Chapter – 3

Partial Product Accumulation

Previous chapter discussed the most common methods for generating partial products. After n partial product rows are generated, they must be added to obtain the final product in the form of sum row and carry row. A simple method is to use stages of carry-propagate adders. But this is very time consuming. To avoid this and reduce the latency, different accumulation schemes are developed.

In this chapter we will discuss mainly Wallace tree method [Wallace 1964] and array method [Baugh 1973, Blanken. 1974] for accumulation of partial product rows. In Wallace tree we will use 3:2 compressors [Shams 2002, Bui 2002], 4:2 compressors [Shen 1978] and RB adders in adder trees [Makino 1996]. In all the cases the latency in terms of normalized gate delay with respect to two input XOR gate is found out. The normalized gate delays are taken from Toshiba library in a systematic manner, which is already summarized in Table 1.2 of chapter 1. As our objective is to analyze various accumulation methods, we have considered that, the partial product rows are available in NB form generated using radix-4 Booth's encoding method or in RB form generated using radix-64 Booth's encoding method, and we will explore the different techniques of accumulation. For a 16x16 multiplication we will do the systematic delay calculations for all the accumulation methods. This systematic method is used to summarize the delay for all other lengths of multiplications. Finally a comparison in terms of normalized gate delay is given for different accumulation methods.

3.1 Array multiplier

Array multipliers [Baugh 1973, Blanken. 1974] are best suited for digital signal processing applications, which need faster computations as well as regular layout and simpler interconnections. In this, wires run only to neighboring cells, and the cells fit nicely into a general data path layout. This simplicity costs a larger number of gates in critical path in comparison to tree structures of accumulation.

In the present work the conventional array multiplier architecture is modified based on the delay analysis of the basic adder unit. By this modification, delay in partial product accumulation of multiplication decreases in comparison to the conventional array multiplier. For a typical 64-bit multiplication the delay advantage will be $15T_{XOR.}$

As the delay analysis of full adder is the basis of architectural improvement, it is discussed first. Then a simple multiplication scheme using Booth encoding is explained. A conventional array multiplier and the delay optimized array multiplier for 16-bit multiplication are discussed and the delay analysis for different sized multiplications is done.

3.1.1 Delay analysis of full adder

The gate level circuit of a standard full adder is shown in Fig. 3.1. For the analysis of delay we will use the normalized gate delays with respect to XOR gate obtained from Toshiba library and summarized in Table 1.2 of chapter 1. From Fig. 3.1 it can be seen that the delay from A, B to Sum is twice that of Cin to Sum. Again the delay from A, B, Cin to Cout is equal to 231 ps, which is almost half that of A, B to Sum. The full adder delay summary is given in Table 3.1. This delay analysis will be used in the multiplier architecture.

Table 3.1 Delay summary of full adder

From	To	Normalized delay In terms of T_{XOR}
A, B	Sum	2
Cin	Sum	1
A, B, Cin	Cout	0.788

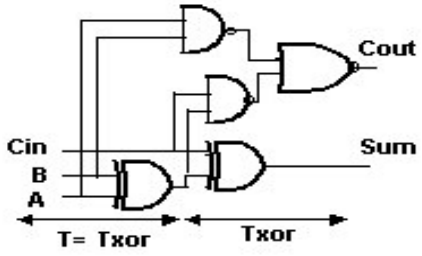


Fig. 3.1 Full adder.

3.1.2 16x16 bit multiplication using Booth algorithm

The multiplication of two 16-bit numbers is shown in Fig 3.2. Here the multiplier is partitioned into overlapping groups of three bits and each group is encoded to a signed digit using radix-4 partial product selection rule described in Table 2.1. This use of Booth's algorithm reduces the number of partial product rows from 16 to only 8. These partial product rows are accumulated to finally one sum row and one carry row. These two rows are then added using a carry-propagate adder (CPA), which will be discussed in chapter 4.

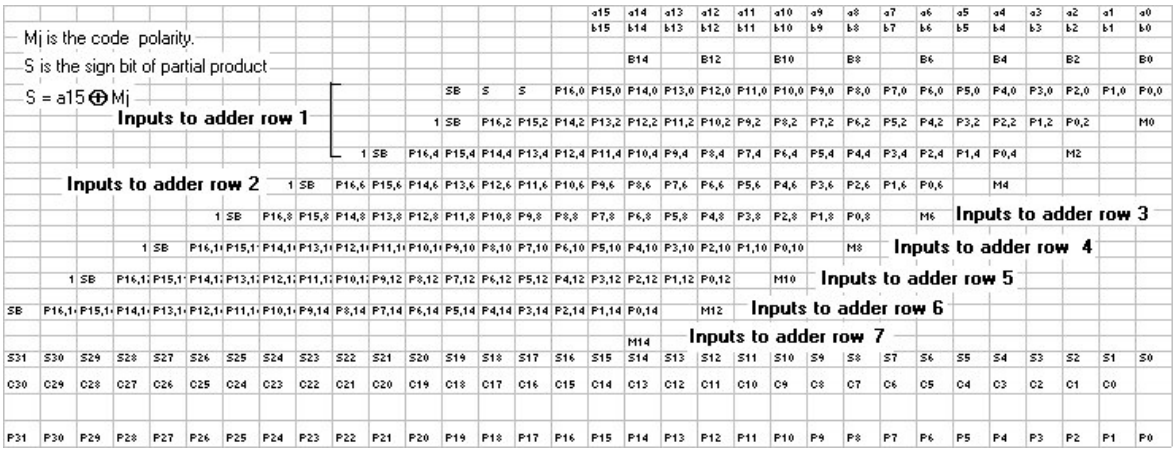
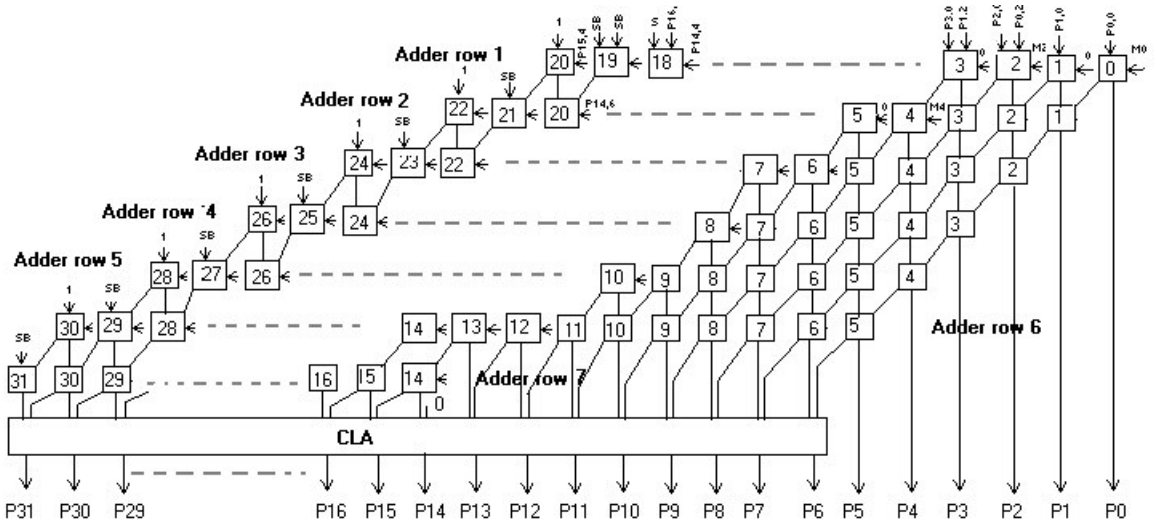


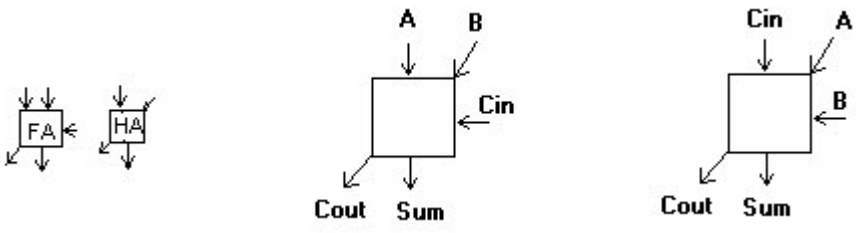
Fig. 3.2. Multiplication of two 16-bit signed binary numbers using radix- 4 Booth encoding.

3.1.3 Conventional and delay optimized array multiplier

First, we will consider the performance issues in a conventional array multiplier [Wang 2003]. Fig. 3.3 (a) depicts a 16x16 size array multiplier. The full adders and the half adders used in the architecture are shown in Fig. 3.3(b). Here all the adder cells used have vertical input A as shown in Fig. 3.3(c). In this the sum signals from the adder cells of row i and column j are fed into row i + 1 of the same column. Thus, to cross each row the signal takes $2T_{XOR}$ time. In general for an n-bit multiplication there are n/2 rows of partial products and an additional row for the Mj bit of last partial product row. The accumulation of n/2 +1 rows is shown in Fig 3.3(a). Here the first row of adders adds the first three partial product rows. So the total number of adder rows needed are n/2 +1 – 2 = n/2 – 1. So the delay in partial product accumulation for the conventional array multiplier is (n/2 –1) * $2T_{XOR}$.



(a)



(b) (c) (d)

Fig. 3.3 (a) Architecture of conventional and delay optimized array multiplier for 16-bit operand multiplication (b) The full adder and half adder used in multiplier (c) Cell structure used in conventional array multiplier (d) Cell structure used in delay optimized array multiplier.

In this conventional multiplier, the sum path, which is the slowest is responsible for the delay path. In the present work the basic idea used is to make proper connections globally so that the delay through out each sum and carry path is approximately the same. For this the long delay path originating from the previous adder (Sum) is connected to the short delay path of next one (i.e. carry-in of full adder). Thus the architecture of the multiplier is essentially similar to the conventional one, but the adder cells used have vertical inputs of carry-in (Cin) as shown in Fig. 3.3(d). It is interesting to note that only this simple replacement optimizes all interconnections.

An example of speed improvement in the 14^{th} column of the above array multiplier is shown in Fig. 3.4. In this, the adder in the 14^{th} column of first row gives the sum and carry output at $2T_{XOR}$ and T_{XOR} delay. Now this adder sum and the carry from 13^{th} row is given as input to the adder of 14^{th} column and second row. This adder is connected as shown in Fig. 3.3(d). So both the sum and carry outputs of this adder are obtained in $3T_{XOR}$ time delay as shown in Fig. 3.4. Because of this type of adder connections in place of the conventional connections, the delay is decreased from $4T_{XOR}$ to $3T_{XOR}$ for every pair (two stages) of adder rows.

In this array multiplier, the number of adder rows will be the same as that in the conventional array multiplier i.e. n/2 - 1. So the delay will be $(n/2 – 1)/2 * 3T_{XOR}=[(3n – 6)/4] * T_{XOR}$ if n/2 –1 is even. If n/2 –1 is odd, the delay will be $(n/2 – 2)/2 * 3T_{XOR} + 2T_{XOR} = [(3n – 4)/4] * T_{XOR}$. In the present example number of stages is odd (n/2 – 1 = 16/2 – 1 = 7), so the delay is $11T_{XOR}$. The stage-by-stage detail of delays is shown in Fig. 3.4.

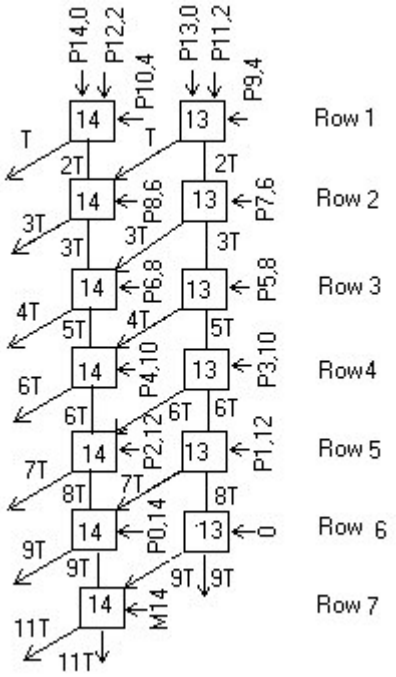


Fig. 3.4. Accumulation of partial products along with delay involved in every row for a delay optimized array multiplier. Here $T = T_{XOR}$

3.1.4 Delay analysis

The architectural comparison of the conventional array multiplier with the delay optimized array multiplier for different bit-widths is given in Table 3.2. A graph of operand size and multiplier delay in terms of XOR gate delay is shown in Fig. 3.5. This shows that with the same hardware, the delay-optimized architecture is faster for all operand sizes. The delay advantage also increases as the operand size increases.

Table 3.2 Architectural comparisons of array multipliers

Adder Type Comparison basis	Conventional	Optimized interconnection
Delay in partial product accumulation step for n-bit multiplication	$(n/2 - 1) * 2T_{XOR}$	$(3n - 4)/4 * T_{XOR}$ for $n/2 - 1$ is odd. $(3n - 6)/4 * T_{XOR}$ for $n/2 - 1$ is even.
Number of adder rows required in partial product accumulation step of multiplication for n-bit multiplication	$n/2 - 1$ row of adder cells.	$n/2 - 1$ row of adder cells.

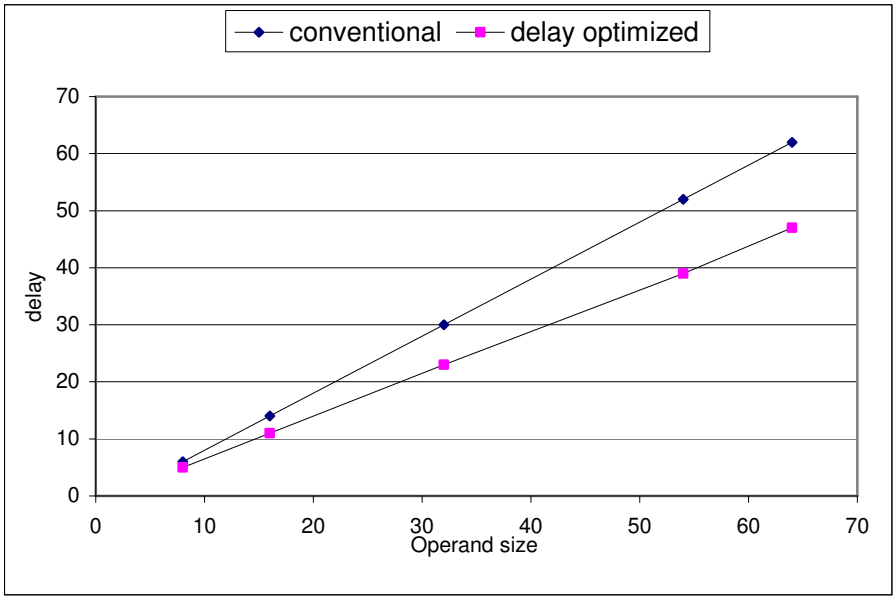
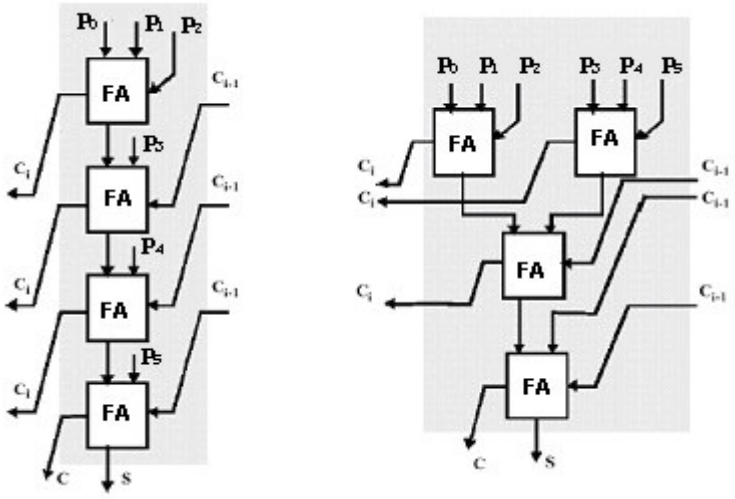


Fig. 3.5 Operand size vs delay (in terms of T_{XOR}) of conventional array multiplier and delay optimized array multiplier.

3.2 Wallace tree

A better way to organize the carry save adders and reduce operation time is in the form of a tree introduced by Wallace [Wallace 1964] and known as Wallace tree. Dadda [Dadda 1965] then introduced the term (n, m) counter (or compressor) and showed the reduction in delay of adder chain. Various researchers then developed (3:2), (4:2) and other high order compressors [Hsiao 1998, Parhi 2001, Song 1991, Wein. 1981]. A simple example of reduction in delay

path is given in Fig. 3.6. Here six partial products i.e. P_0, P_1, P_2, P_3, P_4, P_5 present in a single column are added using an array architecture to give final sum and carry as shown in Fig. 3.6(a). This needs a delay of four-stage adders. Now arranging adders using the Wallace tree method as shown in Fig. 3.6(b), the delay path reduces to three adder delays. In the present work use of 3:2 compressors and 4:2 compressors in Wallace tree arrangements is discussed.



(a) (b)

Fig. 3.6. Accumulation of six partial products using (a) Array and (b) Wallace tree method.

3.3 Partial product accumulation using 3:2 compressor in Wallace tree

In this method of accumulation, a full adder is used as 3:2 compressor. A 3: 2 compressor compresses three inputs a, b, c to produce two outputs, sum and carry. Here carry is not propagated but is given to the next level of adder. This reduces the delay path. In a 16 x 16 bit multiplication using radix-4 Booth encoding the 14^{th} column is of maximum length (as seen in Fig 3.2) and accounts for maximum delay. The accumulation of this 14^{th} column using Wallace tree method with 3:2 compressors is shown in the Fig. 3.7. In this the carries generated in a specific level (let 1^{st} level) of one column with bit value 2^i is given to carry-in of a 3:2 compressor present in the next level (2^{nd} level) of next column with bit value 2^{i+1}. By this, no carry propagation takes place and the critical delay path reduces. In the same way, the

worst-case delay for different lengths of multiplication using Booth encoding is summarized in Table 3.3. In this table, the number of partial products is given by:

Number of partial products = n/2 + 1, where n is the multiplier size.

Here the number of partial product rows is reduced to half i.e. n/2 as we are using radix–4 Booth's encoding and the additional term +1 is because of the sign bit, which is to be added to LSB of $(n/2)^{th}$ partial product row.

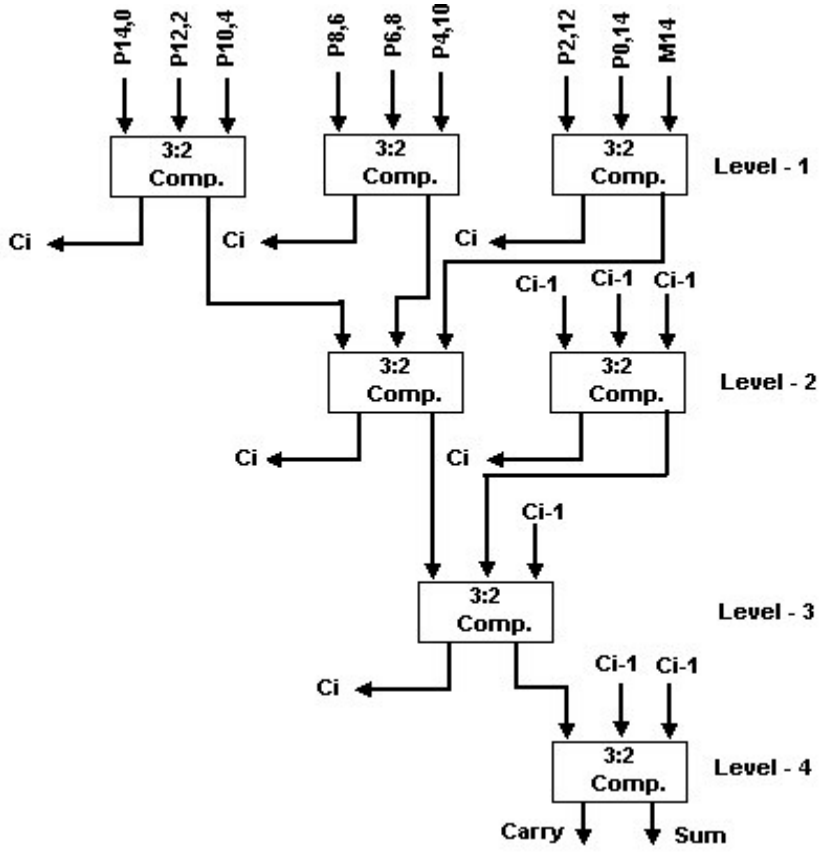


Fig. 3.7 Partial product accumulation of 14^{th} column in a 16x16 multiplication.

Table 3.3 Summary of worst-case delay for partial product accumulation using 3: 2 compressors in Wallace tree

Number of partial products in a column	Number of levels of adder	Delay in terms of XOR gate
≤3	1	2
≤4	2	4
≤6	3	6
≤9	4	8
≤13	5	10
≤19	6	12
≤28	7	14
≤42	8	16
≤63	9	18

3.4. Partial product accumulation using 4:2 compressor in Wallace tree

3.4.1. 4:2 compressor

A 4:2 compressor has five inputs including a carry-in from the neighboring column of one binary bit lower significance. It has three outputs including a carry-out to column of one binary bit of higher significance at same level. This consists of two 3:2 compressors connected in series. General connection of these two 3:2 compressors will give a critical delay of 4 XOR gates [Santoro 1989a]. But optimized interconnection [Goto 1992] as shown in Fig. 3.8(a) will give a critical delay of 3 XOR gates. This will help to reduce the final latency in accumulation stage.

Use of a 4:2 compressor in Wallace tree can reduce the delay path. Besides, improving the power efficiency of such designs can lead to significant savings in the power consumption of the entire multiplier. For accumulating n rows of partial products to give only two rows, we have to use $k=\log_2 (n/2 + 1)$ stages of 4:2 compressors. Thus a delay advantage in 4:2 compressor will result in k times delay advantage in the multiplier. Because of these factors we have proposed two new compressors which are made of less number of transistors and their performances are compared under the same simulation conditions for a fair comparison with the reported ones in [Goto 1997, Ohkubu 1995] and standard CMOS implementation.

The first circuit we have considered is CMOS 4:2 compressor using two full adders as shown in Fig. 3.8(b). This is selected for comparison, because the input signals are only connected to the gates of CMOS circuits. It is robust against voltage and transistor scaling and supports reliable operation at low voltages and minimal transistor sizes. CMOS also has a higher noise margin due to the presence of a static path that restores the correct logic state in the presence of noise. But the drawback of CMOS circuit is the existence of P block with its low mobility (μ_p) devices compared to NMOS devices (μ_n). Therefore, PMOS devices need to be sized up to attain the gate performance. The input capacitance of a CMOS gate is large, because each input is connected to the gate of at least a PMOS and a NMOS transistor. This degrades the speed. Usually the best gate performance is achieved with a PMOS/NMOS width ratio of $\sqrt{\mu_p / \mu_n}$.

In the second circuit in Fig. 3.8(c) we used pass transistor multiplexer for 4:2 compressor [13]. This circuit operates at a higher speed than the CMOS gate based circuit, because of the pass transistor-based design.

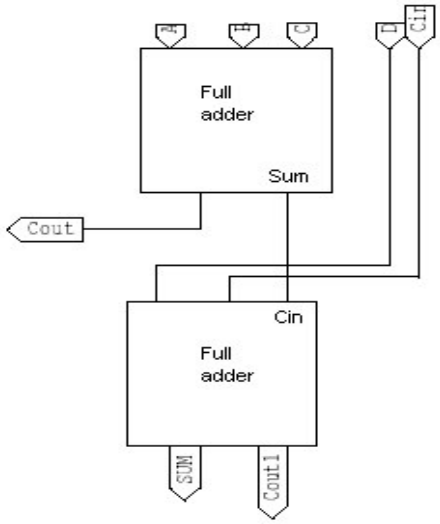
The third *new 4:2 compressor* as shown in Fig. 3.8(d) is based on static energy recovery full adder (SERF) circuit [Bui 2002, Shalem 1999]. The XNOR gate used here has no direct path to ground, which reduces the power consumption. Here the charge stored at load capacitance is reapplied to the control gates. The combination of not having a direct path to ground and re-application of the load charge to the control gate makes the energy recovery full adder an efficient circuit and this can be implemented with only 32 transistors.

The fourth circuit is a DB 4:2 compressor [Shams 2002]. It has no direct path between the power supply and the ground, which eliminates direct short circuit current, but sneak path from previous driving stage output still exists. Though the input is given as A and B

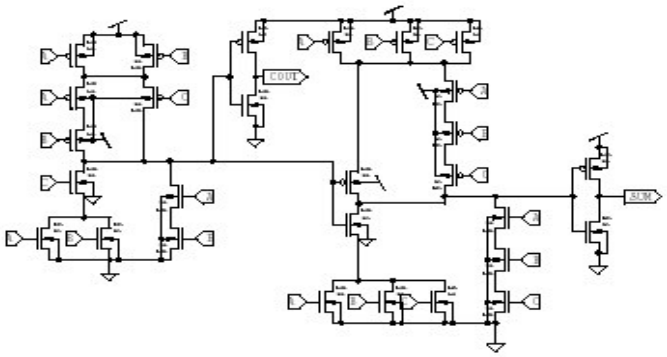
internally, it acts as complementary pass transistor logic to give H and H' which are used to generate sum and carry. This circuit also uses only 32 transistors and results in reduced delay and power consumption. To verify and compare the performance of above circuits we have used the same method as used in section 2.3 for performance comparison of Booth's encoders and selectors. Here five input patterns C1, C2, C3, C4, C5 are formed from inputs I1, I2, I3, I4, I5 as shown in Fig. 3.8(f). Each pattern is simulated 5 times using frequency ranges 2MHz, 5 MHz, 10 MHz, 15 MHz, 20 MHz. All input signals have a rise time and a fall time of 500ps. Thus for each circuit 25 TSPICE simulations are run and in each case the rise time delay and fall time delay for sum are noted down and the maximum delay and average power consumption is reported for the circuits in the same way as is done in section 2.3. The channel length is scaled down and the simulation results in normalized form are noted which are presented in Table 3.4, along with transistor count and energy delay product. With scaling down of channel length the delay, power and energy delay product (EDP) are plotted in Fig. 3.9. Fig. 3.9(c) indicates that the EDP of multiplexer implementation and DB implementation are almost the same. But in terms of transistor count DB uses only 50% of transistors used in multiplexer implementation. Also the transistor count in DB is only 80% of that reported in [Goto 1997].

Table 3.4 4:2 Compressor Comparison

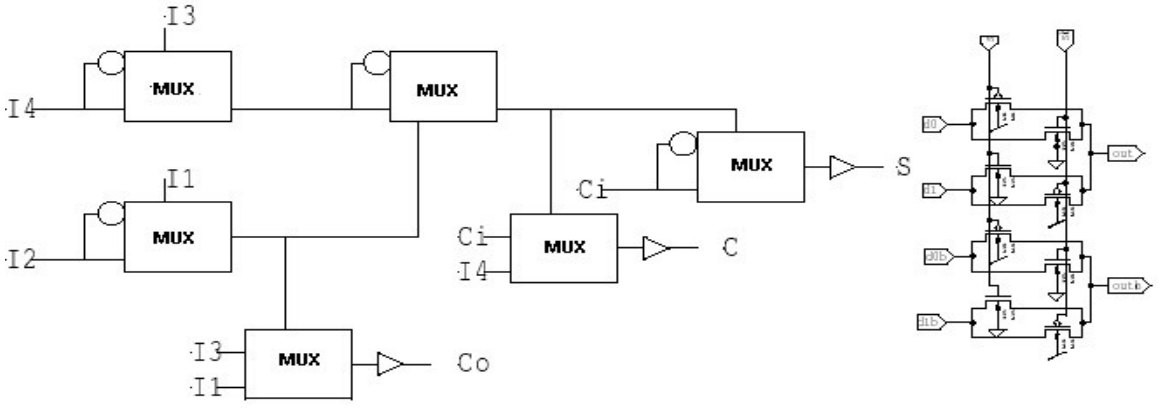
4:2 Comp-ressor type	No of tr. Used	Delay (Normalized)						Power(Normalized)						EDP(Normalized)					
		Channel Length in micron						Channel Length in micron						Channel Length in micron					
		1. 2	1. 3	1. 4	1. 5	1. 6	1. 7	1. 2	1. 3	1. 4	1. 5	1. 6	1. 7	1. 2	1. 3	1. 4	1. 5	1. 6	1. 7
CMOS	56	3.10	3.21	3.29	3.38	3.47	3.56	1.03	1.04	1.06	1.07	1.09	1.11	3.19	3.35	3.49	3.64	3.81	3.98
MUX	68	1.00	1.03	1.07	1.11	1.15	1.19	1.79	1.80	1.85	1.88	1.91	1.96	1.79	1.86	1.98	2.09	2.21	2.35
SERF	32	2.69	2.76	2.83	2.91	2.99	3.08	1.23	1.24	1.26	1.27	1.29	1.31	3.32	3.44	3.57	3.72	3.88	4.04
DB	32	1.85	1.89	1.93	1.97	2.02	2.06	1.00	1.00	1.01	1.02	1.03	1.05	1.85	1.91	1.97	2.03	2.09	2.17


A
B
C
Ci1
Full
adder
Sum
Cout
Cin
Full
adder
SUM
Cout1

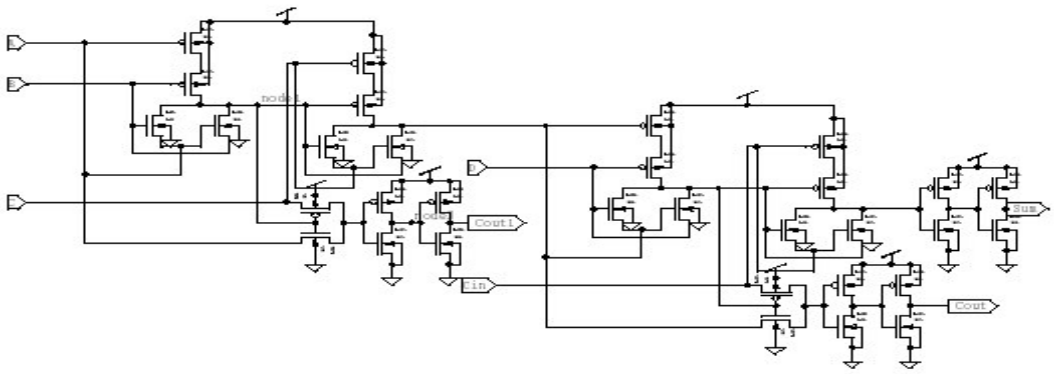
(a)



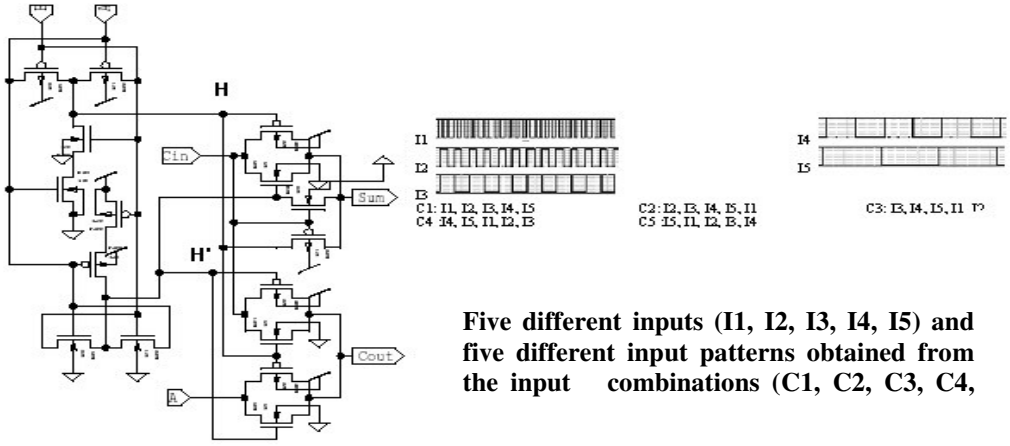
(b)


I3
MUX
I4
MUX
I1
MUX
I2
Ci
MUX
S
Ci
MUX
C
I4
I3
MUX
Co
I1

(c)



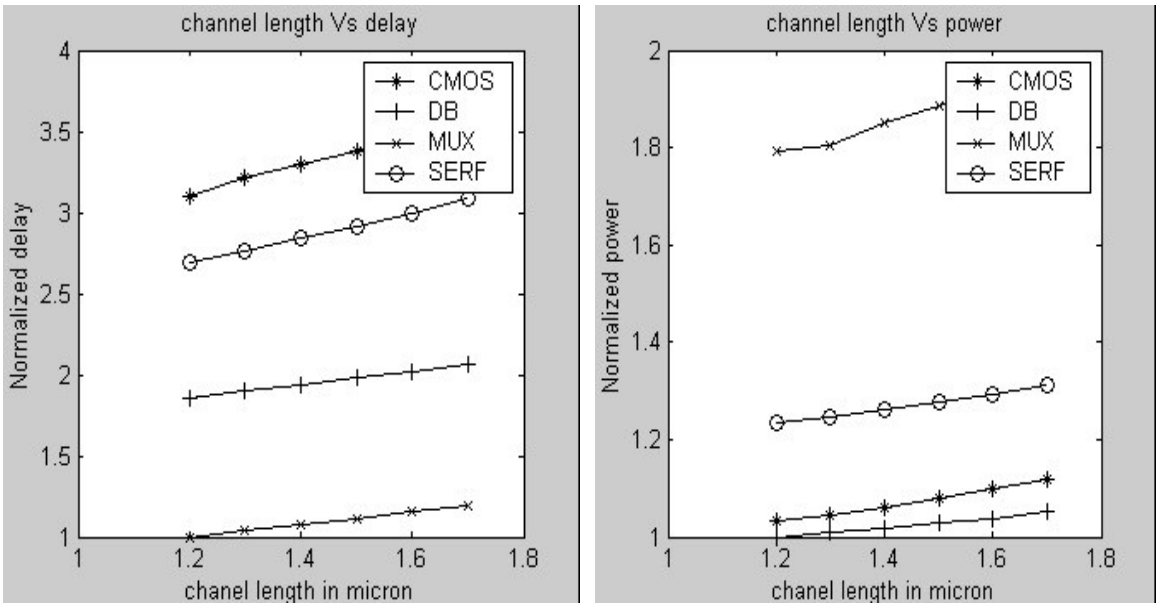
(d)



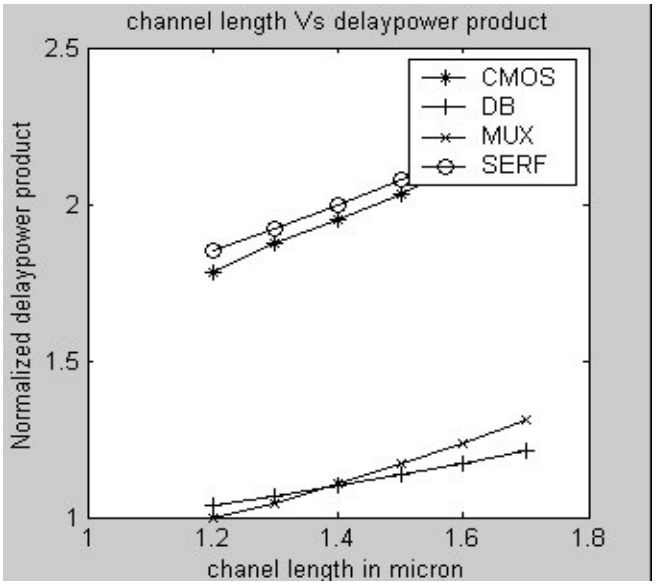
Five different inputs (I1, I2, I3, I4, I5) and five different input patterns obtained from the input combinations (C1, C2, C3, C4,

(e) (f)

Fig. 3.8. Various 4:2 compressors and inputs given to them (a) General 4:2 Compressor using full adder (b) CMOS adder used in 4:2 compressor (c) MUX 4:2 compressor (d) SERF 4:2 compressor (e) DB adder used in 4:2 compressors (f) Input combinations for 4:2 compressors.



(a) (b)



(c)

Fig. 3.9 (a) Channel length vs normalized delay (b) Channel length vs normalized power (c) Channel length vs normalized delay power product.

3.4.2 Use of 4:2 compressors in Wallace tree

The analysis of 4:2 compressor of Fig 3.8(a) shows that, out of the five inputs, three inputs are needed at t=0 and the fourth input and carry is required after one XOR delay. This is because, giving these two inputs at t=0 will not increase the speed. Similarly out of three outputs, one

carry output will be available after one XOR delay. Taking this into consideration the carry-out (Cout) of a 4:2 compressor of one level (let 1st level) of one column with bit value 2^i is given to carry-in of a 4:2 compressor present in the same level of next column with bit value 2^{i+1}. Here the carry-out (Cout) is not dependant on carry-in(Cin). So there will not be any carry propagation. But the other carry Cout1 is given as input to the 4:2 compressor present in the next level (2nd level) of next column with bit value 2^{i+1} and sum also to the next level (2nd level) of 4:2 compressor but present in the same column. Using this mechanism of interconnection the 4:2 compressor tree for accumulating the partial products of 14th column in a 16x16 multiplication is shown in the Fig. 3.10. This indicates that the worst-case delay of this partial product accumulation tree will be that of three 4:2 compressors. The number of levels [35] of 4:2 compressors for different operand sizes and corresponding worst-case delays are given in Table 3.5.

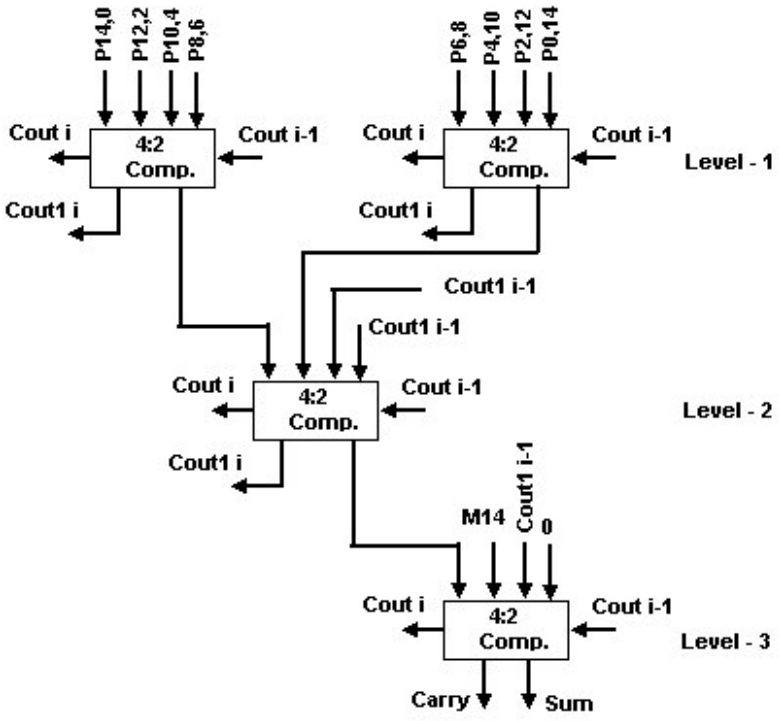


Fig. 3.10 Partial product accumulation of 14th column in a 16x16 multiplication using 4:2 compressors.

Table 3.5 Summary of worst-case delay for partial product accumulation using 4: 2 compressors in Wallace tree

Number of partial products in a single column	Number of levels of adder	Delay in terms of XOR gate
≤4	1	3
≤8	2	6
≤16	3	9
≤32	4	12
≤64	5	15

3.5 Partial product accumulation using RB adder in Wallace tree

Redundant binary number system is a signed-digit number representation system proposed by Avizienis [Avizienis 1961] in 1961. After him various researchers have added their contributions to increase the speed of multipliers using RB number system [Harata 1987, Kuninobu 1987, Edamatsu 1988, Atkin 1970, Takagi 1984, Yannick 2000]. Takagi [Takagi 1985] has given an addition algorithm to add two n-digit RB numbers to give an RB number in constant time independent of operand size without carry propagation. Hiroshi [Makino 1996] has proposed a simple method of RB partial product generation based on methods proposed by [Kuninobu 1987, Edamatsu 1988] just by inverting one of the two partial products and adding (0,1) to the lowest digit. Using this simple method of RB partial product generation from NB partial products and accumulating these using carry free adder makes the RB multiplier faster than some of the conventional NB multiplier implementations. So the different choices of RB adder circuits used in design are discussed. Then, the use of these RB adders in partial product accumulation of a 16x16 bit multiplication is shown. The worst-case delay analysis for this 16x16 bit multiplication is done and the same is extended for different operand sizes. The worst-case delay for different operand sizes in partial product accumulation using RBA tree is given in Table 3.7.

3.5.1 Example of use of RB addition

Multiplication of two numbers A = 11111101= - 3 (multiplicand) and B = 01010010 = 82 (multiplier) is shown in Fig. 3.11. The multiplier is extended by one bit on the LSB side. Then it is partitioned into overlapping groups of three bits as shown in Fig. 3.11 and each group is assigned a digit value as per Table 2.1. Now corresponding to each digit one partial product row is generated. Each partial product row is in NB form. The first and second rows of partial products are added to give a RB partial product PPRB0 using equation 1.1 just by grouping the first row (PP0) and inverted form of second row (PP1). To get the actual addition result we have to subtract 1 (or add –1) at 2^0 positions, which will be done by adding one additional row of control bits at the end. Similarly the third and fourth rows are added by grouping using equation 1.1 to give PPRB1. In this also we have to add –1 in the LSB bit position (in the 2^4 position). The addition of –1 in 2^0 and 2^4 positions is done by adding an additional row of control bits at the end as shown in the figure (Fig. 3.11). Addition of these three rows is done by using carry free addition rule discussed in Table 1.4 to give the result in RB form. Finally RB to NB converter can be used to get the result in NB form.

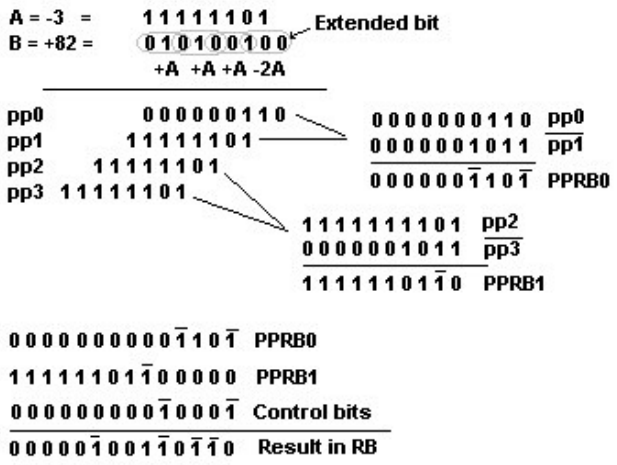


Fig. 3.11 Example of partial product accumulation using RB adder.

For using carry free RB adder in partial product accumulation, three steps are required. In the first step, Booth's encoding reduces the number of partial product rows from n to (n/2 +1). In the second step the NB partial product rows are converted to (n/2 +1)/2 +1 RB rows if n/2 + 1 is even else to (n/2 + 2)/2 +1 RB rows. In third step all these partial product rows are accumulated using carry free adders to give the final result in RB form. This makes the hardware smaller in terms of number of adder rows to be used and reduces delay because of carry free addition.

3.5.2 Redundant Binary adder

An RB adder can add two RB numbers to give a RB number without any carry propagation. The addition rule for this is already discussed in Table 1.4. The same is written in truth table form in Table 3.6. In this all possible combinations of the code for the two digits in the i^{th} position are taken as augend and addend and the intermediate sum and carry are written looking to the augend and addend of $(i-1)^{th}$ position. So the sum and carry digits can have logical value s_{1i} and c_{1i} if any digits at next lower order position is negative else they will be s_{2i} and c_{2i}.

Table 3.6 Rules to generate intermediate sum and intermediate carry

Augend			Addend			Both digits at next lower order position are positive						One digits at next lower order position is negative					
A	B		C	D		Intermediate carry			Intermediate sum			Intermediate carry			Intermediate sum		
x^+_i	x^-_i	x_i	y^+_i	y^-_i	y_i	$c_2{}^+_i$	$c_2{}^-_i$	c_{2i}	$s_2{}^+_i$	$s_2{}^-_i$	s_{2i}	$c_1{}^+_i$	$c_1{}^-_i$	c_{1i}	$s_1{}^+_i$	$s_1{}^-_i$	s_{1i}
0	0	0	0	0	0	0	0	0	0	0	0	dc	dc	dc	dc	dc	dc
0	0	0	0	1	$\bar{1}$	0	0	0	0	1	$\bar{1}$	0	1	$\bar{1}$	1	0	1
0	0	0	1	0	1	1	0	1	0	1	$\bar{1}$	0	0	0	1	0	1
0	0	0	1	1	0	0	0	0	0	0	0	dc	dc	dc	dc	dc	dc
0	1	$\bar{1}$	0	0	0	0	0	0	0	1	$\bar{1}$	0	1	$\bar{1}$	1	0	1
0	1	$\bar{1}$	0	1	$\bar{1}$	0	1	$\bar{1}$	0	0	0	dc	dc	dc	dc	dc	dc
0	1	$\bar{1}$	1	0	1	0	0	0	0	0	0	dc	dc	dc	dc	dc	dc
0	1	$\bar{1}$	1	1	0	0	0	0	0	1	$\bar{1}$	0	1	$\bar{1}$	1	0	1
1	0	1	0	0	0	1	0	1	0	1	$\bar{1}$	0	0	0	1	0	1
1	0	1	0	1	$\bar{1}$	0	0	0	0	0	0	dc	dc	dc	dc	dc	dc
1	0	1	1	0	1	1	0	1	0	0	0	dc	dc	dc	dc	dc	dc
1	0	1	1	1	0	1	0	1	0	1	$\bar{1}$	0	0	0	1	0	1
1	1	0	0	0	0	0	0	0	0	0	0	dc	dc	dc	dc	dc	dc
1	1	0	0	1	$\bar{1}$	0	0	0	0	1	$\bar{1}$	0	1	$\bar{1}$	1	0	1
1	1	0	1	0	1	1	0	1	0	1	$\bar{1}$	0	0	0	1	0	1
1	1	0	1	1	0	0	0	0	0	0	0	dc	dc	dc	dc	dc	dc

The expressions for c_{1i}, s_{1i}, c_{2i} and s_{2i} are given below:

$$c_2^+ = C \cdot \overline{D} \cdot (A + \overline{B}) + A \cdot \overline{B} \cdot (C + \overline{D})$$

$$c_2^- = \overline{A} \cdot B \cdot \overline{C} \cdot D$$

$$s_2^+ = 0 \,,\; s_2^- = (A \oplus C) \oplus (B \oplus D)$$

$$c_1^+ = 0 \,,\; c_1^- = \overline{A} \cdot \overline{C} + B \cdot D \,,\; s_1^+ = 1 \,,\; s_1^- = 0$$

Here the final sum Sf_i can be s_{2i} if both xi and yi are 0 or 1 or $\bar{1}$. Else it depends upon the two augends and addend digits of $(i-1)^{th}$ position. If any one of the x_{i-1} and y_{i-1} are $\bar{1}$ then the Sf_i will be s_{1i}. In the same way the intermediate carry can be selected from c_{2i} and c_{1i}. The expressions for final intermediate sum and intermediate carry are given below:

$$Cf_i^+ = \overline{Sel} \cdot c_2^+ + Sel \cdot c_2^+ \cdot h_{i-1}$$
$$Cf_i^- = \overline{Sel} \cdot c_2^- + Sel \cdot (c_2^- \cdot \overline{h_{i-1}} + c_1^- \cdot h_{i-1})$$
$$Sf_i^+ = Sel \cdot h_{i-1}$$
$$Sf_i^- = Sel \cdot \overline{h_{i-1}}$$

Here

$$Sel = s_2^-, h_{i-1} = x_{i-1}^+ \cdot \overline{x_{i-1}^-} + y_{i-1}^+ \cdot \overline{y_{i-1}^-}$$

Now adding intermediate sum of i[th] position and carry of (i-1) [th] position the digit d_i can be found using the following expression:

$$d_i^+ = Sf_i^- \cdot \overline{Cf_{i-1}^+} + \overline{Sf_i^+} \cdot Cf_{i-1}^-$$
$$d_i^- = \overline{Sf_i^-} \cdot Cf_{i-1}^+ + Sf_i^+ \cdot \overline{Cf_{i-1}^-}$$

This RB adder is used at the first level of RB adder tree, to which all combination of RB code can be given as input. This can be called as RB adder with all code input (RBAA). But the output of this RBAA will have no (1,1) code for digit 0. The delay analysis of this RBAA gives the worst-case delay as:

$$x_{i-1}, y_{i-1} \xrightarrow{586\ ps} s_2^-(Sel) \xrightarrow{294\ ps} Cf_{i-1} \xrightarrow{206\ ps} d_i^+$$

So the total delay in a RBAA is 1086psec or $3.70T_{XOR}$.

The next level of adder will be taking input from first level of adder, which has no (1,1) code for digit 0. So the RB adder will have relatively less complexity. The expressions for RB adder, which will be used in the 2nd and higher levels, can be obtained in the same way as in the previous adder, but there will not be any input digit having (1,1) as the code. So the simplified expressions for c_{1i}, s_{1i}, c_{2i} , s_{2i} and h_{i-1} are given in next page.

$$c_2^+ = A \cdot (C + \overline{D}) + C \cdot (A + \overline{B})$$
$$c_2^- = B \cdot D$$
$$s_2^+ = 0 \,,\; s_2^- = (A + B) \oplus (C + D)$$
$$c_1^+ = 0 \,,\; c_1^- = \overline{A} \cdot B + \overline{C} \cdot D \,,\; s_1^+ = 1 \,,\; s_1^- = 0$$
$$h_{i-1} = x_{i-1}^- + y_{i-1}^-$$

As this adder is taking limited RB code as input, it can be called as RB adder with limited input or RBAL. In this RBAL the worst delay path is:

$$x_{i-1}, y_{i-1} \xrightarrow{494\ ps} s_2^+(Sel) \xrightarrow{294\ ps} Cf_{i-1} \xrightarrow{206\ ps} d_i^+$$

So the total delay in a RBAL is 994psec or 3.39 T_{XOR}.

3.5.3 Use of RB adder in a 16x16 multiplication

In a 16x16 multiplication the number of partial product rows will be PPnb= n/2 +1 =9. Now each set of two successive partial product rows will be grouped to form RB partial product rows resulting ((PPnb+1)/2) 5 number of RB partial product rows. Along with this, one more control bit row is to be added. So total RB rows to be compressed is ((PPnb+1)/2 +1) = 6. To compress 6 RB rows, three levels of RB adders will be required as shown in Fig. 3.12. In this the inputs to the first level are RB digits with any combination of RB code of Table 1.3. So the suitable adder is RBAA for this level. But in all other levels RBAL adder will be used. So the total delay in compressing these six partial product rows to single row of RB code will be 3074psec or 10.49T_{XOR} (summation of one RBAA and two RBAL delay). Similarly the delay analysis of partial product accumulation stage using RB adder for different sizes is done and summarized in Table 3.7.

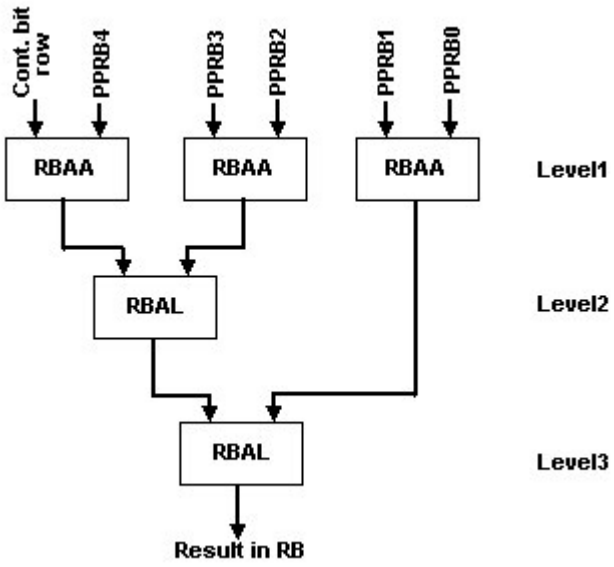


Fig. 3.12 Partial product accumulation of RB partial product rows generated from NB partial product rows in a 16x16 multiplication using RB adder.

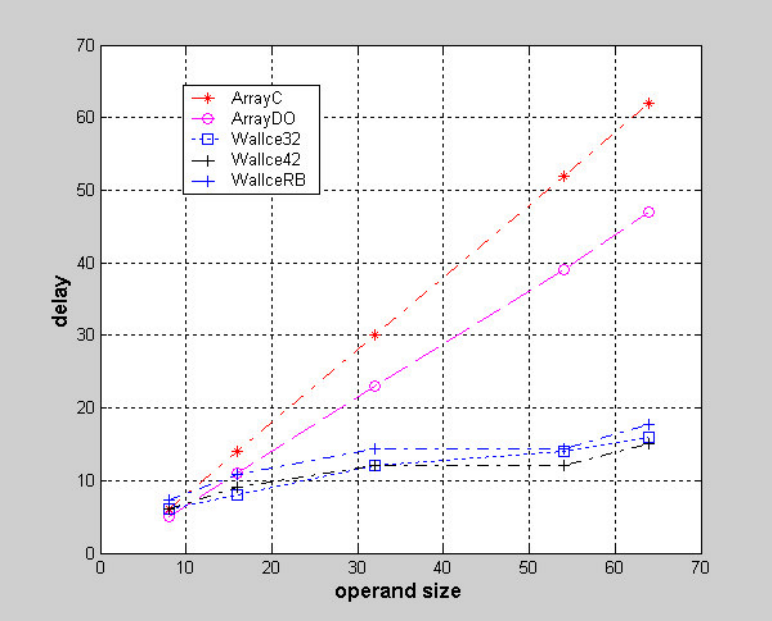
Table 3.7 Summary of worst-case delay for partial product accumulation using RB adders in Wallace tree

Number of NB partial products in a single column	Number of levels of RB adder	Delay in terms of XOR gate
≤2	1	3.70
≤6	2	(3.70+3.39) =7.09
≤14	3	(3.70+2*3.39) =10.49
≤30	4	(3.70+3*3.39) =13.87
≤62	5	(3.70+4*3.39) =17.26

3.6 Delay comparisons of different accumulation methods

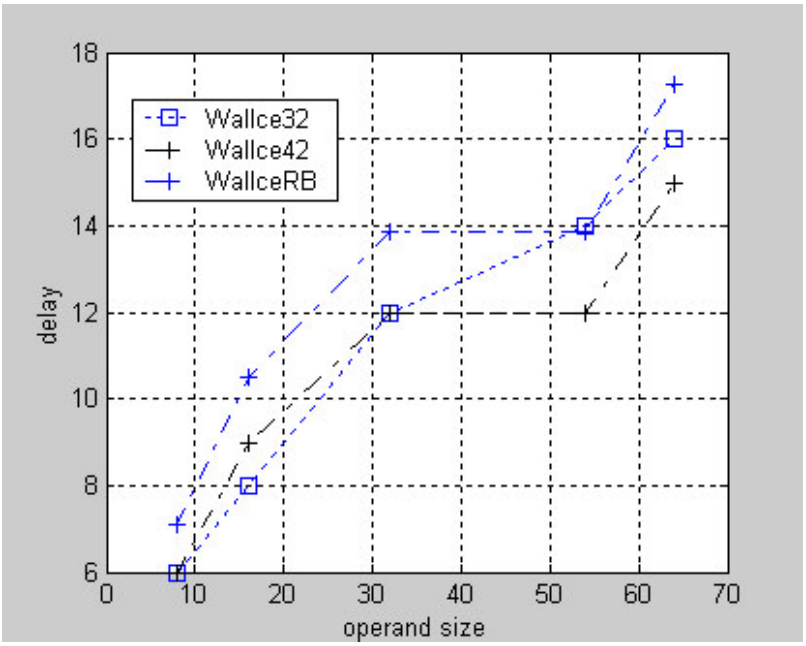
We have summarized the worst-case delay involved in partial product accumulation for frequently used operand sizes in multiplication using different architectures in Table 3.2, 3.3, 3.5, and 3.7. Now for comparison the normalized delay vs operand size is plotted using MATLAB in Fig. 3.13(a) and 3.13(b). In all the cases we expect that radix-4 Booth encoding will be used for generation of partial products. So for n x n multiplication the number of partial product rows will be (n/2 + 1). The plot in Fig. 3.13 (a) shows that for 8x8 multiplication the delay optimized array multiplier will have the least delay. This architecture

implementation is also simplest because of regular structure indicating best choice for 8x8 multiplier. This array accumulation method will have higher delay path for all other operand sizes but can be used for moderate speed application requiring regular layout and simpler interconnection. For 16x16 multiplication Wallace tree method of accumulation using 3:2 compressor is giving the best delay performance. For 32-bit operand size use of 3:2 and 4:2 compressors in Wallace tree are giving same delay performance. The delay performance of accumulation stage using 4:2 compressors in Wallace tree for 54 and 64 operand size is the best. It is observed that the delay of Wallace tree using RB adder is more. But this RB accumulation tree gives the result in one row in RB form. We will see in the next chapter that the conversion from RB to NB will have less delay in comparison to addition of sum and carry row using fast adders. So even if the accumulation delay is relatively more, it will give better overall delay performance.



(a)

Fig. 3.13 Normalized delay comparison of different accumulation methods (a) Delay of conventional array (ArrayC), delay optimized array (ArrayDO), Wallace tree using 3:2 compressor (Wallace32), Wallace tree using 4:2 compressor (Wallace42) and Wallace tree using RB adder (WallaceRB) Vs multiplier operand size



(b)

Fig. 3.13 Normalized delay comparison of different accumulation methods (b) Delay of Wallace32, Wallace42 and WallaceRB with multiplier operand size for better resolution.