P-3457

EFFICIENT PARALLEL MULTIPLIER ACCUMULATOR ARCHITECTURE FOR HIGH SPEED ARITHMETIC

By

K.NAGENDRAN

Reg. No. 0920106010

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 **"EFFICIENT PARALLEL MULTIPLIER ACCUMULATOR ARCHITECTURE FOR HIGH SPEED ARITHMETIC"** is the bonafide work of **Mr.K.NAGENDRAN [Reg. no. 0920106010]** 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.

[illegible] 6/4/11
Project Guide

Ms. K.Kavitha

[illegible] 6/4/11
Head of the Department

Dr. Rajeswari Mariappan

The candidate with university Register no. 0920106010 is examined by us in the project viva-voce examination held on ..21.-.4.-.2011.....

[illegible] 21/4/11
Internal Examiner

[illegible]
External Examiner

ACKNOWLEDGEMENT

A project of this nature needs co-operation and support from many for successful completion. In this regards, 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 the 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 her 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.K.Kavitha M.E (Ph.D),** Assistant Professor(SRG), Project guide for her 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

Fast digital signal processing areas like audio signal processing, video or image processing uses Multiplier-and- Accumulator (MAC) as their basic element in order to perform repeated multiplication and addition. The conventional MAC architecture uses more shift and add operations at multiplier unit this in turn increases delay in the arithmetic operations.

The main objective of the project is to design a new multiplier-and-accumulator (MAC) architecture to perform high speed arithmetic operation. By combining multiplication with accumulation and devising a hybrid type of Carry Save Adder (CSA), the performance can be improved. This is done by merging the accumulator that has the largest delay in MAC into CSA tree. The proposed CSA tree uses radix-4 Modified Booth's Algorithm (MBA) and has the modified array for the sign extension in order to increase the bit density of the operands. The CSA propagates the carries to the least significant bits of the Partial products and generates the least significant bits in advance to decrease the number of the input bits of the final adder. Also, the proposed MAC accumulates the intermediate results in the type of sum and carry bits, instead of the output of the final adder, which made it possible to optimize the pipeline scheme to improve the performance. Based on the theoretical and experimental estimation, results such as the amount of hardware resources, delay, and pipelining scheme is analysed. The proposed MAC shows the superior properties to the standard design in many ways and performance twice as much as the previous research in the similar clock frequency.

As a future work this pipelined MAC architecture is applied to the digital filter design such as FIR and IIR filter. The performance parameter like area, delay and area delay product of the digital filter with the proposed MAC design is to be compared with that of standard design.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

ALU	ARITHMETIC AND LOGIC UNIT
BA	BOOTH ALGORITHM
BWA	BAUGH–WOOLEY ALGORITHM
CLA	CARRY LOOK AHEAD ADDER
CSA	CARRY SAVE ADDER
DCT	DISCRETE COSINE TRANSFORM
DWT	DISCRETE WAVELET TRANSFORM
FA	FULL ADDER
FPGA	FIELD PROGRAMMABLE GATE ARRAY
HA	HALF ADER
MAC	MULTIPLIER-AND-ACCUMULATOR
MBA	MODIFIED BOOTH ALGORITHM
MUX	MULTIPLEXERS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO MAC

With the recent rapid advances in multimedia and communication systems, real-time signal processings like audio signal processing, video/image processing, or large-capacity data processing are increasingly being demanded. The multiplier and multiplier-and-accumulator (MAC) are the essential elements of the digital signal processing such as filtering, convolution, and inner products. Most digital signal processing methods use nonlinear functions such as Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT). Because they are basically accomplished by repetitive application of multiplication and addition, the speed of the multiplication and addition arithmetic's determines the execution speed and performance of the entire calculation. Because the multiplier requires the longest delay among the basic operational blocks in digital system, the critical path is determined by the multiplier.

In general, a multiplier uses Booth's algorithm and array of full adders (FAs), or Wallace tree instead of the array of FAs. This multiplier mainly consists of the three parts: Booth encoder, a tree to compress the partial products such as Wallace tree, and final adder. Because Wallace tree is to add the partial products from encoder as parallel as possible, its operation time is proportional to $O(\log_2 N)$, where N is number of inputs. It uses the fact that counting the number of 1's among the inputs reduces the number of outputs into $\log_2 N$. In real implementation, many (3:2) or (7:3) counters are used to reduce the number of outputs in each pipeline step. The most effective way to increase the speed of a multiplier is to reduce the number of the partial products because multiplication proceeds a series of additions for the partial products. To reduce the number of calculation steps for the partial products, MBA Algorithm has been applied mostly where Wallace tree has taken the role of increasing the speed to add the partial products.

an architecture in which accumulation has been combined with the carry save adder (CSA) tree that compresses partial products. In the architecture, the critical path was reduced by eliminating the adder for accumulation and decreasing the number of input bits in the final adder. It has a better performance because of the reduced critical path, but there is a need to improve the output rate due to the use of the final adder results for accumulation.

Here a new architecture for a high-speed MAC is proposed. In this MAC, the computations of multiplication and accumulation are combined and a hybrid-type CSA structure is proposed to reduce the critical path and improve the output rate. A modified array structure for the sign bits is used to increase the density of the operands. A carry look-ahead adder (CLA) is inserted in the CSA tree to reduce the number of bits in the final adder. In addition, in order to increase the output rate by optimizing the pipeline efficiency, intermediate calculation results are accumulated in the form of sum and carry instead of the final adder outputs.

1.2 MOTIVATION OF THE WORK

Digital signal processing architectures consist of MAC unit as the key block. As the output of the DSP architecture depends on the critical delay of a MAC unit, most of the work is concentrated on the design of efficient adder and multiplier. As multiplier consumes more time unit, work is concentrated on the design of multiplier units for reducing the delay. Conventional multiplier consumes more time and power due to generation of more number of partial products. This leads to more number of adders thus increasing the hardware complexity. So work is concentrated on the design of efficient multiplication algorithms where there is a possibility for reducing the number of partial products. Radix-2 and Radix-4 based multiplication algorithms reduces the number of partial products in which a group of multiplier bits are compared for partial product generation. Further by eliminating separate accumulation and implementing CSA with MAC there is a possibility for reducing hardware complexity and minimizing delay and power dissipation.

- Performance of the MAC unit is improved by either using high speed multipliers or by using improved fast adder architectures. Here multiplication unit combined with accumulation and devising a hybrid type of carry save adder (CSA), for high speed operation.

- Radix-4 modified booth's algorithm is used in the carry save adder tree to reduce the number of partial products.

- The proposed MAC is designed using VHDL code and simulated using Modelsim. Simulation results are used for performance comparison of the proposed multiplier.

- Performance parameters taken for analysis are gate count, delay and power and these parameters are compared with the existing MAC unit using 4-2 compressor circuit.

1.4 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.

1.4.1 STRUCTURAL DESCRIPTIONS

The structural descriptions are explained below with examples.

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.

```
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

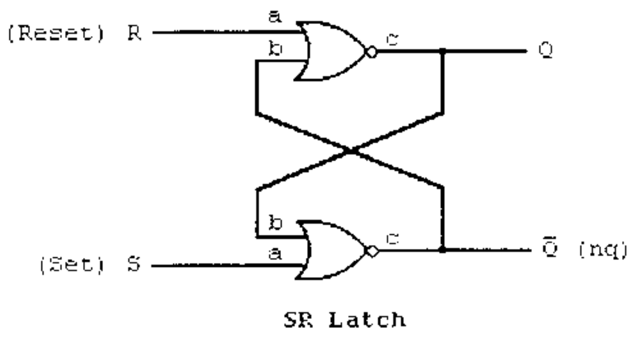


Fig.1.1 Schematic SR Latch

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.

1.4.2 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.

Example

Suppose we were to describe the following SR latch using VHDL as in the following schematic.

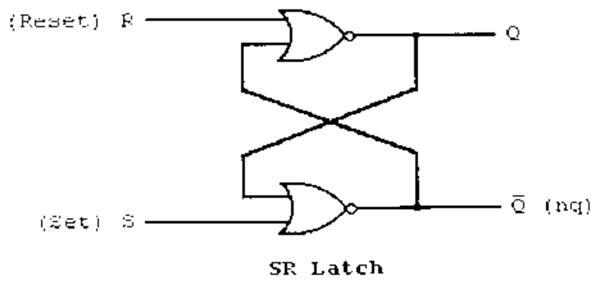


Fig.1.2 Data flow approach in SR Latch

```
  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.

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.

1.4.3 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.

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 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.

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.

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.

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.

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 real.

1.5 SOFTWARES USED

- ModelSim PE 5.4e
- Xilinx ISE 9.2i
- MicroWind 3.1

1.6 ORGANIZATION OF THE REPORT

- **Chapter 2** discusses about the overview of MAC.
- **Chapter 3** discusses about conventional MAC unit with 4:2 compressor.
- **Chapter 4** discusses the Proposed MAC architecture for high speed computation.
- **Chapter 5** presents the simulation output and results.
- **Chapter 6** gives the conclusion and future scope.

CHAPTER 2

OVERVIEW OF MAC

2.1 GENERAL HARDWARE ARCHITECTURE OF MAC

The general hardware architecture of the MAC is shown in Fig.2.1.The basic hardware requirement of the MAC architecture is a multiplier and adder unit. It executes the multiplication operation by multiplying the input multiplier and the multiplicand. This is added to the previous multiplication result as the accumulation step.

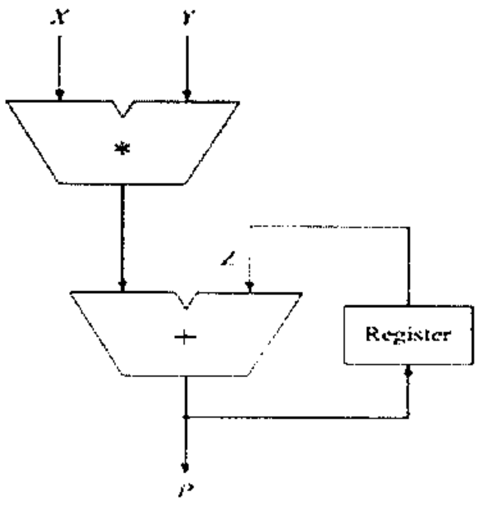


Fig.2.1 Hardware architecture of general MAC.

The multiplication–accumulation results can be expressed as

$$P = X * Y + Z$$

Where X is a multiplicand, Y is a multiplier and Z is the previous accumulated result. A register is used in this structure to store the previously computed results of multiplication and addition. The MAC unit is classified into two types, one is the Parallel MAC unit and the other is a Merged MAC unit.

The general structure of a parallel MAC [3] for multiplying two numbers X and Y adding the result to Z is shown in Fig. 2.2. The partial products in the figure can be generated using any multiplication algorithm using bit-serial, serial-parallel, or full-parallel techniques. Partial-Product Generation can be achieved using several techniques such as the BWA, the Booth algorithm (BA) or the MBA. For an n-bit multiplier, the number of summands is n for BWA, for < n/2 BA, and n/2for MBA. In addition to the encoding step, the BA and MBA algorithms also require generation of the two's complement of the multiplier

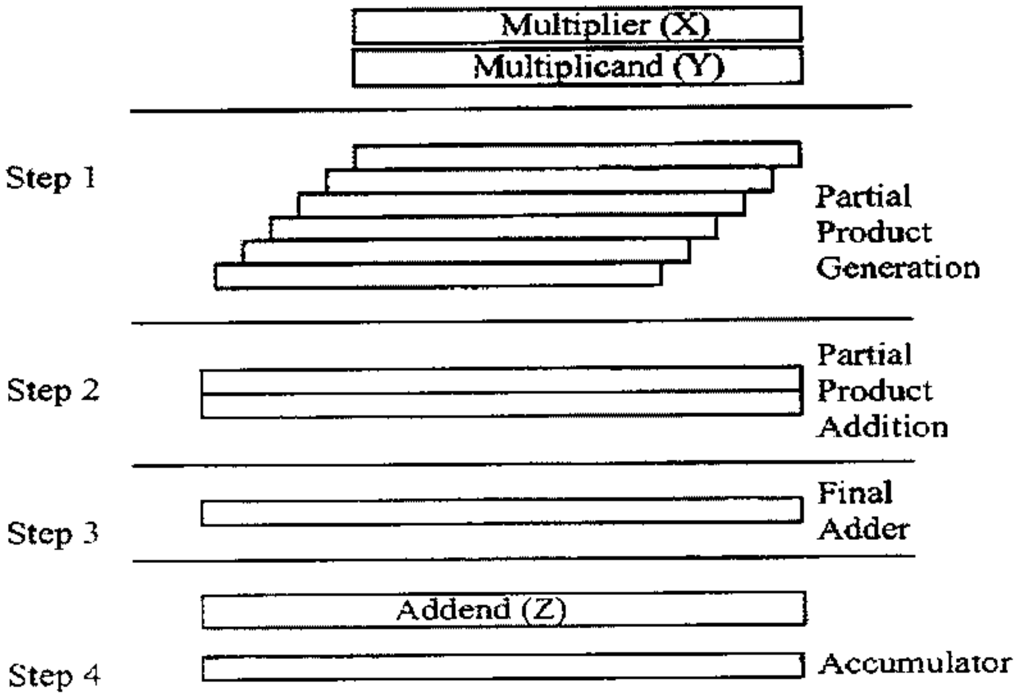


Fig. 2.2 The three major parts of a general parallel multiplier.

Partial-Product Addition is done using carry-save techniques, Wallace trees, or summand skip. When the number of partial products is reduced to sum and carry words, a final adder is required to generate the multiplication result. The final adder produces a double-precision result of 2n bits that must be added to the accumulator content, which is also 2n-bits wide, which is also 2n-bits wide. Typical microprocessors will complete the multiplication operation and follow that with a double-precision accumulation operation. This almost times the delay of a one-bit full adder. Recently, it was realized that the MAC operation can be merged to make the MAC operation take as much delay as a regular multiply operation.

2.3 MERGED MULTIPLIER-AND-ACCUMULATOR UNIT

In merged multiplier and accumulator [6], accumulation operation is merged within the partial products reduction tree used for multiplication. The architecture shown in Fig.2.3 is based on Binary trees constructed using 4-2 compressor circuits. Increasing the speed of operation is achieved by taking advantage of the available free input lines of the 4-2 compressors. The bits of the accumulated value are directly fed into the summation tree. This results in merging the accumulation operation within the multiplication process.

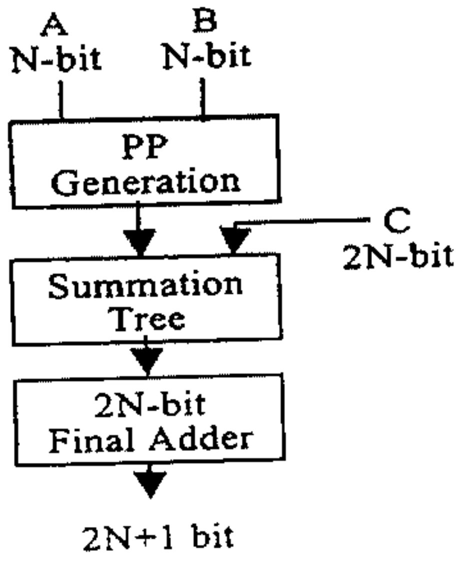


Fig.2.3 Hardware architecture of merged MAC.

The main advantage of this architecture is that increases the overall speed of the MAC operation and eliminates the need for the final accumulator circuit. But some irregularities due to the shifting of the partial products occur as a result of the introduced zeros at various places in the reduction tree. These introduced zeros represent either hardware inefficiency, if they are actually added or irregularities in the tree if special compressor circuits are built to design.

2.4 BAUGH-WOOLEY ALGORITHM

An algorithm for direct 2's complement array multiplication has been proposed by Baugh and Wooley [10] 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.

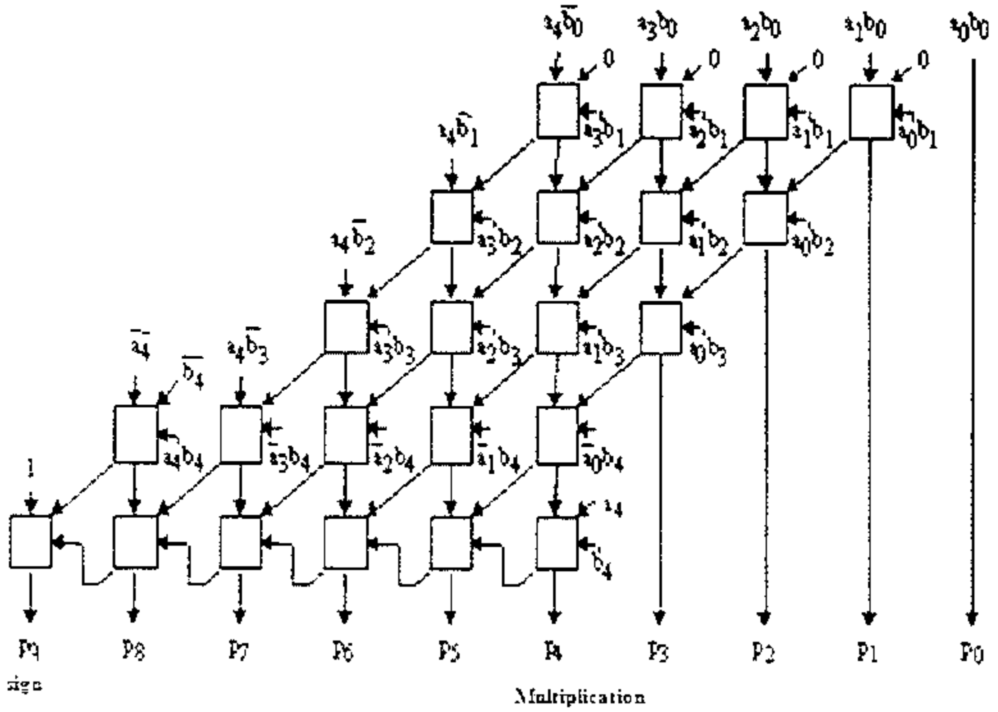


Fig.2.4 Baugh-Wooley Algorithm for Unsigned Multiplier

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 we have two n-bit numbers, *x* and y, their product can be defined as:

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

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.

$$\begin{array}{cccccccccccccccc}
 & & & & & & & & y_7 & y_6 & y_5 & y_4 & y_3 & y_2 & y_1 & y_0 \\
 & & & & & & & & x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & & & & 1 & \overline{p_{70}} & p_{60} & p_{50} & p_{40} & p_{30} & p_{20} & p_{10} & p_{00} \\
 & & & & & & & \overline{p_{71}} & p_{61} & p_{51} & p_{41} & p_{31} & p_{21} & p_{11} & p_{01} & \\
 & & & & & & \overline{p_{72}} & p_{62} & p_{52} & p_{42} & p_{32} & p_{22} & p_{12} & p_{02} & & \\
 & & & & & \overline{p_{73}} & p_{63} & p_{53} & p_{43} & p_{33} & p_{23} & p_{13} & p_{03} & & & \\
 & & & & \overline{p_{74}} & p_{64} & p_{54} & p_{44} & p_{34} & p_{24} & p_{14} & p_{04} & & & & \\
 & & & \overline{p_{75}} & p_{65} & p_{55} & p_{45} & p_{35} & p_{25} & p_{15} & p_{05} & & & & & \\
 & & \overline{p_{76}} & p_{66} & p_{56} & p_{46} & p_{36} & p_{26} & p_{16} & p_{06} & & & & & & \\
 & p_{77} & \overline{p_{67}} & \overline{p_{57}} & \overline{p_{47}} & \overline{p_{37}} & \overline{p_{27}} & \overline{p_{17}} & \overline{p_{07}} & & & & & & & \\
\hline
\overline{S_{15}} & S_{14} & S_{13} & S_{12} & S_{11} & S_{10} & S_9 & S_8 & S_7 & S_6 & S_5 & S_4 & S_3 & S_2 & S_1 & S_0
\end{array}$$

Fig.2.5 Illustration of an 8-bit Baugh-Wooley multiplication.

signed multiplications; Fig. 2.5 illustrates the algorithm for an 8-bit case. 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 N th column.
iii) The MSB of the final result is inverted.

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. 2.6. 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.

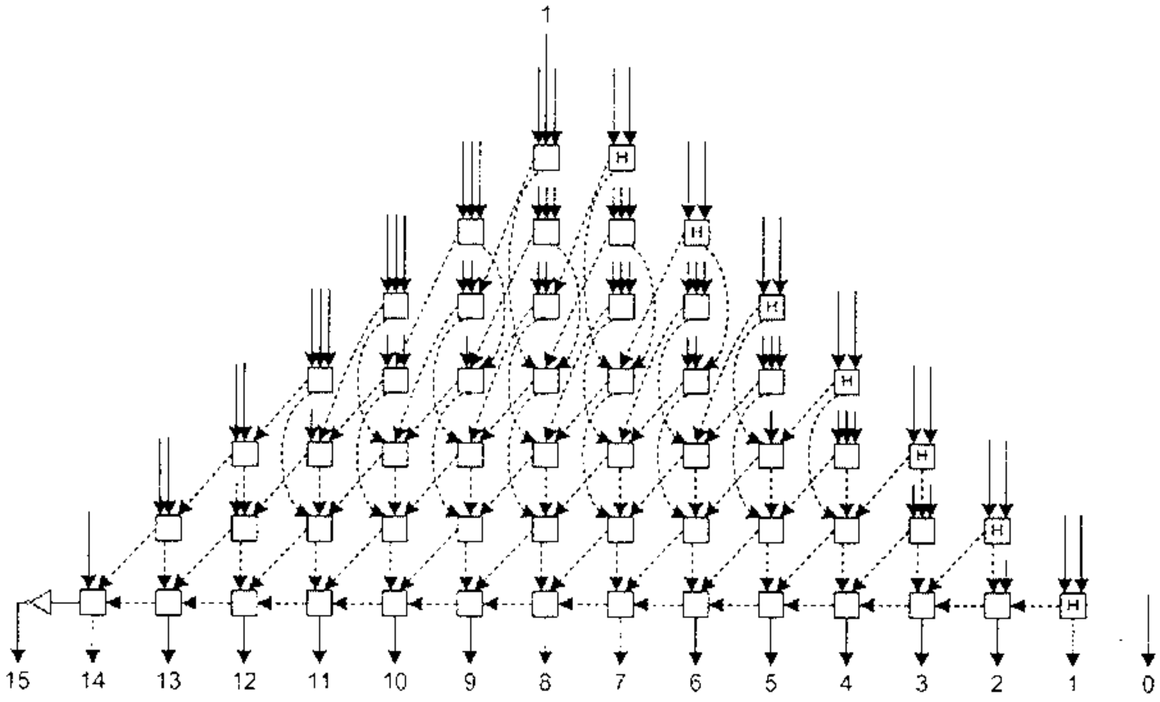
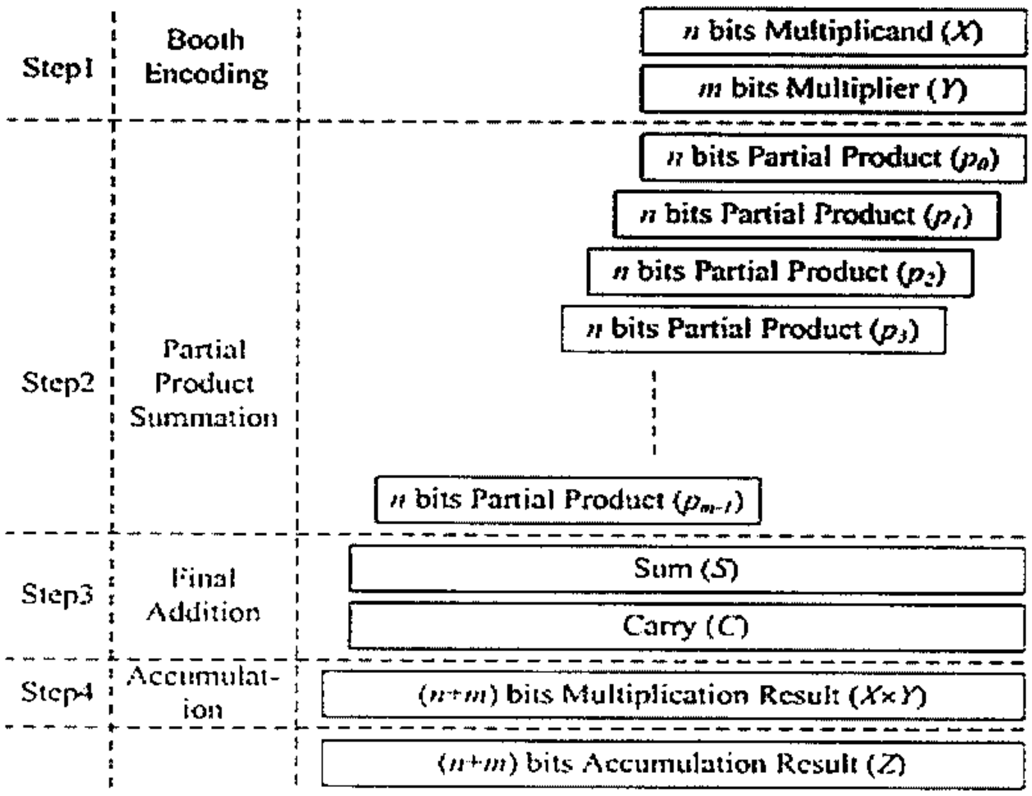


Fig.2.6. 8-bit Baugh-Wooley multiplier using an HPM reduction tree.

A multiplier [4] can be divided into three operational steps. The first step is Booth encoding in which a partial product is generated from the multiplicand and the multiplier. The second step is adder array or partial product compression to add all partial products and convert them into the form of sum and carry. The last is the final addition in which the final multiplication result is produced by adding the sum and the carry. If the process to accumulate the multiplied results is included, a MAC consists of four steps, as shown in Fig. 2.7, which shows the operational steps explicitly.



Step1
Booth Encoding
n bits Multiplicand (X)
m bits Multiplier (Y)
Step2
Partial Product Summation
n bits Partial Product (p_0)
n bits Partial Product (p_1)
n bits Partial Product (p_2)
n bits Partial Product (p_3)
n bits Partial Product (p_{m-1})
Step3
Final Addition
Sum (S)
Carry (C)
Step4
Accumulat-ion
($n+m$) bits Multiplication Result ($X \times Y$)
($n+m$) bits Accumulation Result (Z)

Fig.2.7 Basic arithmetic steps of multiplication and accumulation

After performing booth encoding for the n bits multiplicand and m bit multiplier, the number of partial product generated is less than n/2 for booth algorithm and equal to n/2 in the case of modified booth algorithm (MBA).

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. This 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 we are multiplying four bits numbers.

U	V	X	X-1	
0000	0000	1100	0	Load the value
				1st cycle
				2nd cycle
				3rd Cycle
				4th Cycle

Fig.2.8 U V Method for Calculation of Product of x and y

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.

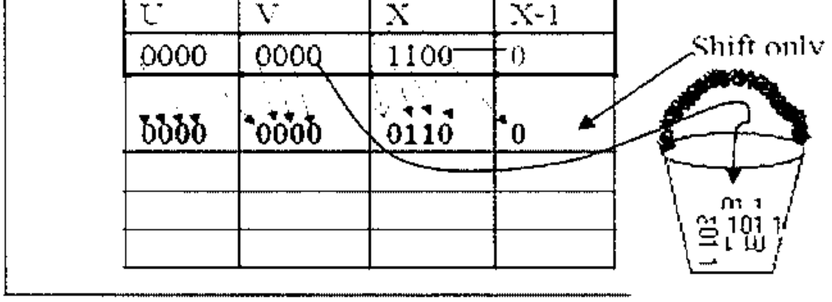
   0 1 Add Y to U, and shift

   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.

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. 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.



U	V	X	X-1
0000	0000	1100	0
0000	0000	0110	0

Repeat the same steps until the four cycles are completed.

U	V	X	X-1	
0000	0000	1100	0	
0000	0000	0110	0	
0000	**0000**	**0011**	**0**	← **Shift only**

Fig.2.9 Intermediate Results of Multiplication in U and V Method

U	V	X	X-1	
0000	0000	1100	0	
0000	0000	0110	0	
0000	0000	0011	0	
1110	**0000**	**0011**	**0**	← Add –Y (0000 + 1110 = 1110)
1111	**0000**	**1001**	**1**	← Shift

U	V	X	X-1	
0000	0000	*1100*	0	
0000	0000	0110	0	
0000	0000	0011	0	
1110	0000	0011	0	
1111	0000	1001	1	
1111	**1000**	***1100***	**1**	← Shift only

Fig.2.10 Final results of multiplication in U and V method

which helps to decrease the number of subsequent calculation stages. The original version of the Booth algorithm had two drawbacks. They are: (i) The number of add or subtract operations and the number of shift operations becomes variable and becomes inconvenient in designing parallel multipliers. (ii) The algorithm becomes inefficient when there are isolated 1's. These problems are overcome by using modified Booth encoding.

2.5.2 Modified Booth Algorithm

Modified Booth Recoding algorithm is one of the most popular techniques to reduce the number of partial products to be added while multiplying two numbers. Reduction in number of partial products depends upon how many bits are encoded. If 3-bit encoding (Radix-4) is used the number of partial products is reduced by half. This is a great saving in terms of silicon area and also speed as number of stages to be added is reduced to half compared to normal add and shift multiplication.

Block	Re-coded digit	Operation on X
000	0	0X
001	+1	+1X
010	+1	+1X
011	+2	+2X
100	-2	-2X
101	-1	-1X
110	-1	-1X
111	0	0X

Table 2.1 Booth Encoding

Booth algorithm which scan strings of three bits with the algorithm given below:

1) Extend the sign bit 1 position if necessary to ensure that n is even.

2) Append a 0 to the right of the LSB of the multiplier.

-X, +2X or -2X.

The negative values of X are made by taking the 2's complement. The multiplication of X is done by shifting X by one bit to the left. Thus, in any case, in designing a n-bit parallel multipliers, only n/2 partial products are generated. Modified Booth algorithm is briefly discussed and its implementation steps follow the discussion.

Let A and B is two n-bit two's complement binary numbers where A is multiplicand and B is multiplier. Product P = A.B

An equivalent base 4 redundant sign digit representation of B is obtained as:

$$B = \sum_{i=0}^{\left(\frac{n}{2}\right)-1} 2^{2i} K_i$$

Where K, is calculated by following equation

$$K_i = -2b_{2i+1} + b_{2i} + b_{2i-1\backslash}$$

At each step 3 bits of Multiplier B, b2,+1, b2,, ba.1, are examined and corresponding K is calculated. B is always appended on the right with zero (b.i=0), and n is always even (B is sign extended if needed). The product A.B is then obtained by adding n/2 partial products.

$$A.B = \sum_{i=0}^{\left(\frac{n}{2}\right)-1} 2^{2i} K_i A$$

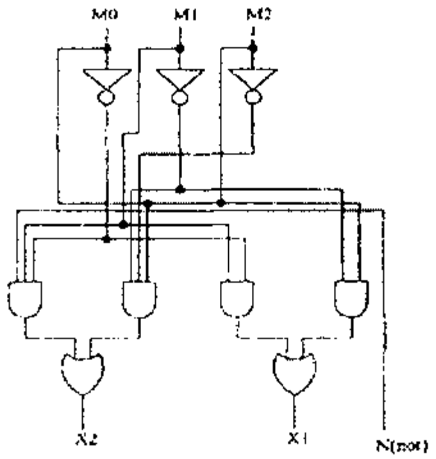


Fig.2.11 Radix-4 Modified Booth's Encoder Module

using circuit diagram shown in Fig.2.12.

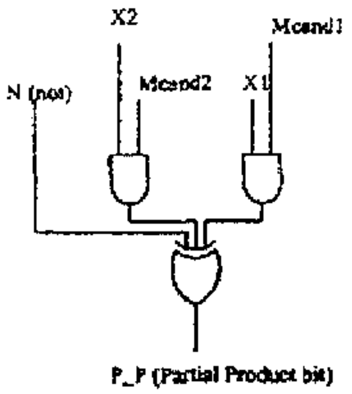


Fig.2.12 Partial Product Bit Generator

2.5.3 Partial Product Summation

For parallel multipliers [5], the addition is accomplished using carry-save techniques, Wallace trees, or summand skip. However, these two techniques require irregular wiring and extra hardware.

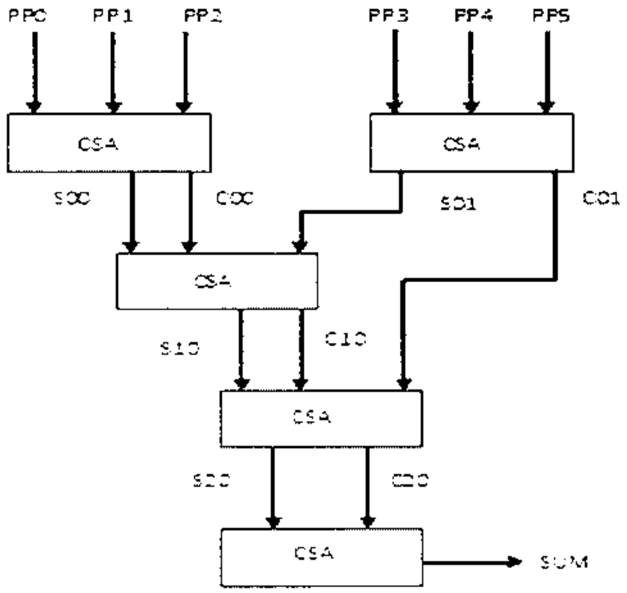


Fig 2.13 Wallace Tree Method

multiplies two terms from the results of the partial product generated as shown in Fig 2.13. In order to perform the multiplication of two numbers with the Wallace method, partial product matrix is reduced to a two-row matrix by using a carry save adder and the remaining two rows are summed using a fast carry-propagate adder to form the product.

CSA performs the addition of m numbers in lesser duration compared to the simple addition. It takes three numbers (a+b+c) to add together and outputs two numbers, sum and carry (s+c).It is carried out in one time unit duration .In carry save adder, the carry(c) is bought until the last step and the ordinary addition carried out in the very last step .The most important application of a carry-save adder is to calculate the partial products in integer multiplication. This allows for architectures, where a tree of carry save adders is used to calculate the partial products very fast. One 'normal' adder is then used to add the last set of carry bits to the last partial products to give the final multiplication result. Usually, a very fast carry-look ahead or carry-select adder is used for this last stage, in order to obtain the optimal performance.

In Wallace tree architecture, all the bits of all of the partial products in each column are added together by a set of counters in parallel without propagating any carries. Another set of counters then reduces this new matrix and so on, until a two-row matrix is generated. Here a 3:2 counter is used. Then, a fast adder is used at the end to produce the final result. The advantage of Wallace tree is speed because the addition of partial products is now 0(logN).

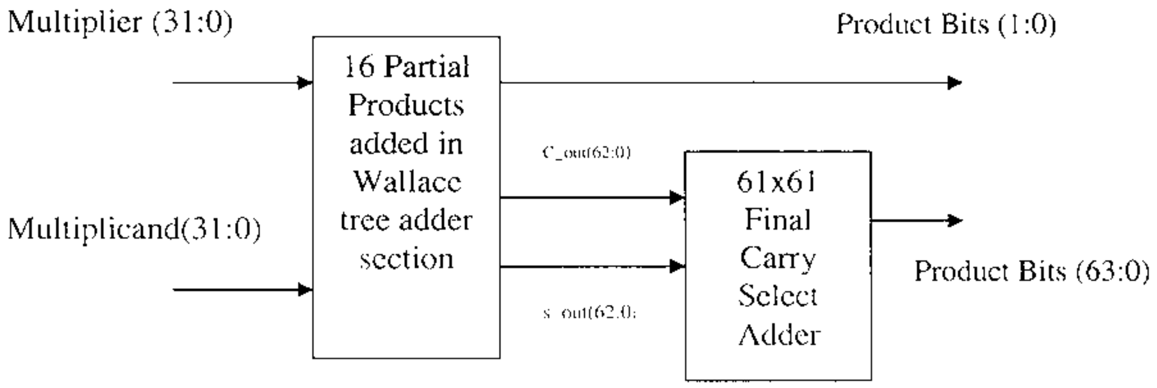
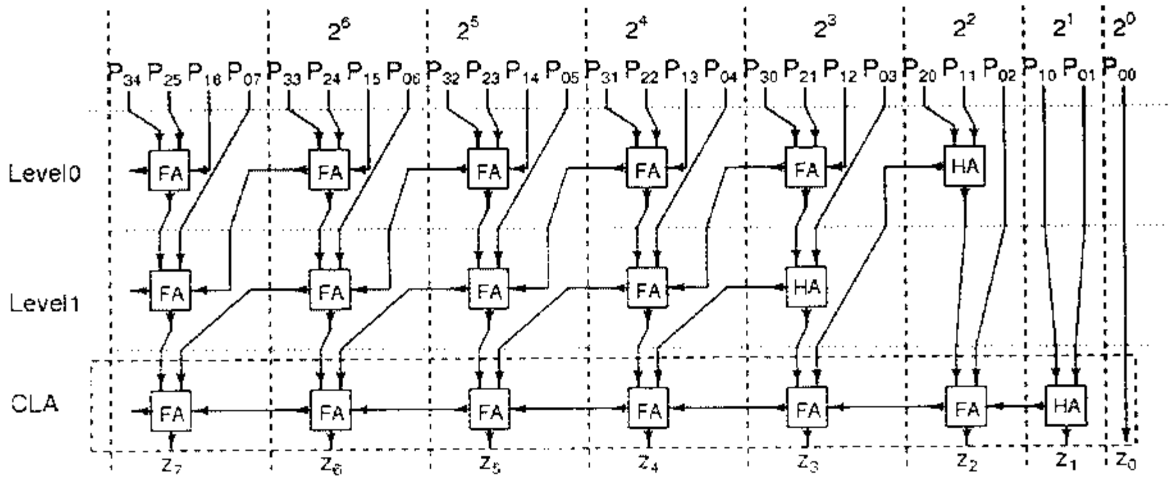


Fig.2.14 Block Diagram of 32-Bit Wallace Tree Multiplier.

from the block diagram partial products are added in Wallace tree block and at the end generates two LSB final product bits and sum and carry array of 61 bits which are added in final fast adder (CSA).

In a full adder delay from one input to one output can be different from the delay from another input to this same output. .So when a full adder is used as 3:2 compressor in Wallace tree, its input and outputs should not be treated equally. For example, if two 3:2 compressor are inter-connected in such a way that the longest path of these two compressor are inter-connected, then the total delay is the sum of the longest delay of these two compressor. A better way to inter-connect these two compressors is to connect the shortest delay path of one 32 compressor to the longest delay path of another 3:2 compressor. An algorithm to generate Wallace tree adder section that takes above point into consideration can be found in and this algorithm is used to implement the Wallace tree adder section to add all partial products and minimize the delay.



2^6
2^5
2^4
2^3
2^2
2^1
2^0
P_{34} P_{25} P_{16} P_{07} P_{33} P_{24} P_{15} P_{06} P_{32} P_{23} P_{14} P_{05} P_{31} P_{22} P_{13} P_{04} P_{30} P_{21} P_{12} P_{03} P_{20} P_{11} P_{02} P_{10} P_{01} P_{00}
Level0
Level1
CLA
FA
HA
Z_7 Z_6 Z_5 Z_4 Z_3 Z_2 Z_1 Z_0

Fig.2.15 Wallace tree for 4 x 4 Multiplication.

Wallace Tree CSA structure [8] in Fig 2.15 has been used to sum the partial products in reduced time. In this regard, when both algorithms are combined in one multiplier, there is a significant reduction in computing multiplications. The Wallace tree has three steps:

1. Multiply each bit of one of the arguments, by each bit of the other, yielding n^2 results. Depending on position of the multiplied bits, the wires carry different weights.

3. Group the wires in two numbers, and add them with a conventional adder.

The benefit of the Wallace tree is that there are only O(log n) reduction layers, and each layer has O(1) propagation delay. As making the partial products is O(1) and the final addition is O(log n), the multiplication is only O(log n), adding partial products with regular adders it would require O(log n)^2 time.

2.5.4 Final Adder and Accumulation

When the number of partial products is reduced to sum and carry words, a final adder is required to generate the multiplication result. The number of bits of the final adder is the sum of the number of bits of the multiplier and multiplicand. Thus, the data path width is usually doubled and the delay of this stage is most severe. Normally 4-bit CLAs can be used to reduce the delay and area requirements. This adder is a practical design with reduced delay at the price of more complex hardware. The carry look ahead design can be obtained by a transformation of the ripple carry design into a design in which the carry logic over fixed groups of bits of the adder is reduced to two-level logic. The main idea behind carry look-ahead addition is an attempt to generate all incoming carries in parallel and avoid waiting until the correct carry propagates from the stage (FA) of the adder where it has been generated. CLA adder is based on the fact that a carry signal will be generated in two cases:

(1) When both bits A_i and B_i are 1, or

(2) When one of the two bits is 1 and the carry-in (carry of the previous stage) is 1.

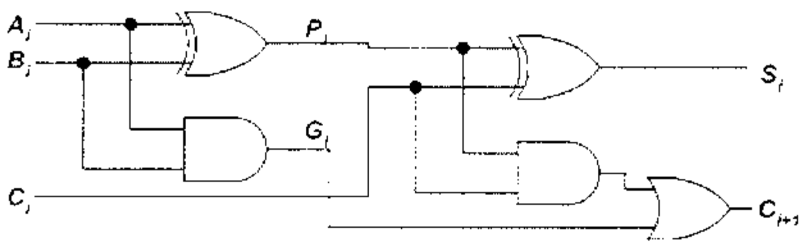


Fig.2.16 Carry Look Ahead Adder Circuit

column: namely A_i & B_i and the carry bit coming from the previous column (C_i).

In this circuit, the 2 internal signals P_i and G_i are given by:

$$P_i = A_i \text{ xor } B_i$$

$$G_i = A_i\, B_i$$

The output sum and carry can be defined as:

$$S_i = P_i \text{ xor } C_i$$

$$C_{i+1} = G_i + P_i\, C_i$$

G_i is known as the carry Generate signal since a carry (C_{i+1}) is generated whenever G_i =1, regardless of the input carry (C_i). P_i is known as the carry propagate signal since whenever P_i=1, the input carry is propagated to the output carry, i.e., $C_{i+1} = C_i$

2.6 MATHEMATICAL MODEL OF MAC

The n-bit 2's complement binary number can be expressed as

$$X = -2^{N-1} X_{N-1} + \sum_{i=0}^{N-2} 2^i X^i, \quad x_i \in 0,1$$

If the multiplicand X is expressed in base-4 type redundant sign digit form in order to apply the Booth's algorithm,

$$X = \sum_{i=0}^{N/2-1} d_i 4_i$$

$$d_i = -2x_{2i+1} + x_{2i} + x_{2i-1}$$

The multiplication result can be expressed as

$$X \times Y = \sum_{i=0}^{N/2-1} d_i 2^{2i} Y$$

$$P = X \times Y + Z = \sum_{i=0}^{N/2-1} d_i 2^i Y + \sum_{j=0}^{2N-1} z_i 2^i$$

In this d_i denotes the recoded bits of the multiplier it is computed using the booth algorithm and the recoded value is multiplied with the multiplicand X to generate partial products and they are given as a input to CSA based Wallace tree structure. Each of the two terms on the final equation P is calculated independently and the final result is produced by adding the two results. The MAC architecture implemented by this equation is called the standard design.

If N-bit data are multiplied, the number of the generated partial products is proportional to N. In order to add them serially, the execution time is also proportional to N. The multiplier architecture, which is the fastest, uses radix-2 Booth encoding that generates partial products and a Wallace tree based on CSA as the adder array to add the partial products. If radix-2 Booth encoding is used, the number of partial products which are the inputs to the Wallace tree, is reduced to half, resulting in the decrease in CSA tree step. In addition, the signed multiplication based on 2's complement numbers is also possible. Due to these reasons, most current used multipliers adopt this Booth encoding.

CHAPTER 3

MAC UNIT WITH 4-2 COMPRESSOR CIRCUIT

3.1 OVERVIEW OF MERGED MAC

In MAC architecture, speed is mainly increased by partial products reduction network and accumulator. But these operations require addition of large operands that involve long paths for carry propagation. Using tree architectures represent an attractive solution to speed up the partial products reduction process. The speed can be increased by applying a merging technique; here the Accumulation operation is merged within the partial products reduction tree used for multiplication.

The architecture is based on Binary trees constructed using 4-2 compressor circuits. Increasing the speed of operation is achieved by taking advantage of the available free input lines of the 4-2 compressors, which result from the parallelogram shape of the generated partial products, and using the bits of the accumulated value to fill in these gaps. This results in merging the accumulation operation within the multiplication process.

Using 4-2 compressors as the basic cell to construct the addition tree in parallel multipliers 2: 1 Compression ratio can be achieved. The 4-2 compressor circuits are fully utilizing the summation tree by feeding the accumulated data bits into the unused inputs of the 4-2 compressors so that the accumulation operation will be merged within the multiplication circuit, which will save the cost of an additional accumulator. This directly results in increasing the overall speed of the MAC operation. Power consumption and circuit area are also reduced.

3.2 CONSTRUCTION OF MERGED MAC UNIT

A basic MAC unit can be divided into two main blocks; the Multiplier and the Accumulator. The multiplier can also be divided into the partial products generation and reduction blocks. The partial product addition block can be further divided into a summation tree and a final adder. The summation network represents the core of the MAC unit. This construction leads to four basic blocks to be implemented. In the merged MAC unit as shown in Fig. 3.1 the addition network reduces the number of partial products into two operands representing a sum and a carry. The final adder is then used to generate the multiplication result out of these two operands. The last block is the accumulator, which is required to perform a double precision addition operation between the multiplication result and the accumulated operand. This block requires a very large adder due to the large operands size. This stage represents a bottleneck in the multiplication process in terms of speed since it involves horizontal carry propagation.

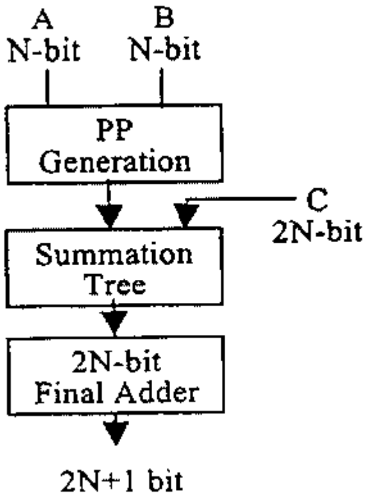


Fig. 3.1 Merged MAC Unit

3.3 CONSTRUCTION OF 4-2 COMPRESSOR CIRCUIT

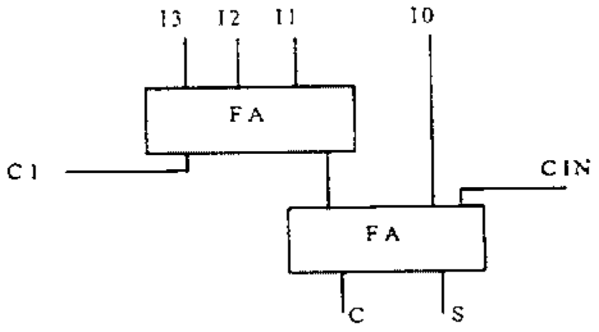


Fig. 3.2 Construction of 4-2 compressor circuit using 2 full adder cells

process by introducing parallelism. The tree structure, which was first introduced using 3-2 compressors suffers from irregular interconnections and is difficult to layout. It also results in high power consumption as a result of the capacitances introduced by large interconnects. A more regular structure is proposed based upon binary trees constructed using 4-2 compressors. Fig. 3.2 shows the construction of a 4-2 compressor circuit using two full adder cells.

3.4 DISTRIBUTION OF DATA BITS WITHIN THE 4-2 COMPRESSOR

The free inputs of the 4-2 compressors are taken to realize the accumulation operation by feeding the bits of the accumulated operand into the summation tree instead of putting zeroes. Some minor enhancements are required for the existing hardware to accommodate the incoming bits. Fig. 3.3 shows the new distribution of data among the tree. The bits of the accumulated value are inserted within the tree as early as possible in those empty locations at the corresponding columns so that those 4-2 compressors are fully utilized.

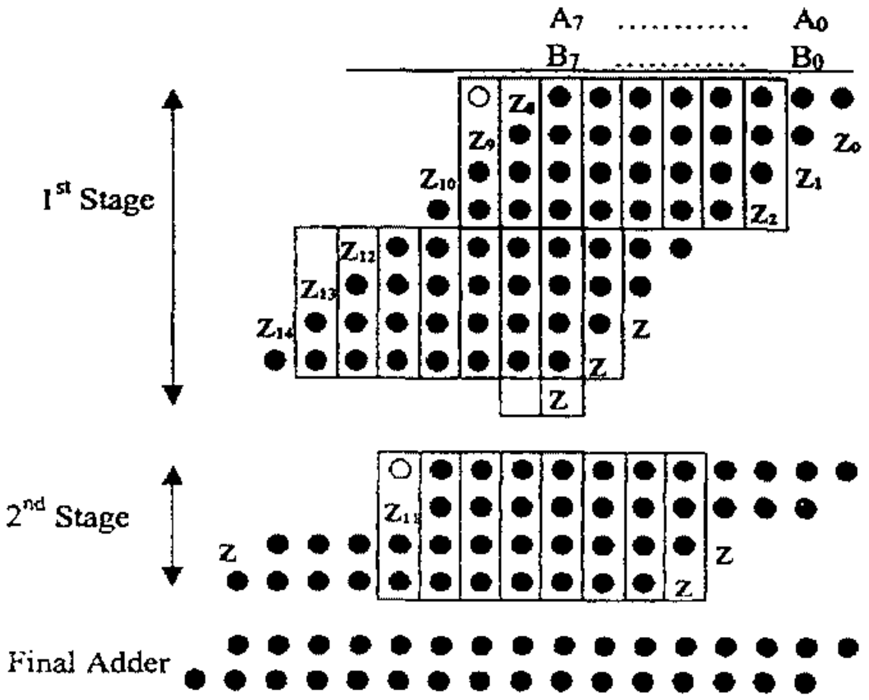


Fig. 3.3 Distribution of data bits

The bits ZO, ZI, Z2, Z3, Z4, Z5, Z6, Z7, Z8, Z9, ZIO, Z11.Z12, Z13, Z14 are inserted in the first stage, the bits Z3, Z4, Z11, Z15 are inserted in then second stage. The bit Z7, from 4 to 5 at the input of the compressors available at this column. To solve this problem, a 5-2 compressor is used to handle this extra bit.

The construction of the 5-2 compressor is shown in Figure 3.4. It is noticed that the delay of the 5-2 compressor is equivalent to that of two cascaded full adders.

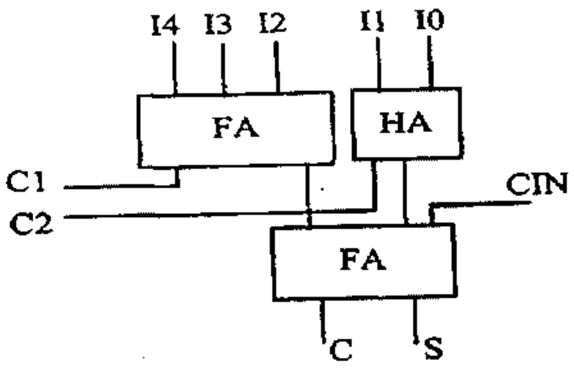


Fig. 3.4 Construction of the 5-2 Compressor.

3.5 DRAWBACKS OF MAC UNIT WITH 4-2 COMPRESSOR CIRCUIT

The major disadvantage that occurs due to the parallelogram shape of the generated partial products is that some of the 4-2 compressor circuits are not fully utilized since they cannot able to get complete 4 inputs which results in hardware inefficiency.

CHAPTER 4

PROPOSED PARALLEL MAC ARCHITECTURE

4.1 DERIVATION OF MAC ARITHMETIC

The expression for the new arithmetic will be derived from equations of the standard design. From this result, a new VLSI architecture for MAC can be designed.

4.1.1 BASIC CONCEPT

If an operation to multiply two N–bit numbers and accumulate into a 2N-bit number is considered, the critical path is determined by the 2N-bit accumulation operation. If a pipeline scheme is applied for each step in the standard design of Fig. 2.7, the delay of the last accumulator must be reduced in order to improve the performance of the MAC. The overall performance of the standard MAC is improved by eliminating the accumulator itself by combining it with the CSA function. If the accumulator has been eliminated, the critical path is then determined by the final adder in the multiplier.

The basic method to improve the performance of the final adder is to decrease the number of input bits. In order to reduce this number of input bits, the multiple partial products are compressed into a summand a carry by CSA. The number of bits of sums and carries to be transferred to the final adder is reduced by adding the lower bits of sums and carries in advance within the range in which the overall performance will not be degraded. A 2-bit CLA is used to add the lower bits in the CSA. In addition, to increase the output rate when pipelining is applied, the sums and carry from the CSA are accumulated instead of the outputs from the final adder in the manner that the sum and carry from the CSA in the previous cycle are inputted to CSA. Due to this feedback of both sum and carry, the number of inputs to CSA increases, compared to the standard design. In order to efficiently solve the increase in the amount of data, CSA architecture is modified to treat the sign bit.

The above mentioned concept is applied to standard design equation to express the new MAC arithmetic. Then, the multiplication would be transferred to a hardware architecture that complies with the proposed concept, in which the feedback value for accumulation will be modified and expanded for the new MAC.

First, if the multiplication is decomposed and rearranged, it becomes

$$X \times Y = d_0 2Y + d_1 2^2 Y + d_2 2^4 Y + ... + d_{N/2-1} 2^{N-2} Y$$

If the above equation is divided into the first partial product, sum of the middle partial products, and the final partial product, it can be re-expressed as

$$X \times Y = d_0 2Y + \sum_{i=0}^{N/2-2} d_i 2^{2i} Y + d_{N/2-1} 2^{N-2} Y$$

If Z is first divided into upper and lower bits and rearranged, then the above equation becomes

$$z = \sum_{i=0}^{N-1} z_i 2^i + \sum_{i=N}^{2N-1} z_i 2^i$$

The second term can be separated further into the carry term and sum term as

$$\sum_{i=N}^{2N-1} z_i 2^i = \sum_{i=0}^{N-1} z_{N+i} 2^i 2^N = \sum_{i=0}^{N-2} (c_i + s_i) 2^i 2^N$$

Thus, the above equation is finally separated into three terms as

$$Z = \sum_{i=0}^{N-1} Z_i 2^i + \sum_{i=0}^{N-2} c_i 2^i 2^N + \sum_{i=0}^{N-2} s_i 2^i 2^N$$

Now the new MAC arithmetic can be expressed as

$$P = \left(d_0 2Y + \sum_{i=0} d_i 2 \; Y + d_{N/2-1} 2 \; Y \right) + \left(\sum_{i=0} Z_i 2 2 + \sum_{i=0} c_i 2 2 + \sum_{i=0} s_i 2 2 \right)$$

The above equation can be re-expressed as

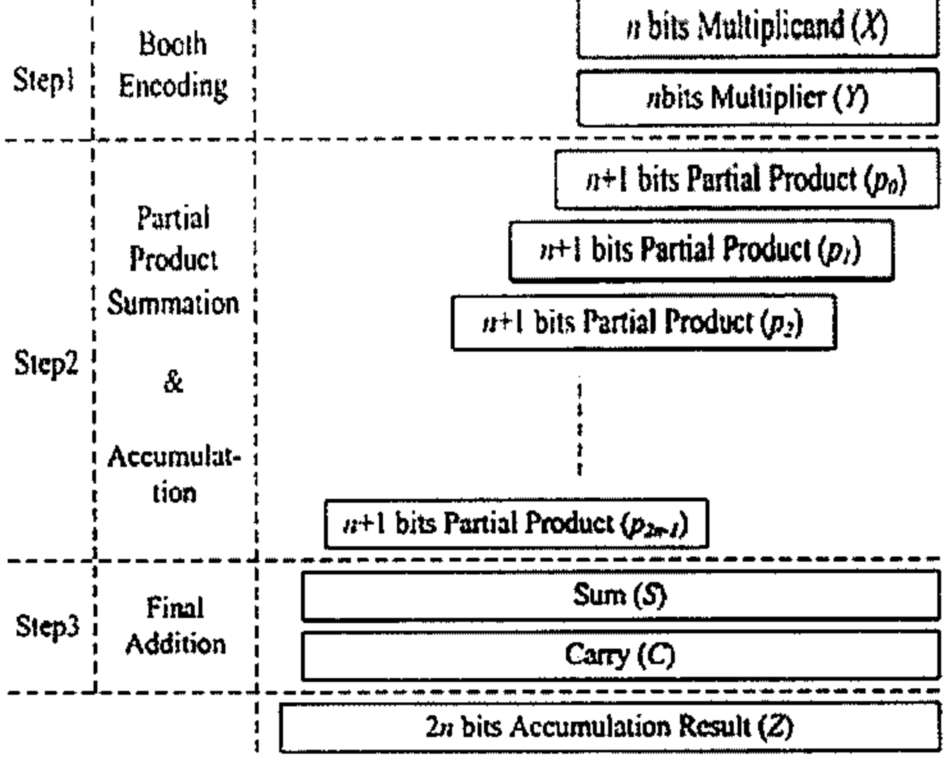
$$P = \left(d_0 2Y + \sum_{i=0}^{N-1} Z_i 2^i \right) + \left(\sum_{i=0}^{N/2-1} d_i 2^{2i} Y + \sum_{i=0}^{N-2} c_i 2^i 2^N \right) + \left(d_{N/2-1} 2^{N-2} Y + \sum_{i=0}^{N-2} s_i 2^i 2^N \right)$$

The first parenthesis on the right is the operation to accumulate the first partial product with the added result of the sum and the carry. The second parenthesis is the one to accumulate the middle partial products with the sum of the CSA that was fed back. Finally, the third parenthesis expresses the operation to accumulate the last partial product with the carry of the CSA.

4.2 PIPELINED MAC ARCHITECTURE

The modified MAC is organized into three steps. When compared with Fig. 2.2, it is easy to identify the difference that the accumulation has been merged into the process of adding the partial products. Another big difference from Fig. 2.7 is that the final addition process in step3 is not always run even though it does not appear explicitly in Fig. 4.1. Since accumulation is carried out using the result from step2 instead of that from step3, step3 does not have to be run until the point at which the result for the final accumulation is needed.

The hardware architecture of the MAC to satisfy the process in Fig.4.1 is shown in Fig.4.2. The n-bit MAC inputs, X and Y, are converted into an (n+1)-bit partial product by passing through the Booth encoder. In the CSA and accumulator, accumulation is carried out along with the addition of the partial products. As a result, n-bit S, C and Z (the result from adding the lower bits of the sum and carry) are generated. These three values are fed back and used for the next accumulation. If the final result for the MAC is needed, P[2n-1: n] is generated by adding S and C in the final adder and combined with P[n-1: n] that was already generated.



Step1
Booth Encoding
n bits Multiplicand (X)
nbits Multiplier (Y)
Step2
Partial Product Summation & Accumulation
n+1 bits Partial Product (p0)
n+1 bits Partial Product (p1)
n+1 bits Partial Product (p2)
n+1 bits Partial Product (p2n-1)
Step3
Final Addition
Sum (S)
Carry (C)
2n bits Accumulation Result (Z)

Fig.4.1 Modified Arithmetic operation of multiplication and accumulation.

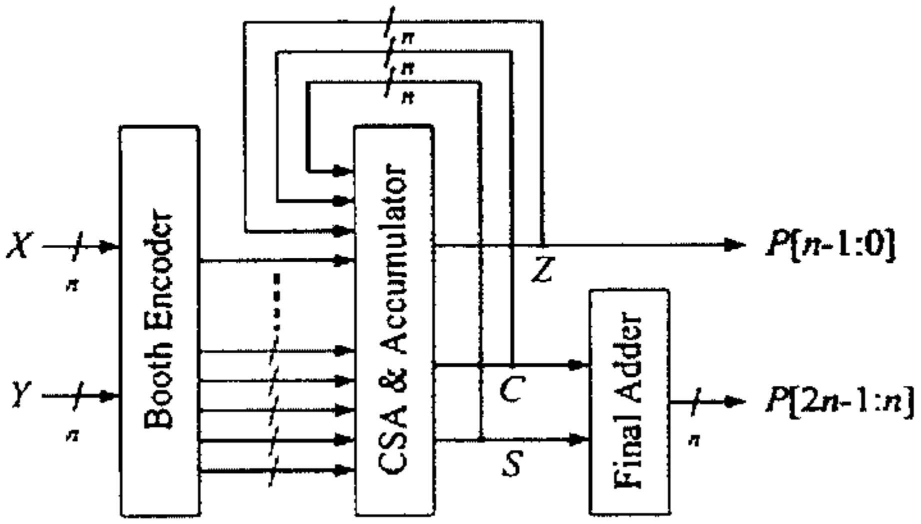


Fig.4.2 Hardware architecture of the modified MAC.

The architecture of the hybrid-type CSA that complies with the operation of the proposed MAC is shown in Fig.3.3, which performs 8x8-bit operation. S_i is to simplify the sign expansion and N_i is to compensate 1's complement number into 2's complement number. S[i] and C[i] correspond to the i-th bit of the feedback sum and carry. Z[i] is the i-th bit of the sum of the lower bits for each partial product that were added in advance and Z'[i] is the previous result.

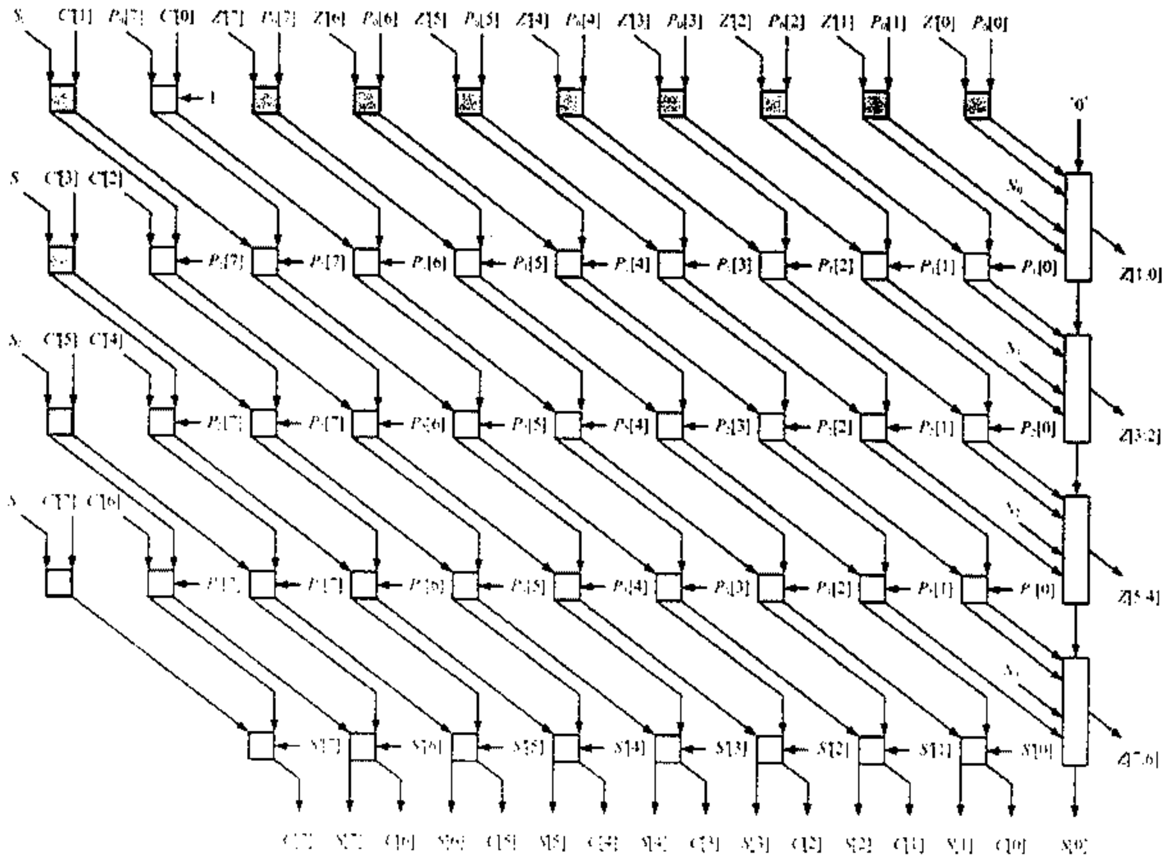


Fig.4.3. Architecture of the Modified CSA tree.

In addition, $P_j[i]$ corresponds to the i-th bit of the j-th partial product. Since the multiplier is for 8 bits, totally four partial products P0[7: 0] ~ P3[7: 0] are generated from the Booth encoder. This CSA requires at least four rows of FAs for the four partial products. Thus, totally five FA rows are necessary since one more level of rows are needed for accumulation. For an n x n-bit MAC operation, the level of CSA is (n/2). The white square in with five inputs is a 2-bit CLA with a carry input.

The critical path in this CSA is determined by the 2-bit CLA. It is also possible to use FAs to implement the CSA without CLA. However, if the lower bits of the previously generated partial product are not processed in advance by the CLAs, the number of bits for the final adder will increase. When the entire Multiplier or MAC is considered, it degrades the performance if CLA is not used.

CSA, CCA and Carry Look-Ahead Adder (CLAs) can be used to implement the final fast addition. CLA is widely used and can be easily implemented in dynamic domino CMOS logic with the limitation of full-custom design. For standard static CMOS circuit, CCA and CSA are preferred and can easily be implemented using a standard cell library. In contrast to the CSA, CCA needs to use XOR logic to produce the final results. This translates in more delay as compared to a same bit-width CSA. The CSA needs to store both the conditional sum and carry together. As a result, more multiplexers are used than for a CCA. To combine the benefits of both adders, a mixed CSA-CCA architecture was implemented to compute a final fast addition.

CHAPTER 5

SIMULATION RESULTS

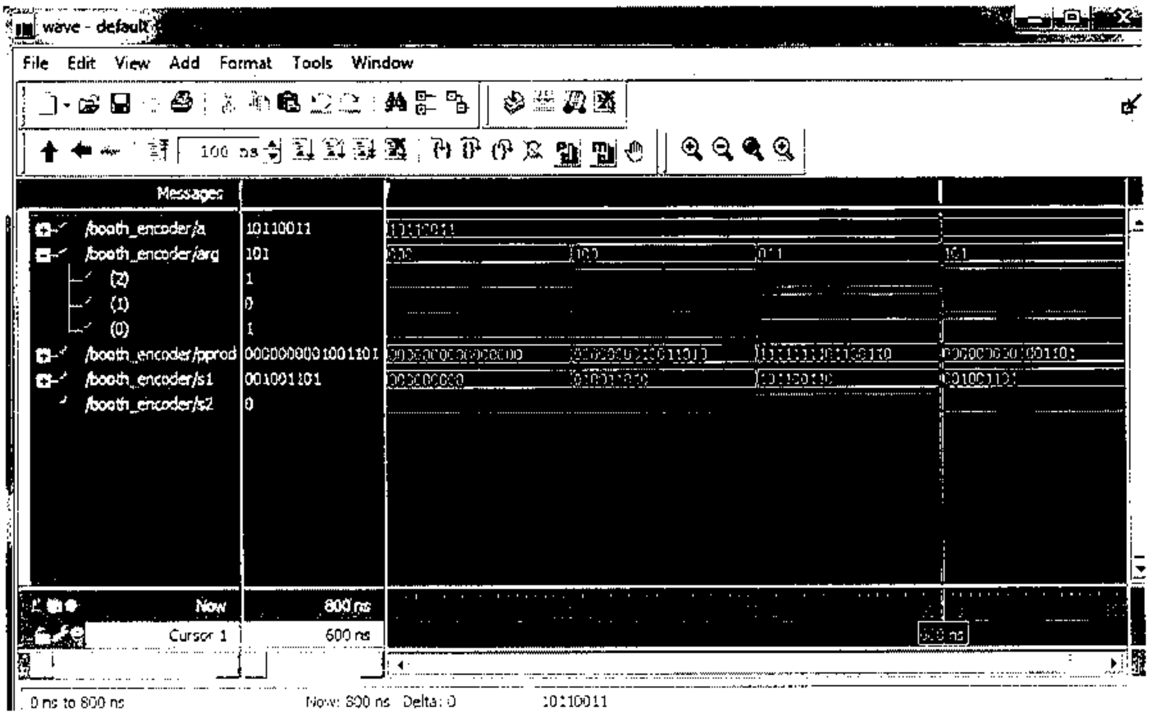
The tools used to obtain the simulated output for pipelined MAC architecture are as follows. They are

1) Modelsim 6.4

2) Microwind

The simulated waveforms are obtained by assigning the input values at various levels of extraction and the corresponding outputs are obtained from the assigned inputs. The outputs obtained are complementary with respect to the corresponding complementary inputs. The simulated waveforms of the proposed work are shown here.

5.1 PARTIAL PRODUCT GENERATION



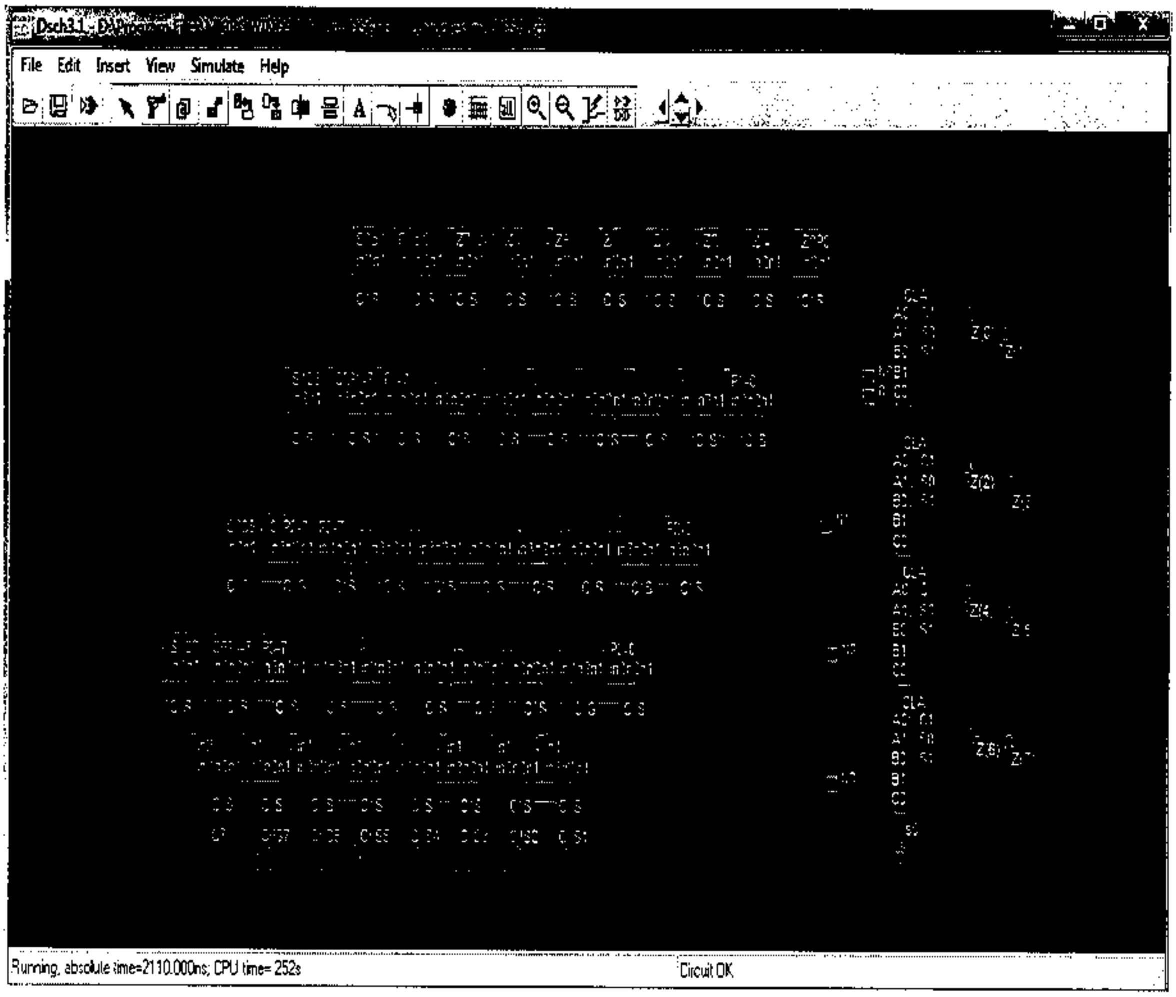
```
/booth_encoder/a        10110011
/booth_encoder/arg      101
   (2)                  1
   (1)                  0
   (0)                  1
/booth_encoder/pprod    0000000001001101
/booth_encoder/s1       001001101
/booth_encoder/s2       0
Now                     800 ns
Cursor 1                600 ns
0 ns to 800 ns          Now: 800 ns  Delta: 0      10110011
```

Fig.5.1 Simulation output of Booth Encoder

modified booth encoding, done by using Modelsim software.

5.2 HYBRID CSA ARCHITECTURE

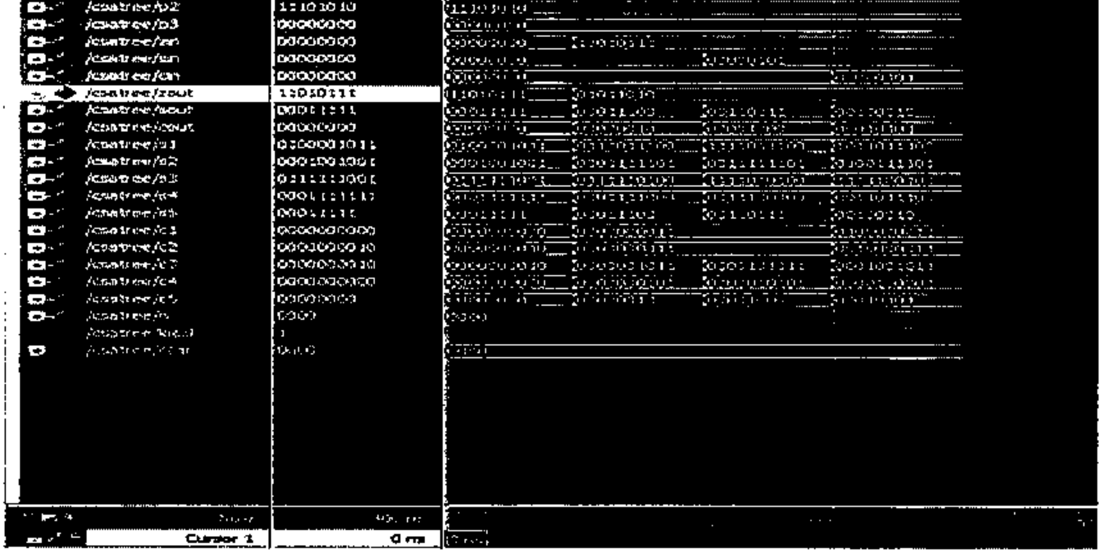
Fig. 5.2 shows the design of hybrid carry save adder architecture, done by using Microwind software.



Running, absolute time=2110.000ns; CPU time= 252s

Circuit OK

Fig.5.2 Design of CSA tree Architecture

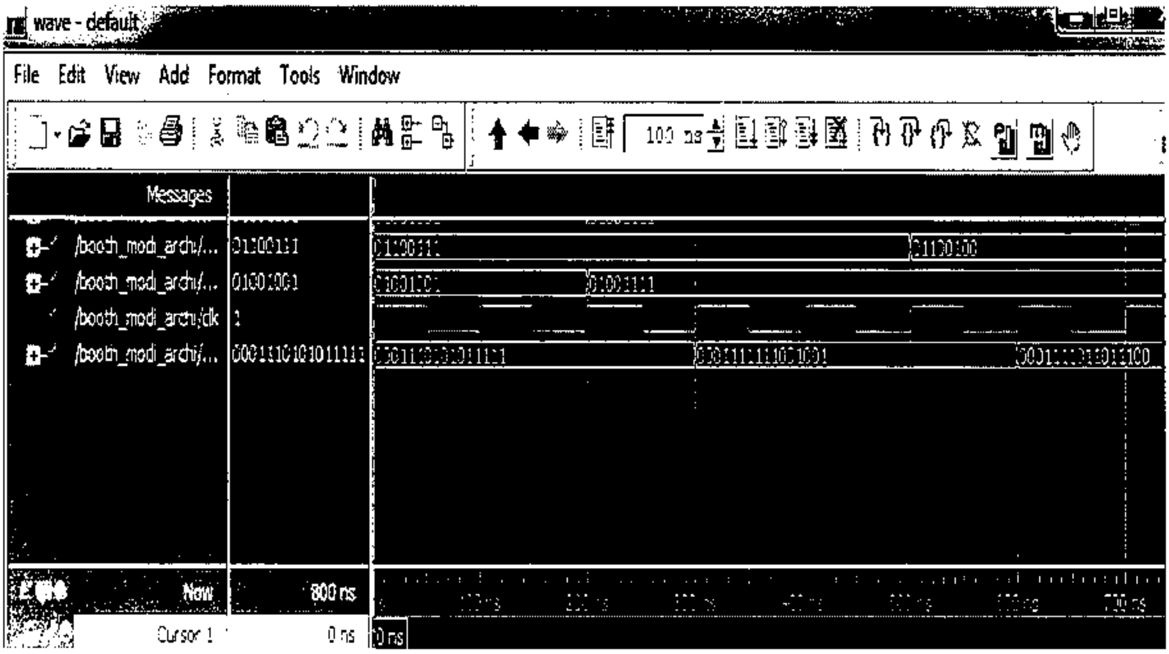


```
/csatree/b2      11101010
/csatree/b3      00000000
/csatree/an      00000000
/csatree/sn      00000100
/csatree/cn      00000000
/csatree/zout    11010111
/csatree/sout    00001111
/csatree/cout    00000000
/csatree/s1      0100001011
/csatree/s2      0001001001
/csatree/s3      0111110001
/csatree/s4      0001111111
/csatree/s5      00011111
/csatree/c1      0000000000
/csatree/c2      0001000010
/csatree/c3      0000000010
/csatree/c4      0000000000
/csatree/c5      00000000
/csatree/n       0000
/csatree/[illegible]  1
/csatree/[illegible]  0000
Cursor 1         0 ns
```

Fig.5.3 Simulation output of Hybrid CSA tree Architecture

5.3 SIMULATED OUTPUT OF PROPOSED MAC

Fig. 5.4 shows the simulated output of proposed architecture of MAC, done by using ModelSim software.



```
Messages
/booth_modi_archi/...   01100111            01100111                                01100100
/booth_modi_archi/...   01001001            01001001          01001111
/booth_modi_archi/clk   1
/booth_modi_archi/...   0001110101011111    0001110101011111  0001111111001001      0001110111011100
Now                     800 ns
Cursor 1                0 ns                0 ns
```

Fig. 5.4 Simulation output of proposed MAC unit

COMPRESSOR CIRCUIT

Design summary:

Selected Device: Xilinx FPGA- 2s300eft256-6

Number of Slices:	142	out of	3072	4%
Number of Slice Flip Flops:	96	out of	6144	1%
Number of 4 input LUTs:	246	out of	6144	4%
Number of bonded IOBs:	33	out of	182	18%
Number of GCLKs:	1	out of	4	25%

Total equivalent gate count for design = 2,505

Timing Summary:

Speed Grade: -6

Minimum input arrival time before clock: 10.014ns

Maximum output required time after clock: 6.514ns

5.5 SYNTHESIS REPORT FOR MAC WITH HYBRID CSA ARCHITECTURE

Design summary:

Selected Device: Xilinx FPGA- 2s300eft256-6

Number of Slices:	105	out of	3072	3%
Number of Slice Flip Flops:	16	out of	6144	0%
Number of 4 input LUTs:	17	out of	6144	2%
Number of bonded IOBs:	33	out of	182	18%
Number of GCLKs:	1	out of	4	25%

Total equivalent gate count for design = 1,304

Timing Summary:

Speed Grade: -6

Minimum input arrival time before clock: 8.026ns

Maximum output required time after clock: 4.264ns

The power calculation of conventional MAC with 4-2 compressor and modified MAC with Hybrid CSA Architecture are shown below.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		67
Vccint 2.50V:	27	67
Vcco25 2.50V:	0	0
Logic:	7	18
Outputs:		
Vcco25	0	0
Signals:	19	48

Table 5.1 Power Calculation of MAC with 4-2 compressor.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		49
Vccint 2.50V:	17	43
Vcco33 3.30V:	2	7
Inputs:	1	3
Logic:	9	23
Outputs:		
Vcco33	0	0
Signals:	6	16
Quiescent Vcco33 3.30V:	2	7

Table 5.2 Power Calculation of modified MAC with Hybrid CSA Architecture.

Performance parameters are used to compare the MAC Unit with 4-2 compressor and with Hybrid CSA Architecture. The table 5.3 shows the difference between MAC Unit with 4-2 compressor and with Hybrid CSA Architecture in terms of gate count, speed and power.

Parameters	**MAC Unit with 4-2 compressor circuit**	**MAC Unit with Hybrid CSA Architecture**
Gate count	2,505	1,304
Delay(ns)	4.26	1.912
Power(mw)	67	49

Table 5.3 Performance Analysis of MAC Unit with 4-2 Compressor Circuit and MAC Unit with Hybrid CSA Architecture.

5.7.1 Gate Count Analysis

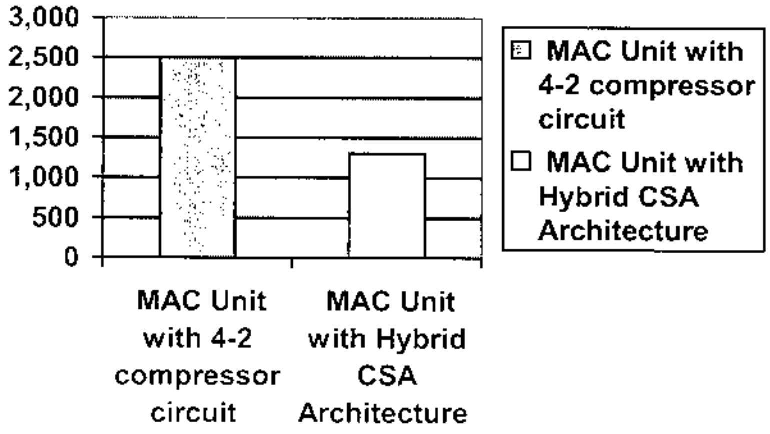


Fig.5.5 Gate Count Analysis of MAC Unit

with Hybrid CSA Architecture. The gate count of the proposed MAC unit with Hybrid CSA Architecture decreases by 47.94%.

5.7.2 Delay Analysis

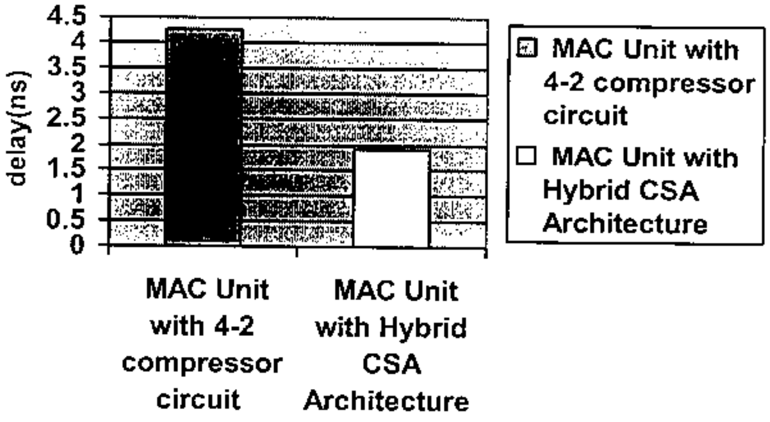


Fig.5.6 Delay Analysis of MAC Unit

Fig.5.6 shows the delay of MAC Unit with 4-2 compressor and with Hybrid CSA Architecture. The delay is reduced by 52.23% hence the speed of the proposed MAC unit increases.

4.7.3 Power Analysis

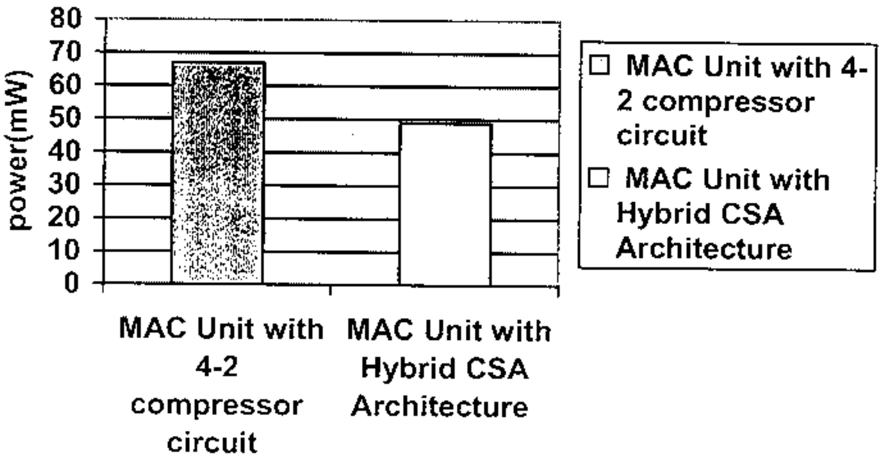


Fig.5.7 Delay Analysis of MAC Unit

and with Hybrid CSA Architecture The power dissipation of MAC unit with hybrid CSA architecture decreases by 31.8%. Hence this pipeline architecture MAC architecture using hybrid CSA tree can be applicable in digital signal processing applications that require low power, low area and high speed.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

The new MAC architecture to execute the multiplication-accumulation operation, which is the key operation, for digital signal processing and multimedia information processing is proposed. By removing the independent accumulation process that has the largest delay and merging it to the compression process of the partial products, the overall MAC performance was improved almost twice as much as in the existing system. The modified CSA architecture increases the output rate by optimizing the pipeline efficiency, intermediate calculation results are accumulated in the form of sum and carry instead of the final adder outputs, this in turn reduces the number of inputs to final adder and thereby minimizes the delay. Consequently, the proposed architecture can be used effectively in the area requiring high throughput such as a real-time digital signal processing.

6.2 FUTURE SCOPE

In Future, the Pipelined MAC architecture may be used in the efficient design of digital signal processing circuits such as FIR and IIR filter by constructing the parallel MAC architecture using the design tool such as Microwind.

[1] Asadee and Pishva. (2009) "A new MAC design using high-speed partial product summation tree" ,IEEE Trans. comput.,Vol 41,No.1, pp. 231–234.

[2] Booth A.D. (1952) "A signed binary multiplication technique," Quart. J. Math., Vol. IV, pp. 236–240.

[3] Cooper A. R. (1988) "Parallel architecture modified Booth multiplier," Proc. Inst. Electr. Eng. G, Vol. 135, pp. 125–128.

[4] Elguibaly F. (2000) "A fast parallel multiplier–accumulator using the modified Booth algorithm," IEEE Trans. Circuits Syst., Vol. 27, No. 9, pp.902–908.

[5] Fadavi-Ardekani J. (1993) "MxN Booth encoded multiplier generator using optimizedWallace trees," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., Vol. 1, No. 2, pp. 120–125.

[6] Fayed A and Bayoumi M. (2002) "A merged multiplier-accumulator for high speed signal processing applications," Proc. ICASSP, Vol. 3, pp. 3212–3215.

[7] Shanbag N. R and Juneja P. (1988) "Parallel implementation of a 4x4-bit multiplier using modified Booth's algorithm," IEEE J. Solid-State Circuits, Vol. 23, No. 4, pp. 1010–1013.

[8] Wallace C. S. (1964) "A suggestion for a fast multiplier," IEEE Trans. Electron Comput.. Vol. EC-13, No. 1, pp. 14–17.

[9] Young-Ho Seo and Dong-Wook Kim. (2010) "A New VLSI Architecture of Parallel Multiplier–Accumulator Based on Radix-2 Modified Booth Algorithm", IEEE Trans. On VLSI, Vol. 18, No.2, pp. 201–208.

[10] G. Goto, T. Sato, M. Nakajima. and T. Sukemura(1992),"A 54X54 regular structured tree multiplier," IEEE J. Solid-State Circuits, vol. 27, no.9,pp. 1229–1236.

structured arithmetic circuits. IEEE Transaction on Very Large Scale Integration (VLSI) System vol. 13, no.6, pp. 686–695.

[12] Zicari P, Perri S, Corsonello P and Cocorullo G. (2005) "An optimized adder accumulator for high speed MACs," Proc. ASICON 2005, Vol.2, pp. 757–760.

[13] J.J.F. Cavanagh, Digital Computer Arithmetic, NewYork:McGrawHill,1984.

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

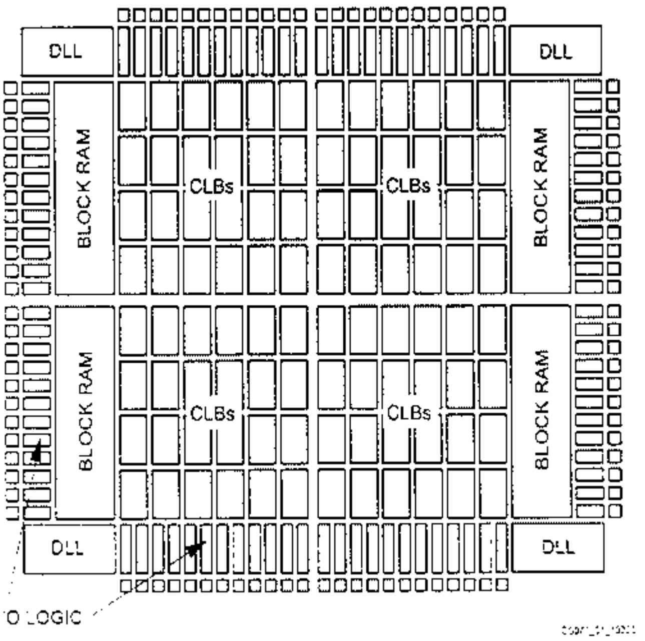


Figure 1: Basic Spartan-IIE Family FPGA Block Diagram

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**

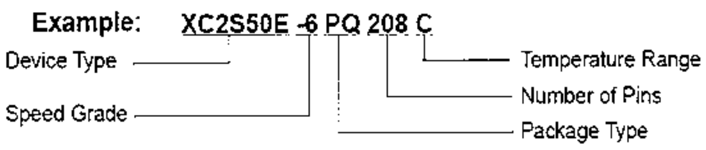
	Pins	144	208	256	456
	Type	Plastic TQFP	Plastic PQFP	Fine Pitch BGA	Fine Pitch BGA
Device	Code	TQ144	PQ208	FT256	FG456
XC2S50E	-6	C, I	C, I	C, I	-
	-7	(C)	(C)	(C)	-
XC2S100E	-6	C, I	C, I	C, I	C, I
	-7	(C)	(C)	(C)	(C)
XC2S150E	-6	-	(C, I)	(C, I)	(C, I)
	-7	-	(C)	(C)	(C)
XC2S200E	-6	-	C, I	C, I	C, I
	-7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