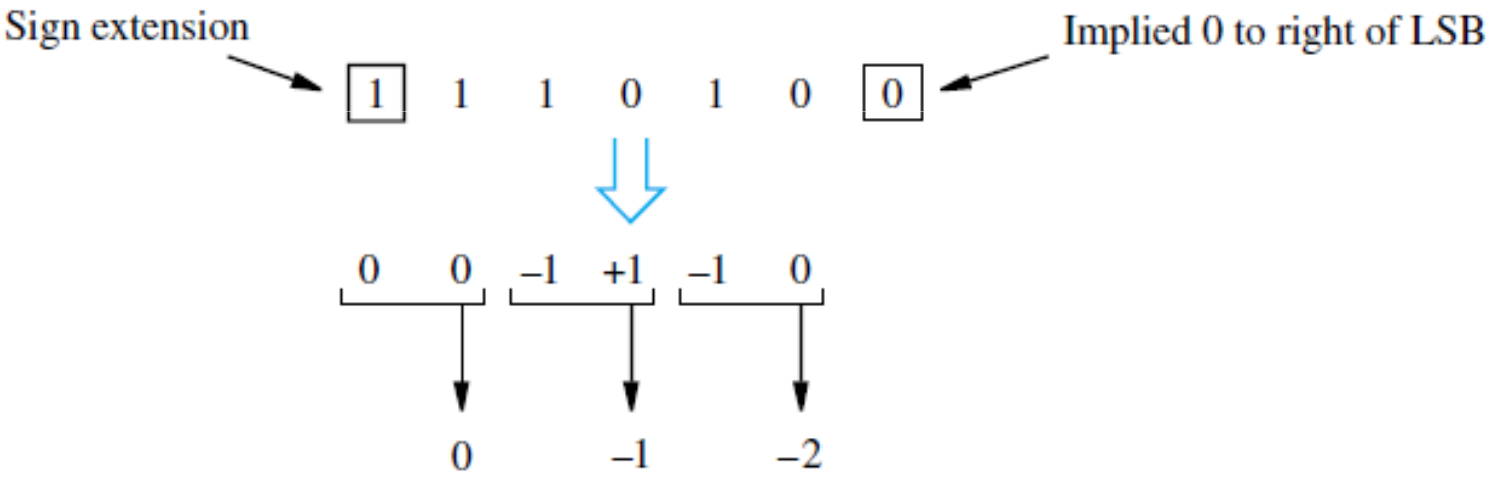
$$
\begin{array}{rrrrrrr}
 & 0 & 1 & 1 & 0 & 1 & (+13) \\
\times & 1 & 1 & 0 & 1 & 0 & (-6) \\
\hline
\end{array}
\quad \Longrightarrow \quad
\begin{array}{rrrrrrrrrrr}
 & & & & & 0 & 1 & 1 & 0 & 1 & \\
 & & & & & 0 & -1 & +1 & -1 & 0 & \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & & \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & & & \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & & & & & \\
\hline
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & (-78)
\end{array}
$$

Booth multiplication with a negative multiplier.

Fast Multiplication

Bit-Pair Recoding of Multipliers

- Derived directly from the Booth algorithm
- Group the Booth-recoded multiplier bits in pairs



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
$i+1$	i	$i-1$	
0	0	0	$0 \times M$
0	0	1	$+1 \times M$
0	1	0	$+1 \times M$
0	1	1	$+2 \times M$
1	0	0	$-2 \times M$
1	0	1	$-1 \times M$
1	1	0	$-1 \times M$
1	1	1	$0 \times M$

(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)

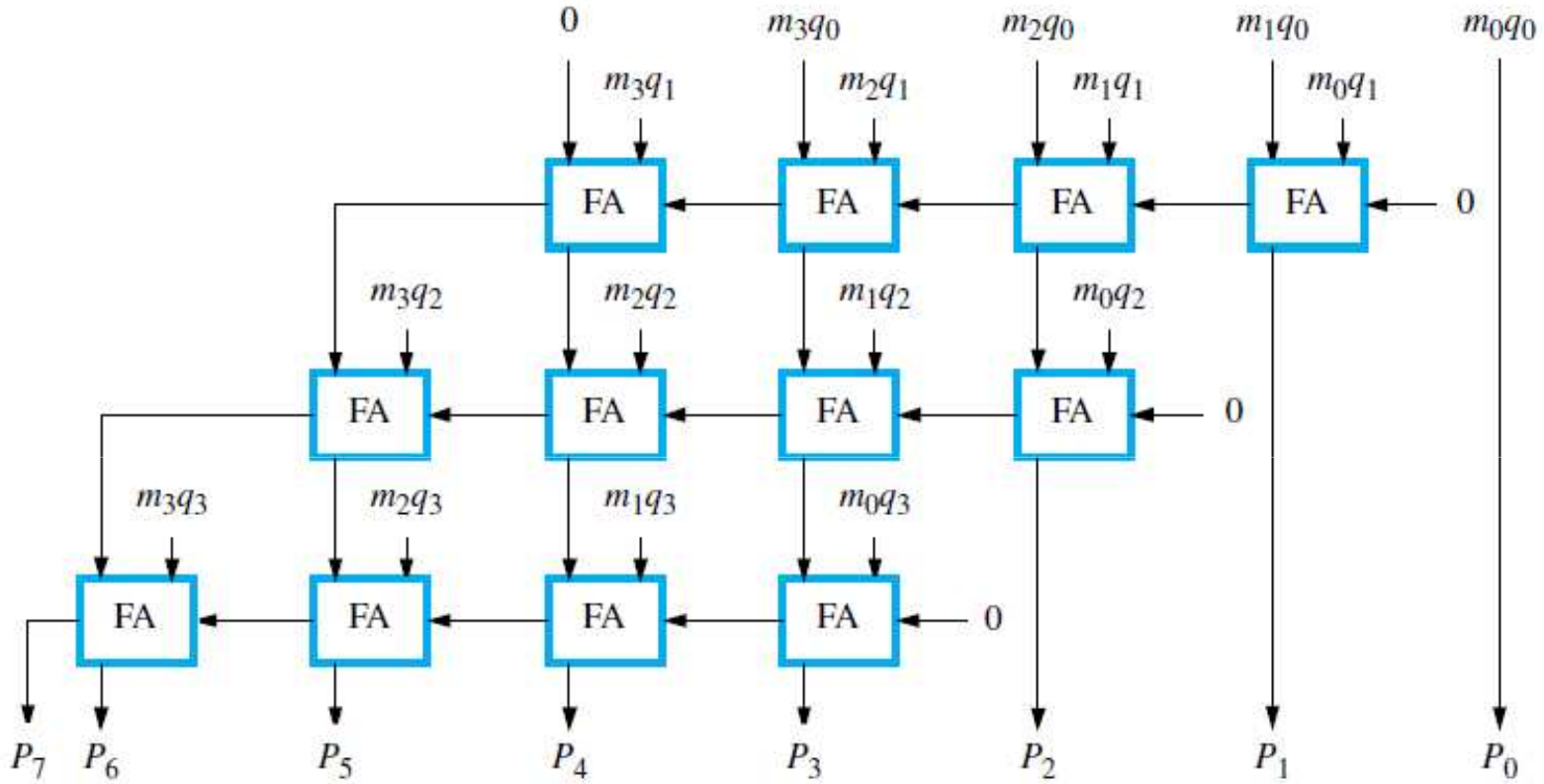
Booth

					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	

Bit-Pair Recoding

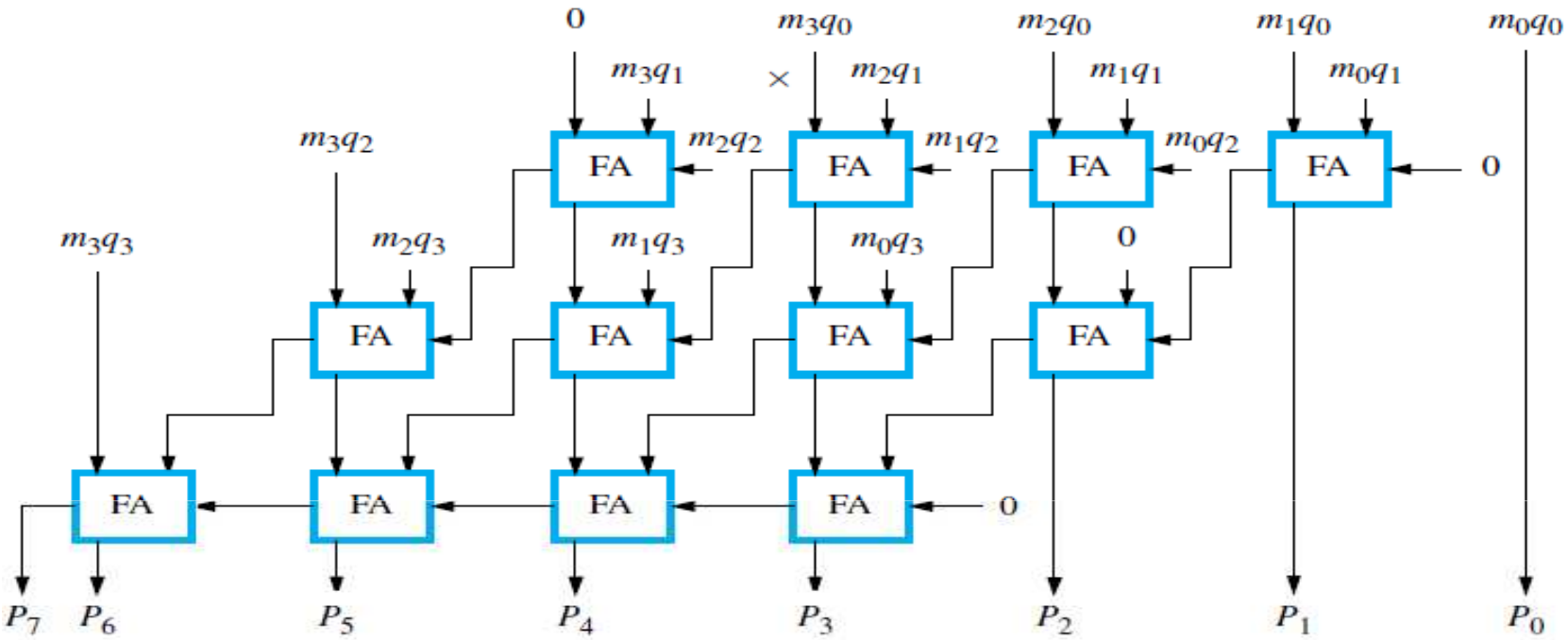
Multiplication requiring only $n/2$ summands.

Carry-Save Addition of Summands



(a) Ripple-carry array

Carry-Save Addition of Summands



(b) Carry-save array

Ripple-carry and carry-save arrays for a 4 × 4 multiplier.

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.

Carry-Save Addition of Summands

- Consider the addition of many summands
  - 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
					×	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

A multiplication example used to illustrate carry-save addition

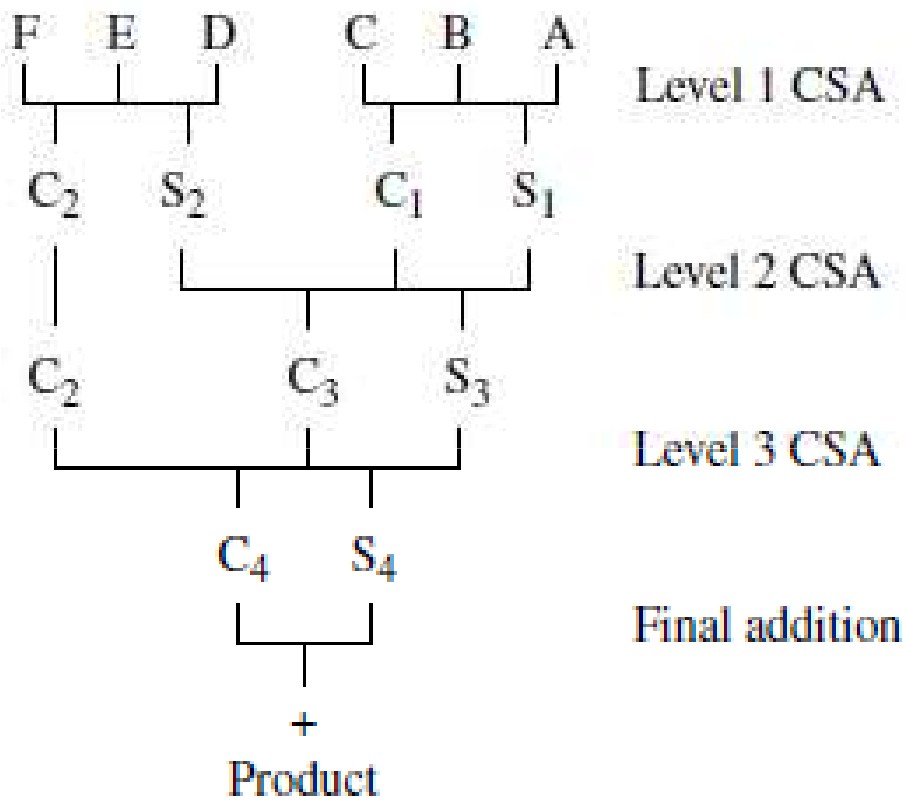
```
                    1  0  1  1  0  1   M
                  × 1  1  1  1  1  1   Q
                  ------------------
                    1  0  1  1  0  1     A
                 1  0  1  1  0  1       B
              1  0  1  1  0  1          C
           ---------------------------
           1  1  0  0  0  0  1  1       S1
        0  0  1  1  1  1  0  0          C1

        1  0  1  1  0  1                D
     1  0  1  1  0  1                   E
  1  0  1  1  0  1                      F
  ------------------------
  1  1  0  0  0  0  1  1                S2
0 0  1  1  1  1  0  0                   C2

           1  1  0  0  0  0  1  1       S1
        0  0  1  1  1  1  0  0          C1
  1  1  0  0  0  0  1  1                S2
  ---------------------------
  1  1  0  1  0  1  0  0  0  1  1       S3
0 0  0  0  1  0  1  1  0  0  0          C3
0 0  1  1  1  1  0  0                   C2
  ---------------------------
0 1  0  1  1  1  0  1  0  0  1  1       S4
+ 0 1  0  1  0  1  0  0  0  0  0        C4
  ---------------------------
1 0  1  1  0  0  0  1  0  0  1  1       Product
```

The multiplication example from Figure 9.17 performed using carry-save addition.

Carry-Save Addition of Summands



Schematic representation of the carry-save addition operations

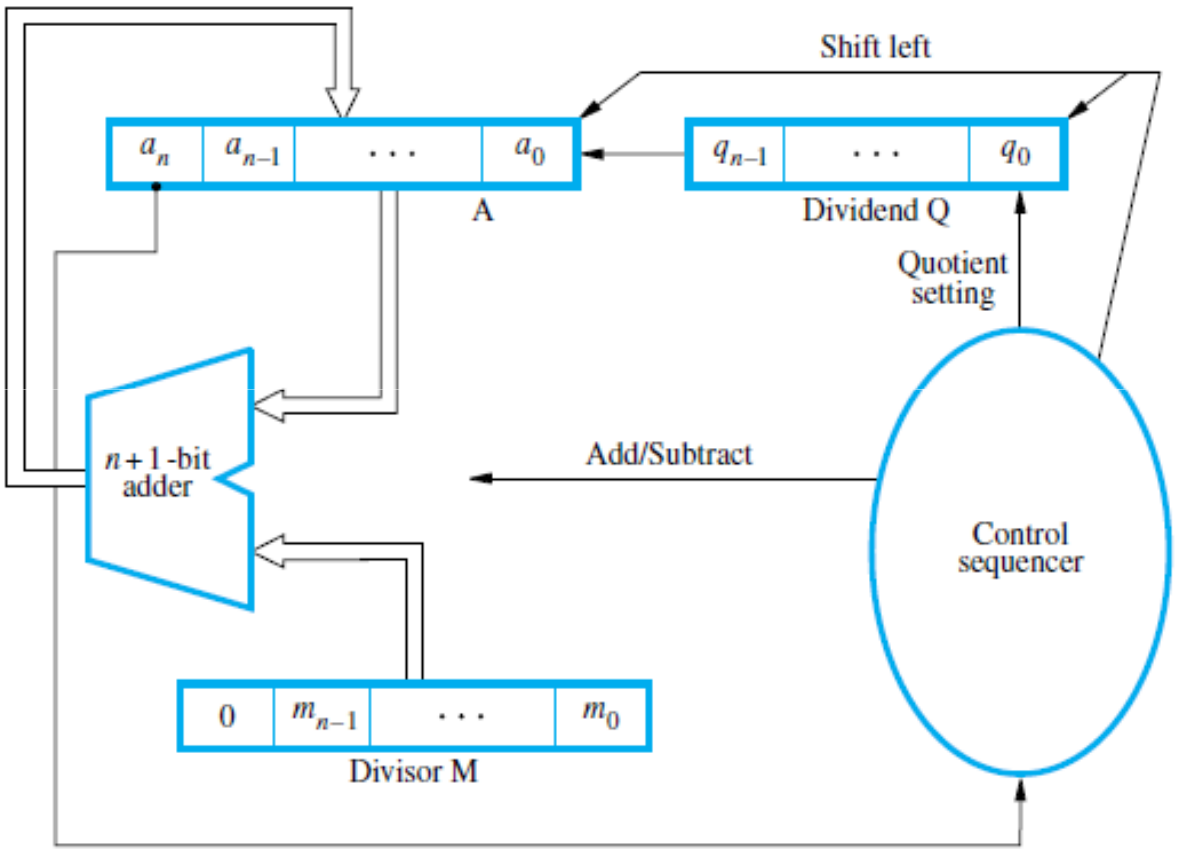
Integer Division

```
      21
13 ) 274
     26
     ---
      14
      13
     ---
       1
```

```
                10101
1101 ) 100010010
       1101
       ----
        10000
         1101
        -----
         1110
         1101
         ----
            1
```

Longhand division examples.

Restoring Division



Circuit arrangement for binary division.

Restoring Division

Do the following three steps n times:

1. Shift A and Q left one bit position.
2. Subtract M from A, and place the answer back in A.
3. If the sign of A is 1, set q_0 to 0 and add M back to A (that is, restore A); otherwise, set q_0 to 1.

Restoring Division

```
        10
11 ) 1000
       11
       --
        10
```

	Remainder	Quotient	
Initially	0 0 0 0 0	1 0 0 0	First cycle
	0 0 0 1 1		
Shift	0 0 0 0 1	0 0 0 □	
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 0		
Restore	1 1		
	0 0 0 0 1	0 0 0 0	
Shift	0 0 0 1 0	0 0 0 □	Second cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1		
Restore	1 1		
	0 0 0 1 0	0 0 0 0	
Shift	0 0 1 0 0	0 0 0 □	Third cycle
Subtract	1 1 1 0 1		
Set q_0	0 0 0 0 1		
Shift	0 0 0 1 0	0 0 0 1	
Subtract	1 1 1 0 1	0 0 1 □	Fourth cycle
Set q_0	1 1 1 1 1		
Restore	1 1		
	0 0 0 1 0	0 0 1 0	

A restoring division example.

Non-Restoring Division

Stage 1: Do the following two steps n times:

1. If the sign of A is 0, shift A and Q left one bit position and subtract M from A; otherwise, shift A and Q left and add M to A.
2. Now, if the sign of A is 0, set q_0 to 1; otherwise, set q_0 to 0.

Stage 2: If the sign of A is 1, add M to A.

Non-Restoring Division

Initially	0 0 0 0 0	1 0 0 0	First cycle
	0 0 0 1 1		
Shift	0 0 0 0 1	0 0 0 □	
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 0	0 0 0 0	
Shift	1 1 1 0 0	0 0 0 □	Second cycle
Add	0 0 0 1 1		
Set q_0	1 1 1 1 1	0 0 0 0	
Shift	1 1 1 1 0	0 0 0 □	Third cycle
Add	0 0 0 1 1		
Set q_0	0 0 0 0 1	0 0 0 1	
Shift	0 0 0 1 0	0 0 1 □	Fourth cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1	0 0 1 0	
		Quotient	
Add	1 1 1 1 1		Restore remainder
	0 0 0 1 1		
	0 0 0 1 0		
	Remainder		

A non-restoring division example.