Chapter – 2

Partial Product Generation

In a multiplication the first step is the partial product generation. The simple method of partial product generation involves the multiplication of every multiplicand bit by every multiplier bit. So for a multiplication of two n-bit numbers n partial product rows of n bits each will be generated. Here no sign bit is taken in to consideration. To generate every bit of partial product in a row an AND gate can be used. In this method of partial product generation, accumulation hardware will be required to accumulate n partial product rows and accordingly time delay will be there.

Using any technique in the above, if the number of partial product rows can be reduced to n/2 or less, then the hardware required to accumulate them as well as the accumulation time will be reduced. Keeping this as objective Booth [Booth 1951] in 1951 had proposed an algorithm which was modified later by Rubinfield [Rubinfield 1975] and given the name Modified Booth's Encoding (MBE) algorithm. Using this algorithm the number of partial product rows will reduce to half. Based on this algorithm various efficient circuit techniques have been developed by a number of researchers[Choi 2001, Khoo 1999, Villeger 1993]. One such circuit technique has been developed and compared with others as a part of this thesis work. The number of partial product rows can be reduced further by using simple extensions to Booth's algorithm by recording larger number of inputs (called as higher radix encoding) [Atkin 1970, Sam 1990]. But for this purpose time-consuming and complex partial product generation circuits are required, which are discussed in detail in this chapter.

It is true that for higher radix encoding, the partial product generator will have higher delay. But in this work a new RB arithmetic based scheme has been proposed to generate partial products using radix-64 encoding. This will reduce the number of partial product rows by a factor of 6, but with significantly smaller delay as compared to the reported work of radix-64 encoding by Sang-Hoon Lee [Hoon 2002].

2.1 Non-Booth method of partial product generation

The simplest method for partial product generation is called non-Booth. This is illustrated by the use of the dot diagram for the 16x16 bit multiplication in Fig. 2.1. Here each row of dots represents a partial product row and each dot represent a bit in the partial product. A partial product row can be either zero or the multiplicand row depending on whether the corresponding multiplier bit value is 0 or 1. The partial products are shifted to account for the weights of multiplier bits with which the multiplicand is multiplied. The partial product generation logic consists of a row of single AND gates for each bit of partial product. The inputs for each AND gate are bits of the multiplier and multiplicand, such that the sum of the bit’s arithmetic weights is equal to the column’s arithmetic weight.

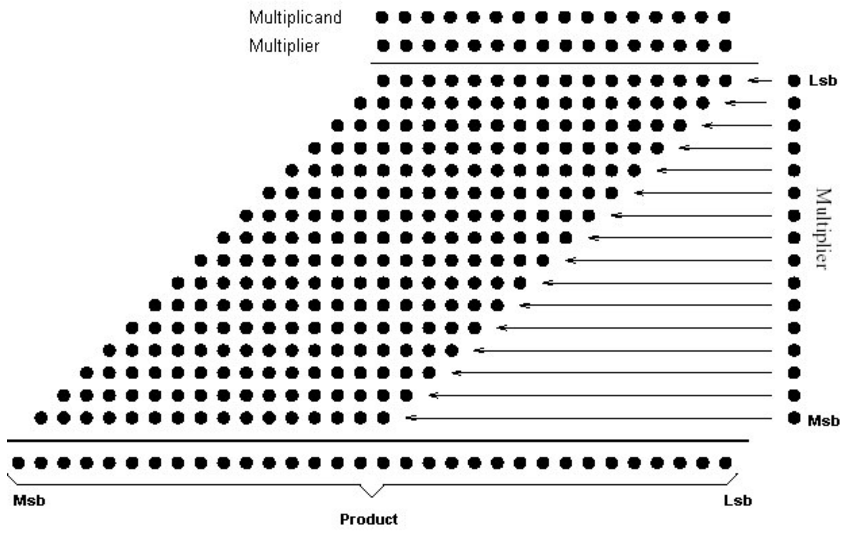


Fig. 2.1 A 16 x 16 bit non-Booth encoding

2.2 Booth's algorithm

In the non-Booth's method of partial product generation the number of partial product rows is same as the number of multiplier bits. But by using Booth's algorithm with different radices the number of partial product rows can be reduced by a factor of two or more. This will lead to reduced accumulation time and smaller area. But the price is paid in terms of the partial product generation complexity and the time required to generate the partial products.

2.2.1 Booth radix-4 encoding

The most widely used technique for partial product generation in parallel multipliers is radix-4 modified Booth encoding (MBE). This can reduce the number of partial product rows to be generated by a factor of two thus reducing the accumulation hardware cost as well as the accumulation time. According to MBE a signed binary number in two's complement form can be partitioned to overlapping group of three bits as shown in the Fig. 2.2. By coding each of these groups into single signed digit as shown in Table 2.1, an n-bit signed binary number can be represented as n/2 sign digits. Here each signed digit takes the possible value of 0, +1, -1, +2, -2 and accordingly the partial product will take a value of 0, +A, -A, +2A, -2A where A is the multiplicand. The truth table for coding used in the radix-4 MBE is shown in Table 2.1.

Table 2.1 Partial product selections in radix-4 encoding

b_{j+1}	b_j	b_{j-1}	Code
0	0	0	0
0	0	1	+A
0	1	0	+A
0	1	1	+2A
1	0	0	-2A
1	0	1	-A
1	1	0	-A
1	1	1	0

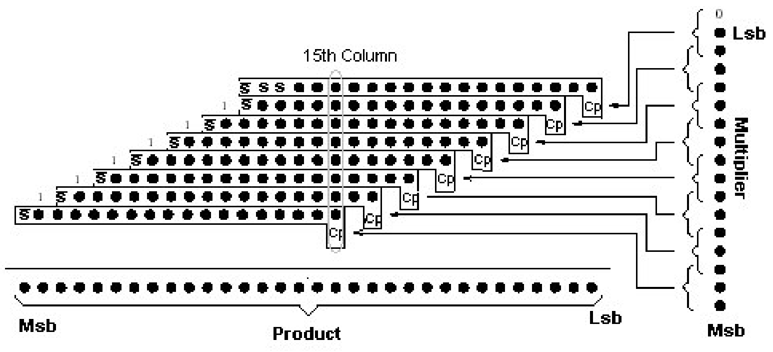


Fig. 2.2 Multiplication of two 16-bit numbers using radix-4 MBE.

Depending upon the coded value of each group the 0, +A, +2A can be obtained easily by selecting zero or A or shifting A one bit position. For –A and –2A each bit is complemented and finally the code polarity bit Cp = 1 is added at least significant bit (LSB) position. This indicates that the maximum possible number of bits in a single column to be added is n/2 +1. In the present example 15th column is having maximum number of bits. But for positive number multiplication the MSB bit of multiplier will be always 0. So the maximum number of bits in a single column in this case is n/2, provided n is even. If n is odd, the maximum number of bits in a single column will be (n+1)/2.

The polarity of a partial product row is dependant on both the multiplicand polarity as well as the code polarity (Cp). A partial product row is negative i.e. the sign bit of the partial product is 1 only when one of multiplicand or the code polarity is negative. So the sign bit can be obtained as S = $a15 \oplus Cp$, where Cp is the code polarity of that specific code and a15 is the sign bit of the multiplicand. To preserve the sign bit of the partial product sign extension of partial products to the left most bit position is to be done. The sign extension can be replaced by what is shown in Fig. 2.2. A proof of this replacement is given in [Vassiliadis 1991].

2.2.2 Booth radix-8 encoding

The radix-8 encoding is the extension of radix-4 encoding. In this the binary number in two's complement form is partitioned into overlapping groups of four bits as shown in Fig. 2.3. By coding each of these groups, a n-bit signed binary number can be represented as a sum of n/3 signed digits. The partial product code generation is given in Table 2.2. Here each signed digit takes the possible value of 0, ± 1, ± 2, ± 3, ± 4 and accordingly the partial product will take a value of 0, $\pm$A, $\pm$2A, $\pm$3A, $\pm$4A, where A is the multiplicand.

Table 2.2 Partial product selections in radix-4 encoding

Multiplier bit	Selection	Multiplier bit	Selection
0000	+0	1000	– 4A
0001	+A	1001	– 3A
0010	+A	1010	– 3A
0011	+2A	1011	– 2A
0100	+2A	1100	– 2A
0101	+3A	1101	– A
0110	+3A	1110	– A
0111	+4A	1111	– 0

Depending upon the coded value of each group partial products 0, +A, +2A, + 4A and –2A, -4A can be obtained easily by selecting 0, A, shifting A or taking two's complement of A and adding 1 in the LSB position. But to get 3A one carry propagate addition is to be done. This will increase the latency of the circuit. The complexity of the selection logic will also increase.

Using the same method of grouping, more bits can be grouped and the number of partial product rows can be reduced. But because of selection complexity and the requirement of carry propagate adder to generate partial product like 3A the performance will degrade. Because of this reason the most widely used Booth encoding is limited to radix-4. So in the next section we will discuss the best circuits for partial product generation using radix-4 MBE.

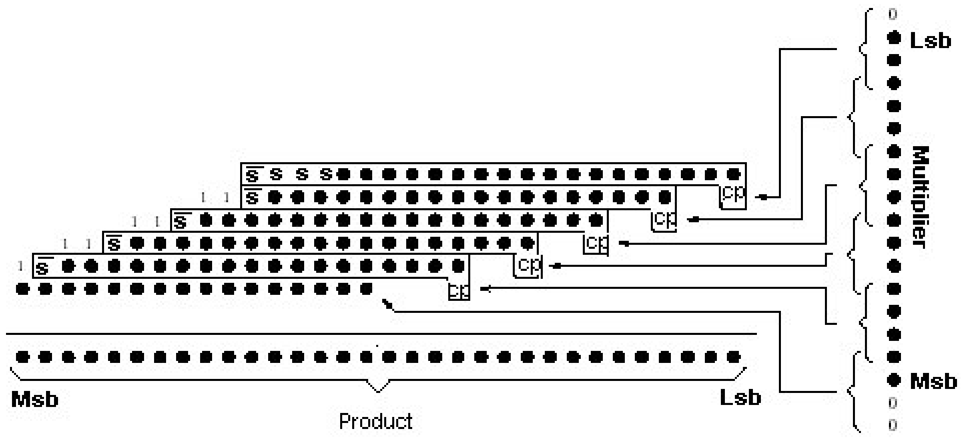


Fig. 2.3 Multiplication of two 16-bit numbers using Radix-8 encoding.

2.3 Circuits for radix – 4 MBE

The most widely used technique for partial product generation in parallel multipliers is radix-4 MBE. The main features of this encoding are that the partial product rows can be generated using low latency simple circuits. Because of this many researchers have concentrated on decreasing latency and area. So in this section we will discuss some of the best circuits proposed by others and a new circuit proposed by us. The circuit performances are compared based on simulation results and our circuit is found to out perform others.

2.3.1 Modified Booth encoding

As discussed earlier a signed binary number in two's complement form can be partitioned into overlapping groups of three bits that are mapped to a signed digit. Here each signed digit takes the possible value of 0, +1, -1, +2, -2, and accordingly, the partial products will take values of 0, +A, -A, +2A, -2A; where A is the multiplicand. The truth table for the radix-4 MBE [Goto 1997] along with the required control bits is shown in Table 2.3. Using these control bits the partial products can be generated by the following expression [Goto 1997].

$$P_{i,j} = (a_i . PL_j + \overline{a_i} . M_j) X_j + (a_{i-1} . PL_j + \overline{a_{i-1}} . M_j) 2X_j \qquad (2.1)$$

Table. 2.3 Truth table of MBE

Inputs				Sign select			
b_{j+1}	b_j	b_{j-1}	Func.	X_j	$2X_j$	PL_j	M_j
0	0	0	0	0	1	0	0
0	0	1	+A	1	0	1	0
0	1	0	+A	1	0	1	0
0	1	1	+2A	0	1	1	0
1	0	0	-2A	0	1	0	1
1	0	1	-A	1	0	0	1
1	1	0	-A	1	0	0	1
1	1	1	0	0	1	0	0

So partial products can be obtained by getting control signals Xj, 2Xj, Mj, PLj by a circuit called as Booth encoder (BE) and applying these to a selector logic, called Booth selector (BS) to get the final partial product.

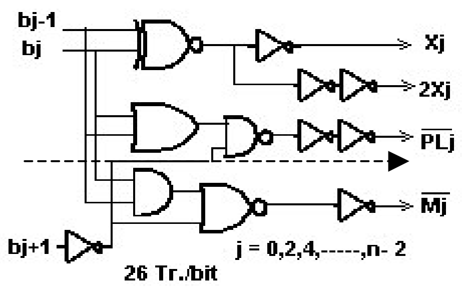
2.3.2 Best proposed circuits by others

The implementations of the best-reported circuits are done in same way as are reported by authors. In case of non-availability of specification the circuit technique used is made uniform for all circuits and also mentioned.

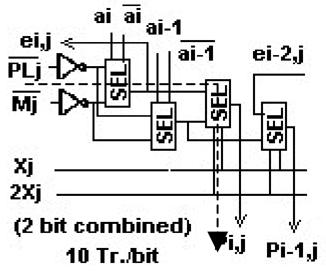
The first circuit pair elements are given the names BE-I and BS-I [Goto 1997] and are shown in Fig. 2.4(a) and (b). Here the XOR gate is implemented using pass transistor logic **a**nd for all other circuits CMOS logic is used. The encoder part uses 38 transistors and for generating a single partial product the selector circuit uses 10 transistors. This circuit was the least transistor circuit reported till date for single partial product generation.

The next circuit pair BE-II and BS-II are reported in [Ohkubu 1995] and are shown in Fig. 2.4 (c) and (d). Here the multiplexer used is based on complementary pass transistor logic and the XOR gate used is based on pass transistor logic. This BE-II uses 36 transistors and BS-II uses 18 transistors. The delay path of encoder and selector, which is responsible for critical delay, is shown in the Fig. 2.4 (c) and (d) by dotted lines.

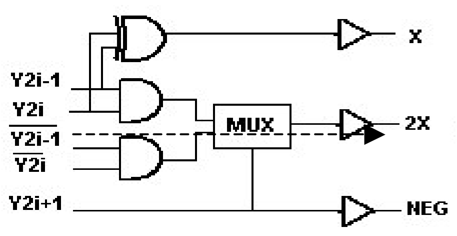
In the third implementation [Changyeh 2000] in Fig. 2.4 (e) and (f) the encoder part BE-III is very simple and having least delay. Here the partial product generation part BS-III uses 18 transistors and encoder part uses 18 transistors. For XOR implementation pass transistor logic is used. Here the critical delay path is from Y to out put. Though the transistor count for the encoder is very small, it is not so for the partial product generation. In fact in a n x n multiplier only n/2 +1 number of encoders are used where as n x (n/2 +1) number of selectors are used. So the transistor advantage in the encoder in this implementation does not give overall advantage in transistor count.



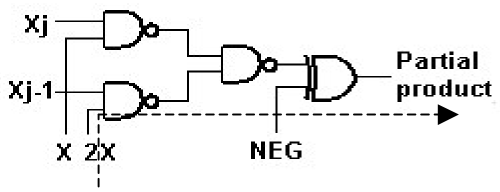
(a)



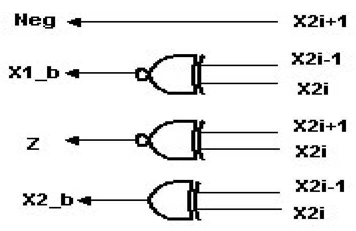
(b)



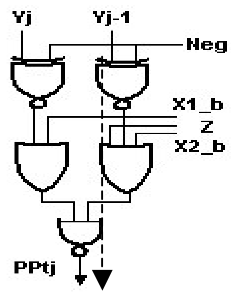
(c)



(d)



(e)

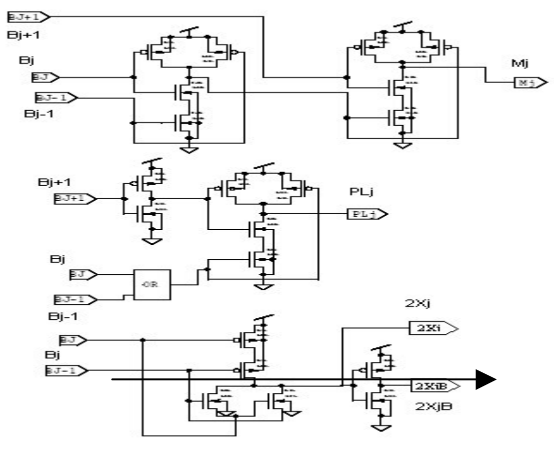


(f)

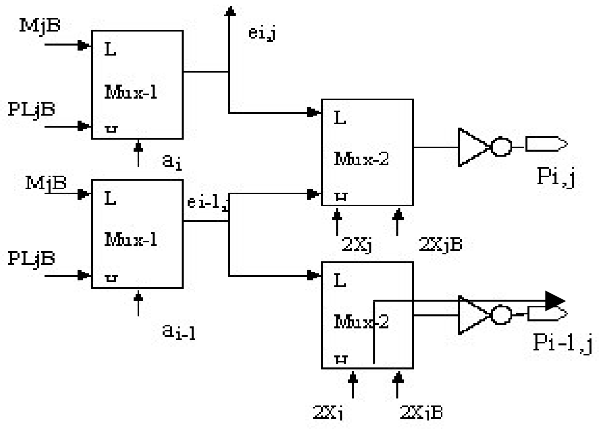
Fig. 2.4. Booth encoders and selectors (a) BE-I (b) BS-I (c) BE-II (d) BS-II (e) BE-III (f) BS-III.

2.3.3 New Booth encoder (BE-new) and new Booth selector (BS-new)

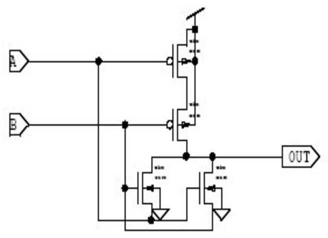
The BE-new proposed by us generates three control signals X_j, M_j, PL_j which are used by BS-new to generate partial products. Looking to the Table 2.3 and using ground less XNOR gates [Bui 2002] its implementation is shown in Fig. 2.5(a). The XNOR is called as ground less XNOR as it has no direct path from V_{dd} to ground as shown in Fig. 2.5(c). Thus the power consumption is less. The BS-new design using high-speed n-channel multiplexer is shown in Fig. 2.5(b). The n-channel multiplexer used here is shown in Fig. 2.5(d). The MUX-1 in BS-new contains an inbuilt inverter to generate the inverse of SEL i.e. SELB from SEL. In this implementation for BE-new only 26 transistors are used and the selector part generates two partial products using 16 transistors. So the new implementation needs only 8 transistors per partial product generation. The critical path for BE and BS combination based on simulation result is shown in Fig. 2.5(a) and (b) by dotted lines.



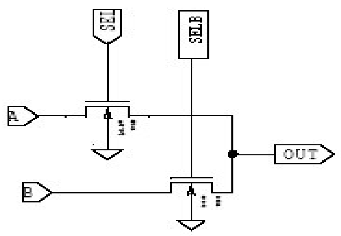
(a)



(b)



(c)



(d)

Fig. 2.5. New circuits (a) BE-new (b) BS-new (c) Ground less XNOR (d) n-channel multiplexer.

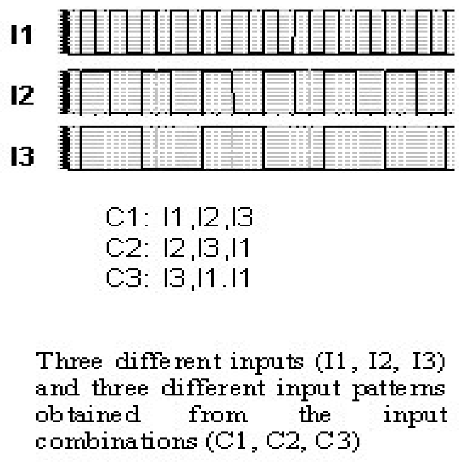
2.3.4 Comparison of the BE-new and BS-new

The above new circuit and the other three reported circuits are implemented using S-edit. The performance evaluation is done based on normalized delay and power obtained from simulation results.

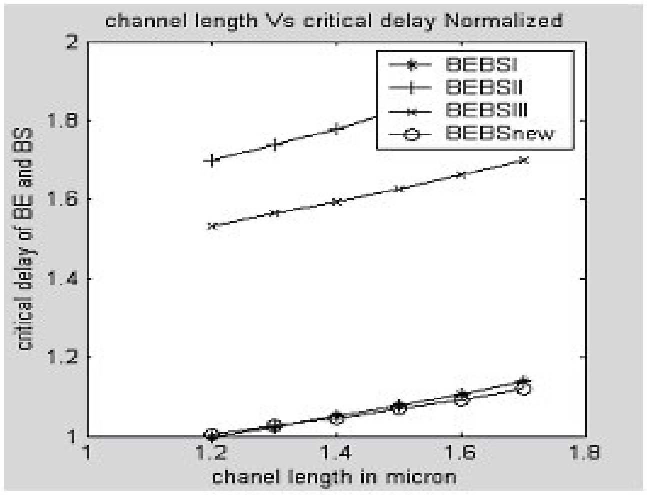
To verify and compare the performance of above circuits each booth encoder and selector is simulated for all possible inputs. For proper understanding a typical input combinations for BE is given in Fig. 2.6(a). As the circuit responds differently to different input patterns we used three input patterns C1, C2, C3 formed from inputs I1, I2, I3 as shown in Fig. 2.6(a). 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 15 TSPICE simulation are run and in each case the rise time delay and fall time delay are noted down and the critical delay and average power consumption is reported for the circuits. During a simulation session a single power measurement is taken by averaging the instantaneous power over a period of three pattern cycles starting from the beginning of second cycle to the end of fourth cycle. The measurement does not include the first cycle to avoid transient glitches. Every time the channel length is scaled down and the simulation results in normalized form are noted in Table 2.4 along with transistor count. With scaling down of channel length the delay, power and energy delay product (EDP) are plotted in Fig. 2.6(b), 2.6(c), 2.6(d).

Table 2.4 BE and BS comparisons

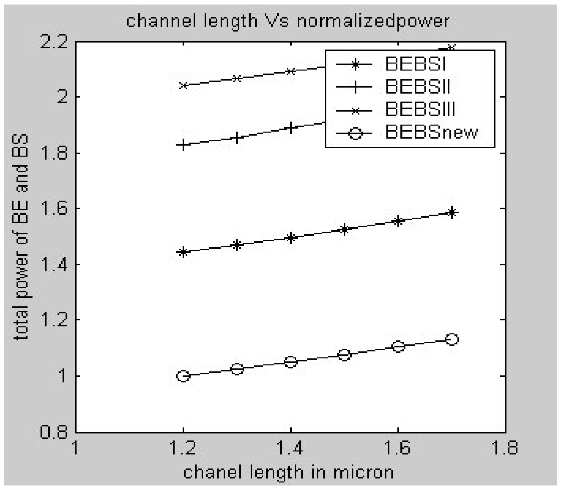
Type of Booth encoder and booth selector	No of tr. Used		Delay (Normalized)						Power (Normalized)						EDP (Normalized)					
			Channel Length in micron						Channel Length in micron						Channel Length in micron					
	BE	BS	1. 2	1. 3	1.4	1. 5	1. 6	1. 7	1. 2	1. 3	1. 4	1. 5	1. 6	1.7	1. 2	1. 3	1. 4	1. 5	1. 6	1.7
BE-I and BS-I	38	10	1.00	1.02	1.05	1.07	1.10	1.13	1.44	1.46	1.49	1.52	1.55	1.58	1.44	1.50	1.57	1.64	1.71	1.80
BE-II and BS-II	36	18	1.69	1.73	1.77	1.82	1.87	1.92	1.82	1.85	1.88	1.92	1.95	1.99	3.10	3.22	3.35	3.50	3.67	3.84
BE-III and BS-III	18	18	1.53	1.56	1.59	1.62	1.66	1.69	2.04	2.06	2.08	2.11	2.14	2.17	3.13	3.22	3.33	3.44	3.56	3.69
BE-new and BS-new	26	8	1.00	1.02	1.04	1.06	1.09	1.11	1.00	1.02	1.05	1.07	1.10	1.13	1.00	1.05	1.09	1.15	1.20	1.26


I1
I2
I3
C1: I1 ,I2 ,I3
C2: I2 ,I3 ,I1
C3: I3 ,I1 .I1
Three different inputs (I1, I2, I3) and three different input patterns obtained from the input combinations (C1, C2, C3)

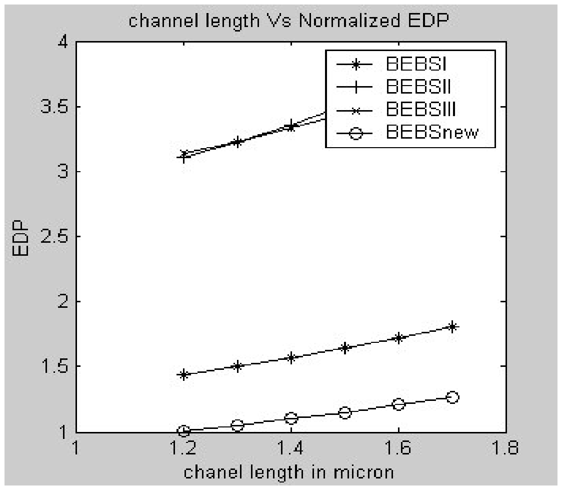
(a)



(b)



(c)



(d)

Fig. 2.6. (a) Inputs applied to BE-BS combinations and the plots for normalized (b) Delay (c) Power (d) Energy delay product.

From the delay plot of Fig. 2.6(b) it is clear that the new design is having delay comparable to that of BE-I and BS-I pair. But this is achieved with less power consumption and less energy delay product (EDP) as observed from Fig. 2.6(c) and 2.6(d) along with transistor advantages. The power advantage is coming because of the use of ground less XOR gate and the delay advantages is coming because of the use of n-channel pass transistor multiplexer.

The delay of BE-new and BS-new pair is compared with the delay of a two input XOR gate as shown in Fig. 2.7, implementing both of them with S-edit. The normalized delay of this BE-new and BS-new pair with respect to this XOR gate is found to be 2.11. This delay is used in chapter 5 for delay calculation of partial product generation stage in terms of T_{XOR}.

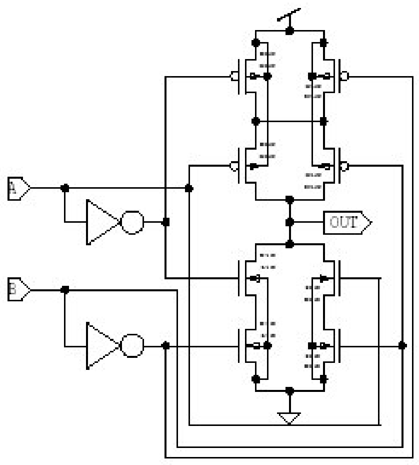


Fig. 2.7 Two input XOR gate

2.4 Radix-64 encoding using redundant arithmetic

It has been discussed in the previous section (section 2.2) that to avoid complexity and delay involved in higher radix encoding radix 4 encoding is mostly preferred. Hoon [Hoon 2002] has proposed an innovative architectural idea for higher radix multiplier. This architecture is improved upon by us to decrease the delay further.

In radix-64 encoding the binary number in two's complement form is partitioned into overlapping groups of seven bits. However this requires 65 multiplying coefficients such as –32, -31, …….., 0,……., 29, 30, 31, 32, and accordingly the partial product rows will take values of –32X, -31X, -30X, ……., 0, ……., 29X, 30X, 31X, 32X, where X is the multiplicand. A partial product row with multiplying coefficients in the form of 2^i (where i

=1, 2, 3, 4, 5) is easy as it can be obtained just by shifting X. Generation of other partial product rows with multiplying coefficients 3, 5, 7, 9, 11, ---, 31 will be difficult. However all the partial product rows can be obtained by single addition of two numbers mX and nX. Here m and n are coefficients, one is chosen from 0, ± 1, ± 2, ± 3, ± 4 (called as T-group or Tgr coefficients) and the other is chosen from 0, ± 8, ± 16, ± 24, ± 32 (called as S-group or Sgr coefficients) as shown in Table. 2.5. The addition of these two numbers (mX and nX) can be redundant binary addition as discussed in Table 1.4 of chapter-1 to avoid any carry propagation.

Now the two basic groups of numbers required as multiplying coefficient of X are 0, 1, 2, 3, 4 (Tgr) and 0, 8, 16, 24, 32 (Sgr). All the numbers of both the groups can be obtained from 0 or by shifting either X or 3X. Obviously 24X can be obtained by shifting 3X three-bit positions. 3X can be generated by adding X and 2X using a carry look-ahead adder as is done by Sang-Hoon. This also can be obtained as 4X-X. The computation of 4X-X using RB as discussed in chapter 1 equation 1.2 is just grouping of 4X and X. So the 3X can be computed with only routing delay in place of large carry propagate adder delay.

Table 2.5 Choosing the S group and T group number from fundamental coefficients

y6	y5	y4	y3	y2	y1	y0	n	Tgr	Sgr	y6	y5	y4	y3	y2	y1	y0	n	Tgr	Sgr
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-32	0	-32
0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	1	-31	1	-32
0	0	0	0	0	1	0	1	1	0	1	0	0	0	0	1	0	-31	1	-32
0	0	0	0	0	1	1	2	2	0	1	0	0	0	0	1	1	-30	2	-32
0	0	0	0	1	0	0	2	2	0	1	0	0	0	1	0	0	-30	2	-32
0	0	0	0	1	0	1	3	3	0	1	0	0	0	1	0	1	-29	3	-32
0	0	0	0	1	1	0	3	3	0	1	0	0	0	1	1	0	-29	3	-32
0	0	0	0	1	1	1	4	4	0	1	0	0	0	1	1	1	-28	4	-32
0	0	0	1	0	0	0	4	-4	8	1	0	0	1	0	0	0	-28	-4	-24
0	0	0	1	0	0	1	5	-3	8	1	0	0	1	0	0	1	-27	-3	-24
0	0	0	1	0	1	0	5	-3	8	1	0	0	1	0	1	0	-27	-3	-24
0	0	0	1	0	1	1	6	-2	8	1	0	0	1	0	1	1	-26	-2	-24
0	0	0	1	1	0	0	6	-2	8	1	0	0	1	1	0	0	-26	-2	-24
0	0	0	1	1	0	1	7	-1	8	1	0	0	1	1	0	1	-25	-1	-24
0	0	0	1	1	1	0	7	-1	8	1	0	0	1	1	1	0	-25	-1	-24
0	0	0	1	1	1	1	8	0	8	1	0	0	1	1	1	1	-24	0	-24
0	0	1	0	0	0	0	8	0	8	1	0	1	0	0	0	0	-24	0	-24
0	0	1	0	0	0	1	9	1	8	1	0	1	0	0	0	1	-23	1	-24
0	0	1	0	0	1	0	9	1	8	1	0	1	0	0	1	0	-23	1	-24
0	0	1	0	0	1	1	10	2	8	1	0	1	0	0	1	1	-22	2	-24
0	0	1	0	1	0	0	10	2	8	1	0	1	0	1	0	0	-22	2	-24
0	0	1	0	1	0	1	11	3	8	1	0	1	0	1	0	1	-21	3	-24
0	0	1	0	1	1	0	11	3	8	1	0	1	0	1	1	0	-21	3	-24
0	0	1	0	1	1	1	12	4	8	1	0	1	0	1	1	1	-20	4	-24
0	0	1	1	0	0	0	12	-4	16	1	0	1	1	0	0	0	-20	-4	-16
0	0	1	1	0	0	1	13	-3	16	1	0	1	1	0	0	1	-19	-3	-16
0	0	1	1	0	1	0	13	-3	16	1	0	1	1	0	1	0	-19	-3	-16
0	0	1	1	0	1	1	14	-2	16	1	0	1	1	0	1	1	-18	-2	-16
0	0	1	1	1	0	0	14	-2	16	1	0	1	1	1	0	0	-18	-2	-16
0	0	1	1	1	0	1	15	-1	16	1	0	1	1	1	0	1	-17	-1	-16
0	0	1	1	1	1	0	15	-1	16	1	0	1	1	1	1	0	-17	-1	-16
0	0	1	1	1	1	1	16	0	16	1	0	1	1	1	1	1	-16	0	-16

y6	y5	y4	y3	y2	y1	y0	n	Tgr	Sgr	y6	y5	y4	y3	y2	y1	y0	n	Tgr	Sgr
0	1	0	0	0	0	0	16	0	16	1	1	0	0	0	0	0	-16	0	-16
0	1	0	0	0	0	1	17	1	16	1	1	0	0	0	0	1	-15	1	-16
0	1	0	0	0	1	0	17	1	16	1	1	0	0	0	1	0	-15	1	-16
0	1	0	0	0	1	1	18	2	16	1	1	0	0	0	1	1	-14	2	-16
0	1	0	0	1	0	0	18	2	16	1	1	0	0	1	0	0	-14	2	-16
0	1	0	0	1	0	1	19	3	16	1	1	0	0	1	0	1	-13	3	-16
0	1	0	0	1	1	0	19	3	16	1	1	0	0	1	1	0	-13	3	-16
0	1	0	0	1	1	1	20	4	16	1	1	0	0	1	1	1	-12	4	-16
0	1	0	1	0	0	0	20	-4	24	1	1	0	1	0	0	0	-12	-4	-8
0	1	0	1	0	0	1	21	-3	24	1	1	0	1	0	0	1	-11	-3	-8
0	1	0	1	0	1	0	21	-3	24	1	1	0	1	0	1	0	-11	-3	-8
0	1	0	1	0	1	1	22	-2	24	1	1	0	1	0	1	1	-10	-2	-8
0	1	0	1	1	0	0	22	-2	24	1	1	0	1	1	0	0	-10	-2	-8
0	1	0	1	1	0	1	23	-1	24	1	1	0	1	1	0	1	-9	-1	-8
0	1	0	1	1	1	0	23	-1	24	1	1	0	1	1	1	0	-9	-1	-8
0	1	0	1	1	1	1	24	0	24	1	1	0	1	1	1	1	-8	0	-8
0	1	1	0	0	0	0	24	0	24	1	1	1	0	0	0	0	-8	0	-8
0	1	1	0	0	0	1	25	1	24	1	1	1	0	0	0	1	-7	1	-8
0	1	1	0	0	1	0	25	1	24	1	1	1	0	0	1	0	-7	1	-8
0	1	1	0	0	1	1	26	2	24	1	1	1	0	0	1	1	-6	2	-8
0	1	1	0	1	0	0	26	2	24	1	1	1	0	1	0	0	-6	2	-8
0	1	1	0	1	0	1	27	3	24	1	1	1	0	1	0	1	-5	3	-8
0	1	1	0	1	1	0	27	3	24	1	1	1	0	1	1	0	-5	3	-8
0	1	1	0	1	1	1	28	4	24	1	1	1	0	1	1	1	-4	4	-8
0	1	1	1	0	0	0	28	-4	32	1	1	1	1	0	0	0	-4	-4	0
0	1	1	1	0	0	1	29	-3	32	1	1	1	1	0	0	1	-3	-3	0
0	1	1	1	0	1	0	29	-3	32	1	1	1	1	0	1	0	-3	-3	0
0	1	1	1	0	1	1	30	-2	32	1	1	1	1	0	1	1	-2	-2	0
0	1	1	1	1	0	0	30	-2	32	1	1	1	1	1	0	0	-2	-2	0
0	1	1	1	1	0	1	31	-1	32	1	1	1	1	1	0	1	-1	-1	0
0	1	1	1	1	1	0	31	-1	32	1	1	1	1	1	1	0	-1	-1	0
0	1	1	1	1	1	1	32	0	32	1	1	1	1	1	1	1	0	0	0

2.4.1 Example of partial product generation using radix-64 algorithm

Assume X (= 010010 100011) and Y (= 010011 101001) are to be multiplied as shown in Fig. 2.8. Now from Table 2.5 grouping of 7 LSB bits after padding 0 in the LSB position gives the code digit =n0 = -23. So the first partial product = -23 X. As to Table. 2.5 this can be represented as the addition of two numbers 1X (one from Tgr) and -24X (other from Sgr).

i.e. –23X = X – 24X.

```
      x11 x10 x9 x8 x7  x6 x5 x4 x3 x2 x1 x0
X =    0   1   0  0  1   0  1  0  0  0  1  1    = 1187

Y =    0   1   0  0  1   1  1  0  1  0  0  1  0
      y11 y10 y9 y8 y7  y6 y5 y4 y3 y2 y1 y0 y-1
                        \_____________________/
                             First grouping
For j=0 ,
code digit Dj = -32y6j+5 + 16y6j+4 + 8y6j+3 + 4y6j+2 + 2y6j+1 +1y6j + 1y6j-1
              = -32y5 + 16y4 + 8y3 + 4y2 + 2y1+ 1y0 + 1y-1
              = -32 + 0 + 8 + 0 + 0 + 1 + 0
              = -23
First partial product = - 23X = X - 24X
24X = 3X shifted by three bit position left with zeros padded at LSB
3X = 4X - X = (4X , X ) in RB form
4X = X left shifted by two bit positions with two zeros padded at LSB
   = 0   1   0   0   1   0   1   0   0   0   1   1   0   0
 X = 0   0   0   1   0   0   1   0   1   0   0   0   1   1

3X = (0,0) (1,0) (0,0) (0,1) (1,0) (0,0)  (1,1) (0,0) (0,1)  (0,0) (1,0) (1,0) (0,1) (0,1)

24X = (0,0) (1,0) (0,0) (0,1) (1,0) (0,0) (1,1) (0,0) (0,1) (0,0) (1,0) (1,0) (0,1) (0,1) (0,0) (0,0) (0,0)

-24X = (1,1) (0,1) (1,1) (1,0) (0,1) (1,1) (0,0) (1,1) (1,0) (1,1) (0,1) (0,1) (1,0) (1,0) (1,1) (1,1) (1,1)
         0    -1    0     1    -1    0     0     0     1     0    -1    -1    1     1     0     0     0
  X =  (0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (1,0) (0,0) (0,0) (1,0) (0,0) (1,0) (0,0) (0,0) (0,0) (1,0) (1,0)
         0     0     0     0     0     0     1     0     0     1     0     1     0     0     0     1     1
      ______________________________________________________________________________________________
I     Sum      0    -1     0     1    -1     0    -1     0    -1     1     1     0    -1    -1     0    -1    -1
n
t
e     Ci  0    0     0     0     0     0     1     0     1     0    -1     0     1     1     0     1     1
r
m
e
d
i
a
t
e
      ______________________________________________________________________________________________
      S   0    0    -1     0     1    -1     1    -1     1    -1     0     1     1     0    -1     1     0    -1

      So Partial product =     -23X = - 27301
```

Fig. 2.8. Example of partial product generation using radix-64 algorithm.

So the partial product generation can be divided into two parts. One part is the generation of 0, 1X, 2X, 3X, 4X(Tgr) and 0, 8X, 16X, 24X, 32X(Sgr) and the other is selecting the correct Tgr and Sgr digit using seven bits (y_{6j+5} , y_{6j+4}, y_{6j+3}, y_{6j+2}, y_{6j+1}, y_{6j}, y_{6j-1}), where j = 0, 1, 2,. …. , n/6.

As we have discussed 2X, 4X, 8X, 16X, 32X can be obtained just by shifting X. 24X can be obtained by shifting 3X, 3 bit position left and padding three zeros at LSB side. So 3X is to be obtained. To get 3X we can add 2X(X shifted by one bit) with X by using CPA. In this case the delay will be same as that of CPA and it increases with the size of X. We also can obtain 3X in RB form as 4X –X using equation 1.2 without any carry propagation delay. So 3X = 4X – X = (4X, X). Grouping of 4X and X to get 3X in the present example is shown in Fig 2.8. So the required 24X is obtained just by shifting of 3X by 3-digit position. Now all the bits in 24X are inverted to get –24X. This –24X is in RB form. So all other Sgr and Tgr elements are to be represented in RB form. In the present example –24X (in RB form) is to be added with X. So X is represented in RB form as X=X-0 = (X, 0). The same method can be used for representing 0, 2X, 4X, 8X, 16X and 32X in RB form.

Now the other part is selecting the correct Tgr digit ($Tgr^{+}_{i,j}$, $Tgr_{i,j}^{-}$) and Sgr digit($Sgr^{+}_{i,j}$, $Sgr^{-}_{i,j}$) using seven bits (y_{6j+5} , y_{6j+4}, y_{6j+3}, y_{6j+2}, y_{6j+1}, y_{6j}, y_{6j-1}) where i = 0, 1, 2, …….., n+5, n+6 and j =0, 1, 2,. ….. , n/6. Tgr digit and Sgr digit are obtained by using the 12 control signals (S1Xj, S2Xj, S3Xj, S4Xj, S8Xj, S16Xj, S24Xj, S32Xj, ZEROA, ZEROB, y_{6j+5}, y_{6j+2}) in selector circuit of Fig. 2.9. These control signals are obtained from seven consecutive bits of Y (y_{6j+5}, y_{6j+4}, y_{6j+3}, y_{6j+2}, y_{6j+1}, y_{6j}, y_{6j-1}) using expressions of equation 2.2.

Here it is assumed that j=0. So the seven consecutive bits of Y are y5, y4, y3, y2, y1, y0, y-1

S1X = $\overline{(y_2 \oplus y_1)}.(y_{-1} \oplus y_0)$

$$S2X = \overline{y_1} \cdot y_0 \cdot y_{-1} + y_1 \cdot \overline{(y_{-1} + y_0)}$$

$$S3X = (y_1 \oplus y_2) \cdot (y_{-1} \oplus y_0)$$

$$S4X = \overline{y_2} \cdot (y_{-1} \cdot y_0 \cdot y_1) + y_2 \cdot \overline{(y_{-1} + y_0 + y_1)}$$

$$ZERA = \overline{(y_{-1} + y_0 + y_1 + y_2)} + (y_{-1} \cdot y_0 \cdot y_1 \cdot y_2)$$

$$ZERB = \overline{(y_2 + y_3 + y_4 + y_5)} + (y_2 \cdot y_3 \cdot y_4 \cdot y_5)$$

$$S8X = \overline{(y_5 \oplus y_4)}.(y_3 \oplus y_2)\text{ s}$$

$$S16X = \overline{y_4} \cdot y_3 \cdot y_2 + y_4 \cdot \overline{(y_3 + y_2)}$$

$$S24X = (y_4 \oplus y_5) \cdot (y_2 \oplus y_3)$$

$$S32X = \overline{y_5} \cdot (y_4 \cdot y_3 \cdot y_2) + y_5 \cdot \overline{(y_4 + y_3 + y_2)} \qquad \text{----(2.2)}$$

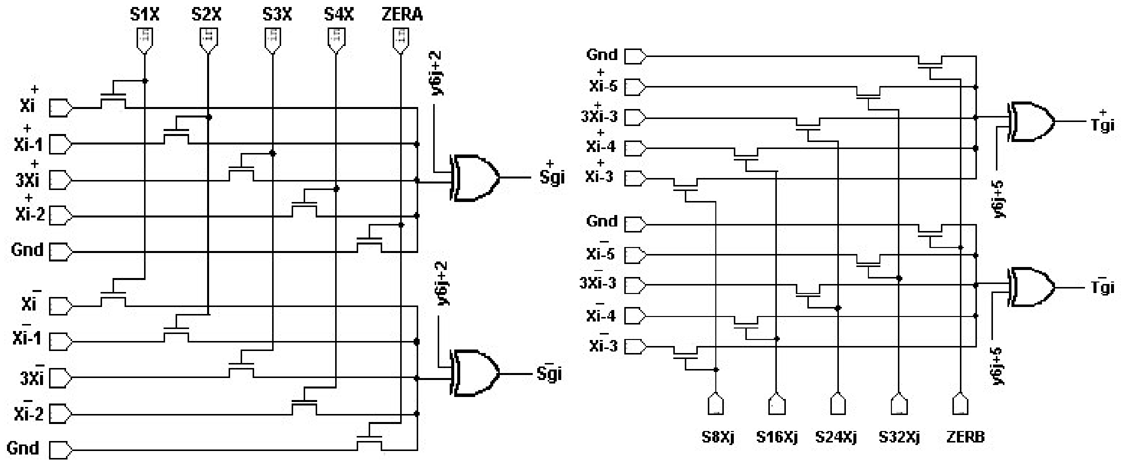


Fig. 2.9 Partial product selector.

In the example of partial product generation in Fig. 2.8 the Sgr coefficient is 1 and the Tgr coefficient is –24. So using encoder and selector circuit, we can get all the Tgr digits as X and Sgr digits as –24X. Now these two RB numbers are added using the rules of Table 1.4 discussed in chapter 1 without any carry propagation. So this method of partial product generation in the higher radix multiplier will have the least delay. The encoder and selector circuits are implemented in S-edit. The worst-case delay of this encoder and selector circuit combination is obtained. This is normalized with respect to the delay of standard XOR gate shown in Fig. 2.7 implemented with same S-edit. The normalized delay is found to be 2.92. This delay is used in chapter 5 for delay calculation of radix-64 partial product generator stage.