Chapter -1

Introduction

Multiplication is one of the basic arithmetic operations used in almost all signal processing algorithms [Oppen. 1999]. As per Oberman [Oberman 1997] 8.72% of all instructions in a typical scientific program are multiplications. In present scenario the increased level of integration brought about by modern VLSI technology has rendered possible the integration of many complex components in a single chip [Abnous 1996, Catthoor 2000, Chien 2001]. This has made the systems faster and thus useful for real time applications like mobile digital signal processing, multimedia applications, scientific computations etc. The growing market for fast floating-point processors, digital signal processing chips, and graphics processors has also created a demand for the high speed, area-efficient multipliers. Current multiplier architectures range from small, low-performance shift and add multipliers, to large, high-performance array and tree multipliers. Taking this into consideration various researchers [Balsara 1996, Gnana. 1985, Saleh 2001, Goto 1992] have developed novel algorithms and circuit techniques to provide higher speed and optimized use of silicon area.

This thesis explores most important circuit techniques, algorithms and architectures available. Along with this, new circuits and two novel architectural improvements are proposed to increase the speed of multiplication

1.1 Multiplication process

Multiplication is a three-step process [Twaijry 1997] as shown in Fig. 1.1.

I. *Generation of partial products:* Partial products are generated from the inputs. A partial product is represented by a row of dots in the figure. The partial products are all produced in parallel. There are several ways of generating the partial products. The different methods are discussed in detail in chapter 2.

II. *Summing up of partial products into two rows:* The generated partial products need to be accumulated to produce the final result. However the simple addition of each partial product involves a relatively long latency carry propagation computation. For the purpose of speeding up, all the partial product rows are first summed up to give only two rows (called sum and carry rows) by using special adder architectures, which are dealt with in detail in chapter 3.

III. *Addition of the two rows (sum and carry rows generated by summing process) by carry propagate adder:* Sum and carry rows together represent the result of multiplication. The final result is obtained only by adding sum and carry rows together. Two architectures for the final addition are explained in chapter 4.

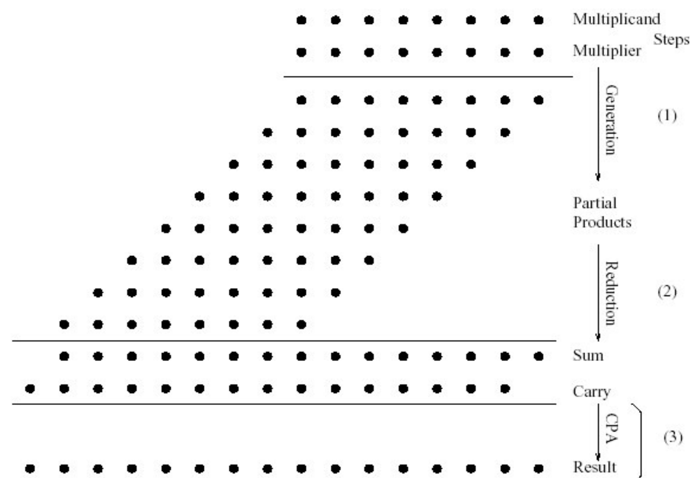

Fig. 1.1 Multiplication steps.

1.2 Multiplier architecture

Multipliers are used starting from very low speed arithmetic calculations, medium speed applications to very high-speed real time multimedia computations. So depending upon the application speed, trade offs are done at architecture and circuit levels to have small silicon area and power consumption. This thesis explores the most important circuit techniques, algorithms and architectures available for hardware implementation of parallel multipliers. Based on synthesis results for a multiplier of required operand size and preferred figure of merit (delay, power, area) the best architecture is proposed. New circuits for partial product generation, accumulation and final addition are also proposed for better performance. A novel partial product generation method using radix-64 encoding is proposed, which outperforms the existing one [Hoon 2002] in terms of delay.

1.3 Terminology

Before we start the main part, defining some terms like normalized gate delay model and basics of redundant binary (RB) number system are discussed to make the understanding of multiplier architectures self contained.

1.3.1 Normalized gate delay model

Any architectural implementation will use some basic gates. By knowing the delay of each gate we are using, we can approximately compute the latency of each specific architecture. It will definitely not give as accurate results as post layout simulation. But getting a good approximation of delay will guide us to choose a specific architecture for our requirement.

The delay model we have used is obtained from Toshiba library [Toshiba] for 130 nanometer technologies. In this, for a specific cell, a number of models are available with different driving strength. An example of a standard cell NAND2 with different models to support the required driving strength in design is given in Table 1.1. In this two input NAND

gate for any model the delay to the output from two different inputs are different. Again the delay measured at rising edge and that measured at falling edge are also different. Among all the rising edge and falling edge delays and for all the models of a standard cell the worst-case delay is taken as the delay of standard cell. In this NAND2 cell, the worst-case delay is rise delay of ND2D0 from A2 to ZN and is 103-pico sec (psec.). So the NAND2 cell delay is taken as 103-psec.

In the same way the delay for other standard cells are obtained from the Toshiba library and summarized in Table 1.2. Now all the delays are normalized with respect to a two input XOR cell delay. This normalized delay will be used for delay approximation for different architecture in chapter 2, 3 and 4.

Table 1.1 Different models of NAND2 gate taken from Toshiba library and their delay characteristic for different load

/tcb013g_8lm/NAND2 (A1 -> ZN)

model	load	gain	input cap	rise delay	fall delay
ND2D0	6	242	2.40	94	92
ND2D1	11	254	4.30	78	91
ND2D2	23	271	8.40	78	91
ND2D3	33	254	13.10	78	91
ND2D4	45	260	17.40	78	91
ND2D8	84	254	33.20	75	93

/tcb013g_8lm/NAND2 (A2 -> ZN):

model	load	gain	input cap	rise delay	fall delay
ND2D0	6	242	2.40	103	94
ND2D1	11	261	4.20	85	94
ND2D2	23	253	9.00	85	94
ND2D3	33	252	13.20	85	94
ND2D4	45	261	17.30	85	94
ND2D8	84	252	33.50	82	96

Table. 1.2 Delay characteristic of different standard cells

Cell name	Actual delay in p.sec	Normalized Delay
Two input NAND	103	$T_{NAND2} = 0.351$
Two input NOR	128	$T_{NOR2} = 0.436$
Two input AND	158	$T_{AND2} = 0.539$
Two input OR	201	$T_{OR2} = 0.686$
Two input XOR	293	$T_{XOR2} = 1$
Two input XNOR	284	$T_{XNOR2} = 0.969$
Inverter	88	$T_{INV} = 0.300$
Three input NAND	126	$T_{NAND3} = 0.430$
Three input NOR	147	$T_{NOR3} = 0.501$

1.3.2 Redundant binary (RB) number system

RB number system is a part of signed digit representation proposed by Avizienis [Avizienis 1961]. It has a fixed radix 2 and a digit set $\{\bar{1},0,1\}$ where $\bar{1}$ denotes –1. An n-digit RB integer $Y = [y_{n-1}\ y_{n-2} \ldots\ldots y_0]_{SD2}$ where $y_i \in \{\bar{1}, 0, 1\}$ has the value $\sum_{i=0}^{n-1} y_i * 2^i$. It is similar to an unsigned binary integer except that y_i can be $\bar{1}$. To represent these three symbols two bits are required. The most commonly used coding scheme, which is also used in this work, is shown in Table 1.3.

Table 1.3 RB coding scheme

X^+	X^-	RB digit
0	0	0
0	1	$\bar{1}$
1	0	1
1	1	0

1.3.3 Addition of two natural binary (NB) numbers to give a RB number

The addition of A and B is expressed as [Makino 1996]

$$A + B = A- (-B) = A – (\bar{B} +1) = (A – \bar{B}) – 1 = (A, \bar{B}) – 1 \quad (1.1)$$

So this expression indicates that two natural numbers can be added to give the result plus one by just taking the combination of A and complement of B. But to find out the actual value of A+B, one is to be subtracted. For finding out subtraction the above expression can be written as:

$$A - B = A +(-B) = A + (\bar{B} +1) = A+\bar{B} +1= (A,B) \quad (1.2)$$

An example of subtraction of B from A is shown in Fig. 1.2 using equations 1.1 and 1.2. The subtraction of B = 31 from A=19 results in –12. Now by using equation 1.1 grouping of A and $\bar{B}$ is done as shown in Fig. 1.2(a) and the RB digit corresponding to each group is

assigned looking to Table 1.3. So the result obtained in RB form has value –11. The addition of A and -B is obtained by subtracting 1 from (A, $\bar{B}$). This equation (1.1) is used to design a novel final adder in a parallel multiplier along with equivalent bit conversion algorithm in chapter 4. Now we also can use equation 1.2 to do this subtraction, which is shown in Fig 1.2(b). The analysis of use of this equation (1.2) with the previous one (1.1) shows an important conclusion that, to get the final result no carry-propagate subtraction of 1 is needed. This equation is used in radix-64 partial product generation in chapter-2 to reduce the latency significantly.

A = 0 0 0 1 0 0 1 1 (= 19)
+ (= - 12)
B = 1 1 1 0 0 0 0 1 (- 31)

A,$\bar{B}$ = (0,0) (0,0) (0,0) (1,1) (0,1) (0,1) (1,1) (1,0)

In RB 0 0 0 0 -1 -1 0 1 = -11

So A + B = (A,$\bar{B}$) -1 = -11-1 =-12

(a)

A = 00010011 (= 19)
– (= - 12)
B = 00011111 (= 31)

A,B = (0,0)(0,0)(0,0)(1,1)(0,1)(0,1)(1,1)(1,1)

In RB 0 0 0 0 -1 -1 0 0

So A - B = - 12

(b)

Fig. 1. 2 Subtraction of 31 from 19 (a) Subtraction using addition of equation 1.1 (b)Subtraction using equation 1.2.

1.3.4 Addition of two-RB numbers

The RB number representation allows representing an integer in several ways. For example, $[0101]_{SD2}$, $[011\bar{1}]_{SD2}$, $[1\bar{1}01]_{SD2}$, $[1\bar{1}1\bar{1}]_{SD2}$, $[10\bar{1}\bar{1}]_{SD2}$ all represents decimal ‘5’. Because of this property addition of two RB numbers can be performed without carry propagation in two steps [Takagi 1985].

Carry propagation free addition is performed in two steps:

First step: In this step intermediate carry C_i {$\in$($\bar{1}$, 0,1)} and the intermediate sum digit S_i {$\in$($\bar{1}$, 0,1)} are determined to satisfy the equation $a_i + b_i = 2C_i + S_i$ where a_i and b_i are augend and addend digits, respectively. But C_i and S_i are determined such that both S_i and C_{i-1} are

neither 1’s, nor are they $\bar{1}$’s. So the rules for computation for intermediate sum S_i and carry C_i looking to the augend and addend digit (X_i, Y_i) and the digits at the next lower –order position (X_{i-1}, Y_{i-1}) is given in Table 1. 4.

<u>Second step</u>: In the second step, the sum digit Z_i $\{\in(\bar{1},0,1)\}$is obtained at each position by adding the intermediate sum digit S_i and the intermediate carry C_{i-1} from lower-order position without generating a carry.

Table 1.4 Computation rules for the first step in carry propagation free addition

Augend Digit (X_i)	Addend digit (Y_i)	Digits at the next-lower-order position (X_{i-1},Y_{i-1})	Intermediate carry (C_i)	Intermediate Sum (S_i)
1	1	Any value	1	0
1 0	0 1	Both are nonnegative.	1	$\bar{1}$
		Otherwise	0	1
0	0	Any value	0	0
1	$\bar{1}$			
$\bar{1}$	1			
0 $\bar{1}$	$\bar{1}$ 0	Both are nonnegative.	0	$\bar{1}$
		Otherwise	$\bar{1}$	1
$\bar{1}$	$\bar{1}$	Any value	$\bar{1}$	0

An example of addition of two RB numbers is given in Fig.1.3. Here the two numbers 67 and 157 are represented in RB form. In the first step digits of same bit location of augend and addend are added to get the intermediate sum and intermediate carry following the rule given in Table 1.4, just looking at the digits of next lower-order position. In the next step intermediate sum and intermediate carry are added without any carry generation to give the final result.

Augend	$1\ \bar{1}\ 0\ 1\ \bar{1}\ \bar{1}\ 0\ \bar{1}$ (67)
Addend	$+\ 1\ 1\ \bar{1}\ 0\ 0\ 0\ \bar{1}\ \bar{1}$ (157)
Intermediate sum Si	$0\ 0\ \bar{1}\ 1\ 1\ 1\ 1\ 0$
Intermediate carry Ci	$+\ 1\ 0\ 0\ 0\ \bar{1}\ \bar{1}\ \bar{1}\ \bar{1}$
Sum Zi	$1\ 0\ 0\ \bar{1}\ 0\ 0\ 0\ 0\ 0$ (224)

Fig. 1.3 Example of carry propagation free addition.

Thus in redundant binary number system, parallel addition of two numbers by a combinational circuit is performed in a constant time, independent of word length of operands.

1.3.5 Some properties of RB number

<u>Getting negative of a RB number:</u> The negation of a RB number is directly derived by changing the sign of all non zero digits in the number i.e. by inverting all the bits. This is explained through an example shown in Fig. 1.4(a).

<u>Multiplying a RB number by 2^n:</u>

A RB number can be multiplied by 2^n where n is an integer by just left shifting the bits and padding zero as is done in case of NB number. The Fig.1.4 (b) explains this.

RB number = (0,0) (1,1) (0,1) (0,1) (1,1) (1,1) = -12
Negative of RB number = (1,1) (0,0) (1,0) (1,0) (0,0) (0,0) = +12

(a)

RB number = (0,0) (1,1) (0,1) (0,1) (1,1) (1,1) = -12
Left shifted RB number = (0,1) (0,1) (1,1) (1,1) (0,0) (0,0) = - 48

(b)

Fig. 1.4 Properties of RB number (a) The –ve of –12 is obtained by inverting all the bits i.e. same as changing the sign of all non zero numbers (b) Multiplication of -12 by 4 is shifting it left by two position and padding by (0,0) in two lest significant bit positions.

1.3.6 RB to NB conversion

In Fig. 1.2 the NB numbers are added or subtracted and the result obtained is in RB form. But finally the RB number must get converted back to NB number to be used for further processing. For this conversion from RB to NB, equivalent binary conversion algorithm

(EBCA) is used. The EBCA [Kim 2001, Rulling 2003] is based on the fact that the NB numbers are subset of the RB numbers. So in this algorithm attempt is made to find a suitable representation of the final RB product, which is also a valid representation in the NB domain. The algorithm is a three-step process. First the blocks of an RB word which contains 0 0 0 ----0$\bar{1}$ are replaced by an equivalent form of $\bar{1}$11 ----11. In the second step, blocks of 1$\bar{1}\bar{1}$ -----$\bar{1}\bar{1}$ are replaced by an equivalent RB form of 0 0 0 ----0 1. Replace the most significant bit (MSB) $\bar{1}$ by 1 in third step if MSB is $\bar{1}$. Use of this to convert a RB number to NB number is shown in the Fig. 1.5. The truth table [Kim 2003] for converting RB to NB is shown in the Table 1.5.

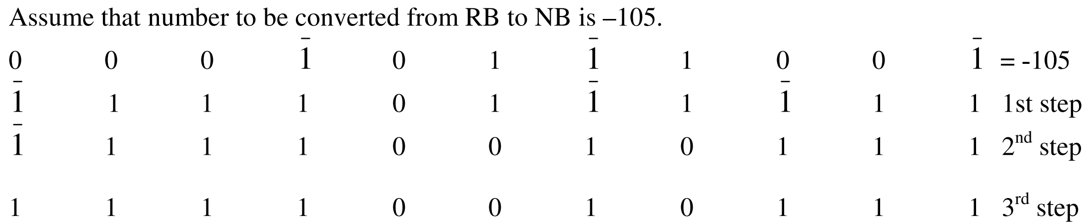

Fig. 1.5 RB to NB conversion using the equivalent bit conversion algorithm.

So the final result obtained in third step is –105 in two's complement form.

Table 1.5 Truth table for RB to NB conversion

X	Y	ENI = 0			ENI = 1		
		Z1	Z2	ENO	Z1	Z2	ENO
0	0	0	0	0	1	1	1
0	$\bar{1}$	1	1	1	1	0	1
0	1	0	1	0	0	0	0
$\bar{1}$	0	1	0	1	0	1	1
$\bar{1}$	$\bar{1}$	0	1	1	0	0	1
$\bar{1}$	1	1	1	1	1	0	1
1	0	1	0	0	0	1	0
1	$\bar{1}$	0	1	0	0	0	0
1	1	1	1	0	1	0	0

1.4 Thesis structure

The thesis is structured as follows:

Chapter 2: Deals with the circuit techniques and algorithms of partial product generation. In this various algorithms are discussed and for each algorithm circuit implementation techniques are also explored.

Chapter 3: In this chapter frequently used adder architectures for summing the generated partial products into two rows (the sum and carry rows) are discussed in terms of normalized gate delay. The advantages of one technique over the other in specific conditions are pointed out which will be useful in a complete multiplier implementation.

Chapter 4: Deals with the addition of finally generated sum and carry rows. Fastest carry-lookahead adder architectures are explored and also a novel EBCA based adder circuit is proposed which outperforms exiting circuits in terms of delay.

Chapter 5: This chapter draws upon all the circuits for parallel multipliers discussed in chapter 2, 3, 4 and explores various possible combinations for parallel multiplier implementations that are possible. Their delay performance in terms of normalized delay with respect to two input XOR gate (obtained from Toshiba library) is analyzed.

Chapter 6: In this chapter multipliers are implemented for different word lengths using different ways as discussed in chapter 5. In every case the cell count, the delay and the power consumed are estimated. Finally analyzing the post route estimates the best possible architecture for smallest delay (T), area (A), power (P), A*T*P, AT^2 is proposed.

Chapter 7: Gives conclusion of the entire study.