P-3461

A NEW LOOK-UP-TABLE OPTIMIZATION FOR MEMORY BASED MULTIPLICATION

By

K.A.SIVAPRAKASHAM

Reg No. 0920106014

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 **"A NEW LOOK-UP-TABLE OPTIMIZATION FOR MEMORY BASED MULTIPLICATION"** is the bonafide work of **Mr.K.A.SIVAPRAKASHAM (Reg. No. 0920106014)** 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.

S. Umamaheswari

Project Guide

Ms. S. Umamaheswari

Head of the Department

Dr. Rajeswari Mariappan

The candidate with university Register No. 0920106014 is examined by us in the project viva-voce examination held on ..21.-.4.-.2011...

21/4/11

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 us 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 and 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. S. Umamaheswari M.E,** 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

The look-up-table (LUT) memory-based multipliers to be used in digital signal processing in many applications. A novel anti-symmetric product coding (APC) scheme is to reduce the LUT size by further half, where the LUT output is added with or subtracted from a fixed value. The optimized LUTs for small input width could be used for efficient implementation of high-precision LUT-multipliers, where the total contribution of all such fixed offsets could be added to the final result or could be initialized for successive accumulations.

The anti-symmetric product coding (APC) and odd-multiple-storage (OMS) techniques for lookup-table (LUT) design for memory-based multipliers to be used in Digital signal processing applications. The proposed combined approach provides a reduction in LUT size to one-fourth of the conventional LUT. The simple technique is used for selective sign reversal to be used in the proposed design. The proposed LUT design for small input sizes can be used for efficient implementation of high-precision multiplication by input operand decomposition.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

APC	----	Anti-symmetric Product Coding
ASIC	----	Application Specific Integrated Circuit
EDA	----	Electronic Design Automation
FIR	----	Finite Impulse Response
FPGA	----	Field Programmable Gate Array
HDL	----	Hardware Description Language
LUT	----	Look Up-Table
OMS	----	Odd Multiple Storage
RTL	----	Register Transfer Level
SoC	----	System on Chip

CHAPTER 1

INTRODUCTION

1.1 MULTIPLICATION

The multiplication is a commonly-used operation of 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 high speed devices. Such trend has to design a low-power multiplier to reach power saving.

Multipliers are key components of many high performance systems such as FIR filters, microprocessors, digital signal processors, etc. A system's performance is generally determined by the performance of the multiplier because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. 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 [1]. It is also clear that lowering the power can result much cheaper packaging, resulting cost effective ASIC design. For these reasons, low power design has become as important as delay optimization.

Along with the progressive device scaling, semiconductor memory has become cheaper, faster, and more power-efficient. Moreover, according to the projections of the international technology roadmap for semiconductors, embedded memories will have dominating presence in the system on- chips (SoCs), which may exceed 90% of the total SoC content [2].

It has also been found that the transistor packing density of memory components is not only higher but also increasing much faster than those of logic components. Apart from that, memory-based computing structures are more regular than the multiply–accumulate structures and offer many other advantages, e.g., greater potential for high-throughput and low-latency implementation and less dynamic power consumption. Memory-based computing is well suited for many digital signal processing (DSP) algorithms, which involve multiplication with a fixed set of coefficients.

Table lookup can replace any coefficient multiplication or unary operation .Although table lookup is often simpler than the actual calculation, the table size grows exponentially with the input signal range [3]. However, for image and video applications, most signals are unsigned 8 bit values, which require only 256 possible cases, so the table based approach can be implemented with a reasonable cost. To implement coefficient multiplication, where the coefficient is constant, avoid using a multiplier, traditional lossless transforms approximate the given coefficient with Look up table multiplication. Then the coefficient multiplication can be implemented using shifts and additions.

1.2 LOOK UP TABLE

The logic block in an FPGA typically has a small number or inputs and outputs. A variety of FPGA products are on the market, featuring different type of logic blocks. The most commonly used logic block is Look Up Table (LUT), which contains storage circuits that are used to implement a small logic function. Each cell is capable of holding a single logic values, either 0 or1.

The stored value is produced as the output of the storage cell. LUTs of various sizes may be created, where the size is defined by the number of inputs. Figure 3.5 shows the structure of a small LUT. It has two inputs, x1 and x2, and one output f. It is capable of implementing any logic function of two variables. A two variable truth table has four row s. this LUT has four storage cells. One cell corresponds to the output value in each row of the truth table.

The input variables x1 and x2 are used as the select inputs of three multiplexers, which is depending on the valuation of x1 and X2. Select the content of one f the four storage cells as the output of the LUT.

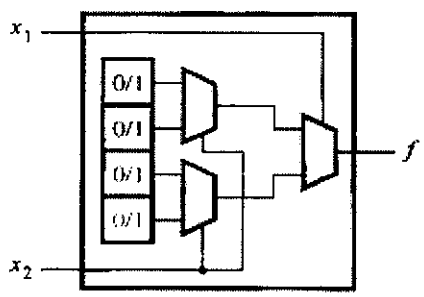


Figure 1.1: Circuit for Two Inputs LUT

X1	X2	F1
0	0	1
0	1	0
1	0	0
1	1	1

Table 1.1: Truth Table of Two Inputs LUT.

The logic functions can be realized in the two-input LUT shown in the truth table of table 1.1. The function f1 from this table can be stored in the LUT as illustrated in figure 1.2.

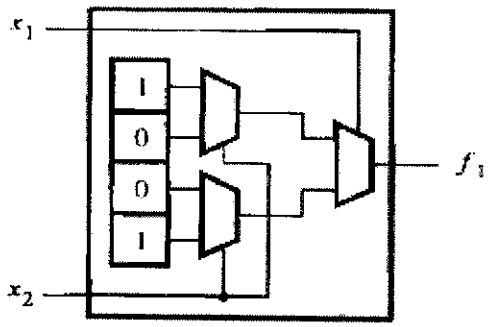


Figure 1.2: Storage Cell Content in the LUT

The arrangement or multiplexers in the LUT correctly realizes the function f1. When .x1=x2=0, the output or the LUT is driven by the top storage cell, which represents the entry in the truth table for x1x2=0. Similarly, for all valuations of *x1* and x2, the logic value stored in the storage cell corresponding to the entry in the truth table chosen by the particular valuation appears on the LUT output.

Providing access to the contents or storage cells is only one way in which multiplexers can he used to implement logic functions. Figure 1.3 shows a three-input LUT. It has eight storage cells because a three-variable truth table has eight rows. In commercial FPGA chips, LUTs usually have either four or five inputs, which require 16 and 32 storage cells, respectively.

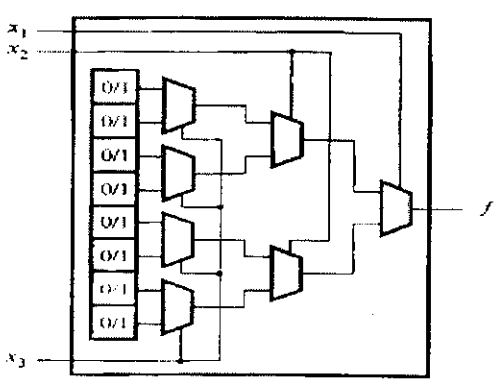


Figure 1.3: A Three Input LUT

1.3 LOOK UP TABLE BASED MULTIPLIER

Along with the device scaling over the years, semiconductor memory has become heaper, faster and more power-efficient. Moreover, according to the projections of the international technology roadmap for semiconductors (ITRS) , embedded memories will have dominating presence in the system-on chips (SoC), which may exceed 90% of total SoC content. It has also been found that the transistor packing density of memory components is not only high but also increasing much faster than the transistor density of logic components. The logic block in an FPGA typically has a small number or inputs and outputs. A variety of FPGA products are on the market, featuring different type of logic blocks. The most commonly used logic block is Look Up Table (LUT), which contains storage circuits that are used to implement a small logic function. Each cell is capable of holding a single logic values, either 0 or1.

Apart from that, the memory-based computing structures are more regular than the multiply-accumulate structures; and offer many other advantages, e.g., greater potential for high throughput and low-latency implementation; and less dynamic power consumption. Memory-based computing is well-suited for many digital signal processing (DSP) algorithms, which involve multiplication with fixed set of coefficients.

The basic functional model of memory-based multiplier is shown in Fig.1.1. Let A be a fixed coefficient and X be an input word to be multiplied with A. Assuming X to be a positive binary number with word-length L, there can be 2L possible values of X, and accordingly, there can be 2L possible values of product C = A·X. Therefore, for memory based multiplication, an LUT of 2L words consisting of precompiled product values corresponding to all possible values of X, is conventionally used. The product-word A·Xi is stored at the location whose address is the same as Xi for $0 \leq 2L-1$, such that if L-bit binary value of Xi is used as address for the LUT, then the corresponding product value A·Xi is available as its output.

Several architectures have been reported in the literature for memory-based implementation of DSP algorithms involving orthogonal transforms and digital filters. But, we do not find any significant work on LUT optimization for memory based multiplication. It is shown that although 2L possible values of X correspond to 2L

possible values of C = A · X, only (2L/2) words corresponding to the odd multiples of A may only be stored in the LUT while the even multiples of A could be derived by left-shift operations of one of those odd multiples.

We have referred to this scheme as odd-multiple storage LUT in the rest of this paper. Using this approach, one can reduce the LUT size to half, but it has significant combinational overhead since it requires a barrel-shifter along with a control-circuit to generate the control-bits for producing a maximum of (L – 1) left-shifts, and an encoder to map the L-bit input word to (L – 1)-bit LUT address. In this paper, we propose two new schemes for optimization of LUT with lower area- and time-overhead. One of the proposed optimization is based on exclusion of sign-bit from the LUT address, and the other optimization is based on a recoding of stored product word.

A conventional lookup-table (LUT)-based multiplier is shown in Fig. 1.4, where A is a fixed coefficient, and X is an input word to be multiplied with A. Assuming X to be a positive binary number of word length L, there can be 2^L possible values of X, and accordingly, there can be 2^L possible values of product $C = A \cdot X$. Therefore, for memory-based multiplication, an LUT of 2^L words, consisting of pre-computed product values corresponding to all possible values of X, is conventionally used.

The logic block in an FPGA typically has a small number or inputs and outputs. A variety of FPGA products are on the market, featuring different type of logic blocks. The most commonly used logic block is Look Up Table (LUT), which contains storage circuits that are used to implement a small logic function. Each cell is capable of holding a single logic values, either 0 or1.

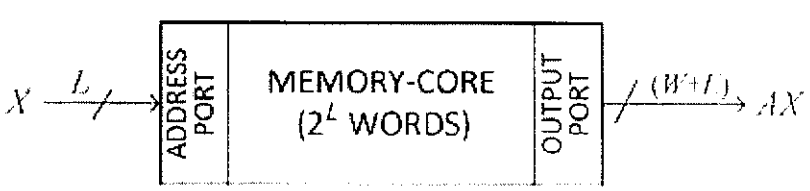


Figure 1.4: Conventional LUT-Based Multiplier.

The product word $A \cdot Xi$ is stored at the location Xi for $0 \leq Xi \leq 2^L - 1$, such that if an L-bit binary value of Xi is used as the address for the LUT, then the corresponding product value $A \cdot Xi$ is available as its output. Several architectures have been reported in the literature for memory-based implementation of DSP algorithms involving orthogonal transforms and digital filters. However, we do not find any significant work on LUT optimization for memory-based multiplication. the memory-based computing structures are more regular than the multiply-accumulate structures; and offer many other advantages, e.g., greater potential for high throughput and low-latency implementation; and less dynamic power consumption. Memory-based computing is well-suited for many digital signal processing (DSP) algorithms, which involve multiplication with fixed set of coefficients.

A new approach to LUT design, where only the odd multiples of the fixed coefficient are required to be stored, which we have referred to as the odd-multiple-storage (OMS) scheme in this brief. In addition, we have shown that, by the anti-symmetric product coding (APC) approach, the LUT size can also be reduced to half, where the product words are recoded as anti-symmetric pairs.

The APC approach, although providing a reduction in LUT size by a factor of two, incorporates substantial overhead of area and time to perform the two's complement operation of LUT output for sign modification and that of the input operand for input mapping. The APC approach is combined with the OMS technique, the two's complement operations could be very much simplified since the input address and LUT output could always be transformed into odd integers.

The OMS technique in cannot be combined with the APC scheme in, since the APC words generated according to are odd numbers. Moreover, the OMS scheme in does not provide an efficient implementation when combined with the APC technique. A different form of APC and combined that with a modified form of the OMS scheme for efficient memory based multiplication. The modified APC and the combined OMS–APC are combined to approach the implementation of combined OMS–APC scheme.

1.4 ORGANIZATION OF THE REPORT

- **Chapter 2** discusses the LUT optimization techniques.
- **Chapter 3** discusses about Hardware description language.
- **Chapter 4** discusses about design of LUT Based multiplier.
- **Chapter 5** presents the simulation output and results
- **Chapter 6** gives the conclusion.

CHAPTER 2

LUT OPTIMIZATION TECHNIQUES

2.1 ANTI-SYMMETRIC PRODUCT CODING

For simplicity of presentation, we assume both X and A to be positive integers. The product words for different values of X for $L = 5$ are shown in Table 2.1. It may be observed in this table that the input word X on the first column of each row is the two's complement of that on the third column of the same row. In addition, the sum of product values corresponding to these two input values on the same row is $32A$. Let the product values on the second and fourth columns of a row be u and v, respectively. Since one can write $u = [(u + v)/2 - (v - u)/2]$ and $v = [(u + v)/2 + (v - u)/2]$, for $(u + v) = 32A$, we can have

$$u = 16A - \left[\frac{v-u}{2}\right] \qquad v = 16A + \left[\frac{v-u}{2}\right]$$

The product values on the second and fourth columns of Table I therefore have a negative mirror symmetry. This behavior of the product words can be used to reduce the LUT size, where, instead of storing u and v, only [(v − u)/2] is stored for a pair of input on a given row. The 4-bit LUT addresses and corresponding coded words are listed on the fifth and sixth columns of the table, respectively. Since the representation of the product is derived from the antisymmetric behavior of the products, we can name it as antisymmetric product code. The 4-bit address $X' = (X'_3X'_2X'_1X'_0)$ of the APC word is given by

$$X' = \begin{cases} X_L, & \text{if } x_4 = 1 \\ X'_L, & \text{if } x_4 = 0 \end{cases}$$

where $XL = (x3x2x1x0)$ is the four less significant bits of X, and X_L is the two's complement of XL.

The desired product could be obtained by adding or subtracting the stored value $(v - u)$ to or from the fixed value $16A$ when $x4$ is 1 or 0, respectively, i.e.,

The desired product could be obtained by adding or subtracting the stored value ($v - u$) to or from the fixed value $16A$ when $x4$ is 1 or 0, respectively, i.e.,

$$\text{Product word} = 16A + (\text{sign value}) \times (\text{APC word})$$

where sign value = 1 for $x4 = 1$ and sign value = -1 for $x4 = 0$. The 4-bit LUT addresses and corresponding coded words are listed on the fifth and sixth columns of the table, respectively. Since the representation of the product is derived from the antisymmetric behavior of the products, we can name it as antisymmetric product code. The product value for $X = (10000)$ corresponds to APC value "zero," which could be derived by resetting the LUT output, instead of storing that in the LUT

Input, X	product values	Input, X	product values	address $x'_3x'_2x'_1x'_0$	APC words
0 0 0 0 1	A	1 1 1 1 1	$31A$	1 1 1 1	$15A$
0 0 0 1 0	$2A$	1 1 1 1 0	$30A$	1 1 1 0	$14A$
0 0 0 1 1	$3A$	1 1 1 0 1	$29A$	1 1 0 1	$13A$
0 0 1 0 0	$4A$	1 1 1 0 0	$28A$	1 1 0 0	$12A$
0 0 1 0 1	$5A$	1 1 0 1 1	$27A$	1 0 1 1	$11A$
0 0 1 1 0	$6A$	1 1 0 1 0	$26A$	1 0 1 0	$10A$
0 0 1 1 1	$7A$	1 1 0 0 1	$25A$	1 0 0 1	$9A$
0 1 0 0 0	$8A$	1 1 0 0 0	$24A$	1 0 0 0	$8A$
0 1 0 0 1	$9A$	1 0 1 1 1	$23A$	0 1 1 1	$7A$
0 1 0 1 0	$10A$	1 0 1 1 0	$22A$	0 1 1 0	$6A$
0 1 0 1 1	$11A$	1 0 1 0 1	$21A$	0 1 0 1	$5A$
0 1 1 0 0	$12A$	1 0 1 0 0	$20A$	0 1 0 0	$4A$
0 1 1 0 1	$13A$	1 0 0 1 1	$19A$	0 0 1 1	$3A$
0 1 1 1 0	$14A$	1 0 0 1 0	$18A$	0 0 1 0	$2A$
0 1 1 1 1	$15A$	1 0 0 0 1	$17A$	0 0 0 1	A
1 0 0 0 0	$16A$	1 0 0 0 0	$16A$	0 0 0 0	0

Table 2.1: APC Words of different input values for L=5

2.2 MODIFIED OMS FOR LUT OPTIMIZATION

The multiplication of any binary word X of size L, with a fixed coefficient A, instead of storing all the $2L$ possible values of $C = A \cdot X$, only $(2L/2)$ words corresponding to the odd multiples of A may be stored in the LUT, while all the even multiples of A could be derived by left-shift operations of one of those odd multiples. Based on the above assumptions, the LUT for the multiplication of an L-bit input with a W-bit coefficient could be designed by the following strategy.

1) A memory unit of $[(2L/2) + 1]$ words of $(W + L)$-bit width is used to store the product values, where the first $(2L/2)$ words are odd multiples of A, and the last word is zero.

2) A barrel shifter for producing a maximum of $(L - 1)$ left shifts is used to derive all the even multiples of A.

3) The L-bit input word is mapped to the $(L - 1)$-bit address of the LUT by an address encoder, and control bits for the barrel shifter are derived by a control circuit.

In Table 2.2, we have shown that, at eight memory locations, the eight odd multiples, $A \times (2i + 1)$ are stored as Pi, for i =0, 1, 2, . . . , 7. The even multiples $2A$, $4A$, and $8A$ are derived by left-shift operations of A. Similarly, $6A$ and $12A$ are derived by left shifting $3A$, while $10A$ and $14A$ are derived by left shifting $5A$ and $7A$, respectively.

A barrel shifter for producing a maximum of three left shifts could be used to derive all the even multiples of A. As required by [3], the word to be stored for X = (00000) is not 0 but $16A$, which we can obtain from A by four left shifts using a barrel shifter. However, if $16A$ is not derived from A, only a maximum of three left shifts is required to obtain all other even multiples of A. A maximum of three bit shifts can be implemented by a two-stage logarithmic barrel shifter, but the implementation of four shifts requires a three-stage barrel shifter. Therefore, it would be a more efficient strategy to store $2A$ for input X = (00000), so that the product $16A$ can be derived by three arithmetic left shifts.

The product values and encoded words for input words X = (00000) and (10000) are separately shown in Table III. For X = (00000), the desired encoded word $16A$ is derived by 3-bit left shifts of $2A$ [stored at address (1000)]. For X = (10000), the APC word "0" is derived by resetting the LUT output, by an active-high RESET signal given by

$$\mathbf{RESET} = (x0 + x1 + x2 + x3) \cdot x4.$$

input X' $x'_3x'_2x'_1x'_0$	product value	# of shifts	shifted input, X''	stored APC word	address $d_3d_2d_1d_0$
0 0 0 1	A	0	0 0 0 1	$P0 = A$	0 0 0 0
0 0 1 0	$2 \times A$	1			
0 1 0 0	$4 \times A$	2			
1 0 0 0	$8 \times A$	3			
0 0 1 1	$3A$	0	0 0 1 1	$P1 = 3A$	0 0 0 1
0 1 1 0	$2 \times 3A$	1			
1 1 0 0	$4 \times 3A$	2			
0 1 0 1	$5A$	0	0 1 0 1	$P2 = 5A$	0 0 1 0
1 0 1 0	$2 \times 5A$	1			
0 1 1 1	$7A$	0	0 1 1 1	$P3 = 7A$	0 0 1 1
1 1 1 0	$2 \times 7A$	1			
1 0 0 1	$9A$	0	1 0 0 1	$P4 = 9A$	0 1 0 0
1 0 1 1	$11A$	0	1 0 1 1	$P5 = 11A$	0 1 0 1
1 1 0 1	$13A$	0	1 1 0 1	$P6 = 13A$	0 1 1 0
1 1 1 1	$15A$	0	1 1 1 1	$P7 = 15A$	0 1 1 1

Table 2.2: OMS Based Design Of The LUT Of APC Word

input X $x_4x_3x_2x_1x_0$	product values	encoded word	stored values	# of shifts	address $d_3d_2d_1d_0$
1 0 0 0 0	$16A$	0	– – –	––	– – –
0 0 0 0 0	0	$16A$	$2A$	3	1 0 0 0

Table 2.3: Product and Encoded Word for X= (00000) AND (10000)

It may be seen from Tables 2.2 and 2.3 that the 5-bit input word *X* can be mapped into a 4-bit LUT address ($d_3\ d_2\ d_1\ d_0$), by a simple set of mapping relations

$$d_i = x''_{i+1}, \qquad \text{for } i = 0, 1, 2 \quad \text{and} \quad d_3 = \overline{x''_0}$$

Where *X"* = (X''$_3$X''$_2$X'$_1$X''$_0$) is generated by shifting-out all the leading zeros of *X_* by an arithmetic right shift followed by address mapping, i.e.,

$$X'' = \begin{cases} Y_L, & \text{if } x_4 = 1 \\ Y'_L, & \text{if } x_4 = 0 \end{cases}$$

Where *YL* and *Y _L* are derived by circularly shifting-out all the leading zeros of *XL* and *X_L*, respectively.

CHAPTER 3

HARDWARE DESCRIPTION LANGUAGE

3.1 INTRODUCTION

Hardware Description Language (HDL) is a language that can describe the behavior and structure of electronic system, but it is particularly suited as a language to describe the structure and the behavior of the digital electronic hardware design, such as ASICs and FPGAs as well as conventional circuits. HDL can be used to describe electronic hardware at many different levels of abstraction such as Algorithm, Register transfer level (RTL) and Gate level. Algorithm is un synthesizable, RTL is the input to the synthesis, and Gate Level is the input from the synthesis. It is often reported that a large number of ASIC designs meet their specification first time, but fail to work when plunged into a system. HDL allows this issue to be addressed in two ways, a HDL specification can be executed in order to achieve a high level of confidence in its correctness before commencing design and may simulate one specification for a part in the wider system context. This depends upon how accurately the specialization handles aspects such as timing and initialization.

3.2 ADVANTAGES OF HDL

A design methodology that uses HDLs has several fundamental advantages over traditional Gate Level Design Methodology. The following are some of the advantages:

- One can verify functionality early in the design process and immediately simulate the design written as a HDL description. Design simulation at this high level, before implementation at the Gate Level allows testing architectural and designing decisions.
- FPGA synthesis provides logic synthesis and optimization, so one can automatically convert a VHDL description to gate level implementation in a given technology.

- HDL descriptions provide technology independent documentation of a design and its functionality. A HDL description is more easily read and understood than a net-list or schematic description.
- HDLs typically support a mixed level description where structural or net-list constructs can be mixed with behavioral or algorithmic descriptions. With this mixed level capabilities one can describe system architectures at a high level or gate level implementation.

3.3 VHDL

VHDL is a hardware description language. It describes the behavior of an electronic circuit or system, from which the physical circuit or system can then be attained.

VHDL stands for VHSIC Hardware Description Language. VHSIC is itself an abbreviation for Very High Speed Integrated Circuits, an initiative funded by United States Department of Defense in the 1980s that led to creation of VHDL. Its first version was VHDL 87, later upgraded to the VHDL 93. VHDL was the original and first hardware description language to be standardized by Institute of Electrical and Electronics Engineers, through the IEEE 1076 standards. An additional standard, the IEEE 1164, was later added to introduce a multi-valued logic system.

VHDL is intended for circuit synthesis as well as circuit simulation. However, though VHDL is simulating fully, not all constructs are synthesizable. The two main immediate applications of VHDL are in the field of Programmable Logic Devices and in the field of ASICs (Application Specific Integrated Circuits). Once the VHDL code has been written, it can be used either to implement the circuit in a programmable device or can be submitted to a foundry for fabrication of an ASIC chip.

VHDL is a fairly general-purpose language, and it doesn't require a simulator on which to run the code. There are many VHDL compilers, which build executable binaries. It can read and write files on the host computer, so a VHDL program can be written that generates another VHDL program to be incorporated in the design being developed. Because of this general-purpose nature, it is possible to use VHDL to write a

test bench that verifies the functionality of the design using files on the host computer to define stimuli, interacts with the user, and compares results with those expected.

The key advantage of VHDL when used for systems design is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). The VHDL statements are inherently concurrent and the statements placed in a PROCESS, FUNCTION or PROCEDURES are executed sequentially. The different approaches in VHDL are structural, data flow, and behavioral methods of hardware description.

3.3.1 STRUCTURAL DESCRIPTIONS

The structural descriptions are explained below with examples. 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

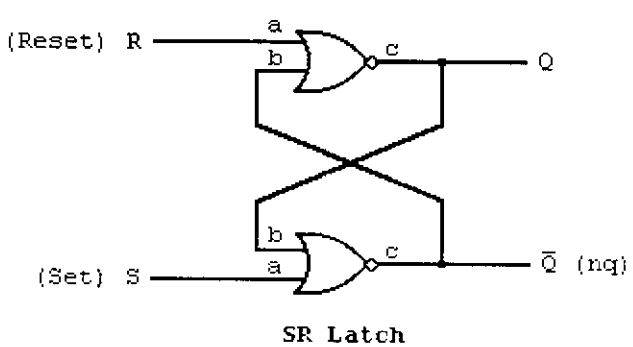


Fig.3.1: Schematic SR Latch

The VHDL statements are inherently concurrent and the statements placed in a PROCESS, FUNCTION or PROCEDURES are executed sequentially. The same connections that occur in the schematic 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 net list. The structural description of a design in VHDL is one of many means of specifying net lists.

3.3.2 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. To describe the following SR latch using VHDL as in the following schematic.

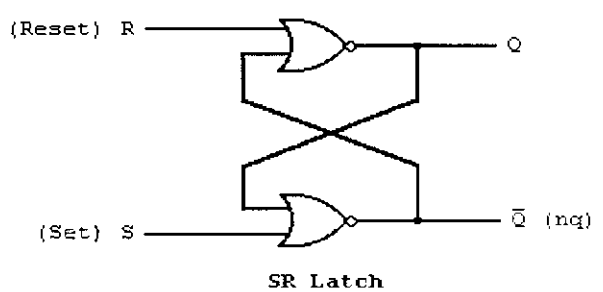


Fig.3.2: Data flow approach in SR Latch

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.

There are 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.

3.3.3 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.

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 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. 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

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. 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.

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. Every VHDL language system will contain a standard library. In VHDL, common code can be put in a separate file to be used by many designs. This common code is called a library. 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.

3.4 SOFTWARES USED

- ModelSim PE 6.1e
- Xilinx ISE 9.2i
- Micro Wind

3.5 ELECTRONIC DESIGN AUTOMATION TOOLS

There are several EDA (Electronic Design Automation) tool available for circuit synthesis, implementation and simulation using VHDL. Some tools are offered as part of a vendor's design suite such as Altera's Quatus II which allows the synthesis of VHDL code onto Altera's CPLD/FPGA chips, or Xilinx's ISE suite, for Xilinx's CPLD/FPGA chips. The tools used were either ISE combined with ModelSim.

CHAPTER 4

DESIGN OF LUT BASED MULTIPLIER

4.1 IMPLEMENTATION OF THE LUT MULTIPLIER USING APC

The APC approach, although providing a reduction in LUT size by a factor of two, incorporates substantial overhead of area and time to perform the two's complement operation of LUT output for sign modification and that of the input operand for input mapping. The APC approach is combined with the OMS technique, the two's complement operations could be very much simplified since the input address and LUT output could always be transformed into odd integers.

The structure and function of the LUT-based multiplier for $L = 5$ using the APC technique is shown in Fig. 4.1. It consists of a four-input LUT of 16 words to store the APC values of product words as given in the sixth column of Table 2.1, except on the last row, where $2A$ is stored for input $X = (00000)$ instead of storing a "0" for input $X = (10000)$.

Besides, it consists of an address-mapping circuit and an add/subtract circuit. The address-mapping circuit generates the desired address $(X'_3X'_2X'_1X'_0)$. A straightforward implementation of address mapping can be done by multiplexing XL and $X\,L`$ using $x4$ as the control bit. The address-mapping circuit, however, can be optimized to be realized by three XOR gates, three AND gates, two OR gates, and a NOT gate, as shown in Fig. 2. Note that the RESET can be generated by a control circuit.

The output of the LUT is added with or subtracted from $16A$, for $x4 = 1$ or 0, respectively, according by the add/subtract cell. Hence, $x4$ is used as the control for the add/subtract cell.

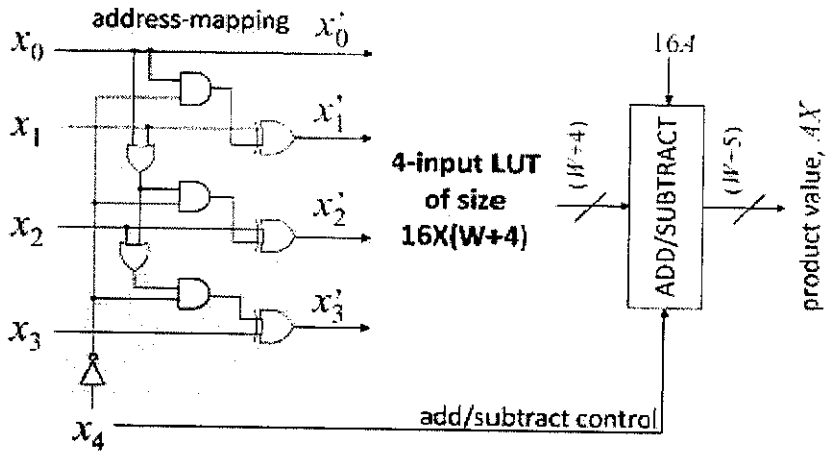


Figure 4.1: LUT based multiplier for L=5

4.2 IMPLEMENTATION OF THE OPTIMIZED LUT USING MODIFIED OMS

The proposed APC–OMS combined design of the LUT for $L = 5$ and for any coefficient width W is shown in Fig. 4.2. It consists of an LUT of nine words of $(W + 4)$-bit width, a four-to-nine-line address decoder, a barrel shifter, an address generation circuit, and a control circuit for generating the RESET signal and control word $(s1s0)$ for the barrel shifter.

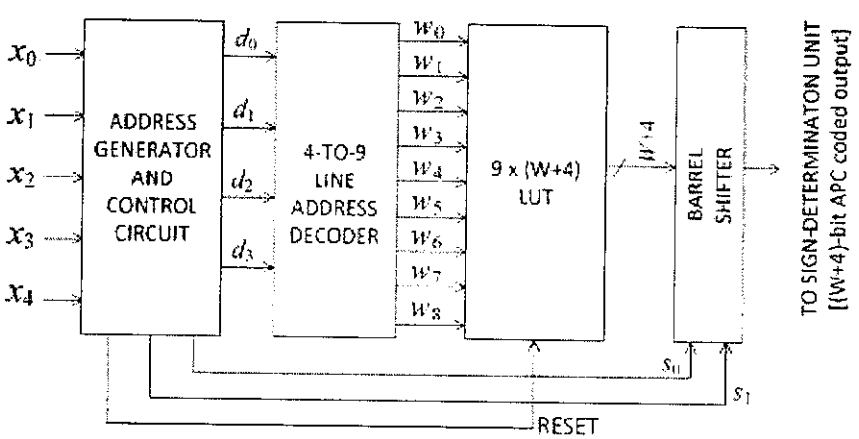


Figure 4.2: Four To Nine Line Address Decoders

The pre-computed values of $A \times (2i + 1)$ are stored as Pi, for $i = 0, 1, 2, \ldots, 7$, at the eight consecutive locations of the memory array, as specified in Table 2.2, while $2A$ is stored for input $X = (00000)$ at LUT address "1000," as specified in Table 2.3. The decoder takes the 4-bit address from the address generator and generates nine word-select signals, i.e., $\{wi$, for $0 \leq i \leq 8\}$, to select the referenced word from the LUT. The 4-to-9-ine decoder is a simple modification of 3-to- 8-line decoder, as shown in Fig.4.2.

The control bits $s0$ and $s1$ to be used by the barrel shifter to produce the desired number of shifts of the LUT output are generated by the control circuit, according to the relations

$$s_0 = \overline{x_0 + \overline{(x_1 + \overline{x_2})}}$$
$$s_1 = \overline{(x_0 + x_1)}.$$

Note that $(s1s0)$ is a 2-bit binary equivalent of the required number of shifts specified in Tables II and III. The RESET signal given by [4] can alternatively be generated as (d_3 AND x_4). The address-generator circuit receives the 5-bit input operand X and maps that onto the 4-bit address word $(d_3d_2d_1d_0)$, according to [5] and [6]. A simplified address generator is presented later in this section.

4.3 OPTIMIZED LUT DESIGN FOR SIGNED AND UNSIGNED OPERANDS

The APC–OMS combined optimization of the LUT can also be performed for signed values of A and X[7]. When both operands are in sign-magnitude form, the multiples of magnitude of the fixed coefficient are to be stored in the LUT, and the sign of the product could be obtained by the XOR operation of sign bits of both multiplicands. When both operands are in two's complement forms, a two's complement operation of the output of the LUT is required to be performed for $x4 = 1$.

There is no need to add the fixed value $16A$ in this case, because the product values are naturally in antisymmetric form[8]. The add/subtract circuit is not required in

Fig. 4.2, instead of that a circuit is required to perform the two's complement operation of the LUT output.

For the multiplication of unsigned input X with signed, as well as unsigned, coefficient A, the products could be stored in two's complement representation, and the add/subtract circuit in Fig. 4.3 could be modified as shown in Fig. 4.3. A straightforward implementation of sign- modification circuit involves multiplexing of the LUT output and its two's complement.

To reduce the area–time complexity over such straightforward implementation, we discuss here a simple design for sign modification of the LUT output. Note that, except the last word, all other words in the LUT are odd multiples of A. The fixed coefficient could be even or Fig.4.3 Optimized implementation of the sign modification of the odd LUT output. (b) Address-generation circuit. odd, but if we assume A to be an odd number, then the all the stored product words (except the last one) would be odd. If the stored value P is an odd number, it can be expressed as $P = PD-1\ PD-2 \cdot \cdot \cdot P1$ and its two's complement is given by

$$P = P_{D-1}\ P_{D-2} \cdot \cdot \cdot P_1$$

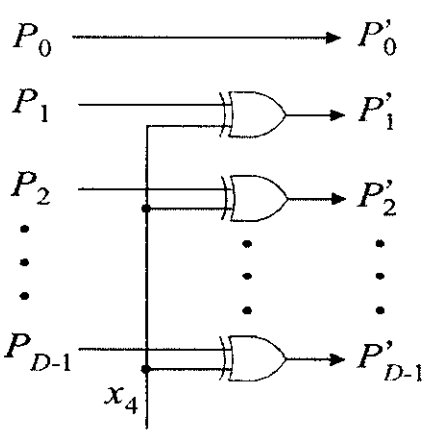


Figure 4.3: Optimized Implementation of address generation

and its two's complement is given by

$$P' = P'_{D-1}\, P'_{D-2} \cdots P'_1$$

where $P'i$ is the one's complement of Pi for $1 \le i \le D - 1$, and $D = W + L - 1$ is the width of the stored words. If we store the two's complement of all the product values and change the sign of the LUT output for $x4 = 1$, then the sign of the last LUT word need not be changed [9]. We can therefore have a simple sign-modification circuit when A is an odd integer.

The fixed coefficient A could be even as well. When A is a nonzero even integer, we can express it as $A\_ \times 2l$, where $1 \le l \le D - 1$ is an integer, and $A\_$ is an odd integer. Instead of storing multiples of A, we can store multiples of $A\_$ in the LUT, and the LUT output can be left shifted by l bits by a hardwired shifter. Similarly, we can have an address-generation circuit as shown in Fig. 4.4, since all the shifted-address YL is an odd integer.

CHAPTER 5

RESULTS

The simulation of this whole project has been done using the Model Sim of version 6.2. Modelsim is a simulation tool for programming {VLSI} {ASIC}s, {FPGA}s, {CPLD}s, and {SoC}s. Modelsim provides a comprehensive simulation and debug environment for complex ASIC and FPGA designs. Support is provided for multiple languages including Verilog, SystemVerilog, VHDL and SystemC.

5.1 SIMULATION RESULT OBTAINED FOR CONVENTIONAL METHOD

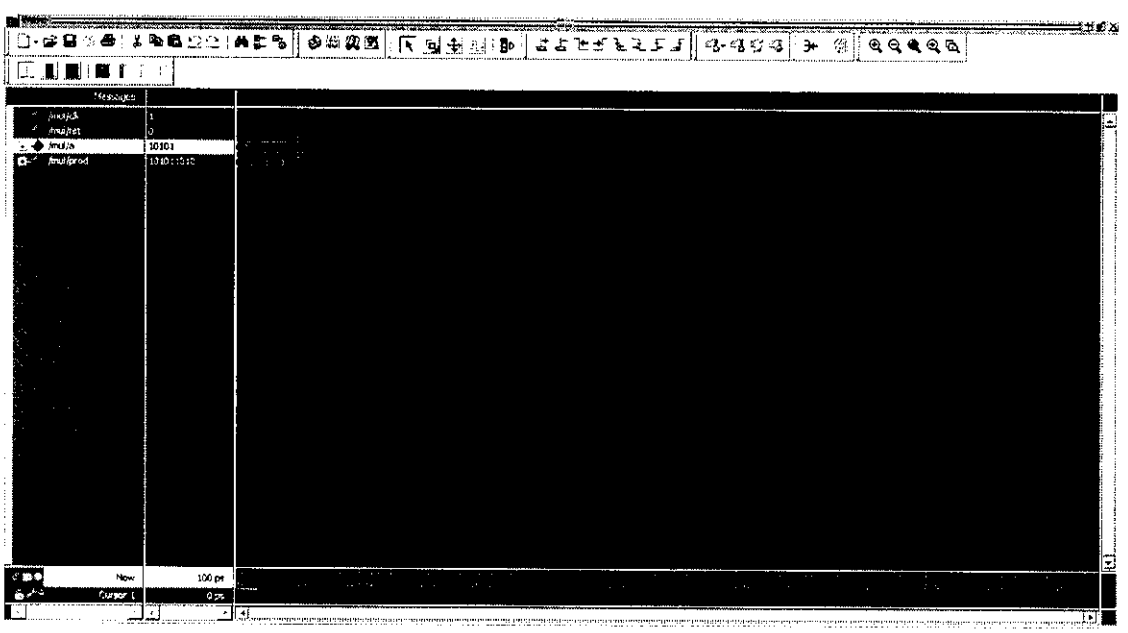


Fig 5.1: Simulation Result Obtained for Conventional Method

5.2 SIMULATION RESULTS OBTAINED FOR APC

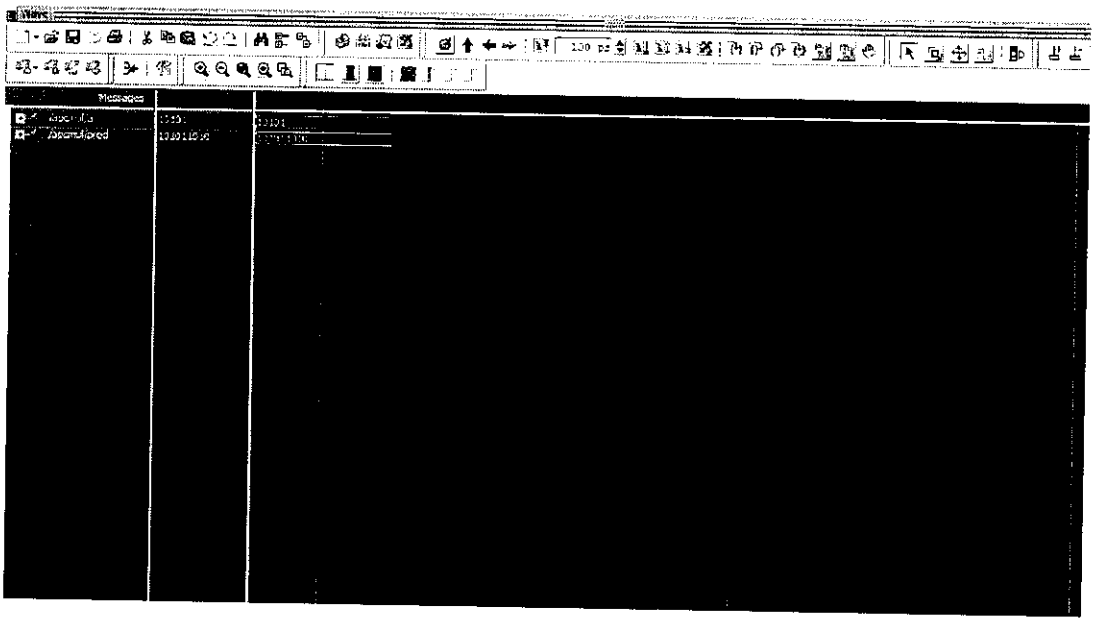


Fig 5.2: Simulation Results Obtained for APC

5.3 SIMULATION RESULTS OBTAINED FOR ADDRESS GENERATION UNIT

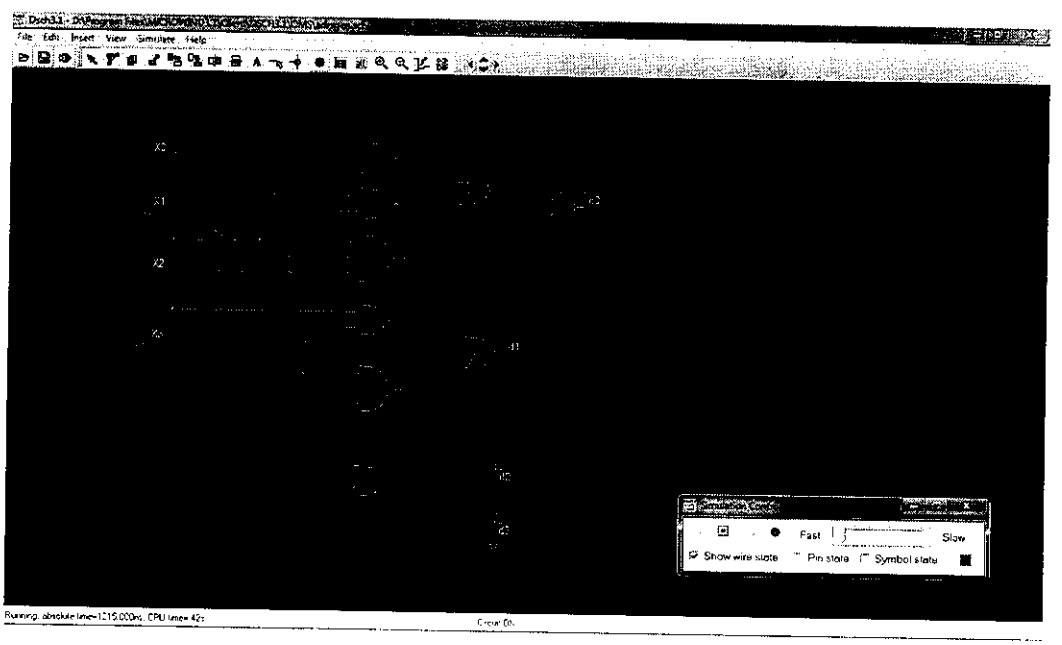


Fig 5.3: Simulation Results Obtained for Address Generation Unit

5.4 SIMULATION RESULTS OBTAINED FOR BARREL SHIFTER

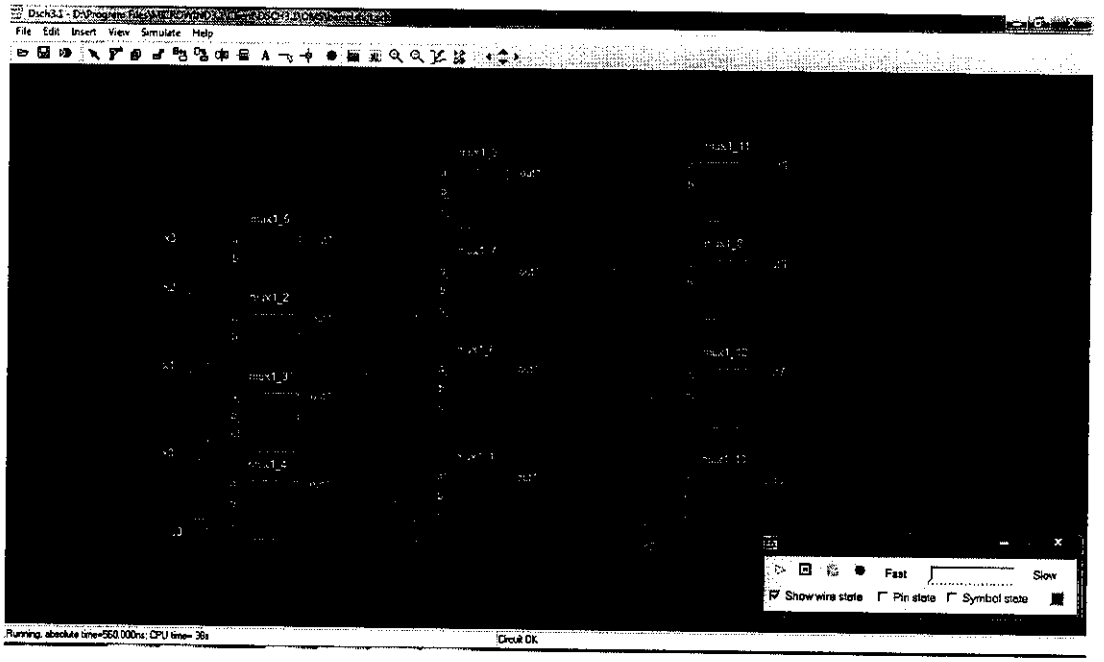


Fig 5.4: Simulation Results Obtained for Barrel Shifter

5.5 SIMULATION RESULTS OBTAINED FOR ADDRESS DECODER

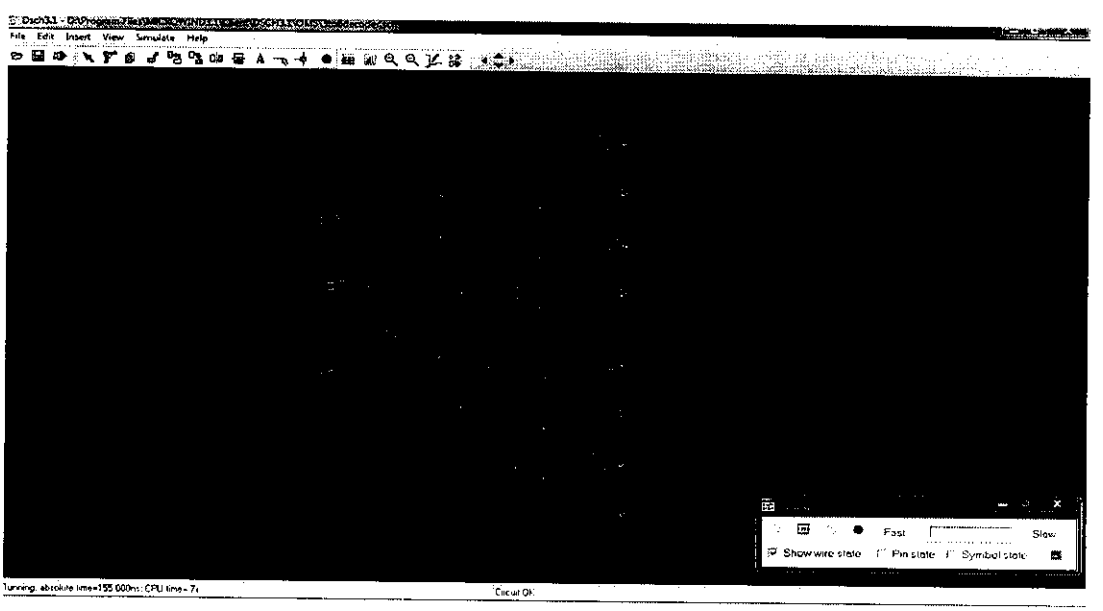


Fig 5.5: Simulation Results Obtained for Address Decoder

5.6 SIMULATION RESULTS OBTAINED FOR OMS BASED LUT

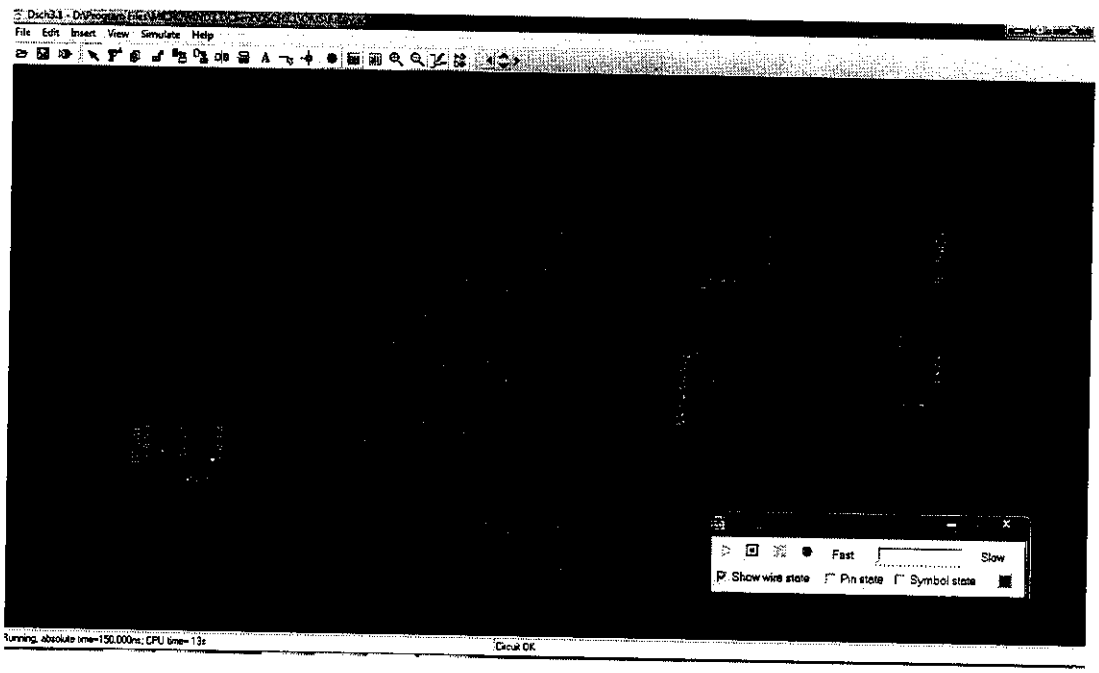


Fig 5.6: Simulation Results Obtained for OMS Based LUT

5.7 SYNTHESIS REPORT FOR CONVENTIONAL LUT

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slices:	144	3584	4%	
Number of Slice Flip Flops:	56	7168	1%	
Number of 4 input LUTs:	2	7168	0%	
Number of bonded IOBs:	33	221	17%	
Number of GCLKs:	1	8	12%	

Table 5.1: Synthesis Report for Conventional LUT

5.8 SYNTHESIS REPORT FOR APC

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slices:	74	3584	2%	
Number of Slice Flip Flops:	24	7168	1%	
Number of 4 input LUTs:	2	7168	0%	
Number of bonded IOBs:	17	221	8%	
Number of GCLKs:	1	8	12%	

Table 5.2: Synthesis Report for APC

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

The proposed LUT multipliers for word size $L = W = 5$ bits are coded in VHDL, where the LUTs are implemented as arrays of constants, and additions are implemented. The area and delay complexities of the multipliers estimated from the synthesis results. It is found that the proposed LUT design involves comparable area and time complexities for a word size of 5 bits, but for higher word sizes, it involves significantly less area and less multiplication time than the CSD-based multiplier. For $L = W = 5$ bits, respectively, it offers more than 30% and 50% of saving in area–delay product (ADP) over the CSD multiplier.

The LUT based multipliers to implement the constant multiplication for DSP applications. The full advantages of proposed LUT based design, however, could be derived if the LUTs are implemented as NAND or NOR read-only memories and the arithmetic shifts are implemented by an array barrel shifter using metal–oxide–semiconductor transistors.

Further work could still be done to derive OMS–APC-based LUTs for higher input sizes with different forms of decompositions and parallel and pipelined addition schemes for suitable area–delay tradeoffs.

REFERENCES

[1]. G.-K. Ma and F. J. Taylor, "Multiplier policies for digital signal processing," *IEEE ASSP Mag.*, pp. 6-20, Jan. 1990.

[2]. J.-I. Guo, C.-M. Liu, and C.-W. Jen, "The efficient memory-based VLSI array design for DFT and DCT," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 39, no. 10, pp. 723–733, Oct. 1992.

[3]. H.-R. Lee, C.-W. Jen, and C.-M. Liu, "On the design automation of the memory-based VLSI architectures for FIR filters," *IEEE Trans. Consum. Electron.*, vol. 39, no. 3, pp. 619–629, Aug. 1993.

[4]. Peter J. Ashenden, "VHDL Standards," IEEE Design and Test of Computers, vol.18, no. 5, pp. 122-123, Sep./Oct. 2001.

[5]. H.-C. Chen, J.-I. Guo, T.-S. Chang, and C.-W. Jen, "A memory-efficient realization of cyclic convolution and its application to DCT ," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 3, pp. 445–453, Mar. 2005.

[6]. D. F. Chiper, M. N. S. Swamy, M. O. Ahmad, and T. Stouraitis, "Systolic algorithms and a memory-based design approach for a unified architecture for the computation of DCT/DST/IDCT/IDST," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 6, pp. 1125–1137, Jun. 2005.

[7]. P. K. Meher, "Memory-based hardware for resource-constrained digital signal processing systems," in *Proc. 6th Int. Conf. ICICS*, Dec. 2007, pp. 1–4.

[8]. P. K. Meher, "New approach to LUT implementation and accumulation for memory-based multiplication," in *Proc. IEEE ISCAS*, May 2009, pp. 453–456.

[9]. P. K. Meher, "New look-up-table optimizations for memory-based multiplication," in *Proc. ISIC*, Dec. 2009, pp. 663–666.

[10]. Application note XAPP 290, Two Flows for Partial Reconfiguration: Module Based or Difference Based, available at www.xilinx.com.

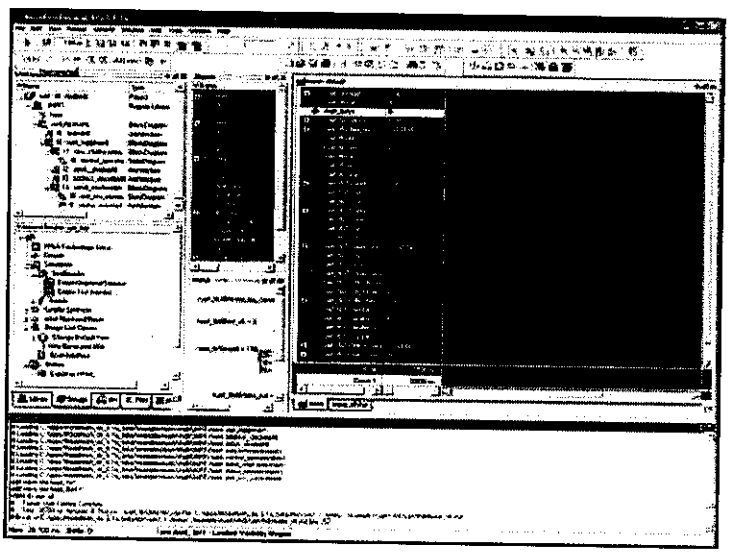
APPENDICES

APPENDIX - I

Design Creation to Realization

D A T A S H E E T

ModelSim Designer



ModelSim Designer combines easy to use, flexible creation with a powerful verification and debug environment.

The Easy to Use Solution for the FPGA Designer

ModelSim® Designer is a Windows®-based design environment for FPGAs. It provides an easy to use, advanced-feature tool at an entry-level price. The complete process of creation, management, simulation, and implementation are controlled from a single user interface, facilitating the design and verification flow and providing significant productivity gains.

ModelSim Designer combines the industry-leading capabilities of the ModelSim simulator with a built-in design creation engine. To support synthesis and place-and-route, ModelSim Designer is plug-in ready for the synthesis and place-and-route tools of your choice. Connection of these additional tools is easy, giving FPGA designers control of design creation, simulation, synthesis, and place-and route from a single cockpit.

A flexible methodology conforms to existing design flows. Designers can freely mix text entry of VHDL and Verilog code with graphical entry using block and state diagram editors. A single design unit can be represented in multiple views, and the management of compilation and simulation at all levels of abstraction is a single click away.

Major product features:

- Project manager, version control, and source code templates and wizards
- Block and state diagram editors
- Intuitive graphical waveform editor
- Automated testbench creation
- Text to graphic rendering facilitates analysis
- HTML Documentation option
- VHDL, Verilog, and mixed-language simulation
- Optimized native compiled architecture
- Single Kernel Simulator technology
- Advanced debug including Signal Spy™
- Memory window
- Easy-to-use GUI with Tcl interface
- Support for all major synthesis tools and the Actel, Altera, Lattice, and Xilinx place-and-route flows
- Built-in FPGA vendor library compiler
- Full support for Verilog 2001
- Windows platform support

Options: SWIFT Interface support graphics-based Dataflow window Waveform Compare, integrated code coverage, and Profiler (for more details on options, please see the ModelSim SE datasheet)

www.model.com/modelsimdesigner

Mentor Graphics

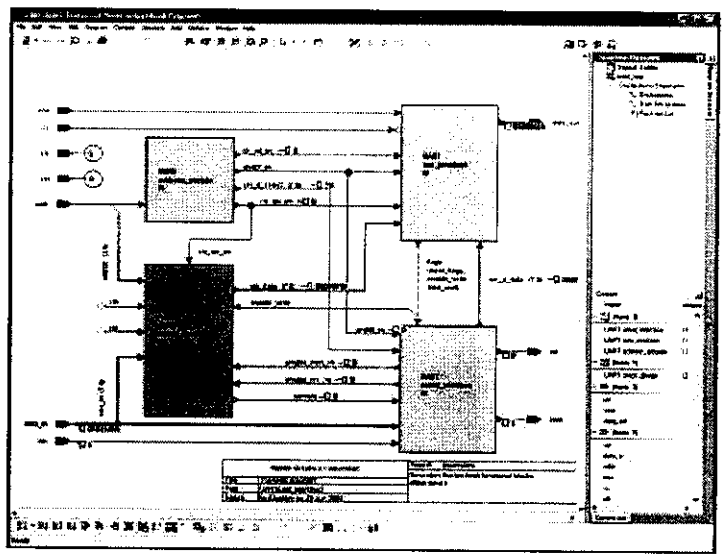
Implementation tasks are achieved through tight integration with the most popular FPGA synthesis and place-and-route tools in the industry. ModelSim Designer can either run these tools directly or externally, via scripts. Either way, the design data is maintained and used in a common and consistent way.

Project Manager

The flexible and powerful project manager feature allows easy navigation through a design in order to understand design content and execute all necessary tasks. The project manager provides easy access to all design data for each individual design unit, throughout the design process. Synthesis results, place-and-route results, back annotated netlists, and SDF files are associated with the relevant design unit. Project archiving is a few simple clicks away, allowing complete version control management of designs. The project manager also stores user-generated design documents, such as Word, PowerPoint, and PDFs.

Templates and Wizards

Creation wizards walk users through the creation of VHDL and Verilog design units, using either text or graphics. In the case of the graphical editor, HDL code is generated from the graphical diagrams created. For text-based design, VHDL and Verilog templates and wizards help engineers quickly develop HDL code without having to remember the exact language syntax. The wizards show how to create parameterizable logic blocks, testbench stimuli, and design objects. Novice and advanced HDL developers both benefit from time-saving shortcuts.



The Block Diagram editor is a convenient way to visually partition your design and have the HDL code automatically generated.

Intuitive Graphical Editors

ModelSim Designer includes block diagram and state machine editors that ensure a consistent coding style, facilitating design reuse and maintenance. These editors use an intuitive, graphical methodology that flattens the learning curve. This shortens the time to productivity for designers who are migrating to HDL methodologies or changing their primary design language.

Designers can automatically view or render diagrams from HDL code in block diagrams or state machines. When the code changes, the diagram can be updated instantly with its accuracy ensured. This helps designers understand legacy designs and aids in the debugging of current designs.

Automated Testbench Creation

ModelSim Designer offers an automated mechanism for testbench generation. The testbench wizard generates VHDL or Verilog code through a graphical waveform editor, with output in either HDL or a TCL script. Users can manually define signals in the waveform editor or use the built-in wizard to define the waveforms. Either way, it is intuitive and easy to use and saves considerable time.

Active Design Visualization Enhances Simulation Debugging

During live simulation, design analysis capabilities are enhanced through graphical design views. From any diagram window, simulations can be fully executed and controlled. Enhanced debugging features include graphical breakpoints, signal probing, graphics-to-text-source cross-highlighting, animation, and cause analysis. The ability to overlay live simulation results in a graphical context speeds up the debug process by allowing faster problem discovery and shorter design iterations.

HTML Documentation

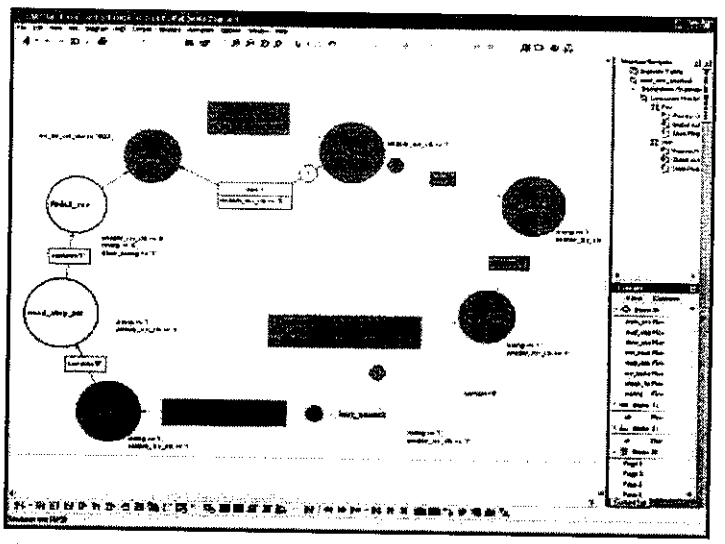
The HTML (HyperText Markup Language) Documentation option allows the user to output the complete design in HTML format. This allows the graphics, HDL, associative design files, and design hierarchy to be shared with anybody that uses an HTML browser. Design hierarchy and details can be viewed and navigated in any HTML browser, aiding communication and documentation.

Memory Window

The memory window enables flexible viewing and switching of memory locations. VHDL and Verilog memories are auto-extracted in the GUI, delivering powerful search, fill, load, and save functionality. The memory window allows pre-loading of memories, thus saving the time-consuming step of initializing sections of the simulation to load memories. All functions are available via the command line, making them available for scripting.

Intelligent GUI

An intelligently engineered GUI makes efficient use of desktop real estate. ModelSim Designer's intuitive layout of graphical elements (windows, toolbars, menus, etc.) makes it easy to step through the design flow. There are also wizards in place which help set up the design environment and make going through the design process seamless and efficient.



The State Diagram Editor uses interactive animation to track the simulation through the state machine and to check that all states were exercised.

Synthesis and Place-and-Route Integration

The industry's popular FPGA synthesis tools, such as Mentor Graphics Precision-RTL and most FPGA vendor synthesis tools, can be integrated into ModelSim Designer with push button convenience. Actel Designer and Libero, Altera Quartus, Lattice ispLEVEL, and Xilinx ISE place-and-route software are similarly integrated. Place-and-route results, together with SDF information, are automatically managed by ModelSim Designer after the process is complete, making them ready for post-place-and-route, gate-level simulation.

FPGA Vendor Library Compiler

ModelSim Designer provides an intuitive mechanism to compile the necessary vendor libraries for post-place-and-route simulation. The compiler detects which FPGA vendor tools have been installed and compiles the necessary libraries as soon as the tool is launched, taking care of this step once and for all.

Visit our web site at www.model.com/modelsimdesigner for more information.

APPENDIX - II

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

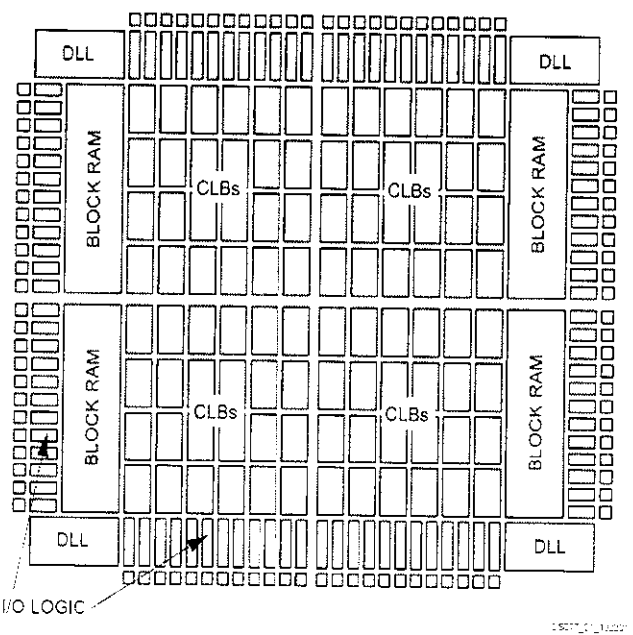


Figure 1: Basic Spartan-IIE Family FPGA Block Diagram

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**

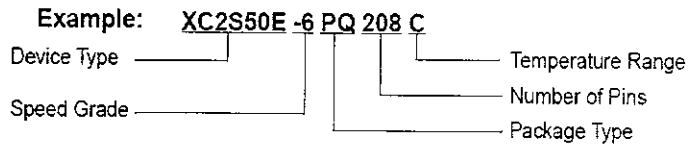
Device	Pins	144	208	256	456
	Type	Plastic TQFP	Plastic PQFP	Fine Pitch BGA	Fine Pitch BGA
	Code	TQ144	PQ208	FT256	FG456
XC2S50E	-6	C, I	C, I	C, I	-
	-7	(C)	(C)	(C)	-
XC2S100E	-6	C, I	C, I	C, I	C, I
	-7	(C)	(C)	(C)	(C)
XC2S150E	-6	-	(C, I)	(C, I)	(C, I)
	-7	-	(C)	(C)	(C)
XC2S200E	-6	-	C, I	C, I	C, I
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