§5 - MULTIPLIER CIRCUITS

5.1 Introduction

Arithmetic circuits form an important class of circuits in digital systems. In this section, we will examine simple complementers, different types of adder/subtractor circuits and their trade-off between speed and complexity, multiplier circuits and floating point circuits. Arithmetic circuits alone could form a 3rd year course on its own, therefore the treatment here will be selective (but at a reasonably detail level). However, circuits will be treated at gate levels or above, but not at transistor level. Emphasis will be placed in the techniques, algorithms and ideas, not circuit tricks. In this chapter, I assume that you known two's complement representation of numbers and the basic ideas behind binary addition and subtraction. If you have forgotten, consult 1st or 2nd year notes.

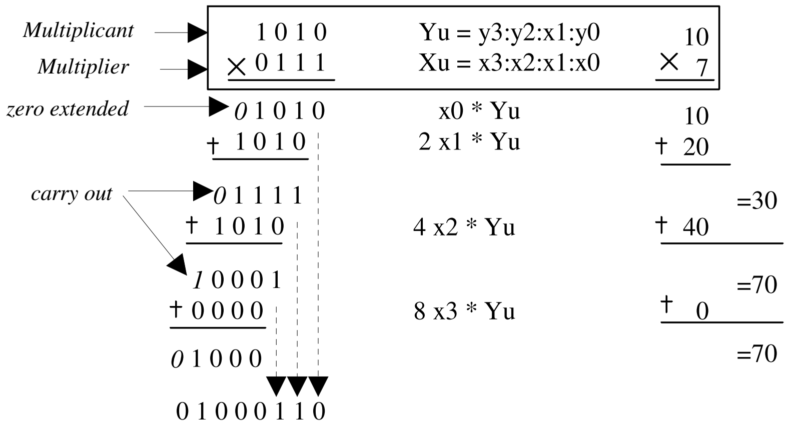
Perhaps the best book on this subject is "*Computer Arithmetics - Principles, Architecture and Design*" by K Kwang. This book covers all the materials discussed in this section and beyond.

Another recently published book in this topic is "*Computer Arithmetic – Algorithms and Hardware Designs*" by Behrooz Parhami. This book covers almost all aspects of computer arithmetics and circuits.

5.2 Multipliers

5.2.1 Unsigned Multiplication

Unsigned multiplication can be done using serial addition as shown below:



At each stage, starting from the LSB of the multiplier, a one-bit multiplier is perform to give a partial product. This is shifted right by one bit before adding to the next bit multiplication. Note that we need 4-bit adder at each stage, yielding 5 bit results. The carry out is used to provide the fifth bit. At each stage, the LSB of the addition becomes one bit of the answer.

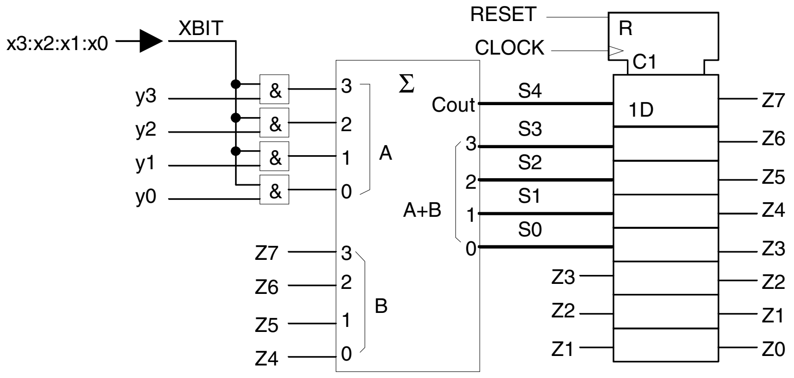


Figure 5.16 4-bit Unsigned Serial-add Multiplier

The operation of this circuit is:

- Reset the Flip-flops
- XBIT = x0; CLOCK ↑
- XBIT = x1; CLOCK ↑
- XBIT = x2; CLOCK ↑
- XBIT = x3; CLOCK ↑
- Answer: Z7:0 = Xu * Yu

At each stage, most significant 4 bits S4:1 are saved and fed back to adder as Z7:4. S0 becomes part of the final answer.

5.2.2 Signed Multiplication

Before considering signed multiplication, let us first examine 4-bit signed addition. Adding two 4 bit numbers gives a 5-bit sum. For an unsigned adder, the carry out provides the fifth bit. Unfortunately, this bit is wrong for signed addition if the sign of the two numbers are different. Therefore, to ensure that a 4-bit signed addition gives the correct 5 bit result, we actually need a 5-bit adder as shown in figure 5.1. The input numbers must first be ***sign extended*** to form a 5-bit signed number before the addition.

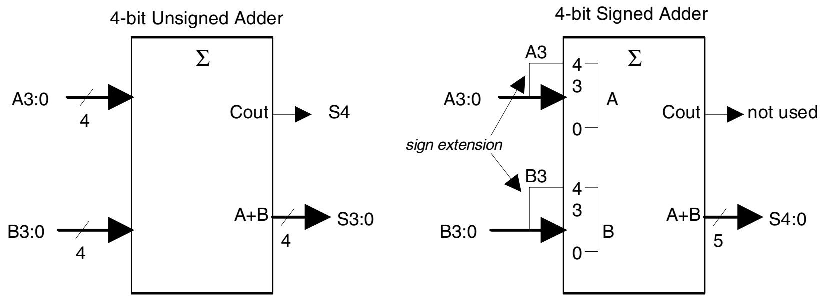
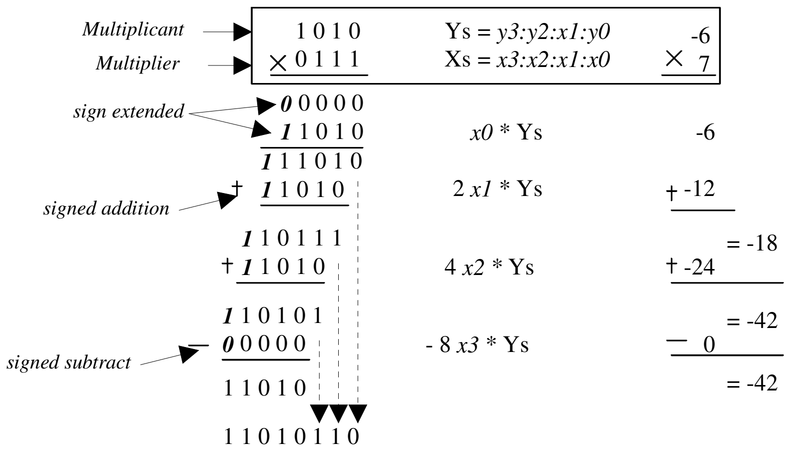


Figure 5.1 Unsigned and Signed Addition

Similar to unsigned multiply, signed multiplication can be done using serial addition as shown below:



The procedure here is similar to that for unsigned multiply except that:

- Each input to the 4-bit sign adder must be sign extended by 1 bit
- The last operation must be a subtraction, not an add.

Therefore the circuit need to use a 4-bit signed adder/subtractor as shown in figure 5.2.

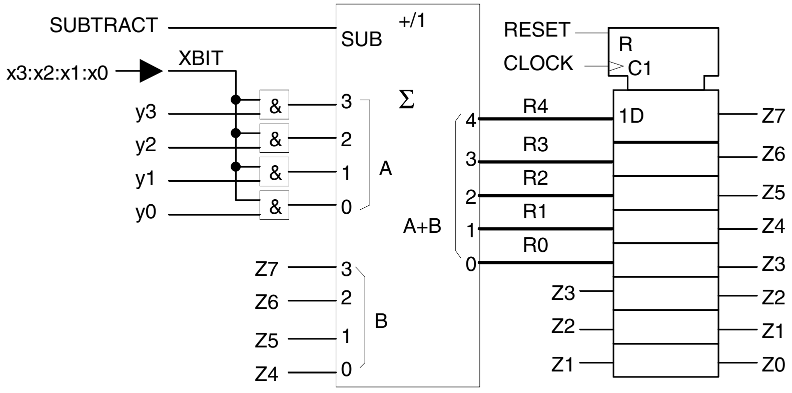


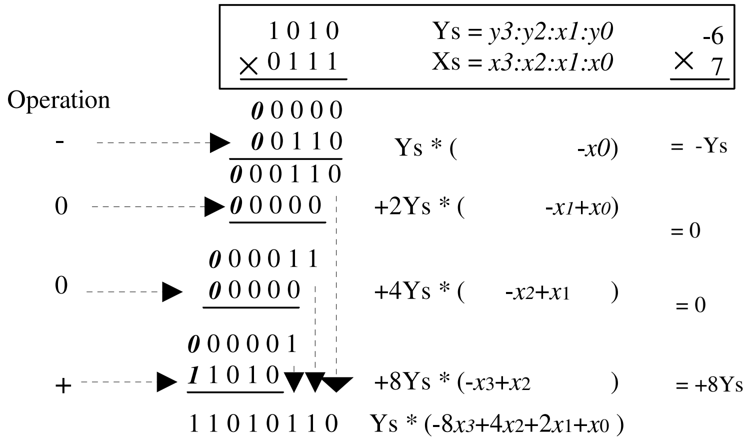
Figure 5.2 4-bit Signed Serial-add Multiplier

The operation of this circuit is:

- Reset the Flip-flops
- XBIT = x0; SUBTRACT=0; CLOCK ↑
- XBIT = x1; SUBTRACT=0; CLOCK ↑

- XBIT = x2; SUBTRACT=0; CLOCK ↑
- XBIT = x3; SUBTRACT=1; CLOCK ↑
- Answer: Z7:0 = Xs * Ys

5.2.3 Recoded Multipliers - Booth Algorithm



Instead of treating the MSB differently from all other bits, it is possible to rearrange the binary bits and code them differently. ***Booth Algorithm*** is one of many algorithms that group together a number of bits in the multiplier and perform a ***recoding*** of the binary bits before the actual addition/subtraction. The table above shows how the Booth algorithm work:

- At each stage, we add Ys * 2^i * $(-x_i + x_{i-1})$
- We assume that $x_{-1} = 0$
- The algorithm is defined by the following equation:

$$\sum_{i=0}^{N-1} 2^i(-x_i + x_{i-1})$$
$$= -\sum_{i=0}^{N-1} 2^i x_i + \sum_{i=0}^{N-1} 2^i x_{i-1}$$
$$= -\sum_{i=0}^{N-1} 2^i x_i + \sum_{i=-1}^{N-2} 2^{i+1} x_i$$
$$= -2^{N-1} x_{N-1} - \sum_{i=0}^{N-2} 2^i x_i + \sum_{i=0}^{N-2} 2^{i+1} x_i + 2^0 x_{-1}$$
$$= -2^{N-1} x_{N-1} + \sum_{i=0}^{N-2} (-2^i + 2^{i+1}) x_i$$
$$= -2^{N-1} x_{N-1} + \sum_{i=0}^{N-2} 2^i x_i$$

x_i	x_{i-1}	$(-x_i+x_{i-1})$	Comments
0	0	0	Do nothing
0	1	+1	Add Ys
1	0	-1	Subtract Ys
1	1	0	Do nothing

The implementation of the Booth Algorithm is shown in figure 5.3. It is very similar to the previous implementation, but we now remove the need to treat the last cycle as special.

An extra register is need to store PREVBIT. It is initialized to 0 (with RESET).
The XOR gate and the AND gates are used to implement the above coding table.

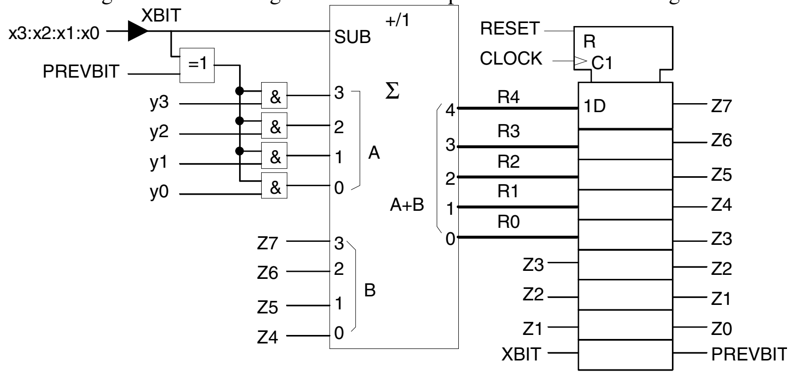


Figure 5.3 Booth Serial Multiplier

5.2.4 Multi-bit Multiplier

All the circuits considered so far handle only one bit multiplication at each clock cycle. There are no reasons why we could not deal with two (or more) bits at a time. Shown in figure 5.4 is a 2-bit serial multiplier for unsigned numbers:

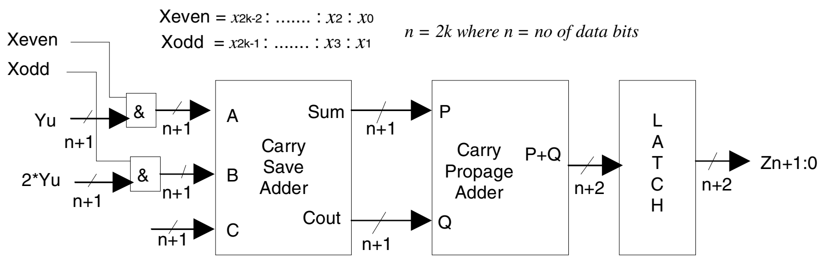


Figure 5.4 2-bit serial multiplier

5.2.5 Modified Booth Algorithm

We can combine the multi-bit idea with the original Booth algorithm as shown below:

Booth Algorithm: bit i $2^i(-x_i + x_{i-1})$

bit i+1 $2^{i+1}(-x_{i+1} + x_i)$

Modified Booth: bit i & i+1 $2^i(-x_i + x_{i-1}) + 2^{i+1}(-x_{i+1} + x_i)$

$= 2^i(-2x_{i+1} + x_i + x_{i-1})$

x_{i+1}	x_i	x_{i-1}	$-2x_{i+1}+x_i+x_{i-1}$	Comments
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Therefore we need a circuit that can add/subtract either 0, Ys or 2*Ys. Such a circuit can be implemented as shown in figure 5.5.

- The multiplex chooses either to double Y3:0 or not.
- R5:0 = X3:0 ± k * Y3:0, where k=1 or 2.
- We need 6 bit adder output to accommodate the answer without overflow.
- X3:0 is sign-extended to make it 5 bits, the same as kY.

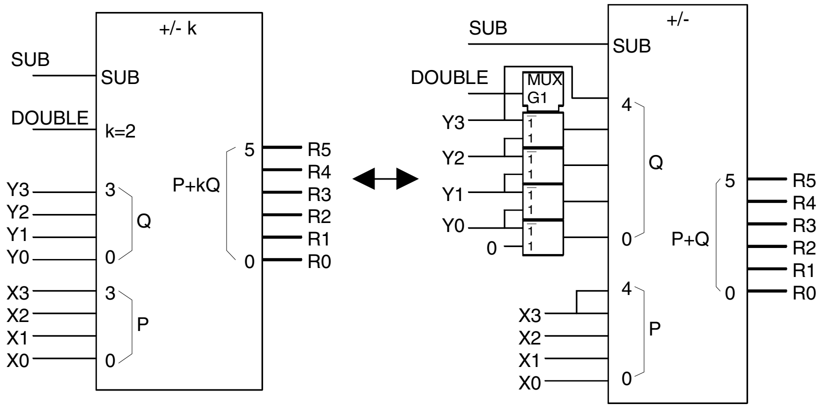


Figure 5.5 Scaling Add/Subtract Circuit

The circuit to implement the modified Booth algorithm is shown in figure 5.6. This circuit basically implement the following function:

XBIT1	XBIT0	PREVBIT	Mult. factor
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

We can derive the following equations from the above table:

$$SUB = XBIT1$$

$$DOUBLE = \overline{XBIT0 \oplus PREVBIT}$$

$$ZERO = (XBIT1 = XBIT0 = PREVBIT)$$

$$= \overline{XBIT1 \oplus XBIT0} \bullet DOUBLE$$

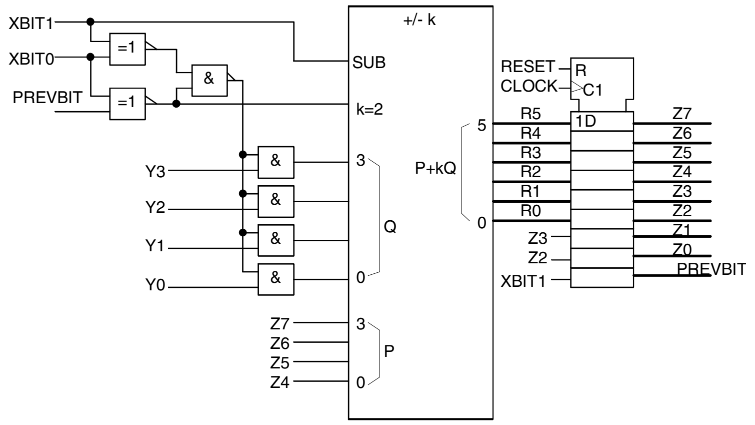


Figure 5.6 Modified Booth Multiplier Circuit

The functional sequence for this circuit is:

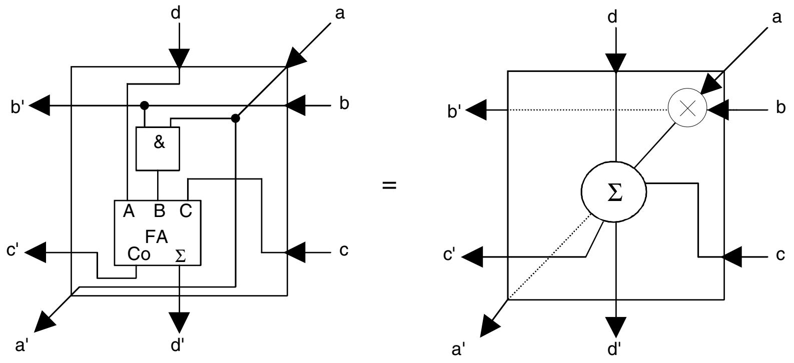
- RESET
- XBIT1:0 = X1:0; CLOCK ↑
- XBIT1:0 = X3:2; CLOCK ↑

5.2.6 Array Multiplier

For fast multiplication, an array of 1-bit multipliers (AND gate) and full-adders can be used. First, let us examine a 4x4 unsigned product:-

$$\begin{array}{ccccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 & = A \\
 & & & X & b_3 & b_2 & b_1 & b_0 & = B \\
\hline
 & & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 & \\
 & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & & \\
 & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & & \\
+ & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 & P = A \times B
\end{array}$$

Next, let us design a 1-bit multiply cell as shown below (figure 5.7). Each cell has a full-adder and an AND gate. The signal feed through shown here helps to build a multiplier as a 2-D array:-



$$a' = a$$

$$b' = b$$

$$c' = a \cdot b \cdot d + a \cdot b \cdot c + c \cdot d$$

$$d' = (a \cdot b \oplus d) \oplus c$$

Figure 5.7 One bit Multiply Cell

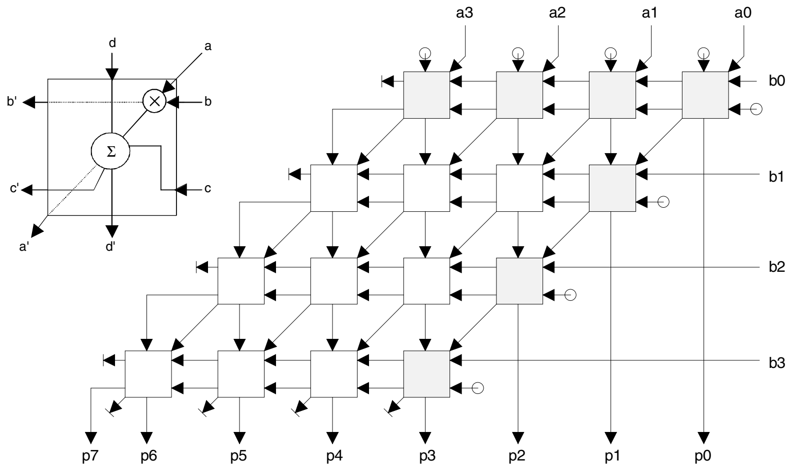


Figure 5.8 4 x 4 bit unsigned multiplier array

We can map the tabulated form of the 4x4 multiplication almost directly into the 2-D array shown in figure 5.8. However, this array is slow. Assuming Δ is the worst case delay per cell, the total worst case delay of the array is 10Δ. For a n x n bit array, the worst case delay can be shown to be (3n-2)Δ. (Why? Work it out yourself!)

Note also that the shaded cells are simpler - they only need a half adder instead of a full adder circuit.

This array could be modified to work much faster by apply the carry-save principle. Instead of propagating the carry length-wise, we could feed it forward diagonally as in the carry-save adder circuit. This results in a better circuit as shown in figure 5.9.

Now the worst-case delay is only 2n,Δ which is much better than (3n-2)Δ (especially when n is large).

This circuit can further be simplified (hence making it works faster) by replacing the shaded cells with simpler circuits as shown in figure 5.10.

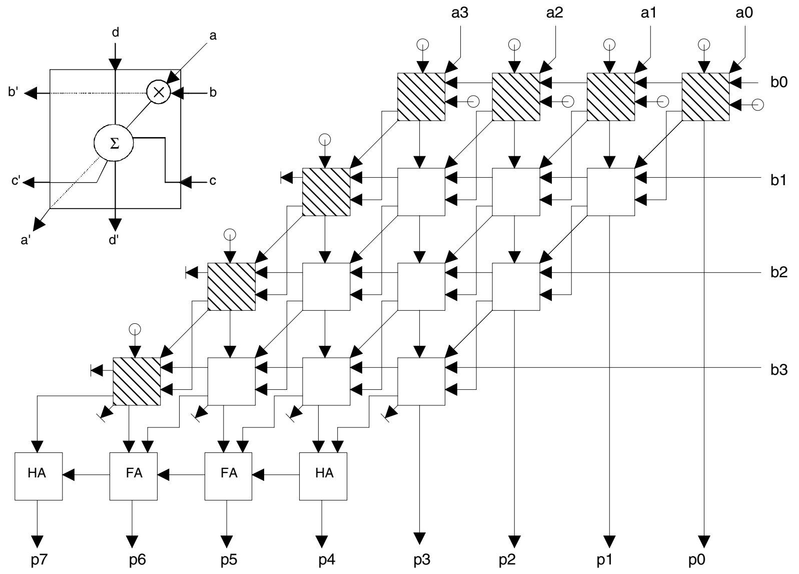


Figure 5.9 Carry-save 4x4 unsigned multiplier array

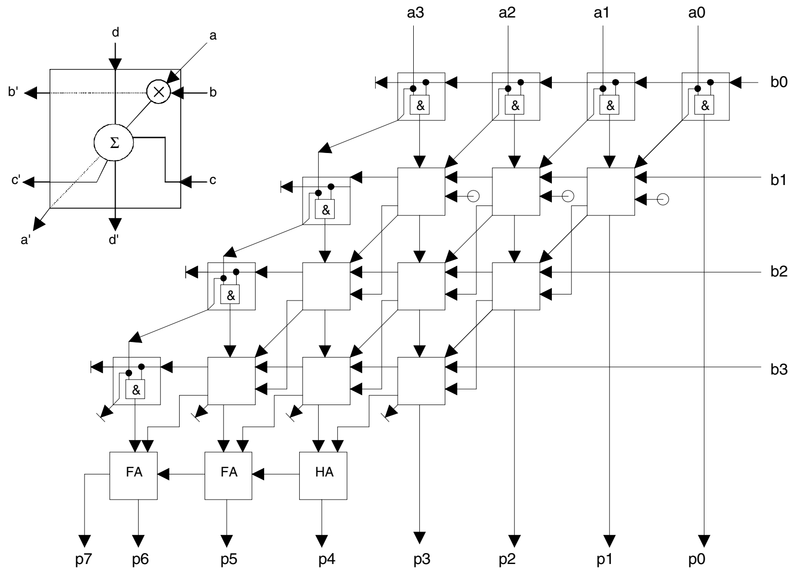


Figure 5.10 Simplied carry-save unsigned multiplier array

5.2.7 2's Complement Multiplier Array

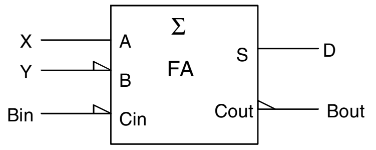
To handle 2's complement signed multiplication, one of many algorithms is depicted in the table below. As before, let us consider 4 bit numbers. The lower 3 bits are multiplied as in the unsigned case. The sign bits are handled separately. The bits with negative weighting are enclosed in parenthesis. They are ***subtracted*** instead of added.

				(a_3)	a_2	a_1	a_0	= A
			X	(b_3)	b_2	b_1	b_0	= B
					a_2b_0	a_1b_0	a_0b_0	
				a_2b_1	a_1b_1	a_0b_1		
+	a_3b_3	0	a_2b_2	a_1b_2	a_0b_2			
		(a_3b_2)	(a_3b_1)	(a_3b_0)				
+		(a_2b_3)	(a_1b_3)	(a_0b_3)				
(p_7)	p_6	p_5	p_4	p_3	p_2	p_1	p_0	P = A x B

In order to build an array for this multiplication, we need ***subtract*** cells. The truth table of a subtract cell is:-

X	Y	Bin	D	Bout
0	0	0	0	0
0	1	0	1	1
1	0	0	1	0
1	1	0	0	0
0	0	1	1	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	1

A subtract cell can be built from a full adder cell as shown in figure 5.11.



$$D = (x \oplus \overline{y}) \oplus \overline{B}_{in}$$

$$\overline{B}_{out} = x \cdot \overline{y} + x \cdot \overline{B_{in}} + \overline{y} \cdot \overline{B}_{in}$$

Figure 5.11 One bit subtractor with borrow

Therefore, the subtract cells in the bottom two rows of the multipliers become (figure 5.12):-

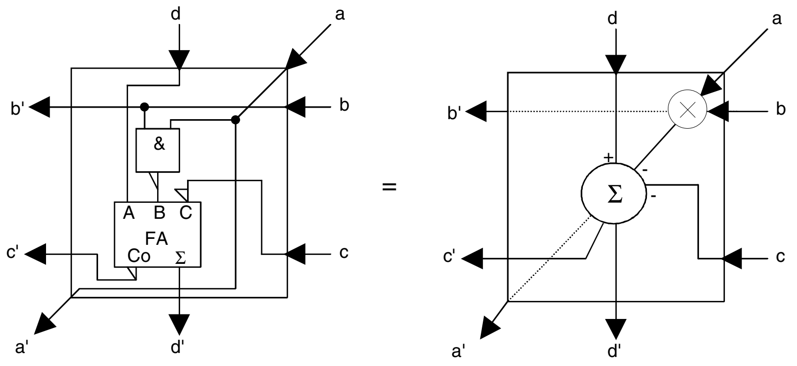


Figure 5.12 Multiplier cell with subtraction

The detail arrangement of the array is left as an exercise for you.