P-3465

LOW POWER MULTIPLIER OPTIMIZED BY MODIFIED PARTIAL PRODUCT SUMMATION

By

S.VISWANATHAN

Reg. No. 0920106018

of

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution affiliated to Anna University of Technology, Coimbatore)

COIMBATORE - 641049

A PROJECT REPORT

Submitted to the

FACULTY OF ELECTRONICS AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements

for the award of the degree

of

MASTER OF ENGINEERING

IN

APPLIED ELECTRONICS

APRIL 2011

BONAFIDE CERTIFICATE

Certified that this project report entitled "**LOW POWER MULTIPLIER OPTIMIZED BY MODIFIED PARTIAL PRODUCT SUMMATION**" is the bonafide work of **Mr.S.Viswanathan [Reg. No. 0920106018]** who carried out the research under my supervision. Certified further that, to the best of my knowledge the work reported herein does not form part of any other project or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

06-04-2011
Project Guide

Dr. A. Vasuki

Head of the Department

Dr. Rajeswari Mariappan

The candidate with university Register No. 0920106018 was examined by us in the project viva-voce examination held on 21.4.2011....

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

A project of this nature needs co-operation and support from many for successful completion. In this regard, I am fortunate to express my heartfelt thanks to Chairman **Padmabhusan Arutselvar Dr.N.Mahalingam B.Sc.,F.I.E.,** and Co-Chairman **Dr.B.K.Krishnaraj Vanavarayar B.Com,B.L.,** for providing necessary facilities throughout the course.

I would like to express my thanks and appreciation to many people who have contributed to the successful completion of this project. First I thank **Dr.J.Shanmugam Ph.D,** Director, for providing me an opportunity to carry out this project work.

I would like to thank **Dr.S.Ramachandran Ph.D,** Principal, who gave his continual support and opportunity for completing the project work successfully.

I would like to thank **Dr.Rajeswari Mariappan Ph.D,** Prof of Head, Department of Electronics and Communication Engineering, who gave her continual support for us throughout the course of study.

I would like to thank **Ms.A.Vasuki Ph.D**, Professor, Project guide for his technical guidance, constructive criticism and many valuable suggestions provided throughout the project work.

My heartfelt thanks to **Ms.R.Latha M.E (Ph.D),** Associate Professor, Project coordinator, for her contribution and innovative ideas at various stages of the project to successfully complete this work.

I express my sincere gratitude to my family members, friends and to all my staff members of Electronics and Communication Engineering Department for their support throughout the course of my project.

ABSTRACT

Multiplication is most important in digital signal processing. The objective of a good multiplier is to provide a physically compact, high speed and low power consuming chip. A low power multiplier using a dynamic range determination unit and a modified structure in the partial product summation unit is designed. The proposed multiplier is based on the modified booth algorithm and parallel operation of partial product summation which accelerates the multiplication speed. The effective dynamic ranges of two input data are estimated by the dynamic range determination unit to determine that these input data with smaller and larger dynamic range are multiplier and multiplicand for booth decoding. An efficient radix-4 recoding logic generates the partial products in a left-to-right order. The partial products are split into upper and lower groups. The partial products in high precision have high chance of being zero. The upper/lower left-to-right structure is modified by moving the correction bits from the upper part to lower part of partial product summation unit to reduce switching power. Additionally, low power adder cells instead of the conventional full adders are used in the upper and lower parts of partial product summation unit to conserve power. The proposed and conventional multipliers are implemented and simulated using Modelsim, Microwind and Xilinx. The simulated results demonstrate that the proposed multiplier consumes the least power than the conventional ones.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

CPA	Carry Propagation Adder
DRD	Dynamic Range Determination
DSP	Digital Signal Processing
FA	Full Adder
HA	Half Adder
PPS	Partial Product Summation
SE	Sign Extension
ULLR	Upper/Lower Left-to-Right
VHDL	Very High Speed Circuits Hardware Description Language

CHAPTER 1

INTRODUCTION

1.1 MULTIPLICATION

Multiplication is a commonly used operation in digital signal processing (DSP). Many DSP algorithms such as digital filters, adaptive filters, transforms and correlations, usually employ repetitive multiplication operations. Hence, the multiplier has become a basic unit in various hardware platforms. Additionally, an increase in the demand of mobile applications promotes development of low-power electronic devices. Hence, there is a need to design a low-power multiplier to reach power saving.

Multipliers are key components of many high performance systems such as finite impulse response filters, digital signal processors, etc. A system's performance is generally determined by the performance of the multiplier because it is generally the slowest element in the system. Furthermore, it occupies more area. Hence, optimizing the speed and area of the multiplier is a major design issue. However, area and speed are usually conflicting constraints so that improving speed results mostly in larger areas.

A low power design is essential to achieve long battery life in portable devices. Having minimized power also becomes critical in high performance microprocessor to keep circuit power density within low cost cooling limit. It is also clear that lowering the power can result in much cheaper packaging, resulting in cost effective Application Specific Integrated Circuit design. For these reasons, low power design has become as important as delay optimization.

In the previous work, several techniques were developed to reduce power consumed by the partial-product summation of a multiplication. Chen *et al.* developed a multiplier based on the radix-4 Booth algorithm with dynamic-range detection to minimize

switching activities of partial products, and thus reduce power consumption [2]. Hsu *et al.* adopted the right-to-left structure of a multiplier with a tiled partial product compressor tree to minimize power dissipation [3]. In Huang *et al.*'s multiplier, the upper/lower left-to-right leapfrog structure was demonstrated to be better than the general multiplier structures [4]. As compared to partial products of a conventional multiplier, Kang *et al.* generated fewer partial products using the special two's complement data representation to lower overall operational time and power consumption in Partial-Product Summation (PPS) [5].

In this work, a low-power multiplier using a Dynamic-Range Determination (DRD) unit and a modified Upper/Lower Left-to-Right (ULLR) structure in the PPS unit is developed, which utilizes a different adder cell to further conserve power dissipation. This multiplier adopts the radix-4 Booth algorithm to yield partial products. Additionally, the DRD unit is to select one of two input data with a smaller dynamic range to address Booth decoders that generate multiple partial products based on the other input datum with a larger dynamic range [6]. Such approach enables that partial products in high precision have a high chance of being zero. Additionally, the arrangement of the adder cell in the PPS unit is well addressed to effectively lower power dissipation and critical path of the PPS unit. Compared to the previous work [7], the proposed low-power multiplier not only modifies the upper/lower left-to-right structure but also decreases power consumption and critical delay of the PPS unit.

In this project, upper/lower left-to-right approach is used for reducing power in the partial product summation unit. It also increases multiplication speed to a great extent than the conventional design. This project is designed using VHDL and power analysis is to be done using Xilinx to present practical proofs, that the proposed approach is better in terms of area, critical path and power.

1.2 INTRODUCTION TO VHDL

VHDL is an acronym which stands for VHSIC Hardware Description Language. VHSIC is yet another acronym which stands for Very High Speed Integrated Circuits. It

is being used for documentation, verification, and synthesis of large digital designs. VHDL is a standard (VHDL-1076) developed by IEEE. The different approaches in VHDL are structural, data flow, and behavioral methods of hardware description.

1.2.1 STRUCTURAL DESCRIPTIONS

The structural descriptions are explained below with examples.

1.2.1.1 BUILDING BLOCKS

Every portion of a VHDL design is considered a block. A VHDL design may be completely described in a single block, or it may be decomposed in several blocks. Each block in VHDL is analogous to an off-the-shelf part and is called an entity. The entity describes the interface to that block and a separate part associated with the entity describes how that block operates. The interface description is like a pin description in a data book, specifying the inputs and outputs to the block.

The following is an example of an entity declaration in VHDL.

```
entity latch is
port (s,r: in bit;
q,nq: out bit);
end latch;
```

The first line indicates a definition of a new entity, whose name is latch. The last line marks the end of the definition. The lines in between, called the port clause, describe the interface to the design. The port clause contains a list of interface declarations. Each interface declaration defines one or more signals that are inputs or outputs to the design. Each interface declaration contains a list of names, a mode, and a type.

The following is an example of an architecture declaration for the latch entity.

```
architecture dataflow of latch is
signal q0 : bit := '0';
```

```
signal nq0 : bit := '1';
begin
q0<=r nor nq0;
nq0<=s nor q0;
nq<=nq0;
q<=q0;
end dataflow;
```

The first line of the declaration indicates that this is the definition of a new architecture called dataflow and it belongs to the entity named latch. So this architecture describes the operation of the latch entity. The schematic for the latch might be

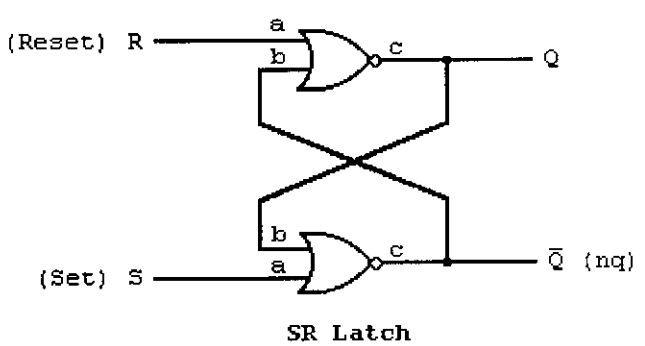


Fig1.1 Schematic SR Latch

The same connections that occur in the schematic can be specified using VHDL with the following architecture declaration:

```
architecture structure of latch is
 component nor_gate
     port (a,b: in bit;
           c: out bit);
    end component;
 begin
 n1: nor_gate
 port map (r,nq,q);
 n2: nor_gate
```

```
port map (s,q,nq);
end structure;
```

The lines between the first and the keyword begin are a component declaration. A list of components and there connections in any language is sometimes called a netlist. The structural description of a design in VHDL is one of many means of specifying netlists.

1.2.2 DATA FLOW DESCRIPTIONS

In the data flow approach, circuits are described by indicating how the inputs and outputs of built-in primitive components are connected together.

1.2.2.1 EXAMPLE

SR latch is described using VHDL as in the following schematic.

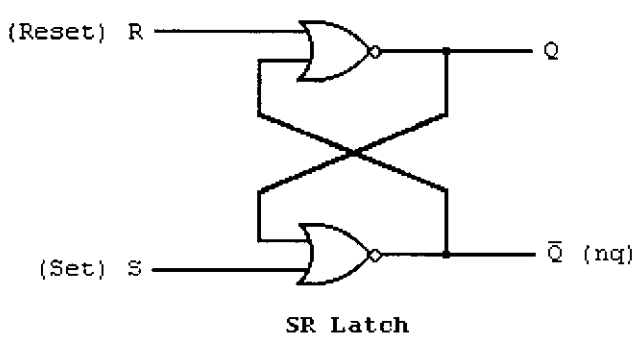


Fig 1.2 Data flow approach in SR Latch

The entity part is written in the following manner.

```
entity latch is

 port (s,r : in bit;
     q,nq : out bit);
end latch;
```

```
architecture dataflow of latch is
begin
 q<=r nor nq;
 nq<=s nor q;
end dataflow;
```

The signal assignment operator in VHDL specifies a relationship between signals, not a transfer of data as in programming languages. The architecture part describes the internal operation of the design. The scheme used to model a VHDL design is called discrete event time simulation. In this the values of signals are only updated when certain events occur and events occur at discrete instances of time. The above mentioned SR latch works with this type of simulation.

1.2.2.2 THE DELAY MODEL

This section refers to the delay model. The two models of delay are used in VHDL. The first is called the inertial delay model. The inertial delay model is specified by adding an after clause to the signal assignment statement. The second is called transport delay model.

1.2.3 BEHAVIORAL DESCRIPTIONS

The behavioral approach to modeling hardware components is different from the other two methods in that it does not necessarily in any way reflect how the design is implemented.

1.2.3.1 THE PROCESS STATEMENT

It is basically the black box approach to modeling. It accurately models what happens on the inputs and outputs of the black box, but what is inside the box (how it works) is irrelevant. The behavioral description is usually used in two ways in VHDL. First, it can be used to model complex components.

Behavioral descriptions are supported with the process statement. The process statement can appear in the body of an architecture declaration just as the signal assignment statement does. The process statement can also contain signal assignments in order to specify the outputs of the process.

1.2.3.2 USING VARIABLES

A variable is used to hold data and also it behaves like you would expect in a software programming language, which is much different than the behavior of a signal. Although variables represent data like the signal, they do not have or cause events and are modified differently. Variables are modified with the variable assignment

1.2.3.3 SEQUENTIAL STATEMENTS

There are several statements that may only be used in the body of a process. These statements are called sequential statements because they are executed sequentially. The types of statements used here are if, if else, for and loop.

1.2.3.4 SIGNALS AND PROCESSES

This section is short, but contains important information about the use of signals in the process statement. A signal assignment, if anything, merely schedules an event to occur on a signal and does not have an immediate effect. When a process is resumed, it executes from top to bottom and no events are processed until after the process is complete.

1.2.3.5 PROGRAM OUTPUT

In most programming languages there is a mechanism for printing text on the monitor and getting input from the user through the keyboard. It can able to give output certain information during simulation. A standard library that comes with every VHDL language system. In VHDL, common code can be put in a separate file to be used by many designs. This common code is called a library. In order to use the library that provides input and output capabilities you must add the statement use textio.all;

immediately before every architecture that uses input and output. The write statement can be used to append constant values and the value of variables and signals of the types bit, bit_vector, time, integer, and real.

1.3 SOFTWARE USED

- Modelsim PE 5.4e
- Xilinx ISE 7.1i
- Microwind

1.4 ORGANIZATION OF THE REPORT

- **Chapter 2** gives an overview of multipliers.
- **Chapter 3** discusses about the low power techniques in multipliers.
- **Chapter 4** discusses about the design of proposed multiplier and compares it with the conventional multipliers.
- **Chapter 5** shows the simulated results.
- **Chapter 6** gives the conclusion and future scope of the project.

CHAPTER 2

OVERVIEW OF MULTIPLIERS

2.1 IMPLEMENTATION OF MULTIPLIERS

As the scale of integration keeps growing, more and more sophisticated signal processing systems are being implemented on a VLSI chip. These signal processing applications not only demand great computation capacity but also consume considerable amount of energy. While performance and Area remain to be the two major design tolls, power consumption has become a critical concern in today's VLSI system design. Power dissipation is recognized as a critical parameter in modern VLSI design field. To satisfy MOORE'S law and to produce consumer electronics goods with more backup and less weight, low power VLSI design is necessary. Dynamic power dissipation which is the major part of total power dissipation is due to the charging and discharging capacitance in the circuit. The golden formula for calculation of dynamic power dissipation is $Pd = C_L V^2 f$. Power reduction can be achieved by, reduction of output Capacitance C_L, reduction of power supply voltage V, reduction of switching activity and clock frequency f.

Fast multipliers are essential parts of digital signal processing systems. The speed of multiply operation is of great importance in digital signal processing as well as in the general purpose processors today, especially since the media processing took off. In the past multiplication was generally implemented by a sequence of addition, subtraction, and shift operations. Multiplication can be considered as a series of repeated additions. The number to be added is the multiplicand, the number of times that it is added is the multiplier, and the result is the product. Each step of addition generates a partial product. In most computers, the operand usually contains the same number of bits. When the operands are interpreted as integers, the product is generally twice the length of operands in order to preserve the information content. This repeated addition method that is

suggested by the arithmetic definition is slow that it is almost always replaced by an algorithm that makes use of positional representation. It is possible to decompose multipliers into two parts. The first part is dedicated to the generation of partial products, and the second one collects and adds them.

The basic multiplication principle is two fold i.e, evaluation of partial products and accumulation of the shifted partial products. It is performed by the successive additions of the columns of the shifted partial product matrix. The 'multiplier' is successfully shifted and gates the appropriate bit of the 'multiplicand'. The delayed, gated instance of the multiplicand must all be in the same column of the shifted partial product matrix. They are then added to form the product bit for the particular form. Multiplication is therefore a multi operand operation.

The multiplier operand Y is often recoded into a radix higher than 2 in order to reduce the number of partial products. The most common recoding is radix-4 recoding with the digit set $\{-2, -1, 0, 1, 2\}$. For a series of consecutive 1's, the recoding algorithm converts them into 0's surrounded by a 1 and a (-1), which has the potential of reducing switching activity. At the binary level, there are many design possibilities.

There are number of techniques that to perform binary multiplication. In general, the choice is based upon factors such as latency, throughput, area, and design complexity. More efficient parallel approach uses some sort of array or tree of full adders to sum partial products. Array multiplier, Booth Multiplier and Wallace Tree multipliers are some of the standard approaches to have hardware implementation of binary multiplier which are suitable for VLSI implementation at complementary metal oxide semiconductor level.

2.2 ARRAY MULTIPLIER

Binary multiplication of positive operands can be implemented in a combinational two dimensional logic array. A 4X4 array multiplier is shown in Fig 2.1. The functions of M0, M1, M2, and M4 are also shown in the figure. X3X2X1X0 is the 4 bit multiplicand

and Y3Y2Y1Y0 is the 4 bit multiplier. The main component in each cell is a full adder. The AND gate in each cell determines whether a multiplicand bit, Xj is added to the incoming partial product bit based on the value of the multiplier bit Yi. Each row adds the multiplicand (appropriately shifted) to the incoming partial product, PPi to generate the outgoing partial product PP(i+1), if Yi =1. If Yi=0, PPi is passed vertically downward unchanged. The worst case signal propagation delay path is from the upper right corner of the array to the high order product bit output at the bottom left corner of the array. Assuming that there are two gate delays from the inputs to the outputs of a full adder block.

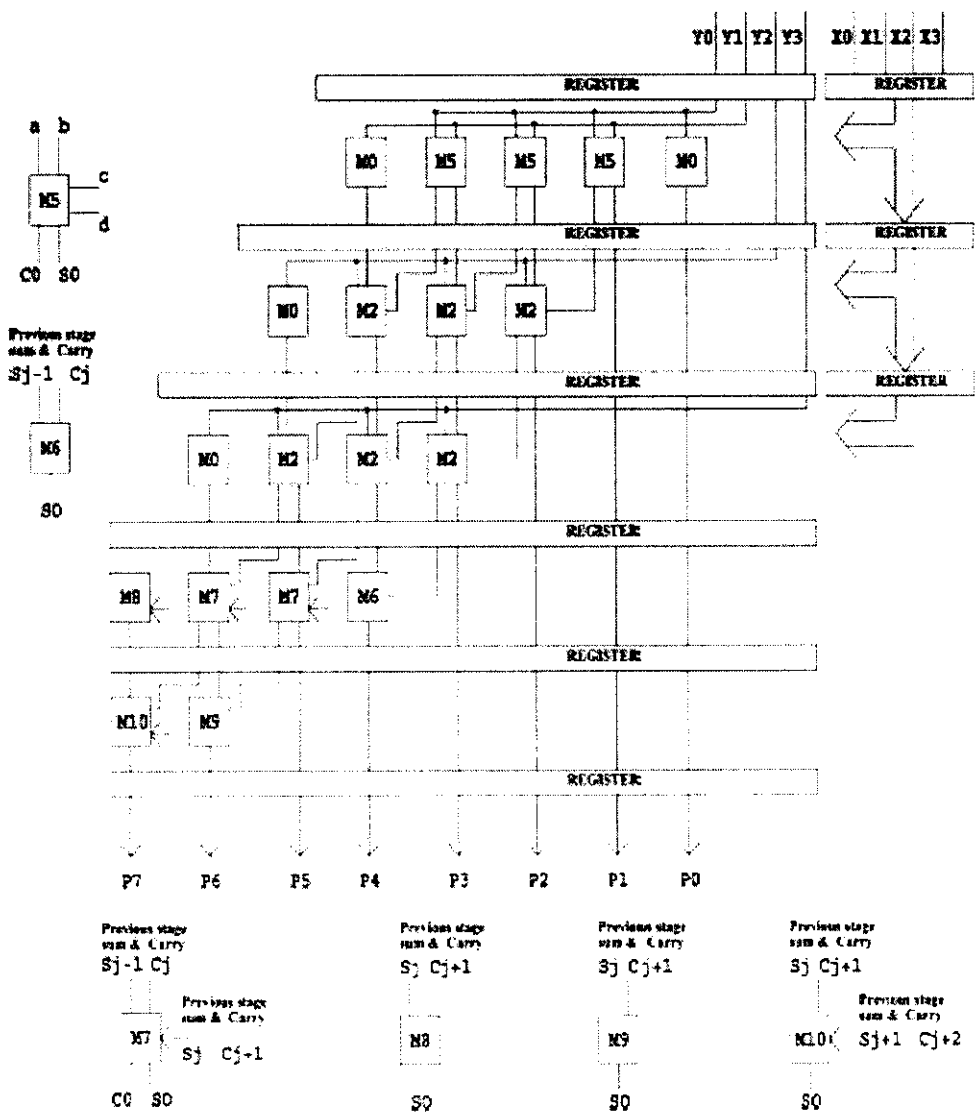


Fig 2.1 array multiplier

Array multipliers gives more power consumption as well as optimum number of components required, but the delay of this multiplier is larger. It also requires larger number of gates because of which the area is larger. Since the array multiplier is having low speed and consumes more power, low power techniques have to be incorporated into the design.

2.3 WALLACE TREE MULTIPLIER

A fast process for multiplication of two numbers was developed by Wallace. Using this method, a three step process is used to multiply two numbers; the bit products are formed, the bit product matrix is reduced to a two row matrix where sum of the row equals the sum of bit products, and the two resulting rows are summed with a fast adder to produce a final product. Three bit signals are passed to a one bit full adder ("3W") which is called a three input Wallace tree circuit, and the output signal (sum signal) is supplied to the next stage full adder of the same bit, and the carry output signal thereof is passed to the next stage full adder of the same no of bit, and the carry output signal thereof is supplied to the next stage of the full adder located at a one bit higher position.

The Wallace tree multiplier is considerably faster than a simple array multiplier because its height is logarithmic in word size, not linear. However, in addition to the large number of adders required, the Wallace tree's wiring is much less regular and more complicated. As a result, Wallace trees are often avoided by designers, while ***design complexity*** is a concern to them.

Wallace tree styles use a ***log-depth*** tree network for reduction. Faster, but irregular, they trade ease of layout for speed. Wallace tree styles are generally avoided for low power applications, since excess of wiring is likely to consume extra power.

While subsequently faster than carry-save structure for large bit multipliers, the Wallace tree multiplier has the disadvantage of being very irregular, which complicates the task of coming with an efficient layout.

The Wallace tree multiplier is a high speed multiplier. The summing of the partial product bits in parallel using a tree of carry-save adders became generally known as the **"Wallace Tree".** Three step processes are used to multiply two numbers.

1. Formation of bit products.

2. Reduction of the bit product matrix into a two row matrix by means of a carry save adder.

3. Summation of remaining two rows using a fast Adder.

Wallace tree is a tree of carry-save adders arranged as shown in Fig 2.2. A carry save adder consists of full adders like the more familiar ripple adders, but the carry output from each bit is brought out to form second result vector rather being than wired to the next most significant bit. The carry vector is 'saved' to be combined with the sum later. In the Wallace tree method, the circuit layout is not easy although the speed of the operation is high since the circuit is quite irregular.

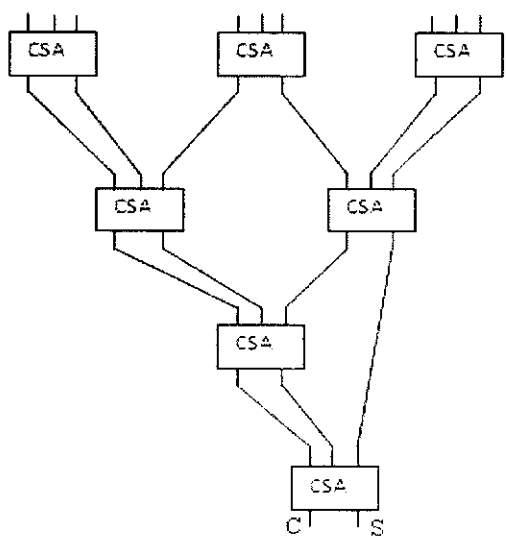


Fig 2.2 Wallace tree multiplier

2.4 BRAUN MULTIPLIER

Fig 2.3 shows the structure of 4*4 Braun multiplier. An n*n bit Braun Multiplier requires n(n-1) adders and n^2 AND gates. In this technique each partial product is added to previous sum of partial products by using row of adders. The Carry-out signals are

shifted one bit to the left and then added to the sum of the first adder which is the addition of partial product bits. The shifting of carry-out bits to the left is done by carry-save adder. As carry bits are passed diagonally downward to the next adder stage, there is no horizontal carry propagation for the first four rows. Instead, the respective carry bit is "saved" for the subsequent adder stage.

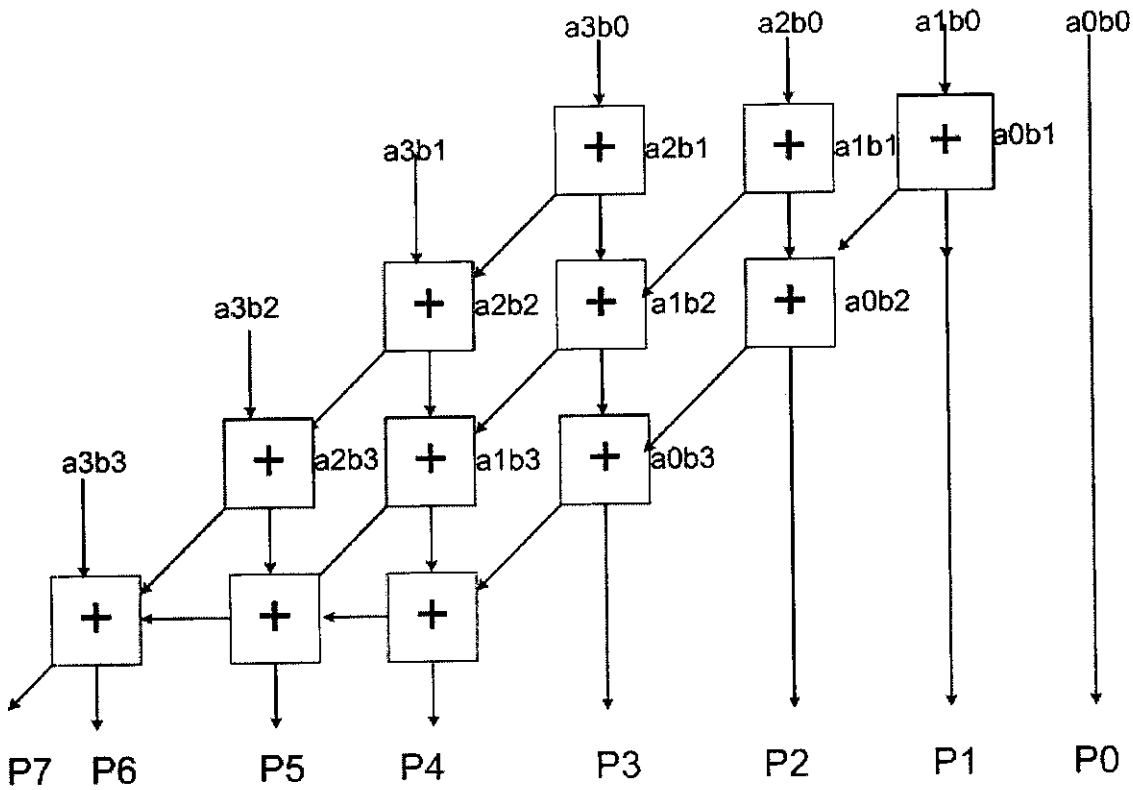


Fig 2.3 Braun multiplier

2.5 BAUGH WOOLEY MULTIPLIER

The Baugh Wooley (BW) algorithm is a relatively straightforward way of doing signed multiplications. Fig 2.3 illustrates the algorithm for an 8-bit case, where the partial product bits have been reorganized according to Hatamian's scheme. The creation of the reorganized partial-product array of an N-bit wide multiplier comprises three steps: *i)* The most significant bit (MSB) of the first $N-1$ partial-product rows and all bits of the last partial-product row, except its MSB, are inverted. *ii)* A '1' is added to the Nth column. *iii)* The MSB of the final result is inverted.

An algorithm for direct 2's complement array multiplication has been proposed by Baugh and Wooley and this algorithm is used in the design of multiplier and accumulator structures. The primary advantage of this algorithm is that the signs of all the partial products are positive, and thus allowing the array to be entirely the same as conventional standard array structures.

The following are some of the highlights of the Baugh-Wooley algorithm

- Algorithm for two's-complement multiplication.
- Adjusts partial products to maximize regularity of array multiplication.
- Moves partial products with negative signs to the last step; also adds negation of partial products rather than subtracts.

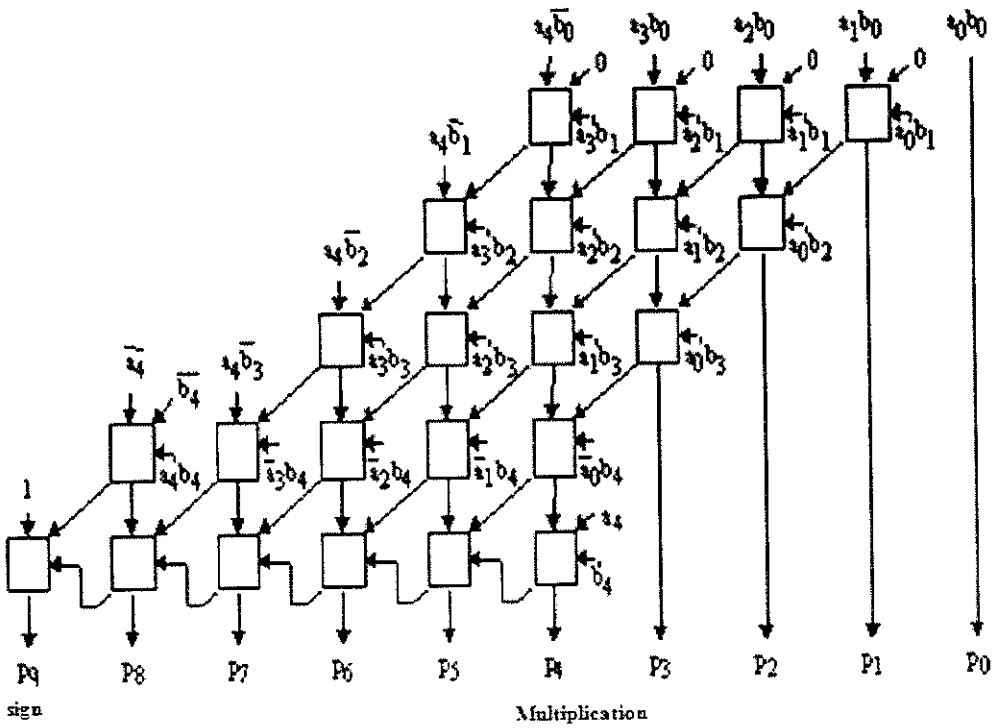


Fig 2.4 Baugh-Wooley Algorithm for Unsigned Multiplier

The Baugh-Wooley algorithm for the unsigned binary multiplication is based on the concept shown in Fig 2.4. The algorithm specifies that all possible AND terms are created first, and then sent through an array of half-adders and full-adders with the carry-outs chained to the next most significant bit at each level of addition.

For signed multiplication (by utilizing the properties of the two's complement system) the Baugh-Wooley algorithm can implement signed multiplication in almost the same way as the unsigned multiplication shown above

The Baugh Wooley algorithmic is used to multiply 2's compliment numbers using a regular iterative adder structure. For example, if x and y are two n-bit numbers, their product can be defined as:

$$P = 2^{2n-2}x_{n-1}y_{n-1} + \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 2^{i+j}x_i y_j$$
$$+ 2^{n-1}\left(\sum_{i=0}^{n-2} 2^i\, y_{n-1}\, x_i + \sum_{i=0}^{n-2} 2^j\, x_{n-1}\, y_j\right)$$
$$+ 2^n + 2^{2n-1}$$

where x and y are in 2's complement format. This algorithm performs the multiplication using only addition of positive bit products. This simplifies the hardware needed to implement the algorithm.

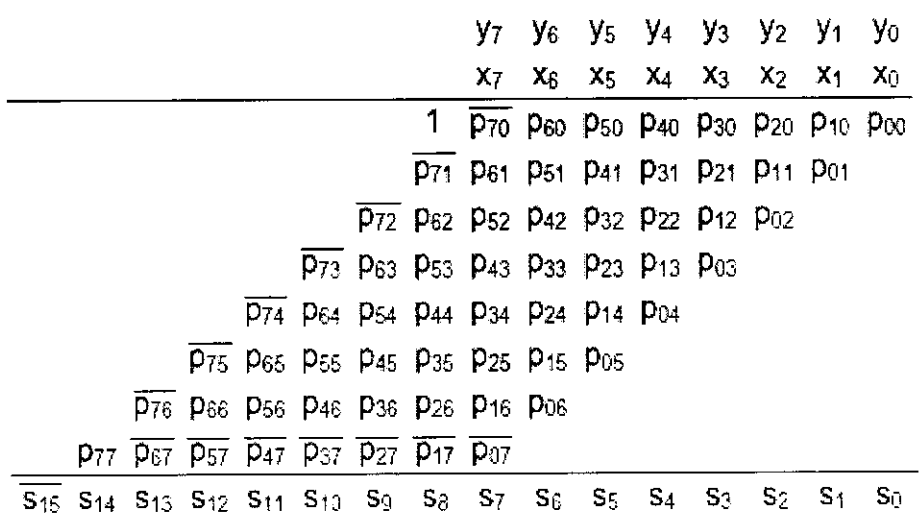


Fig 2.5 Baugh Wooley multiplier

Implementing the BW multiplier based on the HPM tree is as straightforward as the basic algorithm itself. The partial-product bits can be generated by using a 2-input AND gate for each pair of operand bits. In the case a partial-product bit should be inverted, we employ a 2-input NAND gate instead. The insertion of '1' in column N is easily accommodated by changing the half adder at top of row N to a full adder with one

of the input signals connected to '1'. Finally, the inversion of the MSB of the result is done by adding an inverter. The final result of the implementation of the BW algorithm is depicted in Fig 2.5. The Baugh–Wooley algorithm (BWA) has been developed and they have been applied to various digital filtering calculations. But the speed of the multiplier is low.

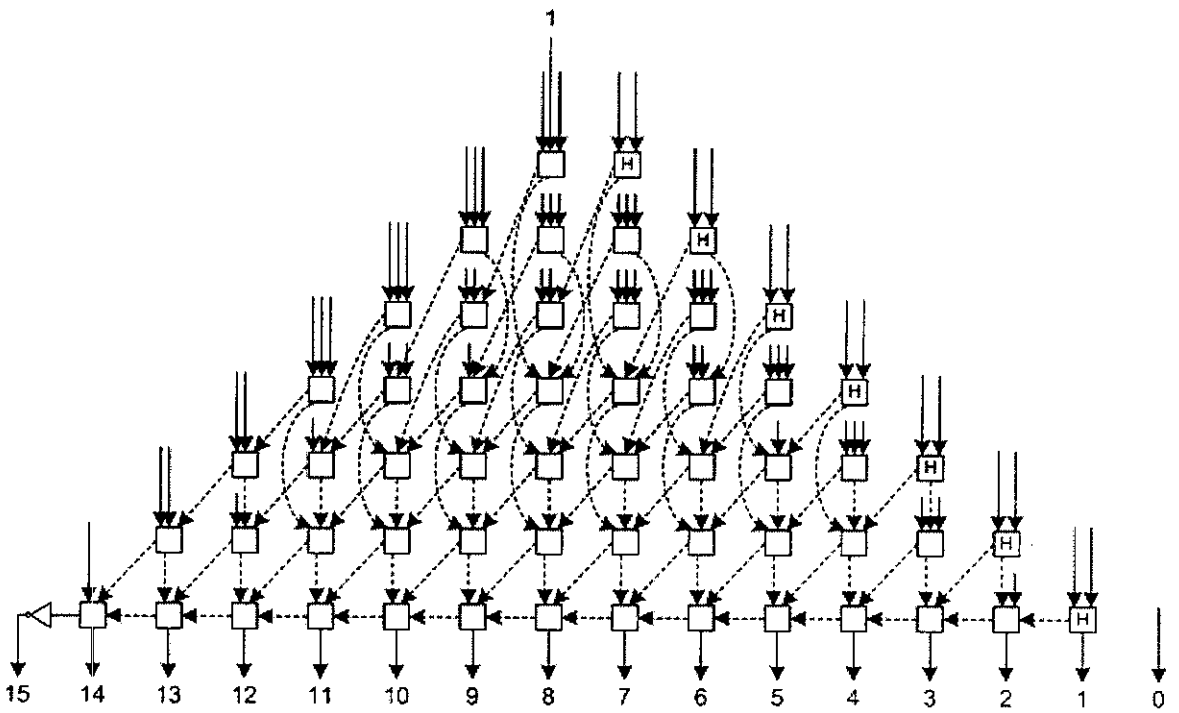


Fig 2.6 8-bit Baugh-Wooley multiplier using an HPM reduction tree.

2.6 BOOTH MULTIPLIER

Though Wallace Tree multipliers were faster than the traditional Carry Save Method, it also was very irregular and hence was complicated while drawing the Layouts. Slowly when multiplier bits gets beyond 32-bits large numbers of logic gates are required and hence also more interconnecting wires which makes chip design large and slows down operating speed. Booth multiplier can be used in different modes such as radix-2, radix-4, radix-8 etc. But Radix-4 Booth's Algorithm reduces the number of partial products from n to *n/2*.

Another improvement in the multiplier is by reducing the number of partial products generated. The Booth recording multiplier is one such multiplier; it scans the three bits at a time to reduce the number of partial products. These three bits are: the two bit from the present pair; and a third bit from the high order bit of an adjacent lower order pair. After examining each triplet of bits, the triplets are converted by Booth logic into a set of five control signals used by the adder cells in the array to control the operations performed by the adder cells. To speed up the multiplication Booth encoding performs several steps of multiplication at once. Booth's algorithm takes advantage of the fact that an adder- subtractor is nearly as fast and small as a simple adder.

From the basics of Booth Multiplication it can be proved that the addition/subtraction operation can be skipped if the successive bits in the multiplicand are same. If three consecutive bits are same then addition or subtraction operation can be skipped. Thus in most of the cases the delay associated with Booth Multiplication are smaller than that with Array Multiplier. However the performance of Booth Multiplier for delay is input data dependant. In the worst case the delay with booth multiplier is on per with Array Multiplier.

The method of Booth recoding reduces the numbers of adders and hence the delay required to produce the partial sums by examining three bits at a time. The high performance of booth multiplier comes with the drawback of power consumption. The reason is large number of adder cells required that consumes large power.

Multiplication means partial product generation and accumulation. Booth algorithm used to reduce the number of partial products to be generated in order to achieve high speed and minimum area. Booth algorithm will speed up the multiplication process. Booth multiplier encoding diagram is shown in Fig 2.6

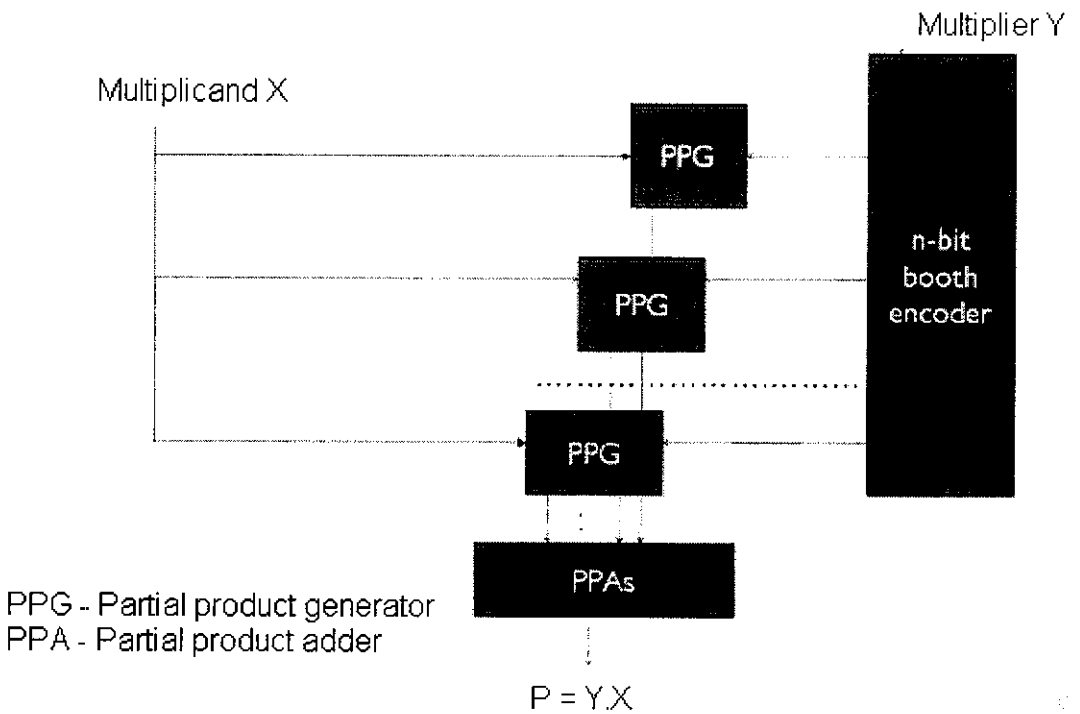


Fig 2.7 Booth Multiplier Structure

2.6.1 BOOTH ALGORITHM STEPS

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation. The Booth algorithm can be explained with the following example:

Example, 2_{ten} x $(-4)_{ten}$

0010_{two}* 1100_{two}

Step 1: Making the Booth table

1. From the two numbers, pick the number with the smallest difference between a series of Consecutive numbers, and make it a multiplier.

   i.e., 0010 -- From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change, and so there are two changes on this one.

   1100 -- From 1 to 1 no change, 1 to 0 one change, 0 to 0 no change, so there is only one change on this one.

2. Let X = 1100 (multiplier)
   Let Y = 0010 (multiplicand)
   Take the 2's complement of Y and call it –Y
   –Y = 1110

3. Load the X value in the table.

4. Load 0 for X-1 value it should be the previous first least significant bit of X.

5. Load 0 in U and V rows which will have the product of X and Y at the end of operation.

6. Make four rows for each cycle; this is because multiplication is for four bit numbers.

U	V	X	X-1
0000	0000	1100	0

Fig 2.8 U V Method

Step 2: Booth Algorithm

Booth algorithm requires examination of the multiplier bits, and shifting of the partial product. Prior to the shifting, the multiplicand may be added to partial product, subtracted from the partial product, or left unchanged according to the following rules:

Look at the first least significant bits of the multiplier "X", and the previous least significant bits of the multiplier "X - 1".

1. 0 0 Shift only

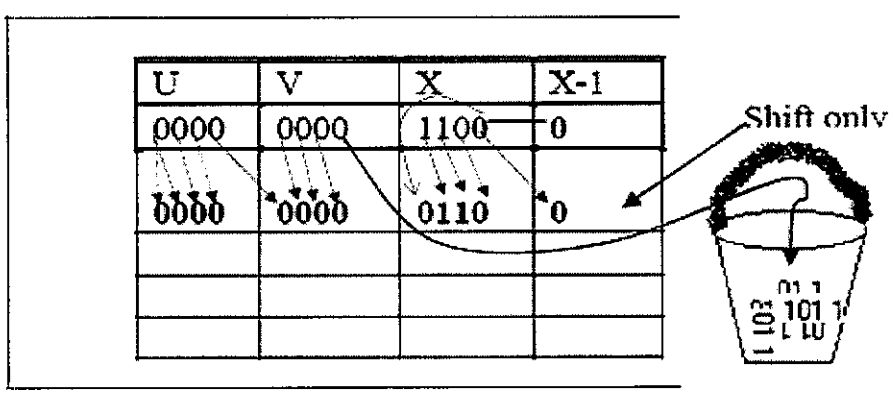
   1 1 Shift only.

   0 1 Add Y to U, and shift

   1 0 Subtract Y from U, and shift or add (-Y) to U and shift

2. Take U & V together and shift arithmetic right shift which preserves the sign bit of 2's complement number. Thus a positive number remains positive, and a negative number remains negative.

3. Shift X circular right shift because this will prevent us from using two registers for the X value.



Repeat the steps until the four cycles are completed

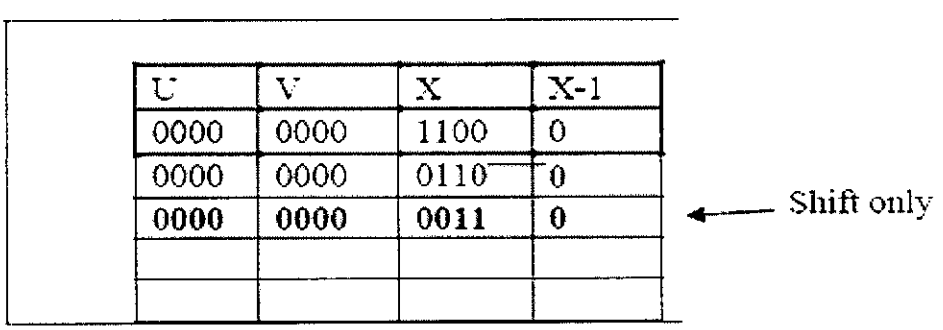


Fig 2.9 Intermediate Results of Multiplication in U and V Method

The process of comparing X and X-1 positions and recode the position with new number and carry the operation up to four cycles. We are doing four cycles since four bits are there in the operands.

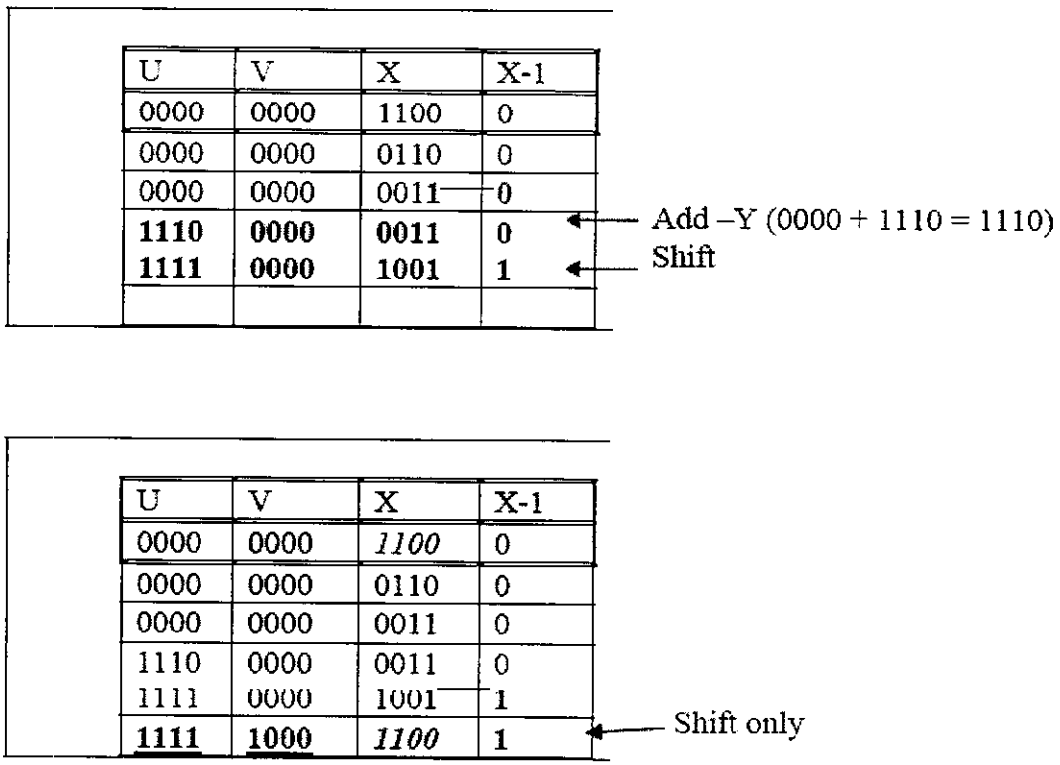


Fig 2.10 Final results of multiplication in U and V method

After doing the operation for four cycles we will get values in the columns u and v. The values of u and v together give the value of the multiplication result. This way of representation is called UV method of multiplication in booth multiplication.

Radix-4 multipliers could consume less power than their radix-2 counterparts as recoding reduces the number of PPs to half. However, the extra recoding logic and the more complex PP generation logic may present significant overheads. In addition, recoding introduces extra unbalanced signal propagation paths because of the additional delay on the paths from operand *Y* to the product output.

2.6.2 MODIFIED BOOTH ALGORITHM

Different multiplication algorithms vary in the approaches of PPG, PPR, and CPA. For PPG, radix-2 is the easiest. To reduce the number of PPs and consequently reduce the area/delay of PP reduction, one operand is usually recoded into high-radix digit sets. The most popular one is the radix-4 digit set {-2,-1, 0, 1, 2}. For PPR, two alternatives exist: reduction by rows, performed by an array of adders, and reduction by columns, performed by an array of counters. The final CPA requires a fast adder scheme because it is on the *critical path.* In some cases, final CPA is postponed if it is advantageous to keep redundant results from PPG for further arithmetic operations. One of the solutions realizing high speed multipliers is to enhance parallelism which helps in decreasing the number of subsequent calculation stages. The Original version of Booth's multiplier (Radix – 2) had two drawbacks.

1. The number of add / subtract operations became variable and hence became inconvenient while designing Parallel multipliers.

2. The algorithm becomes inefficient when there are isolated 1's.

These problems are overcome by using Radix 4 Booth's Algorithm which can scan string of three bits. The partial product generation unit makes use of modified booth algorithm. The steps in the algorithm are

1. Pad the LSB with one zero.
2. Pad the MSB with 2 zeros if n is even and 1 zero if n is odd.
3. Divide the multiplier into overlapping groups of 3-bits.
4. Determine partial product scale factor from modified booth 2 encoding table.
5. Compute the Multiplicand Multiples
6. Sum Partial Product

In Booth algorithm recoding is done by combining two bits at a time. If isolated number of ones is present in booth algorithm then recoding won't reduce number of partial products .there for a modification is done by recoding three bits at the same time this is called modified booth algorithm.

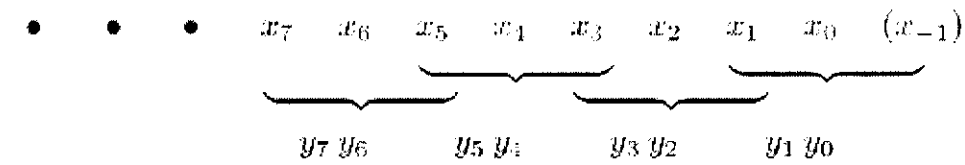


Fig 2.11 Recoding of Modified Booth Multiplier

Table 2.1 Modified booth recoding table

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Operation on X	ADD to LSB
0	0	0	0 . X	0
0	0	1	+1 . X	0
0	1	0	+1 . X	0
0	1	1	+2 . X	0
1	0	0	-2 . X	+1
1	0	1	-1 . X	+1
1	1	0	-1 . X	+1
1	1	1	0 . X	0

Table 2.2 Comparison between multipliers

Parameter	Array Multiplier	Wallace Tree Multiplier	Booth's Multiplier
Operation Speed	Less	High	Highest because the cycle length is as small as possible.
Time Delay	More (n+1)tFA	Log(n)	Less (ntFA/2 + ntFA)
Area	Maximum area because it uses a large number of adders.	Medium area because Wallace Tree used to reduce operands.	Minimum area because adder/subtracter is almost as small/fast as adder.
Complexity	Less complex	More complex	Most complex
Power consumption	Most	More	Less
FPGA implementation	Less efficient	Not efficient	Most efficient

CHAPTER 3

LOW POWER TECHNIQUES

3.1 EXISTING TECHNIQUES

The design techniques such as glitching power minimization by selective gate freezing, Fast power efficient circuit lock switch-off schemes, power-aware scalable pipelined Booth multiplier [6], partially guarded computation (PGC), High performance low power left to right array multiplier design [5] are existing works that reduce the dynamic power consumption by minimizing the switched capacitance.

Glitching power minimization by selective gate freezing replaces some existing gates with functionally equivalent ones that can be frozen by asserting a control signal. Spurious transitions (also called *glitches*) in combinational CMOS logic are a well-known source of unnecessary power dissipation. Reducing glitch power is a highly desirable target because in the vast majority of digital CMOS circuits, only one signal transition per clock cycle is functionally meaningful. Unfortunately, glitch power is heavily dependent on the low-level implementation details, namely, gate propagation delays and input transitions misalignments. Glitches are eliminated by adding some redundant logic that prevents spurious transitions. This can be done by inserting latches in a gate-level netlist and controlling the *latch-enable* pins with redundant signals generated *ad hoc*. Gates with high spurious activity are replaced with functionally equivalent ones (called *F-Gates*) that can be made insensitive to input transitions by asserting a control signal (called *C-Signal*). Although this is functionally equivalent to latch insertion, it can be implemented much more efficiently (area and power-wise) and the modified gates can replace directly the original gates in a placed and routed netlist. This technique has the distinctive feature of being applicable after layout, since it performs *in-place* optimization on the placed and routed description, and it only requires

incremental rewiring. It can only achieve savings of 6.3% in total power dissipation since it operates in the layout environment which is tightly restricted.

```
Procedure GateFreeze (M, L, N) {
1   (RT[ ], AT[ ] = static timing analysis(M);
2   POWER[ ] = Power simulation (M);
3   G = Select candidates (M, Power, AT, N);
    For each (gate g in G) {
4   M = Replace gate (M, L, g);
5   F_G = Compute control circ (M, AT(g), T);
6   M  = Add control circ (M, g, F_G);
    }
    }
```

Fig 3.1 Gate freezing algorithm

The Fast power efficient circuit block switch-off scheme proposes a double switch circuit block switch scheme capable of reducing power dissipation during down time by shortening the settling time after reactivation. The drawbacks of this scheme are the necessity for two independent virtual rails and the necessity for two additional transistors for switching each cell.

A power-aware scalable pipelined Booth multiplier design presents a multiplier using Dynamic Range Detection (DRD) unit is used to select the input operand with a smaller effective dynamic range to yield the Booth codes. There are three separate Wallace trees for the 4X4, 8X8, and 16X16 multiplications. This will certainly induce area and capacitance penalties. Under a 0.13μm CMOS technology, this design can obtain a 20% power reduction over the conventional multipliers. But there is a 44% in area overhead.

Partially guarded computation (PGC) divides the arithmetic units, e.g., adders, and multipliers, into two parts, and turns off the unused part to minimize the power consumption. The functional unit is divided into two parts namely, MSP (Most

Significant Part) and LSP (Least Significant Part). The functional unit performs only the LSP computation if the range of the output data is covered by LSP. This technique disables a part of functional unit based on dynamic range of input operands. The word length of functional unit is determined based on dynamic range of input data such that maximum range of the data does not exceed the word length of functional units. The real data is generally limited to small range in most cases, and the case of maximum range rarely occurs. The reported results show that the PGC can reduce power consumption by 10% to 40% in an array multiplier but with 30% to 36% area overhead. In all the above techniques there is no improvement in speed.

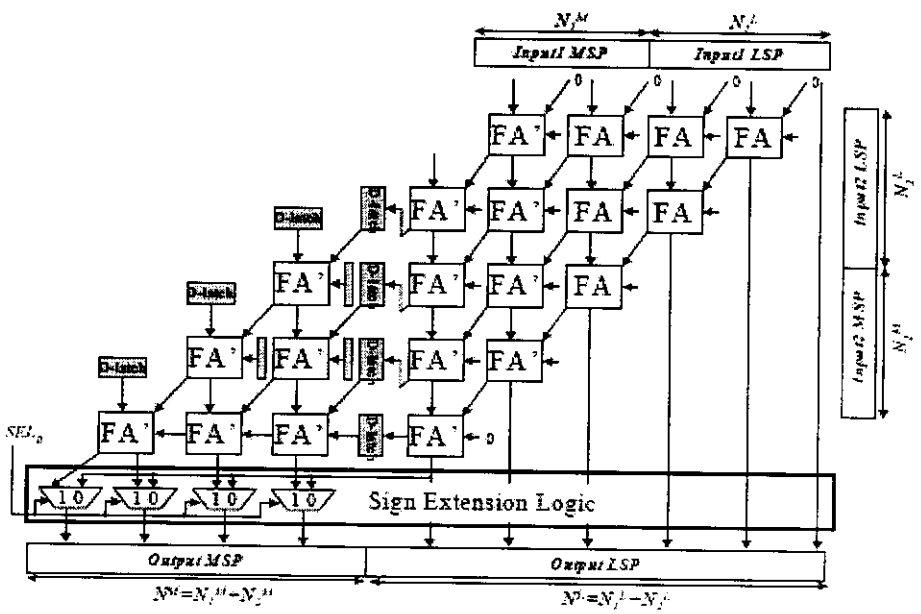


Fig 3.2 Partially guarded circuitry

3.2 LEFT-TO-RIGHT LEAPFROG STRUCTURE

The delay gap between tree multipliers and array multipliers is mainly due to the linear PPR structure in conventional array multipliers. To improve the speed of array multipliers, parallelism is introduced in PPR.

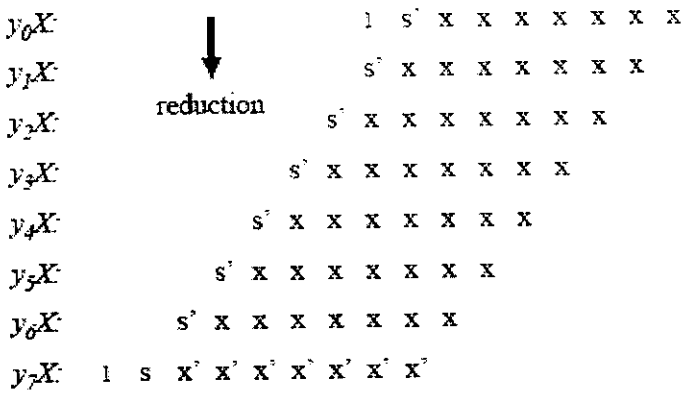


Fig 3.3 Radix 2 PP bit matrix Right to Left (n=8)

For data with a large dynamic range, PPs corresponding to sign extension bits are located in the upper region of an LR array. If sign extension bits switch less frequently than other bits as in many multimedia data, glitches can be reduced because the upper portion is not polluted by frequent switches in the lower portion in LR arrangement.

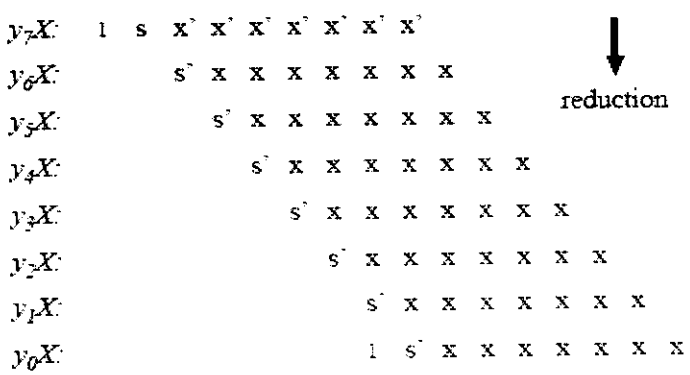


Fig 3.4 Radix 2 PP bit matrix Left to Right (n=8)

To exploit the delay difference between carry and sum signals in adders, the sum signals in the leapfrog structure for R-L array multipliers skip over alternate rows. Because all the carry signals propagate through the entire array, the MSBs of final PPR vectors arrive at the same (latest) time. This unfortunately prevents optimization of the final CPA that is possible in tree multipliers. To allow final CPA optimization in linear array multipliers, L-R computation and leapfrog structure is combined resulting in a new

L-R leapfrog (LRLF) array multiplier scheme. A LRLF multiplier for PP array is shown in Fig 3.1. The dashed lines are carries and solid lines are sum signals. Each adder symbol represents either a FA if all three inputs are variables or a HA if one of three inputs is constant.

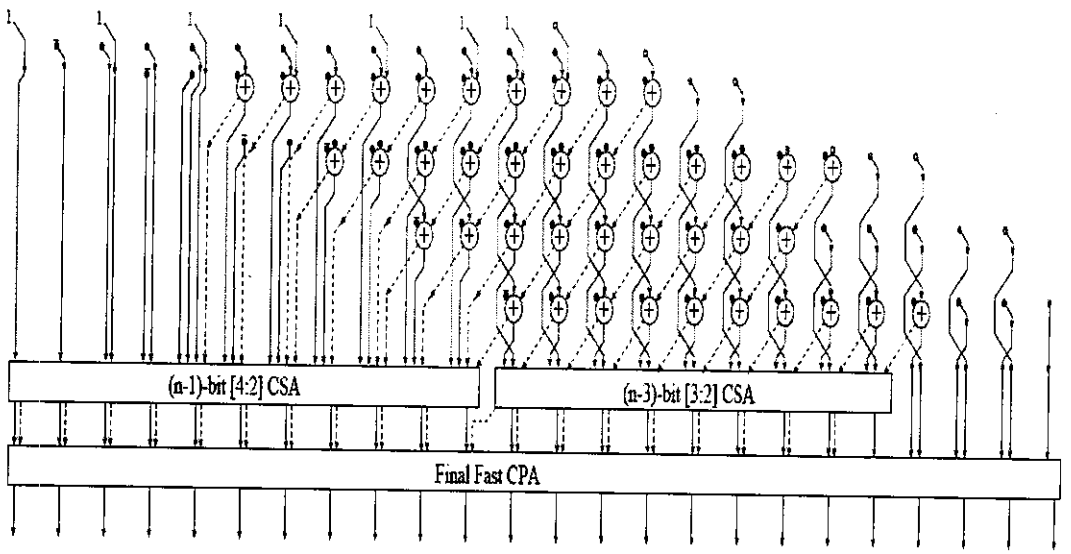


Fig 3.5 LRLF array multiplier

3.2 SPLIT ARRAY LRLF STRUCTURE

From the basic LRLF structure, two types of further splitting are considered: even/odd and upper/lower. Because LRLF well maintains the regularity of array multipliers, this new splitting does not result in a hierarchical tree structure. Instead, Split Array Left Right Leapfrog has a simpler structure than a tree multiplier and requires less design effort.

To reduce PPR delay in LRLF array multiplier, certain level of parallelism is necessary. One approach is to split the PP bit array into even PPs and odd PPs, as shown in Fig 3.2. In each split part, PPs are shifted four bits each row and reduced into two vectors using a LRLF structure. The final vectors from even and odd parts are merged by using an adder. This algorithm is named even/odd LRLF (EOLRLF).

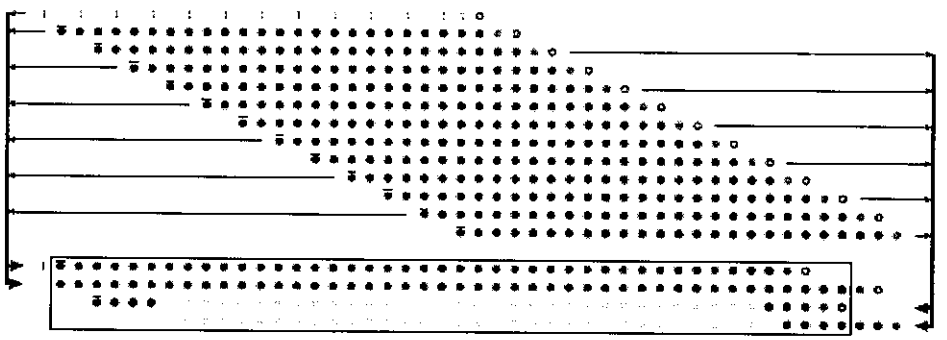


Fig 3.6 EOLRLF array multiplier

Another approach is to split the PP bit array into upper PPs and lower PPs, as shown in Fig 3.3. In each part, PPs are shifted two bits each row and reduced into two vectors using LRLF. The final vectors from upper and lower parts are merged by a carry propagate adder. This algorithm is named upper/lower LRLF (ULLRLF).

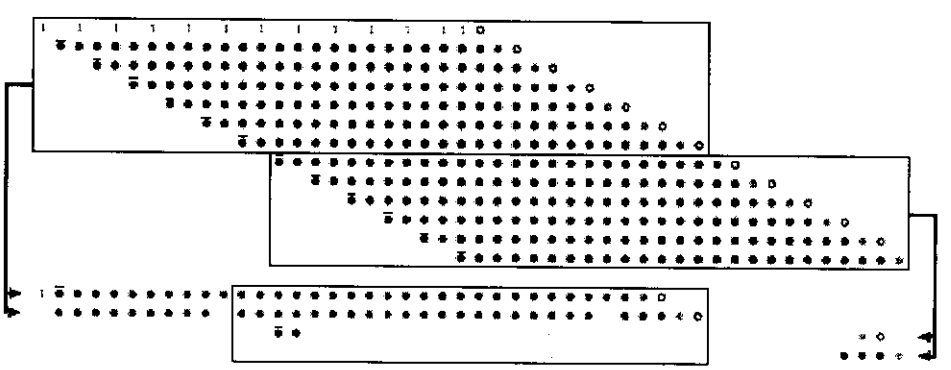


Fig 3.7 ULLR array multiplier

In EOLRLF, the arrival profile of PPR final vectors has fewer latest-arriving bits than that in tree multipliers. Fig 3.4 shows the PPGR delay profiles in a 32*32-bit TDM multiplier, an EOLRLF, and a ULLRLF. The number of latest-arriving bits in EOLRLF is 5 while this number is 8 in TDM. The bit delay distribution in EOLRLF is also more regular. Most bit groups in EOLRLF have 4-5 bits. But the group size varies a lot in TDM.

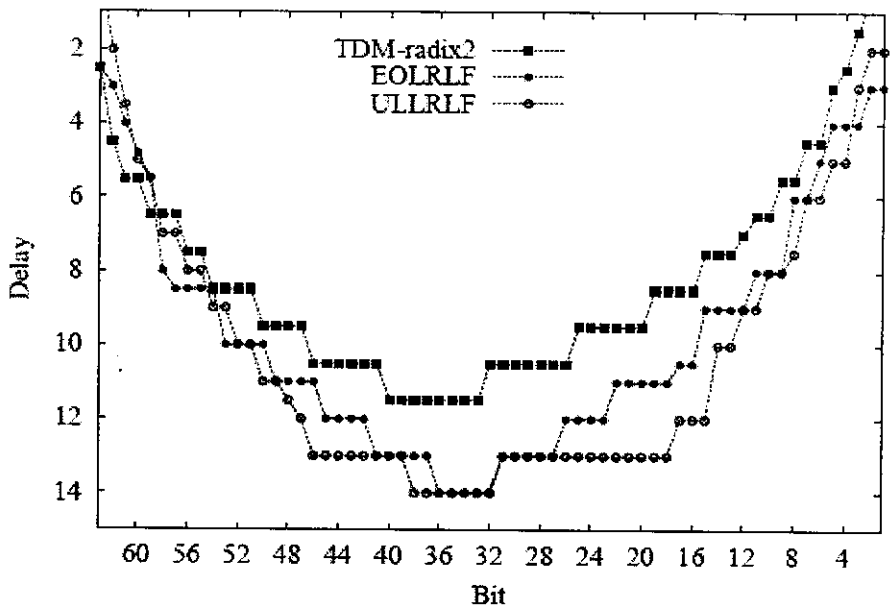


Fig 3.8 PPGR delay profiles

Compared with EOLRLF, ULLRLF has two main advantages. First, the shifting distance between PPs in each upper/lower part is 2 positions instead of 4. Second, the final adder in ULLRLF is only (n+2) bit in contrast to (2n- 3) bit in EOLRLF.

CHAPTER 4

PROPOSED MULTIPLIER

4.1 DESIGN OF PROPOSED MULTIPLIER

Fig 4.1 shows the conventional and proposed multiplier architectures in which word lengths of input data, X and Y, are N bits. The DRD unit is allocated to determine effective dynamic ranges of input data [7]. The data latches are responsible for pipelining to reduce the delay incurred in data access and computation. The Partial-Product Generation (PPG) unit, partial-product summation unit and Carry-Propagation Adder (CPA) are used to fulfill the arithmetic operation of multiplication. The PPG unit generates partial products of a multiplication according to the radix-4 Booth algorithm. The PPS unit is to accumulate the partial products from the PPG unit to yield a sum and a carry that are added by the CPA unit to produce a final product.

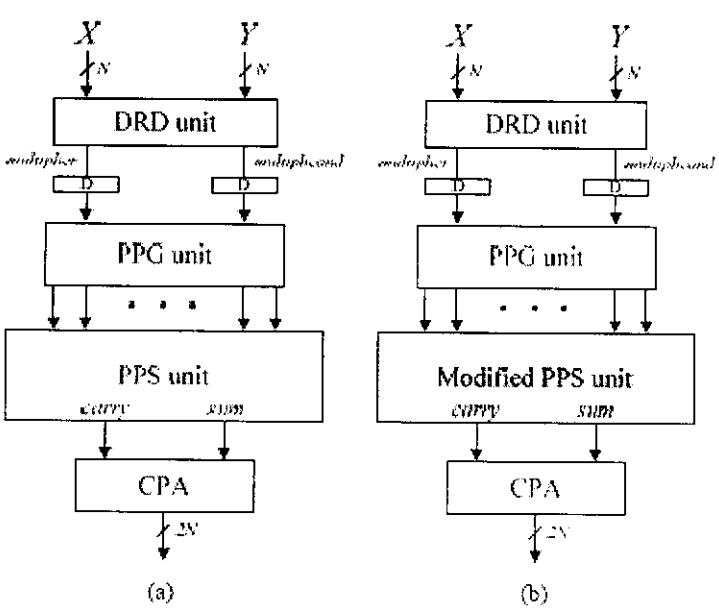


Fig 4.1 Architecture of (a) conventional multiplier (b) proposed multiplier

Fig 4.2 displays an example of a multiplication with two inputs of "2AC9" and "006A" realized by dynamic-range determination, radix-4 Booth decoding and left-to-right summation. The DRD unit can increase the chance of the partial products in high precision being zero. The left-to-right summation structure in the PPS unit can lower switching activities of partial products in high precision being added, and does not increase switching activities of partial-product summation in low precision. Accordingly, the PPS unit based on the left-to-right structure is preferred in our low-power multiplier. As compared to the conventional left-to-right multipliers, the proposed multiplier employs the modified upper/lower left-to-right structure in which a different full adder is adopted to decrease power consumption and critical delay of the PPS unit.

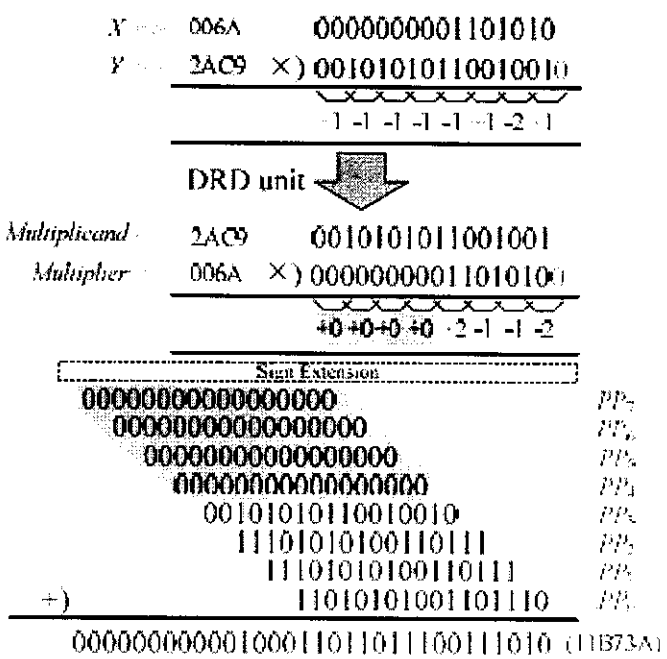


Fig 4.2 Example of a multiplication using DRD, Radix-4 booth decoding and left-to-right summation

4.1.1 DRD AND PPG UNIT

The block diagram of the DRD unit is depicted in Fig 4.3 [7]. In this figure, three bits per group are detected owing to radix-4 Booth decoding. The effective dynamic

ranges of two input data are compared to see which one is smaller, and to select the input datum with a smaller effective dynamic range as a multiplier for Booth decoding. Fig 4.4 shows the PPG unit which can reduce glitches and power consumption of the PPS unit [4].

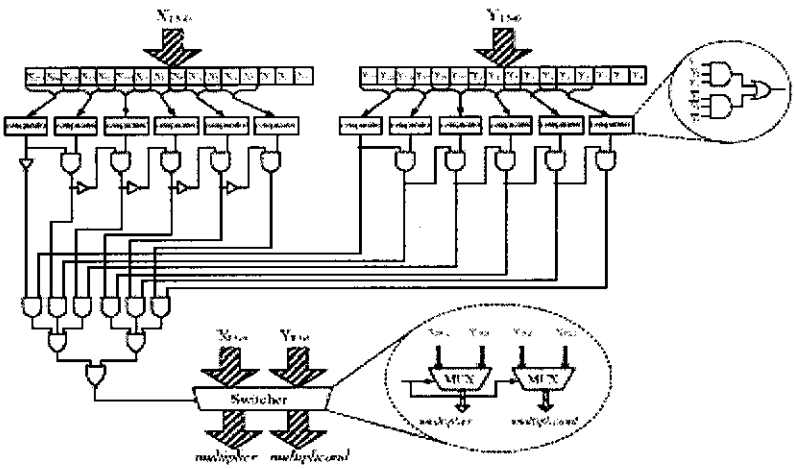


Fig 4.3 Dynamic range determination unit

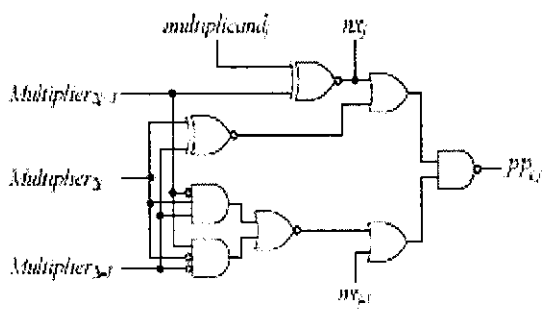


Fig 4.4 Partial product generation unit

4.1.2 MODIFIED PPS UNIT

In order to decrease power consumption and critical delay, the PPS unit using the modified ULLR structure, as shown in Fig 4.5, is adopted in the proposed 16×16-bit multiplier where the partial products, *PP*0~*PP*7, are partitioned into two parts: upper and lower partial products. Such partition can allow parallel summations of these two parts to reduce the delay of the PPS unit with an upper/lower structure. In the conventional left-to-right summation, the summation in the lower part of the PPS unit usually consumes more power than the summation in the upper part of the PPS unit because of the glitch

effect accumulated from the upper part. To minimize the glitch effect, when the upper and lower parts of the PPS unit are individually added in parallel, the power consumed from transient states can be reduced. Fig 4.5(b) and (c) depicts the proposed summations of the upper and lower structures, respectively, which are implemented by many Full Adders (FA) and Half Adders (HA).

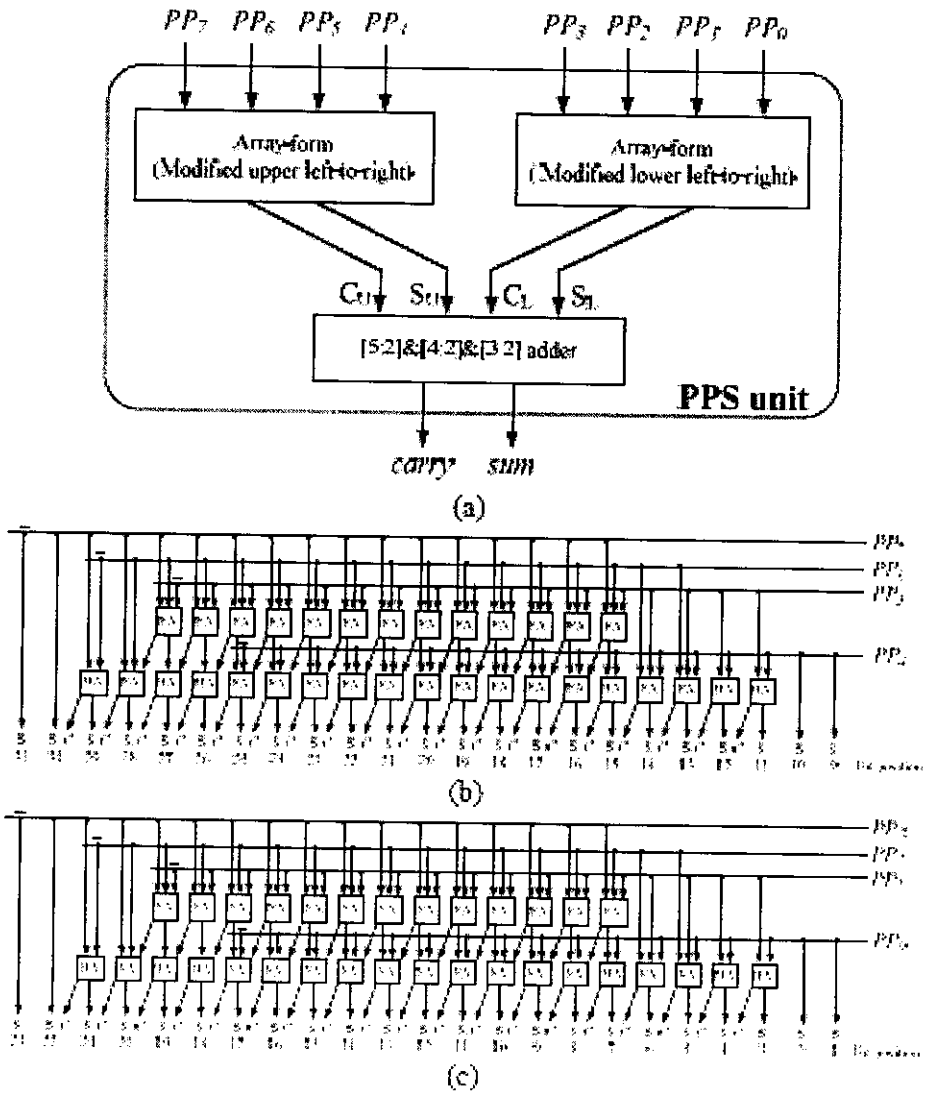


Fig 4.5 Partial product summation unit
(a) proposed architecture (b) Upper left-to-right part (c) Lower left-to-right part

CHAPTER 5

SIMULATION RESULTS

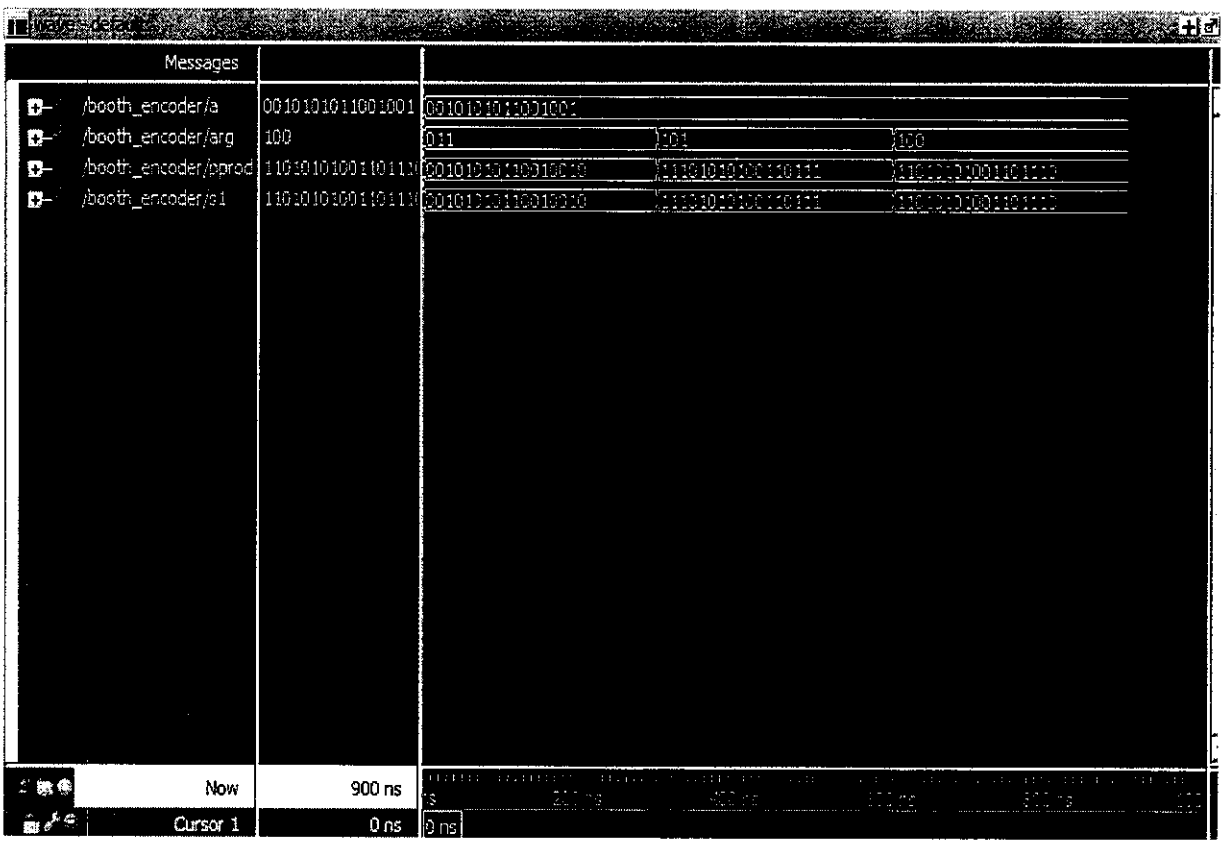


Fig 5.1 Modified booth algorithm

The simulated results are for a multiplication shown in Fig 4.2. Fig 5.1 gives the partial products of a multiplication using Modified booth's algorithm. The generated partial products are given as input to the partial product summation (PPS) unit, which generates sum and carry bits.

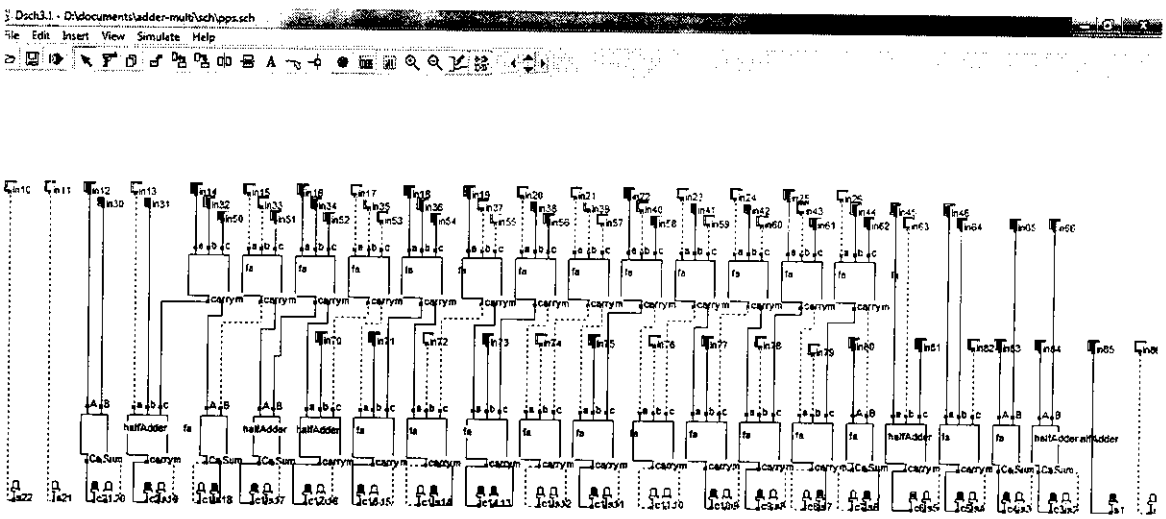


Fig 5.2 Schematic of partial product summation

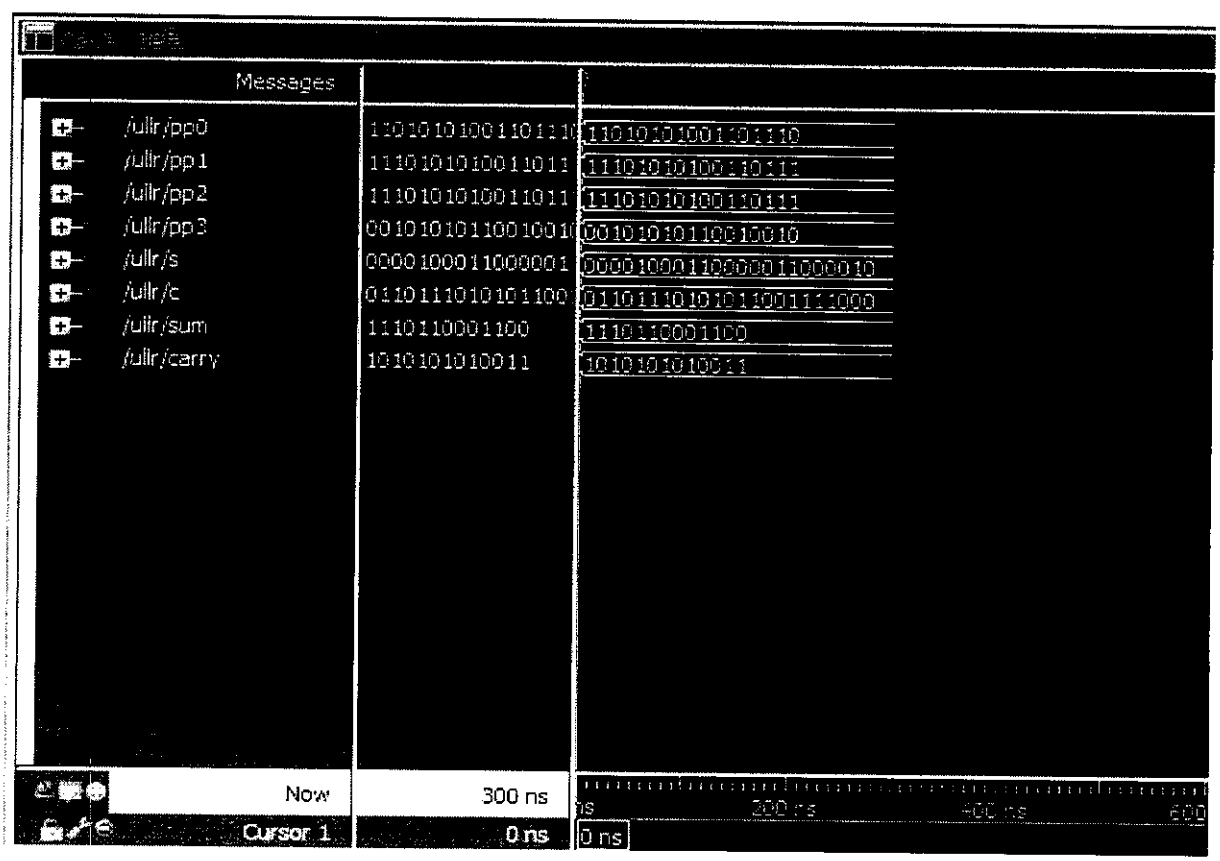


Fig 5.3 Partial product summation unit

The sum and carry bits obtained from PPS unit are added using carry propagate adder which yields the desired product.

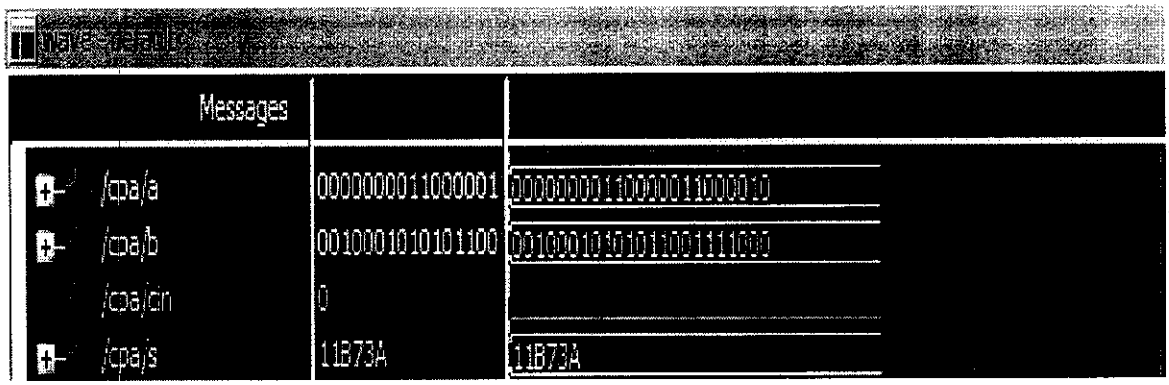


Fig 5.4 Carry propagate adder

The partial product summation unit which makes use of upper/lower left-to-right structure is made of full adders and half adders. The conventional full adders are composed of 24 transistors. To reduce area and power dissipation, the conventional full adders are replaced by a structure composed of 12 transistors.

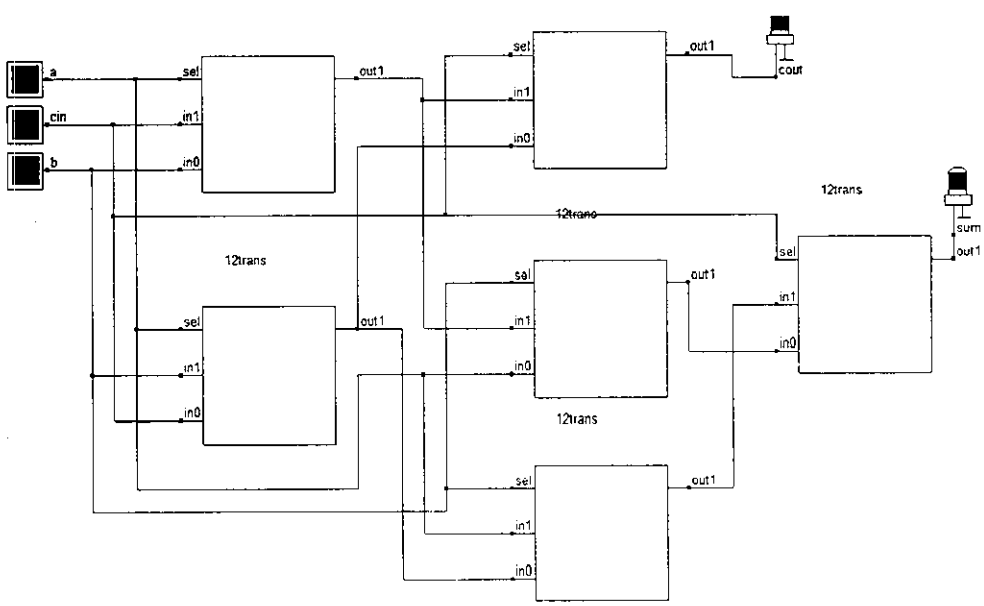


Fig 5.5 Twelve transistor full adder

5.1 POWER REPORT

Table 5.1 Power analysis

PPS UNIT	POWER(mW)
FA (24 transistors)	62
FA(12 transistors)	50

5.2 SYNTHESIS REPORT OF CONVENTIONAL AND PROPOSED MULTIPLIER

Design summary of conventional multiplier:

Selected Device: Xilinx FPGA- 2s300eft256-6

Number of Slices:	142	out of	3072	4%
Number of 4 input LUTs:	246	out of	6144	4%
Number of bonded IOBs:	33	out of	182	18%

Design summary of upper/lower left-to-right multiplier:

Number of Slices:	35	out of	3072	1%
Number of 4 input LUTs:	61	out of	6144	1%
Number of bonded IOBs:	114	out of	182	64%

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

A multiplier using a dynamic range determination unit and a modified upper/lower left-to-right structure in the partial product summation unit is developed. By using Modelsim and Microwind, the units of the proposed multiplier are implemented and the proposed technique has the least power consumption and the least product of area, delay and power consumption than the conventional multipliers. Since the multiplier makes use of parallel operation in partial product summation unit, the computational speed of the multiplier is high than the conventional multipliers. Therefore, the proposed multiplier can be a good candidate to achieve low-power computations for various multimedia applications.

The adder cell using Shannon based technique can be used in the partial product summation unit. This technique makes use of pass transistor logic instead of CMOS logic which reduces the number of transistors.

REFERENCES

[1] Meng-Lin Hsia and Oscal T.-C. Chen, "Low-Power Multiplier Optimized by Partial-Product Summation and Adder Cells", *IEEE Trans. On VLSI Systems, Jun. 2009.*

[2] J.-Y. Kang and J.-L. Gaudiot, "A simple high-speed multiplier design," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1253–1258, Oct. 2006.

[3] S. K. Hsu, S. K. Mathew, M. A. Anders, B. R. Zeydel, V. G. Oklobdzija, R.K. Krishnamurthy and S. Y. Borkar, "A 110 GOPS/W 16-bit multiplier and reconfigurable PLA loop in 90-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 256–264, Jan. 2006.

[4] Z. Huang and M. D. Ercegovac, "High-performance low-power left-to-right array multiplier design," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 272–283, Mar. 2005.

[5] M. D. Ercegovac and T. Lang, Digital Arithmetic. Morgan Kaufmann Publishers, Elsevier Science Ltd., 2004.

[6] J.-Y. Kang and J.-L. Gaudiot, "A Fast and Well-Structured Multiplier," EUROMICRO Symp. Digital System Design, pp. 508- 515, Aug. 2004.

[7] H. Lee, "A power-aware scalable pipelined Booth multiplier," in Proc. IEEE Int. SOC Conf., 2004, pp.123–126.

[8] O. T.-C. Chen, S. Wang and Y. W. Wu, "Minimization of switching activities of partial products for designing low-power multipliers," *IEEE Trans. On VLSI Systems*, vol. 11, no. 3, pp. 418–433, June 2003.

[9] Zhijun Huang, "High level optimization techniques for low power multiplier design" University of california, los angels, 2003.

[10] O. T.-C. Chen, R. R.-B. Sheen and S. Wang, "A low-power adder operating on effective dynamic data ranges," *IEEE Transactions on VLSI Systems*, vol. 10, no. 4, pp. 435–453, Aug. 2002.

APPENDIX

XILINX®

Spartan-IIE 1.8V FPGA Family: Introduction and Ordering Information

DS077-1 (v1.0) November 15, 2001

Preliminary Product Specification

Introduction

The Spartan™-IIE 1.8V Field-Programmable Gate Array family gives users high performance, abundant logic resources, and a rich feature set, all at an exceptionally low price. The five-member family offers densities ranging from 50,000 to 300,000 system gates, as shown in Table 1. System performance is supported beyond 200 MHz.

Spartan-IIE devices deliver more gates, I/Os, and features per dollar than other FPGAs by combining advanced process technology with a streamlined architecture based on the proven Virtex™-E platform. Features include block RAM (to 64K bits), distributed RAM (to 98,304 bits), 19 selectable I/O standards, and four DLLs (Delay-Locked Loops). Fast, predictable interconnect means that successive design iterations continue to meet timing requirements.

The Spartan-IIE family is a superior alternative to mask-programmed ASICs. The FPGA avoids the initial cost, lengthy development cycles, and inherent risk of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary (impossible with ASICs).

Features

- Second generation ASIC replacement technology
  - Densities as high as 6,912 logic cells with up to 300,000 system gates
  - Streamlined features based on Virtex-E architecture
  - Unlimited in-system reprogrammability
  - Very low cost
- System level features
  - SelectRAM+™ hierarchical memory:
    - 16 bits/LUT distributed RAM
    - Configurable 4K-bit true dual-port block RAM
    - Fast interfaces to external RAM
  - Fully 3.3V PCI compliant to 64 bits at 66 MHz and CardBus compliant
  - Low-power segmented routing architecture
  - Full readback ability for verification/observability
  - Dedicated carry logic for high-speed arithmetic
  - Efficient multiplier support
  - Cascade chain for wide-input functions
  - Abundant registers/latches with enable, set, reset
  - Four dedicated DLLs for advanced clock control
  - Four primary low-skew global clock distribution nets
  - IEEE 1149.1 compatible boundary scan logic
- Versatile I/O and packaging
  - Low cost packages available in all densities
  - Family footprint compatibility in common packages
  - 19 high-performance interface standards, including LVDS and LVPECL
  - Up to 120 differential I/O pairs that can be input, output, or bidirectional
  - Zero hold time simplifies system timing
- Fully supported by powerful Xilinx ISE development system
  - Fully automatic mapping, placement, and routing
  - Integrated with design entry and verification tools

Table 1: **Spartan-IIE FPGA Family Members**

Device	Logic Cells	Typical System Gate Range (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O	Maximum Differential I/O Pairs	Distributed RAM Bits	Block RAM Bits
XC2S50E	1,728	23,000 - 50,000	16 x 24	384	182	84	24,576	32K
XC2S100E	2,700	37,000 - 100,000	20 x 30	600	202	86	38,400	40K
XC2S150E	3,888	52,000 - 150,000	24 x 36	864	263	114	55,296	48K
XC2S200E	5,292	71,000 - 200,000	28 x 42	1,176	289	120	75,264	56K
XC2S300E	6,912	93,000 - 300,000	32 x 48	1,536	329	120	98,304	64K

General Overview

The Spartan-IIE family of FPGAs have a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input/Output Blocks (IOBs). There are four Delay-Locked Loops (DLLs), one at each corner of the die. Two columns of block RAM lie on opposite sides of the die, between the CLBs and the IOB columns. These functional elements are interconnected by a powerful hierarchy of versatile routing channels (see Figure 1).

Spartan-IIE FPGAs are customized by loading configuration data into internal static memory cells. Unlimited reprogramming cycles are possible with this approach. Stored values in these cells determine logic functions and interconnections implemented in the FPGA. Configuration data can be read from an external serial PROM (master serial mode), or written into the FPGA in slave serial, slave parallel, or Boundary Scan modes. The Xilinx XC17S00A PROM family is recommended for serial configuration of Spartan-IIE FPGAs. The XC18V00 reprogrammable PROM family is recommended for parallel or serial configuration.

Spartan-IIE FPGAs are typically used in high-volume applications where the versatility of a fast programmable solution adds benefits. Spartan-IIE FPGAs are ideal for shortening product development cycles while offering a cost-effective solution for high volume production.

Spartan-IIE FPGAs achieve high-performance, low-cost operation through advanced architecture and semiconductor technology. Spartan-IIE devices provide system clock rates beyond 200 MHz. Spartan-IIE FPGAs offer the most cost-effective solution while maintaining leading edge performance. In addition to the conventional benefits of high-volume programmable logic solutions, Spartan-IIE FPGAs also offer on-chip synchronous single-port and dual-port RAM (block and distributed form), DLL clock drivers, programmable set and reset on all flip-flops, fast carry logic, and many other features.

Spartan-IIE Family Compared to Spartan-II Family

- Higher density and more I/O
- Higher performance
- Unique pinouts in cost-effective packages
- Differential signaling
  - LVDS, Bus LVDS, LVPECL
- V_{CCINT} = 1.8V
  - Lower power
  - 5V tolerance with 100Ω external resistor
  - 3V tolerance directly
- PCI, LVTTL, and LVCMOS2 input buffers powered by V_{CCO} instead of V_{CCINT}
- Unique larger bitstream

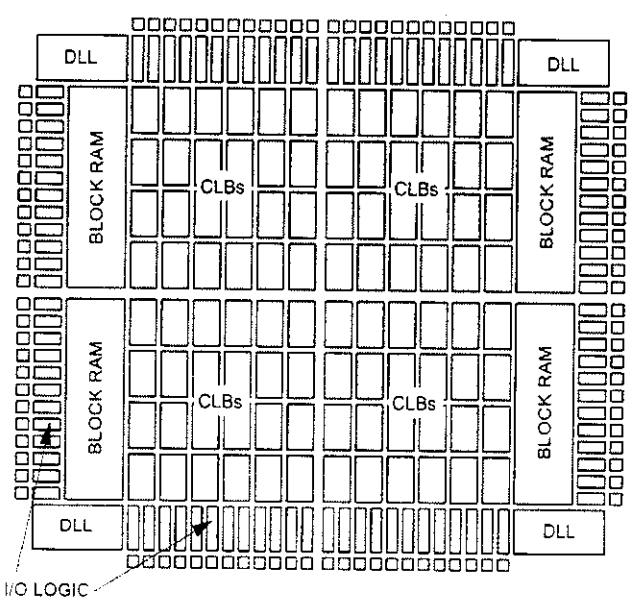


Figure 1: Basic Spartan-IIE Family FPGA Block Diagram

Spartan-IIE Product Availability

Table 2 shows the package and speed grades available for Spartan-IIE family devices. Table 3 shows the maximum user I/Os available on the device and the number of user I/Os available for each device/package combination.

Table 2: **Spartan-IIE Package and Speed Grade Availability**

Device	Pins	144	208	256	456
	Type	Plastic TQFP	Plastic PQFP	Fine Pitch BGA	Fine Pitch BGA
	Code	TQ144	PQ208	FT256	FG456
XC2S50E	-6	C, I	C, I	C, I	-
	-7	(C)	(C)	(C)	-
XC2S100E	-6	C, I	C, I	C, I	C, I
	-7	(C)	(C)	(C)	(C)
XC2S150E	-6	-	(C, I)	(C, I)	(C, I)
	-7	-	(C)	(C)	(C)
XC2S200E	-6	-	C, I	C, I	C, I
	-7	-	(C)	(C)	(C)
XC2S300E	-6	-	C, I	C, I	C, I
	-7	-	(C)	(C)	(C)

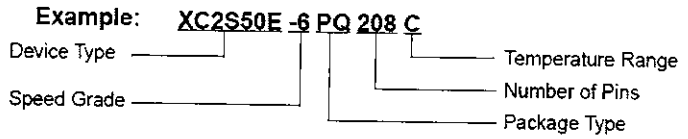
Notes:

1. C = Commercial, T_J = 0° to +85°C: I = Industrial, T_J = –40°C to +100°C
2. Parentheses indicate product not yet released. Contact sales for availability.

Table 3: **Spartan-IIE User I/O Chart**

Device	Maximum User I/O	Available User I/O According to Package Type			
		TQ144	PQ208	FT256	FG456
XC2S50E	182	102	146	182	-
XC2S100E	202	102	146	182	202
XC2S150E	263	-	146	182	263
XC2S200E	289	-	146	182	289
XC2S300E	329	-	146	182	329

Ordering Information



Device Ordering Options

Device
XC2S50E
XC2S100E
XC2S150E
XC2S200E
XC2S300E

Speed Grade	
-6	Standard Performance
-7	Higher Performance

Package Type / Number of Pins	
TQ144	144-pin Plastic Thin QFP
PQ208	208-pin Plastic QFP
FT256	256-ball Fine Pitch BGA
FG456	456-ball Fine Pitch BGA

Temperature Range (T_J)	
C = Commercial	0°C to +85°C
I = Industrial	–40°C to +100°C

Revision History

Version No.	Date	Description
1.0	11/15/01	Initial Xilinx release.

The Spartan-IIE Family Data Sheet

DS077-1, *Spartan-IIE 1.8V FPGA Family: Introduction and Ordering Information* (Module 1)

DS077-2, *Spartan-IIE 1.8V FPGA Family: Functional Description* (Module 2)

DS077-3, *Spartan-IIE 1.8V FPGA Family: DC and Switching Characteristics* (Module 3)

DS077-4, *Spartan-IIE 1.8V FPGA Family: Pinout Tables* (Module 4)