DESIGNING OF 8 BIT ALU AND IMPLEMENTING ON XILINX VERTEX 4 FPGA

SUBMITTED BY
PREETI TAKHAR
PRIYANKA RAJPAL
RAHUL BORTHAKUR
SAKSHI AGARWAL

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS
AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH
NOIDA (U.P.)
APRIL 2011

DESIGNING OF 8 BIT ALU AND IMPLEMENTING ON XILINX VERTEX 4 FPGA

Submitted to
Amity University Uttar Pradesh

in partial fulfillment of the requirements for the award of the Degree of
Bachelor of Technology in
Department Of Electronics and Communications

by
Preeti Takhar
Priyanka Rajpal
Rahul Borthakur
Sakshi Agarwal

Under the guidance of
Dr. Anu Mehra
Department of Electronics and Communications,
Amity School of Engineering and Technology
DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS,
AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH
NOIDA (U.P.)

DECLARATION

We , Preeti takhar, Priyanka Rajpal, Rahul Borthakur and Sakshi Agarwal , students of B.Tech (Electronics and Communication) hereby declare that the project titled “Designing of 8 bit ALU and implementing it on Xilinx Vertex4 FPGA” which is submitted by me to Department of Electronics and Communication, Amity School of Engineering and Technology, Amity University, Uttar Pradesh, Noida, in partial fulfillment of requirement for the award of the degree of Bachelor of Technology in Electronics and Communication , has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition.

Noida

Date

Submitted By:

Preeti Takhar

Priyanka Rajpal

Rahul Borthakur

Sakshi Agarwal.

CERTIFICATE

On the basis of declaration submitted by Preeti Takhar, Priyanka Rajpal, Rahul Borthakur and Sakshi Agarwal, students of B. Tech Electronics and Communications, I hereby certify that the project titled "Designing of 8 bit ALU and implementing it on Xilinx Vertex4 FPGA" which is submitted to Department of Electronics and Communications , Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Electronics and Communications, is an original contribution with existing knowledge and faithful record of work carried out by her under my guidance and supervision.

To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Noida

Date

Dr. Anu Mehra.

Department of Electronics and Communications,

Amity School of Engineering and Technology

Amity University Uttar Pradesh, Noida.

ACKNOWLEDGEMENT

We wish to express my sincere gratitude to **Dr. Balvinder Shukla,** D.G ASET & Pro Vice Chancellor, Amity University, Uttar Pradesh, **Mr.K.M.Soni**, HOD ECE Department and **Mr.Lala Bhaskar**, Program Leader, ECE(2007-11) Amity School of Engineering & Technology for giving us the opportunity to do our final year project **"DESIGNING OF ALU AND ITS IMPLEMENTATION ON FPGA".**

We sincerely thank our faculty guide **Dr.Anu Mehra**, Professor, Amity School of Engineering & Technology for guiding us in every way she could. We would also like to express our gratitude to Mr. Ashutosh Gupta and Mrs Nidhi Gaur for their magnificent help and support throughout the project.

Last but not the least, we wish to avail this opportunity to express our gratitude and love to our friends and our beloved parents for their moral support, strength, help and for everything.

PLACE:

DATE:

Preeti Takhar (A2305107163)

Priyanka Rajpal(A2305107164)

Rahul Borthakur(A2305107167)

Sakshi Agarwal (A2305107184)

ABSTRACT

The main objective of project is to design and verify different operations of Arithmetic and Logical Unit (ALU).

We have designed an 8 bit ALU which accepts two 8 bits numbers and the code corresponding to the operation which it has to perform from the user. The ALU performs the desired operation and generates the result accordingly. The different operations that we dealt with, are arithmetical, logical and relational. Arithmetic operations include arithmetic addition, subtraction, multiplication and division. Logical operations include AND, OR, NAND, XOR, NOT and NOR. These take two binary inputs and result in output logically operated. The operations like the greater than, less than, equal to, exponential etc are also included.

To implement ALU, the coding was written in VHDL and verified in ModelSim. The waveforms were obtained successfully. After the coding was done, the synthesis of the code was performed using Xilinx-ISE. Synthesis translates VHDL code into netlist (a textual description). Thereafter, the simulation was done to verify the synthesized code. And it was, then converted into binary format. Components and connections are mapped to CLB design and is placed and routed to fit onto FPGA. User constraint file is generated and also bit file to load design on FPGA when the later was connected to the laptop. Then device was configured and using FPGA verification, debugging was done. Thus we successfully verified our code using FPGA.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I: INTRODUCTION

1.1 GENERAL:

In this project, our scope was to design an 8 bit ALU and its implement it on FPGA.

An arithmetic logic unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs.

Most of a processor's operations are performed by one or more ALUs. An ALU loads data from input registers, an external Control Unit then tells the ALU what operation to perform on that data, and then the ALU stores its result into an output register. The inputs to the ALU are the data to be operated on (called operands) and a code from the control unit indicating which operation to perform. Its output is the result of the computation.

The coding of the ALU has been done in VHDL.VHDL (Very High Speed Integrated Circuits Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. It is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design.

After the coding had been completed in VHDL, the synthesis part was done using Xilinx-ISE. Xilinx ISE is a software tool for synthesis and analysis of HDL designs, which enables the developers to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

1.2 ABOUT ALU

Microprocessors/Microcontrollers have a single module that performs arithmetic operations on integer values. This is because many of the different arithmetic and logical operations can be performed using similar (if not identical) hardware. The component that performs the arithmetic and logical operations is known as the **Arithmetic Logic Unit**, or ALU.

The ALU is one of the most important components in a microprocessor, and is typically the part of the processor that is designed first. Once the ALU is designed, the rest of the microprocessor is implemented to feed operands and control codes to the ALU.

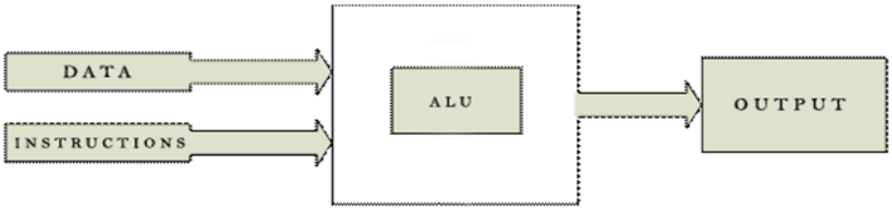


Figure1.1 : ALU block diagram

The arithmetic and logic unit (ALU) performs all arithmetic operations (addition, subtraction, multiplication, and division) and logic operations. Logic operations test various conditions encountered during processing and allow for different actions to be taken based on the results. The data required to perform the arithmetic and logical functions are inputs from the designated CPU registers and operands. The ALU relies on basic items to perform its operations. These include number systems, data routing circuits (adders/subtractors), timing, instructions, operands, and registers. **Figure1.1** shows a representative block diagram of an ALU. An ALU loads data from input registers, an external Control Unit then tells the ALU what operation to perform on that data, and then the ALU stores its result into an output register. The Control Unit is responsible for moving the processed data between these registers, ALU and memory.

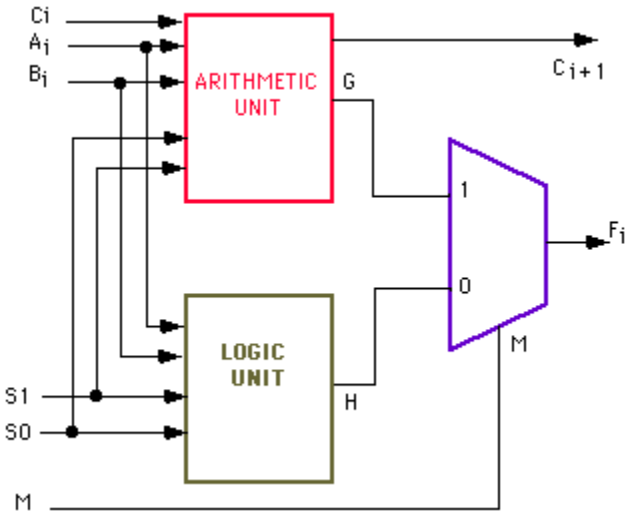


Figure1.2: ALU Architecture

An ALU must process numbers using the same format as the rest of the digital circuit. The format of modern processors is almost always the two's complement binary number representation. Early computers used a wide variety of number systems, including ones' complement, two's complement sign-magnitude format, and even true decimal systems, with ten tubes per digit.

ALUs for each one of these numeric systems had different designs, and that influenced the current preference for two's complement, as this is the representation that makes it easier for the ALUs to calculate additions and subtraction.

1.2.1 FLOWCHART

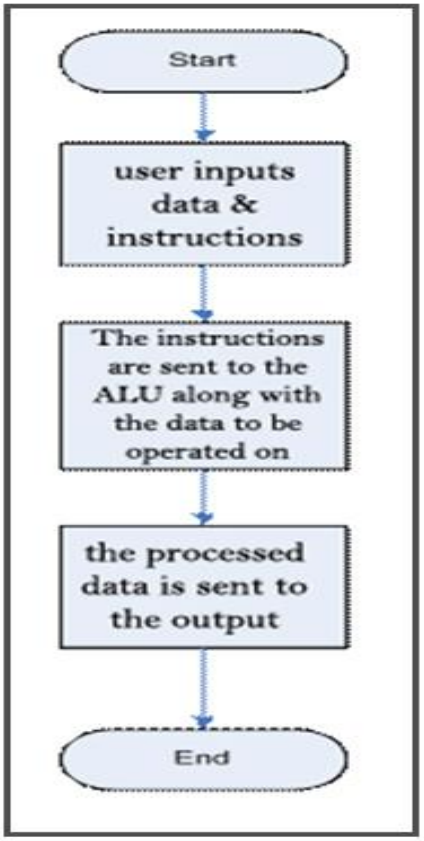


Figure1.3 – ALU Flowchart

Most ALUs can perform the following operations:

- Bitwise logic operations (AND, NOT, OR, XOR, NAND, NOR)
- Integer arithmetic operations (addition, subtraction, and sometimes multiplication and division, though this is more expensive).
- Other operations like greater than, equal to, exponential, modulus etc.

1.3 THE VARIOUS OPERATIONS

ARITHMETIC OPERATIONS	LOGICAL OPERATIONS
ADDITION	AND
SUBTRACTION	OR
MUTLIPLICATION	NOT
DIVISION	NAND
REMAINDER	NOR
MODULUS	XOR
UNARY ADDITION	
UNARY SUBTRACTION	
EXPONENTIAL	

Table-1: Arithmetic & Logical Operators

1.3.1 ARITHMETICAL OPERATIONS

ADDITION

Addition is the basic operation of arithmetic. In its simplest form, addition combines two numbers, the addends or terms, into a single number, the sum of the numbers. Adding more than two numbers can be viewed as repeated addition; this procedure is known as summation and includes ways to add infinitely many numbers in an infinite series; repeated addition of the number one is the most basic form of counting

0101 (decimal 5) + 0011 (decimal 3) = 0100 (decimal 8)

SUBTRACTION

Subtraction is one of the four basic arithmetic operations; it is the inverse of addition, meaning that if we start with any number and add any number and then subtract the same number we added, we return to the number we started with. Subtraction is denoted by a minus sign in infix notation.

MULTIPLICATION

Multiplication is the second basic operation of arithmetic. Multiplication also combines two numbers into a single number, the product. The two original numbers are called the multiplier and the multiplicand, sometimes both simply called factors.

DIVISION

Division is essentially the opposite of multiplication. Division finds the quotient of two numbers, the dividend divided by the divisor. Any dividend divided by zero is undefined. For positive numbers, if the dividend is larger than the divisor, the quotient is greater than one, otherwise it is less than one (a similar rule applies for negative numbers). The quotient multiplied by the divisor always yields the dividend.

REMAINDER

The binary % operator is said to yield the remainder of its operands from an implied division; the left-hand operand is the dividend and the right-hand operand is the divisor.

5%3 produces 2 (note that 5/3 produces 1)
5 %(-3) produces 2 (note that 5/(-3) produces -1)

(-5)%3 produces -2 (note that (-5)/3 produces -1)

(-5)%(-3) produces -2 (note that (-5)/(-3) produces 1)

MODULUS

In computing, the modulo operation, sometimes also called "remainder" or "rest", gives the remainder from a division. It finds the remainder of division of one number by another. Given two numbers, a (the dividend) and n (the divisor), a modulo n abbreviated as a mod n) is the remainder, on division of a by n. For example, if you divide 7 by 3, 3 goes in 7 two times. But there is a remainder of 1, and that is the result of the modulo operation.

UNARY ADDITION/ SUBTRACTION

Unary addition and subtraction operators are unary operators that add or subtract one from their operand, respectively. They are commonly implemented in imperative programming languages. The increment operator increases the value of its operand by 1. The operand must have an arithmetic data type, and must refer to a modifiable data object. Similarly, the decrement operator decreases the value of its modifiable arithmetic operand by 1.

EXPONENTIAL

In mathematics, the exponential function is the function ex, where e is the number (approximately 2.718281828) such that the function ex is its own derivative.[1][2] The exponential function is used to model phenomena when a constant change in the independent variable gives the same proportional change (i.e., percent increase or decrease) in the dependent variable. The function is often written as exp(x), especially when it would be impractical to write the input expression as an exponent.

1.3.2 LOGICAL OPERATIONS

AND

A **bitwise AND** takes two binary representations of equal length and performs the logical AND operation on each pair of corresponding bits. In each pair, the result is 1 if the first bit is 1 **AND** the second bit is 1. Otherwise, the result is 0. For example:

0101 (decimal 5) AND 0011 (decimal 3) = 0001 (decimal 1)

OR

A **bitwise OR** takes two bit patterns of equal length, and produces another one of the same length by matching up corresponding bits (the first of each; the second of each; and so on) and performing the logical inclusive OR operation on each pair of corresponding bits. In each pair, the result is 1 if the first bit is 1 **OR** the second bit is 1 **OR** both bits are 1, and otherwise the result is 0. For example:

0101 (decimal 5) OR 0011 (decimal 3) = 0111 (decimal 7)

NOT

The bitwise NOT, or complement, is a unary operation that performs logical negation on each bit, forming the ones' complement of the given binary value. Digits which were 0 become 1, and vice versa. For example:

NOT 0111 (decimal 7) = 1000 (decimal 8)\

NAND

A **bitwise NAND** takes two binary representations of equal length and performs the logical NAND operation on each pair of corresponding bits. In each pair, the result is 0 if the first bit is 1 **AND** the second bit is 1. Otherwise, the result is 1. For example:

0101 (decimal 5) AND 0011 (decimal 3) = 1110 (decimal 14)

NOR

A **bitwise NOR** takes two bit patterns of equal length, and produces another one of the same length by matching up corresponding bits (the first of each; the second of each; and so on) and performing the logical inclusive OR operation on each pair of corresponding bits. In each pair, the result is 1 if the both bits are zero otherwise the result is 0. For example:

0101 (decimal 5) OR 0011 (decimal 3) = 1000 (decimal 8)

XOR

A **bitwise exclusive or** takes two bit patterns of equal length and performs the logical XOR operation on each pair of corresponding bits. The result in each position is 1 if the two bits are different, and 0 if they are the same. For example:

0101 (decimal 5) XOR 0011 (decimal 3) = 0110 (decimal 6)

CHAPTER II: VHDL & MODELSIM

2.1 ABOUT VHDL

VHDL ((Very High Speed Integrated Circuits) Hardware Description Language) is a hardware description language used in the electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits.

VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. This collection of simulation models is commonly called a **testbench**.

We designed hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE, Altera Quartus, Synopsys Synplify or Mentor Graphics HDL Designer) to produce the RTL schematic of the desired circuit. After that, the generated schematic verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate testbench. To generate an appropriate testbench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required.

2.2 VHDL vs VERILOG

There are now two industry standard hardware description languages, VHDL and Verilog. The complexity of ASIC and FPGA designs has meant an increase in the number of specialist design consultants with specific tools and with their own libraries of macro and mega cells written in either VHDL or Verilog

- **Compilation**

  VHDL: Multiple design-units (entity/architecture pairs), that reside in the same system file, may be separately compiled if so desired. However, it is good design practice to keep each design unit in its own system file in which case separate compilation should not be an issue.

  Verilog:. The Verilog language is still rooted in its native interpretative mode. Compilation is a means of speeding up simulation, but has not changed the original nature of the language. As a result care must be taken with both the compilation order of code written in a single file and the compilation order of multiple files. Simulation results can change by simply changing the order of compilation.

- **Design Reusability**

  VHDL: Procedures and functions may be placed in a package so that they are available to any design-unit that wishes to use them.

  Verilog: There is no concept of packages in Verilog. Functions and procedures used within a model must be defined in the module. To make functions and procedures generally accessible from different module statements the functions and procedures must be placed in a separate system file and included using the `include compiler directive.

- **High level constructs**

  VHDL: There are more constructs and features for high-level modeling in VHDL than there are in Verilog.

  Abstract data types can be used along with the following statements:

  * package statements for model reuse,
  * configuration statements for configuring design structure,
  * generate statements for replicating structure,

* generic statements for generic models that can be individually characterized, for example, bit width.

All these language statements are useful in synthesizable models.

Verilog: Except for being able to parameterize models by overloading parameter constants, there is no equivalent to the high-level VHDL modeling statements in Verilog.

- **Managing large designs**
  VHDL: Configuration, generate, generic and package statements all help manage large design structures.
  Verilog: There are no statements in Verilog that help manage large designs

- **Procedures and tasks**

  VHDL: allows concurrent procedure calls.

  Verilog: does not allowconcurrent task calls.

- **Test harnesses**
  Designers typically spend about 50% of their time writing synthesizable models and the other 50% writing a test harness to verify the synthesizable models. Test harnesses are not restricted to the synthesizable subset and so are free to use the full potential of the language. VHDL has generic and configuration statements that are useful in test harnesses that are not found in Verilog.

2.3 ADVANTAGES OF VHDL

1.) 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).

2.) Another benefit is that VHDL allows the description of a concurrent system. VHDL is a dataflow language, unlike procedural computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time.

3.) VHDL project is multipurpose. Being created once, a calculation block can be used in many other projects. However, many formational and functional block parameters can be tuned (capacity parameters, memory size, element base, block composition and interconnection structure).

4.) VHDL project is portable. Being created for one element base, a computing device project can be ported on another element base, for example VLSI with various technologies.

An example of VHDL coding....

```
library IEEE;
use IEEE.std_logic_1164.all;

-- this is the entity
entity ANDGATE is
  port (
       IN1 : in std_logic;
       IN2 : in std_logic;
       OUT1: out std_logic);
end ANDGATE;
```

```
architecture RTL of ANDGATE is
begin

OUT1 <= IN1 and IN2;

End RTL;
```

2.4 MODELSIM TOOL

TOOL USED: MODELSIM 6.0d

Mentor Graphics ModelSim 6.0d HDL Simulator is a source-level verification tool, allowing you to verify HDL code line by line. We can perform simulation at all levels: behavioral (pre-synthesis), structural (post-synthesis), and back-annotated, dynamic simulation.

Coupled with the most popular HDL debugging capabilities in the industry, ModelSim is known for delivering high performance, ease of use, and outstanding product support.

An easy-to-use graphical user interface enables you to quickly identify and debug problems, aided by dynamically updated windows. For example, selecting a design region in the Structure window automatically updates the Source, Signals, Process, and Variables windows. These cross linked ModelSim windows create an easy-to-use debug environment. Once a problem is found, you can edit, recompile, and re-simulate without leaving the simulator. ModelSim fully supports current VHDL and Verilog language standards. You can simulate behavioral, RTL, and gate-level code separately or simultaneously. ModelSim supports all Actel FPGA libraries, ensuring accurate timing simulations.

The comprehensive user interface makes efficient use of desktop real estate. The intuitive arrangement of interactive graphical elements (windows, toolbars, menus, etc.) makes it easy to view and access the many powerful capabilities of ModelSim. The result is a feature-rich user interface that is easy to use and quickly mastered.

Project Flow

A project is a collection mechanism for an HDL design under specification or test. Even though you don't have to use projects in ModelSim, they may ease interaction with the tool and are useful for organizing files and specifying simulation settings. The following diagram shows the basic steps for simulating a design within a ModelSim project.

As you can see, the flow is similar to the basic simulation flow. However, there are two important differences:
• You do not have to create a working library in the project flow; it is done for you automatically.
• Projects are persistent. In other words, they will open every time you invoke ModelSim unless you specifically close them.

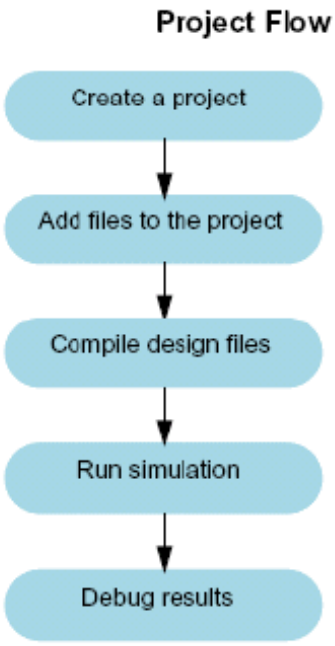


Figure2.1 – Project Flowchart

CHAPTER III: XILINX VERTEX 4 FPGA

3.1 INTRODUCTION

The ML401/ML402/ML403 evaluation platform enables designers to investigate and experiment with features of the Vertex™-4 families of FPGAs. This user guide describes features and operation of the ML401, ML402, and ML403 (ML40*x*) evaluation platforms.

Figure3.1 - XILINX VERTEX 4 FPGA

FEATURES:

• Virtex-4 FPGA:
    - ♦ ML401: XC4VLX25-FF668-10
    - ♦ ML402: XC4VSX35-FF668-10
    - ♦ ML403: XC4VFX12-FF668-10

• 64-MB DDR SDRAM, 32-bit interface running up to 266-MHz data rate

• One differential clock input pair and differential clock output pair with SMA Connectors.

• One 100-MHz clock oscillator (socketed) plus one extra open 3.3V clock oscillator Socket

• General purpose DIP switches (ML401/ML402 platform), LEDs, and push buttons

• Expansion header with 32 single-ended I/O, 16 LVDS capable differential pairs,14 spare I/Os shared with buttons and LEDs, power, JTAG chain expansion capability, and IIC bus expansion

• Stereo AC97 audio codec with line-in, line-out, 50-mW headphone, and microphone-in (mono) jacks

• RS-232 serial port

• 16-character x 2-line LCD display

• One 4-Kb IIC EEPROM

• VGA output:

- ♦ ML401: 50 MHz / 24-bit video DAC
- ♦ ML402: 140 MHz / 24-bit video DAC
- ♦ ML403: 140 MHz / 15-bit video DAC

3.2 Xilinx ISE 9.1i

The version includes all the features of the 9.1i release of the popular Xilinx ISE Foundation™ software with full support for optional embedded, digital signal processing (DSP) and real-time debug design flows. Most notably, ISE WebPACK 9.1i software includes the new Xilinx SmartCompile™ technology, which significantly improves run times by up to 6x faster than the previous version, while maintaining exact design preservation of unchanged logic. ISE WebPACK 9.1i software also includes support for all devices in the Spartan™-3A family of FPGAs and select Virtex™-4 and Virtex-5 FPGA devices. New power optimization features help designers reduce dynamic power by an average of 10 percent.

FPGA Industry's Most Complete Design Solution for Windows and Linux

ISE WebPACK 9.1i software offers a complete front-to-back FPGA design solution allowing users to immediately begin projects. By providing integrated tools for HDL entry, synthesis, implementation, and verification in a free downloadable environment, ISE 9.1i helps users rapidly achieve design goals while reducing overall project cost. This release includes ISE Simulator Lite on both Windows and Linux. The free MXE-III Starter version is available for download from the Xilinx website giving designers a choice in free HDL verification solutions.

3.3 ADVANTAGES

3.3.1 Increased Productivity

ISE WebPACK 9.1i software includes new SmartCompile technology to help designers address the problems associated with re-implementing an entire design with each incremental change. Such re-implementations take time and introduce risk of disrupting portions of the design not directly involved with the change. Xilinx SmartCompile technology addresses these issues with the following technologies:

Partitions: minimize effects of minor changes late in design cycles with copy-and-paste functionality that automatically provides exact preservation of existing placement and routing and reduces re-implementation time.

SmartGuide™: reduces time for re-implementation for small changes by leveraging prior implementation results.

SmartPreview™: enables users to pause and resume place-and-route process and save intermediate results to evaluate design state. By previewing implementation information such as routing status and timing results, users can make important trade-off decisions without waiting for complete implementation.

3.3.2 Faster Timing Closure

New features in ISE WebPACK 9.1i software build on the capabilities of Fmax technology, especially designed to deliver unparalleled performance and timing closure results for high density, high performance designs. ISE WebPACK 9.1i software includes integrated timing closure flow which incorporates enhanced physical synthesis optimizations to provide higher quality of results.

ISE WebPACK 9.1i software includes the expanded timing closure environment of the standard ISE 9.1i version – a virtual ‘Timing Closure Cockpit’ – that enables intuitive cross-probing between constraint entry, timing analysis, floor planning and report views

so designers can more easily analyze timing problems. The integrated timing closure flow incorporates enhanced physical synthesis with improved timing correlation between synthesis and placement timing, resulting in higher quality of results.

3.3.3 Power Optimization

New power optimization in Xilinx Synthesis Technology (XST) and placement, together with improvements in routing, deliver an average of 10 percent lower dynamic power for the Spartan-3 generation of FPGAs. Power optimization improvements in XST also provide power-aware logic optimizations for macro processing on blocks such as multipliers, adders and BRAMs. Implementation algorithms deploy power-efficient placement strategies and lower capacitance nets within the device to minimize power without sacrificing performance.

3.4 DESIGN FLOW OVERVIEW

Xilinx tools can be installed on your own PC so that you can draw schematics or write the code, perform simulations without coming to the lab.You may download a free, compatible version from the Xilinx web page called ISE WebPACK 10.1

The design steps are explained here

3.4.1 Design Entry

The first step is to enter your design. This can be done by creating “Source” files. Source files can be created in different formats such as a schematic, or a Hardware Description Language (HDL) such as VHDL, Verilog . A project design will consist of a top-level source file and various lower level source files. Any of these files can be either a schematic or a HDL file.

3.4.2 Design Simulation

Verification of the functionality can be done using Behavioral Simulation. Create a test bench waveform containing input stimulus you can use to verify the functionality your module.

3.4.3 Design Synthesis

The synthesis step creates EDIF or NGC netlist files from the various source files. The netlist files can serve as an input to the implementation module. Popular synthesis tools include: Synplify, Precision, PGA Compiler II, and XSTgn.

3.4.4 Functional and Timing simulation

Design verification can be done at various stages. The simulator is used to verify the functionality of a design (functional simulation), the behavior and the timing (timing simulation) of your circuit. Timing simulation is run after implementing your circuit in the FPGA since it needs to know the actual placement and routing to find out the exact speed and timing of the circuit.

3.4.5 Design Implementation

After generating the netlist file (synthesis step), the implementation will convert the logic design into a physical file that can be downloaded on the target device. This step involves three sub-steps: Translating the netlist, Mapping and Place & Route. There are several outputs of implementation: Reports, Timing simulation netlists, Floor plan files, FPGA Editor files.

3.4.6 Download Design to the FPGA Board

Once a design is implemented, you must create a file that the FPGA can understand. This file is called a bitstream: a BIT file (.bit extension). The BIT file can be downloaded directly into the FPGA, or the BIT file can be converted into a PROM file, which stores the programming information.

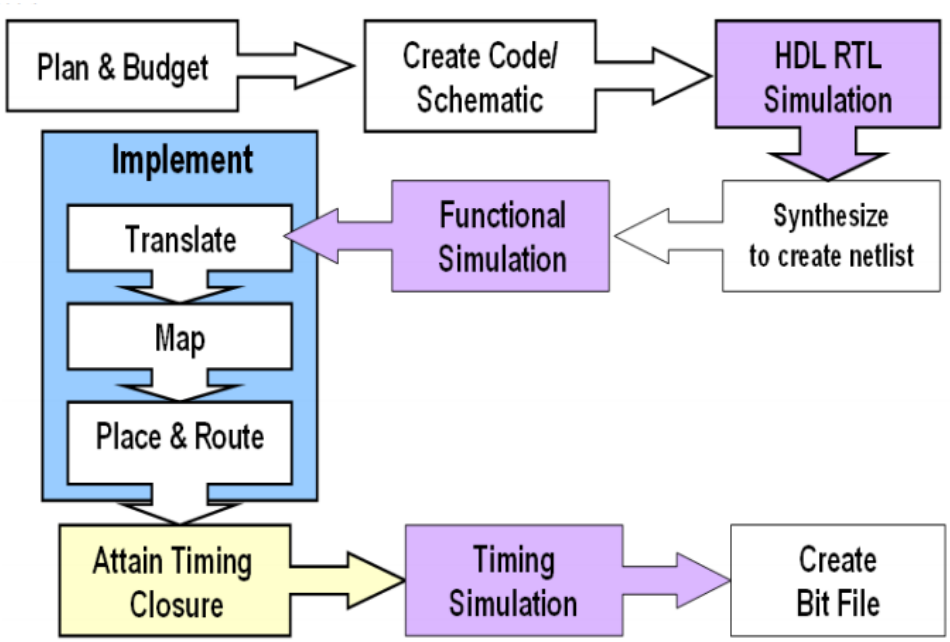


Figure – 3.2 Design Flow

3.5 CREATE A NEW PROJECT

Use this dialog box to create a new project, as described in Creating a Project. To access this dialog box, select **File > New Project**.

- **Name:** Specifies the name for the project. Follow the naming conventions in Naming Conventions.
- **Location:** Specifies the location of the project. You can browse to a directory or enter the name of a directory. If you enter the name of a directory that does not exist, Project Navigator creates the directory. By default, the software automatically creates a subdirectory based on the name entered in the Name field.
- **Working Directory:** Specifies the location of the working directory. By default, the working directory is the same as the project directory. However, you can specify a working directory if you want to keep your ISE project file (.xise extension) separate from your working area.

- **Description:** Allows you to add a description for your project. This field is optional.
- **Product Category:** Specifies an applicable product category. This selection filters the device families and devices that are available in the Family and Device fields.
- **Family:** Specifies the device family, or Xilinx® architecture, into which you will implement your design.
- **Device:** Specifies the device into which you will implement your design.
- **Package:** Specifies the package for the device being targeted.
- **Speed:** Specifies the speed grade of the device being targeted.
- **Top-Level Source Type:** Specifies the source type for the top-level design.
    - **HDL** Select this option if your top-level design file is a VHDL or Verilog file. An HDL Project can include lower-level modules of different file types, such as other HDL files, schematics, and "black boxes," such as IP cores and EDIF files.
    - **Schematic** Select this option if your top-level design file is a schematic file. A schematic project can include lower-level modules of different file types, such as HDL files, other schematics, and "black boxes," such as IP cores and EDIF files. Project Navigator automatically converts any schematic files in your design to structural HDL before implementation; therefore, you *must* specify a synthesis tool when working with schematic projects.
    - **EDIF** Select this option if you want to use an EDIF netlist as the top-level source for the project. This may be the case if you are using a synthesis tool outside of Project Navigator to synthesize the design.
    - **NGC/NGO** Select this option if you want to use an NGC or NGO netlist as the top-level source for the project.
- **Synthesis Tool:** Specifies the synthesis tool and synthesis language used for your design. For synthesis tools that support only single-language designs, select the

appropriate language for your design (for example, Synplify (VHDL) or Synplify (Verilog)).

- **XST** Xilinx Synthesis Technology (XST) is provided with the ISE® software. It supports projects that include VHDL, Verilog, and schematic design files. Mixed-language designs are supported **Synplify and Synplify Pro** Synplify and Synplify Pro are integrated third party synthesis tools that must be purchased separately from Synplicity, Inc. The Synplify software does *not* support projects that include mixed language source files. The Synplify Pro software supports projects that include mixed language source files, such as VHDL and Verilog sources files in the same project. The Synplify and Synplify Pro software do not support projects that include schematic design files.
- **Precision:** Precision is an integrated third party synthesis tool that must be purchased separately from Mentor Graphics, Inc. The Precision software supports projects that include schematic design files and projects that include mixed language source files, such as VHDL and Verilog sources files in the same project.

- **Simulator:** Specify the tool used for simulation and the language used for generating simulation netlists.
  - **ISim:** ISim is the simulator delivered with the ISE software. For more information about this tool
  - **ModelSim:** ModelSim is a third party tool that can be used in an integrated flow within the ISE software.
  - **NC-Sim** The NC-Sim simulator is a third party simulation tool that must be purchased separately from Cadence. It is *not* integrated with the ISE software and must be run standalone. For more information, see the documentation provided with the simulator.

- **VCS:** The VCS simulator is a third party simulation tool that must be purchased separately from Synopsys. It is *not* integrated with the ISE software and must be run standalone. For more information, see the documentation provided with the simulator.
- **Other:** Select other if you are using a simulator that is not listed.

- **Preferred Language:** Controls the default setting for process properties that generate HDL output, such as source files, intermediate files, or structural simulation netlists. If the Synthesis Tool and Simulator options are set to a single-language tool, the default language for generated HDL output files is automatically set. If both the Synthesis Tool and Simulator options are set to mixed-language (VHDL/Verilog) tools, you can use the Preferred Language property to select the language in which generated HDL output is created.
  - **Verilog:** Select this option if both the Synthesis Tool and Simulator are set to mixed-language and you want the default language to be Verilog.
  - **VHDL:** Select this option if both the Synthesis Tool and Simulator are set to mixed-language and you want the default language to be VHDL.
  - **N/A:** This option appears if both the Synthesis Tool and Simulator are set to a single language, because the generated language defaults are set based on the languages you selected for the Synthesis Tool and Simulator.
- **Property Specification in Project File:** Controls how properties are stored in the .xise file.
  - **Store non-default values only:** Select this option to store only non-default property settings in the .xise project file.
  - **Store all values:** Select this option to store *all* property settings, including those set to default values, in the .xise project file. This option is useful when working with source control systems and when moving projects between different ISE software versions, because the values for all properties are stored explicitly.

- **Manual Compile Order:** By default, the RTL compilation order is automatically determined based on the ISE design hierarchy. This option allows you to override the default behavior and set the compilation order manually. Selecting this option disables all hierarchical parsing of HDL source files when they are added to the project, and the design source is displayed in a single flat list rather than hierarchically. For designs with a large number of HDL sources, this can make adding the source files faster.

- **Enable Enhanced Design Summary:** Shows the number of errors and warnings for the entire project and for each of the Detailed Reports.

- **Enable Message Filtering:** Shows the number of messages you filtered. You must enable this option, filter messages, and then run the software to show the number of filtered messages.

- **Display Incremental Messages:** Shows the numbers of new messages for the most recent software run. You must enable this option, and then run the software to show the number of new messages.

3.6 EDITING THE HDL SOURCE FILE

The source file is displayed in the Project Navigator window. The source file window can be used as a text editor to make any necessary changes to the source file. Make sure you enter the pin type for all output pins have to (combinational or sequential). Save the HDL program periodically by selecting the File->Save from the menu.

3.7 TIMING CONSTRAINTS UCF

The UCF file is an ASCII file specifying constraints on the logical design. You create this file and enter your constraints in the file with a text editor. You can also use the Xilinx Constraints Editor to create constraints within a UCF file. These constraints affect how the logical design is

implemented in the target device. You can use the file to override constraints specified during design entry.

UCF Flow

The following figure illustrates the UCF flow.

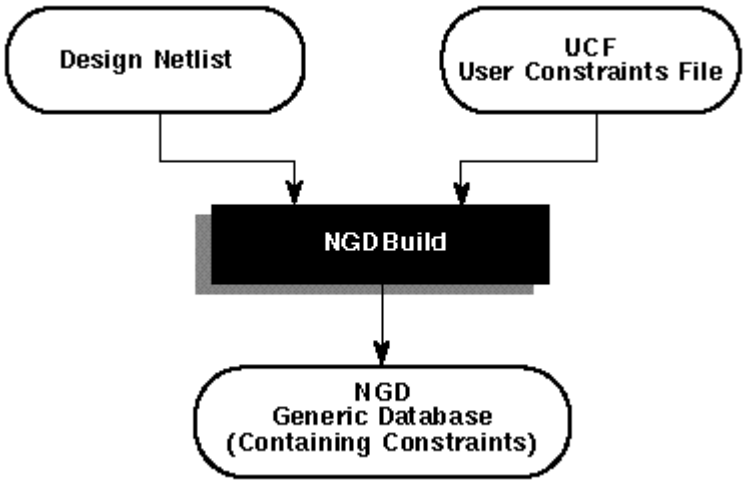


Figure – 3.3 UCF Design Flow

The UCF file is an input to NGDBuild (see the preceding figure). The constraints in the UCF file become part of the information in the NGD file produced by NGDBuild. For FPGAs, some of these constraints are used when the design is mapped by MAP and some of the constraints are written into the PCF (Physical Constraints File) produced by MAP.

The constraints in the PCF file are used by the each of the physical design tools (for example, PAR and the timing analysis tools), which are run after your design is mapped.

Manual Entry of Timing Constraints: You can manually enter timing specifications as constraints in a UCF file. When you then run NGDBuild on your design, your timing constraints are added to the design database as part of the NGD file.

To avoid manually entering timing constraints in a UCF file, use the Xilinx Constraints Editor, a tool that greatly simplifies constraint creation. For a detailed description of how to use the editor, see the Xilinx Constraints Editor online help.

UCF/NCF File Syntax: Logical constraints are found in:

- a Netlist Constraint File (NCF), an ASCII file generated by synthesis programs
- User Constraint File (UCF), an ASCII file generated by the user

This section describes the rules for entering constraints in a UCF or NCF file.

It is preferable to place any user-generated constraint in the UCF file -- *not* in an NCF or PCF file.

General Rules

Following are some general rules for the UCF and NCF files.

- The UCF and NCF files are case sensitive. Identifier names (names of objects in the design, such as net names) must exactly match the case of the name as it exists in the source design netlist. However, any Xilinx constraint keyword (for example, LOC, PERIOD, HIGH, and LOW) may be entered in all upper-case, all lower-case, or mixed case.
- Each statement is terminated by a semicolon (;).
- No continuation characters are necessary if a statement exceeds one line, since a semicolon marks the end of the statement.
- You can add comments to the UCF/NCF file by beginning each comment line with a pound (#) sign. Following is an example of part of a UCF/NCF file containing comments.
- # file TEST.UCF
- # net constraints for TEST design
- NET "$SIG_0" MAXDELAY 10;

- NET "$SIG_1" MAXDELAY 12 ns;

  C and C++ style comments (/* */ and respectively) are also supported.

- Statements do not have to be placed in any particular order in the UCF/NCF file.

- Although not required, Xilinx recommends that NET and INST names be enclosed in double quotes to avoid errors. Additionally, inverted signal names that contain a tilde, for example, ~OUTSIG1, must always be enclosed in double quotes.

Conflict in Constraints

The constraints in the UCF/NCF files and the constraints in the schematic or synthesis file are applied equally. It does not matter whether a constraint is entered in the schematic or synthesis file or in the UCF/NCF files. If the constraints overlap, UCF overrides NCF and schematic constraints. NCF overrides schematic constraints.

If by mistake two or more elements are locked onto a single location, the mapper detects the conflict, issues a detailed error message, and stops processing so that you can correct the mistake.

Syntax

The syntax for constraints in the UCF/NCF files is:

{**NET**|**INST**|**PIN**} "*full_name*" *constraint***;**

or

SET *set_name set_constraint***;**

where

- *full_name* is a full hierarchically qualified name of the object being referred to. When the name refers to a pin, the instance name of the element is also required.

- *Constraint* is a constraint in the same form as it would be used if it were attached as an attribute on a schematic object. For example, LOC=P38 or FAST, and so forth.
- *set_name* is the name of an RLOC set.
- *set_constraint* is an RLOC_ORIGIN or RLOC_RANGE constraint

Specifying Attributes for TIMEGRP and TIMESPEC

To specify attributes for TIMEGRP, the keyword TIMEGRP precedes the attribute definitions in the constraints files.

TIMEGRP "input_pads"=pads EXCEPT output_pads;

3.8 DOWNLOAD DESIGN TO THE FPGA BOARD

This is the last step in the design verification process. This section provides simple instructions for downloading the counter design to the Spartan-3 Starter Kit demo board.

1. Connect the 5V DC power cable to the power input on the demo board (J4).

2. Connect the download cable between the PC and demo board (J7).

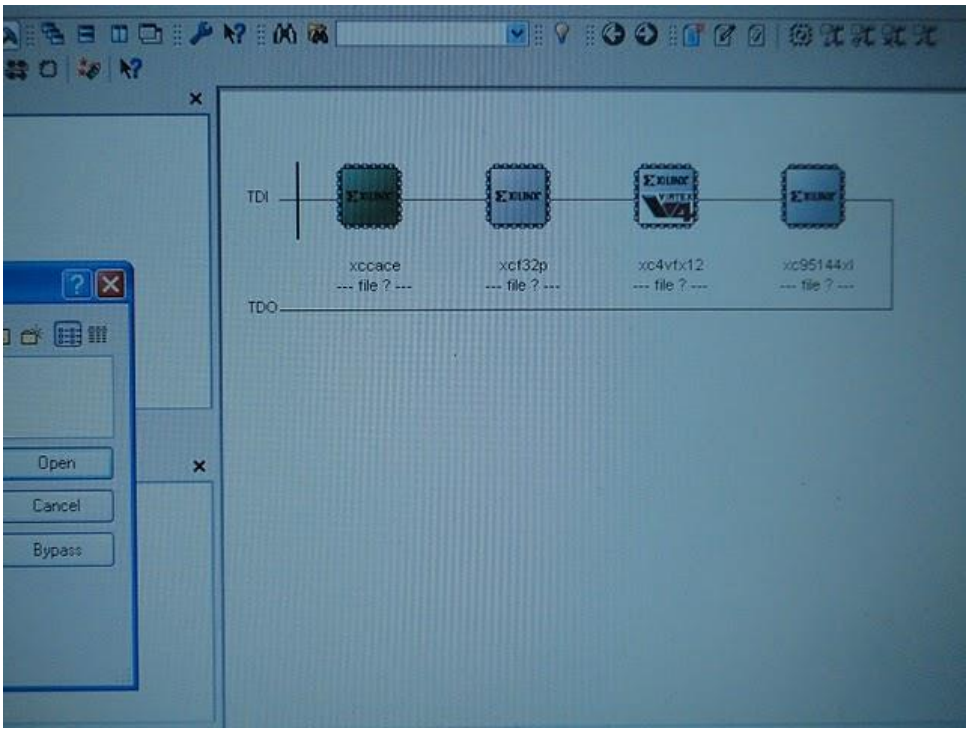
3. Select **Synthesis/Implementation** from the drop-down list in the Sources window.

4. Select **counter** in the Sources window.

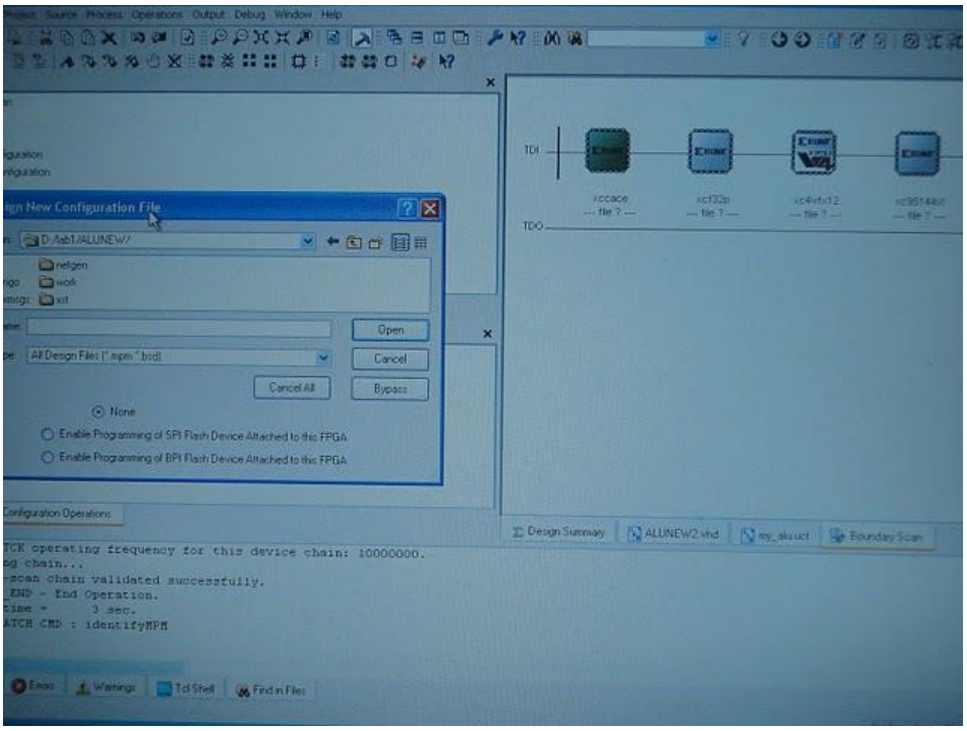
5. In the Processes window, click the "+" sign to expand the Generate Programming

File processes.

6. Double-click the **Configure Device (iMPACT)** process.

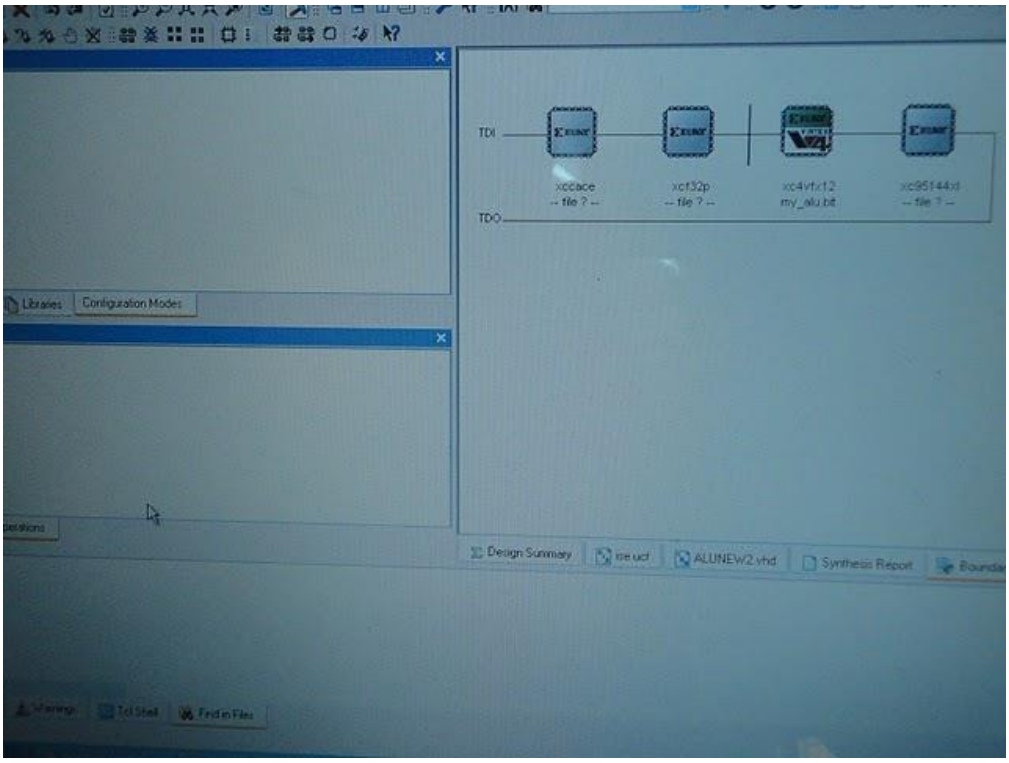


1. The Xilinx WebTalk Dialog box may open during this process. Click **Decline**.



8. Select **Disable the collection of device usage statistics for this project only** and click **OK**.

9. In the Welcome dialog box, select **Configure devices using Boundary-Scan (JTAG).**



10. Verify that **automatically connect to a cable and identify Boundary-Scan chain** is selected.

11. Click **Finish**.

12. If you get a message saying that there are two devices found, click OK to continue.

The devices connected to the JTAG chain on the board will be detected and displayed in the iMPACT window.

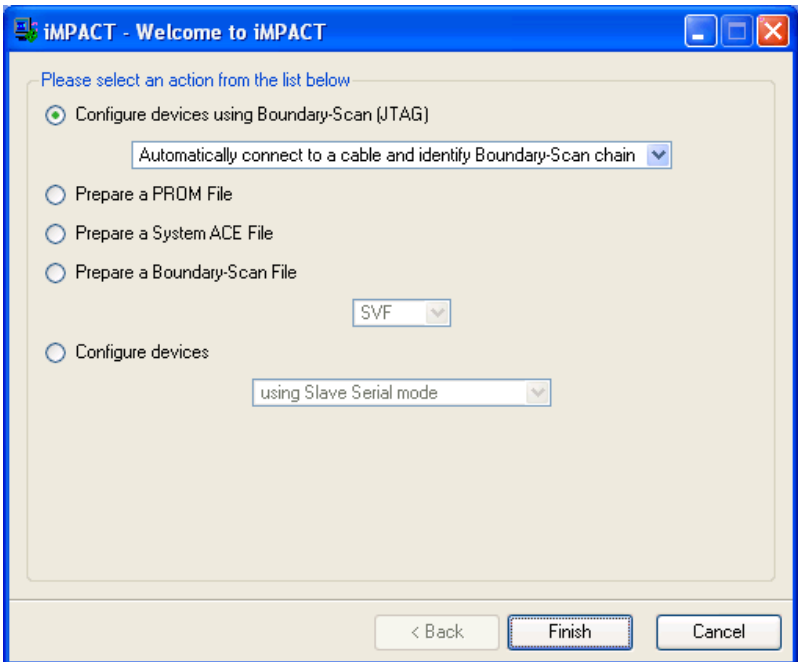
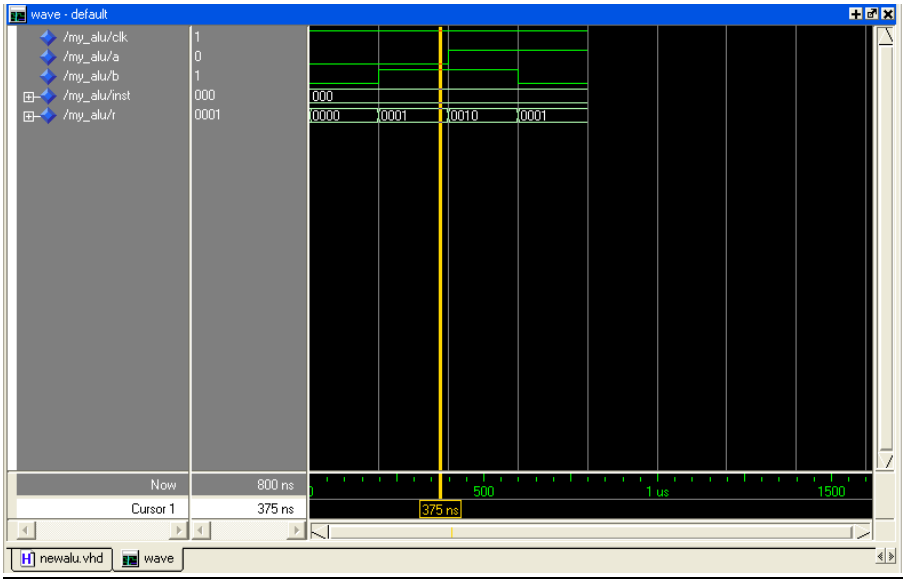


Figure – 3.4 impact Welcome Dialog Box

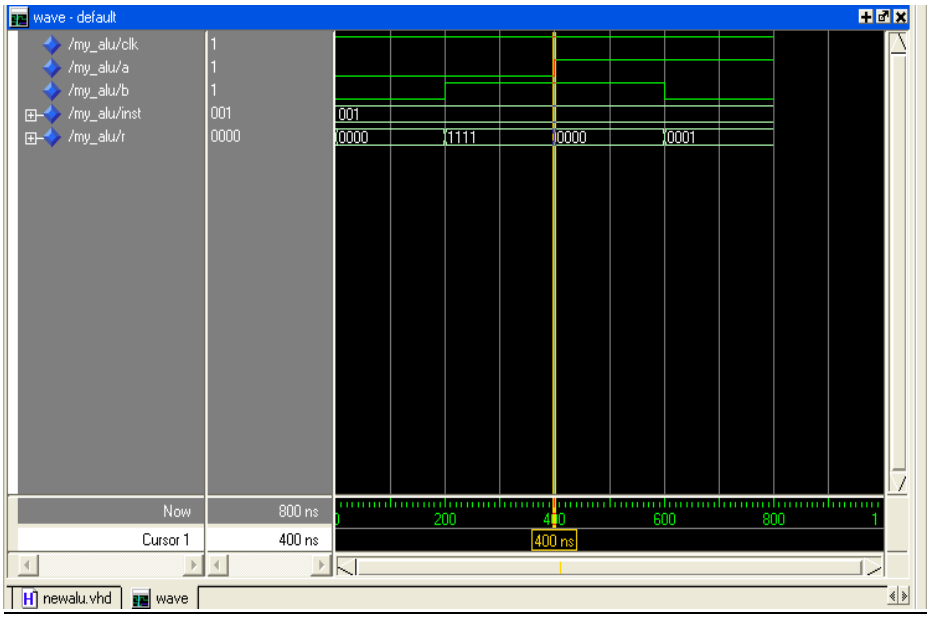
CHAPTER IV: RESULTS AND DISCUSSIONS

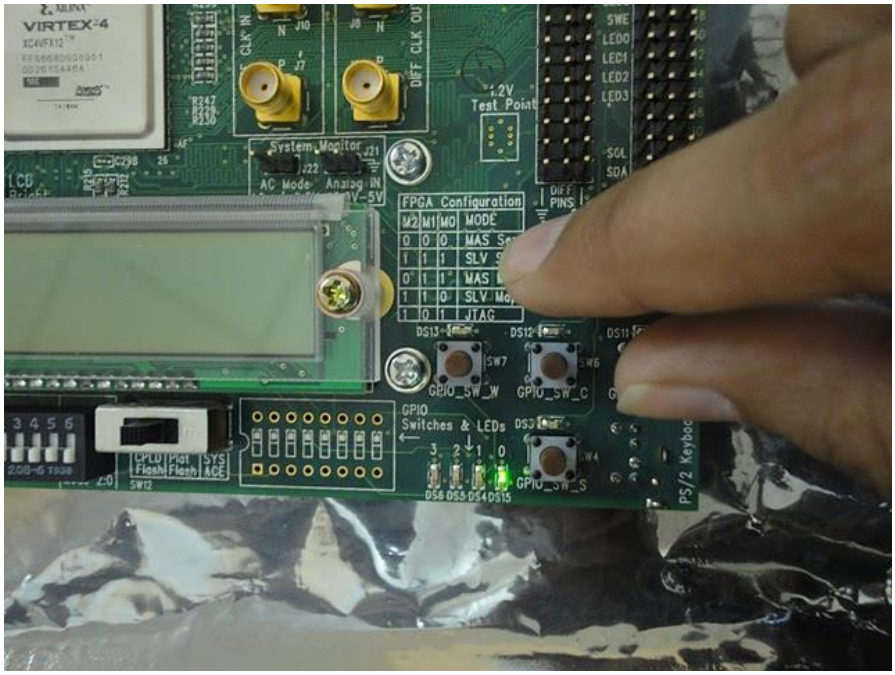
4.1 RESULTS OF 1-BIT ALU-

1. A+B when inst=000

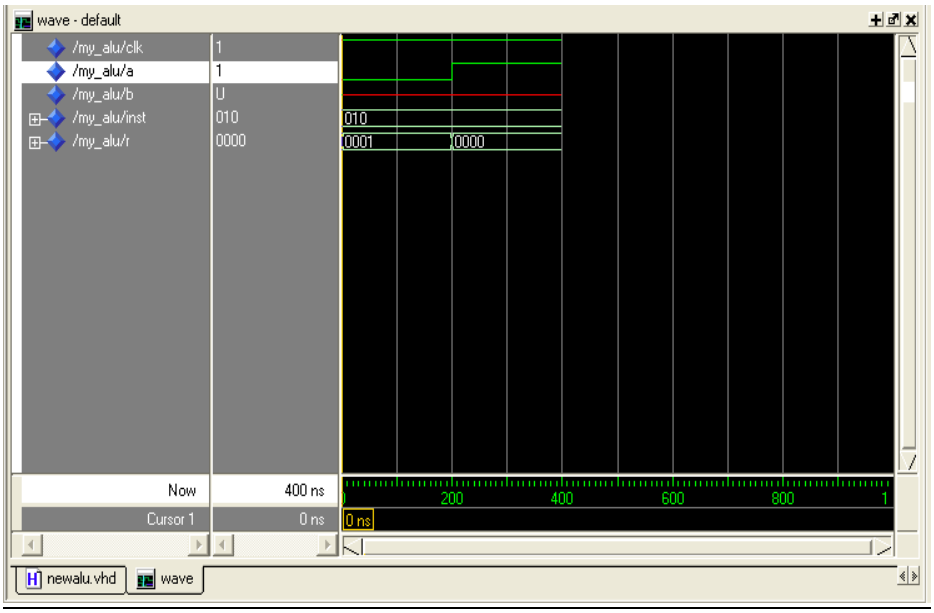


2.) A-B when inst = 001

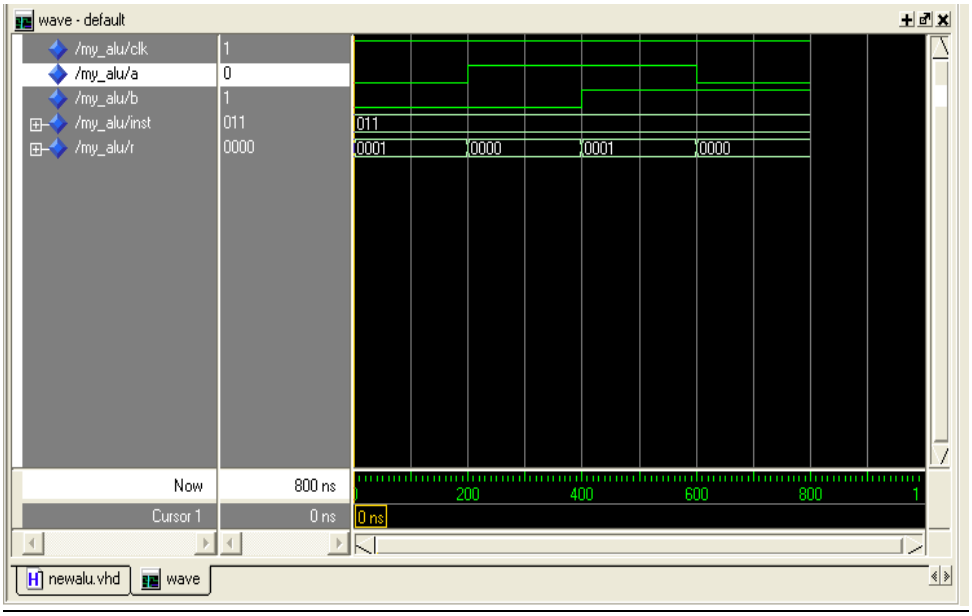




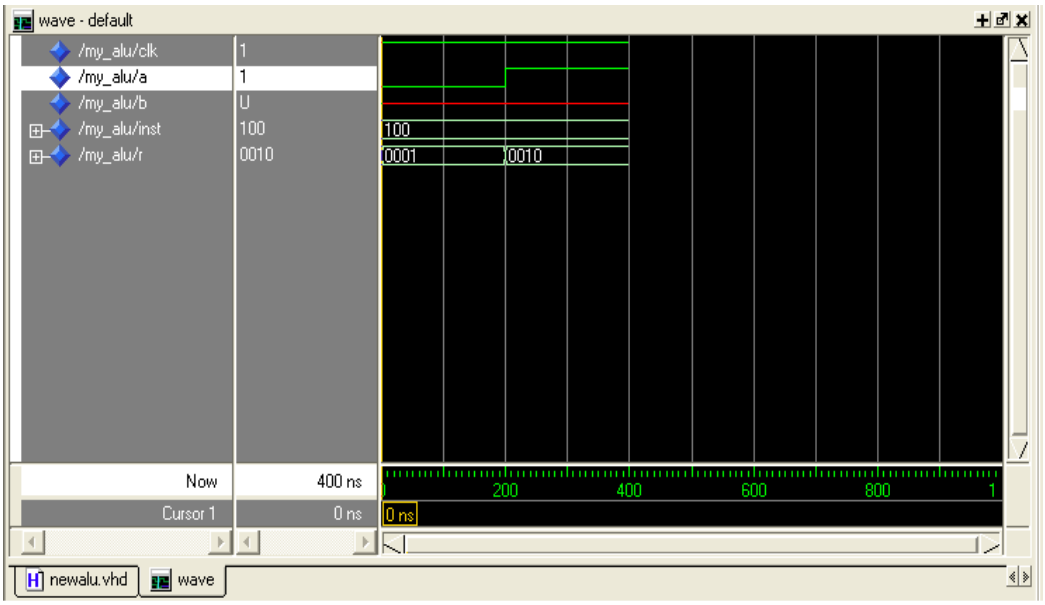
3.) not A when inst= 010



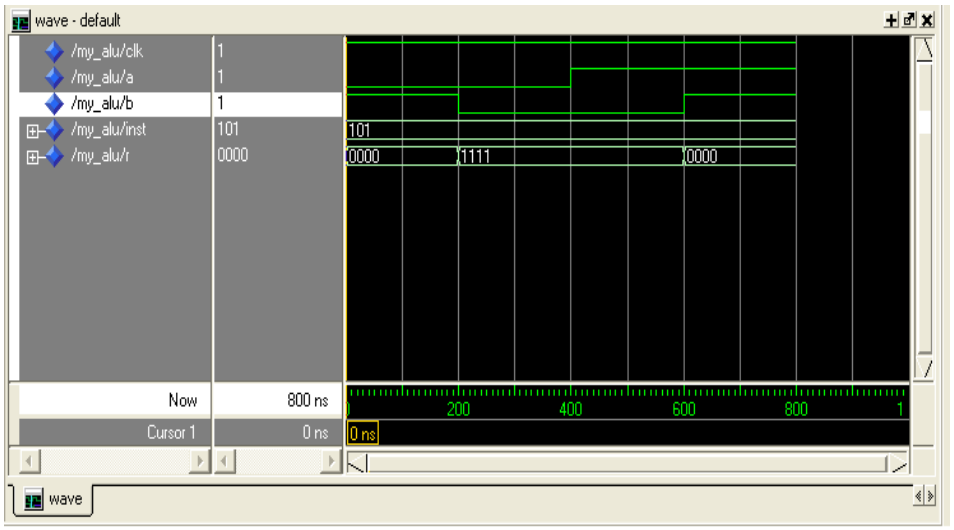
4.) Is A=B when inst = 011

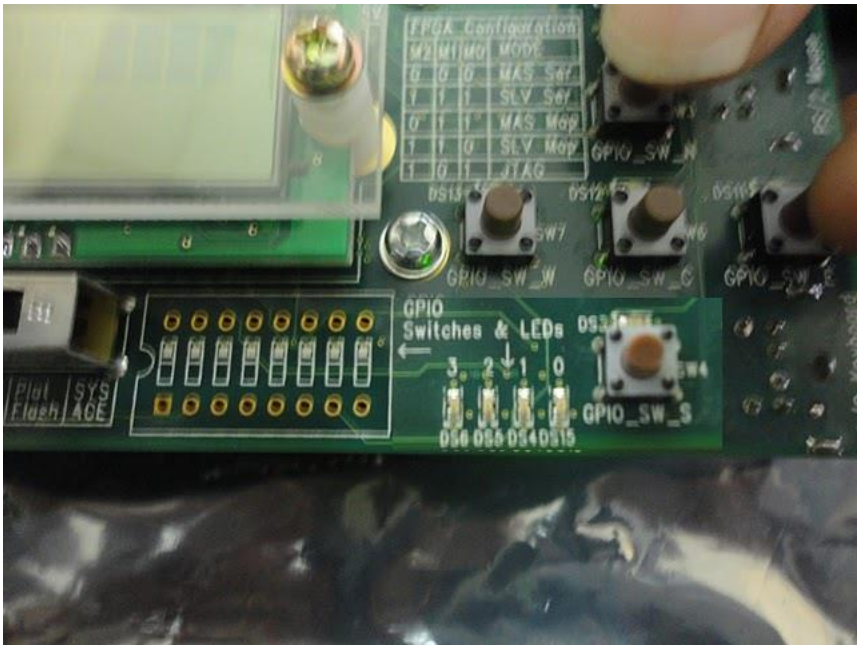


5.) A= A+ 1 when inst=100

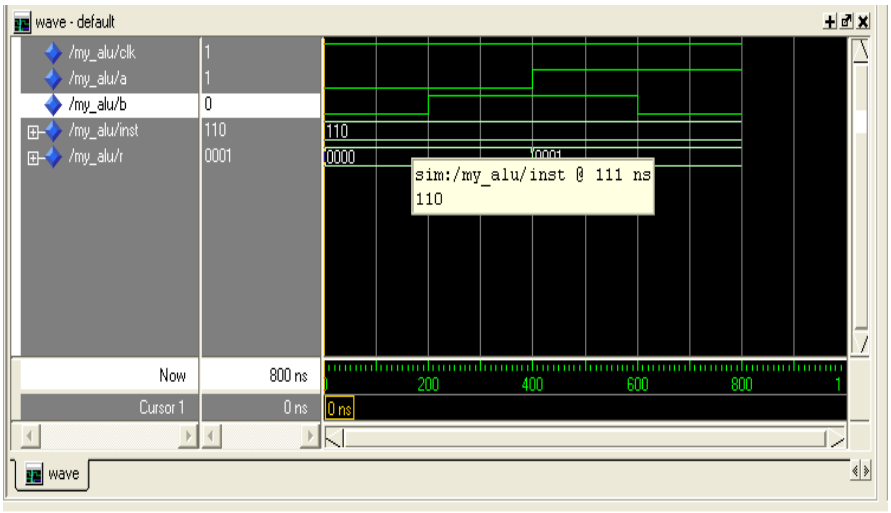


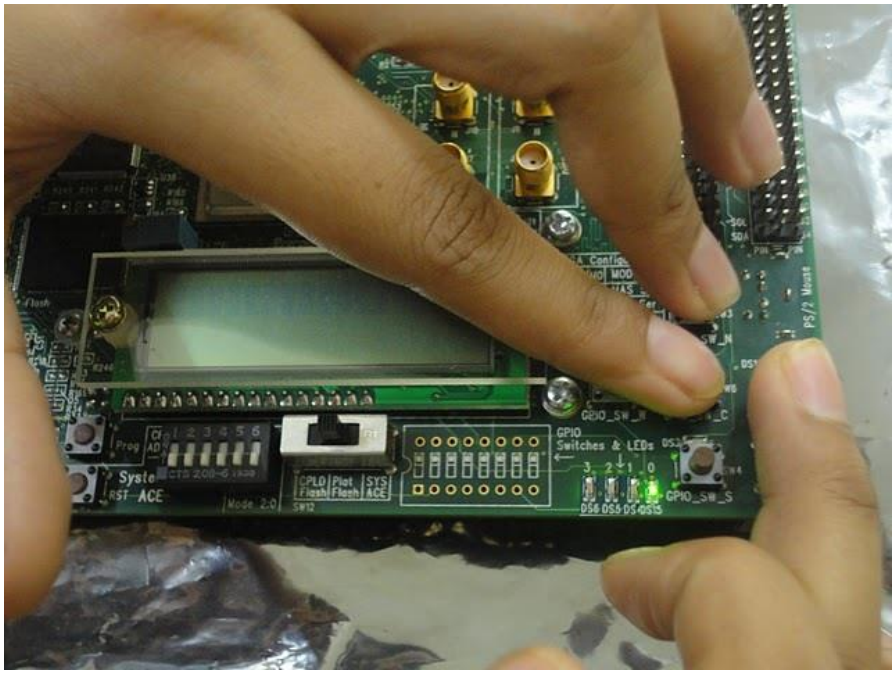
6. A mod B when inst=101



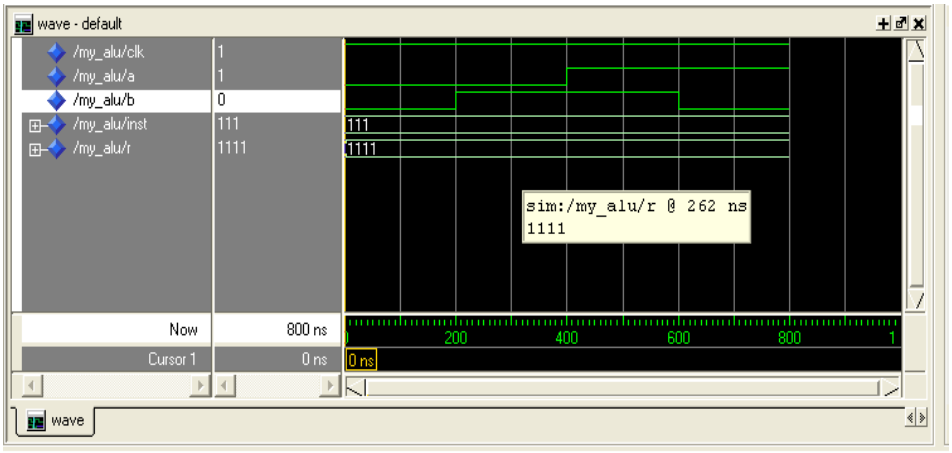


7. A **2 when inst=110

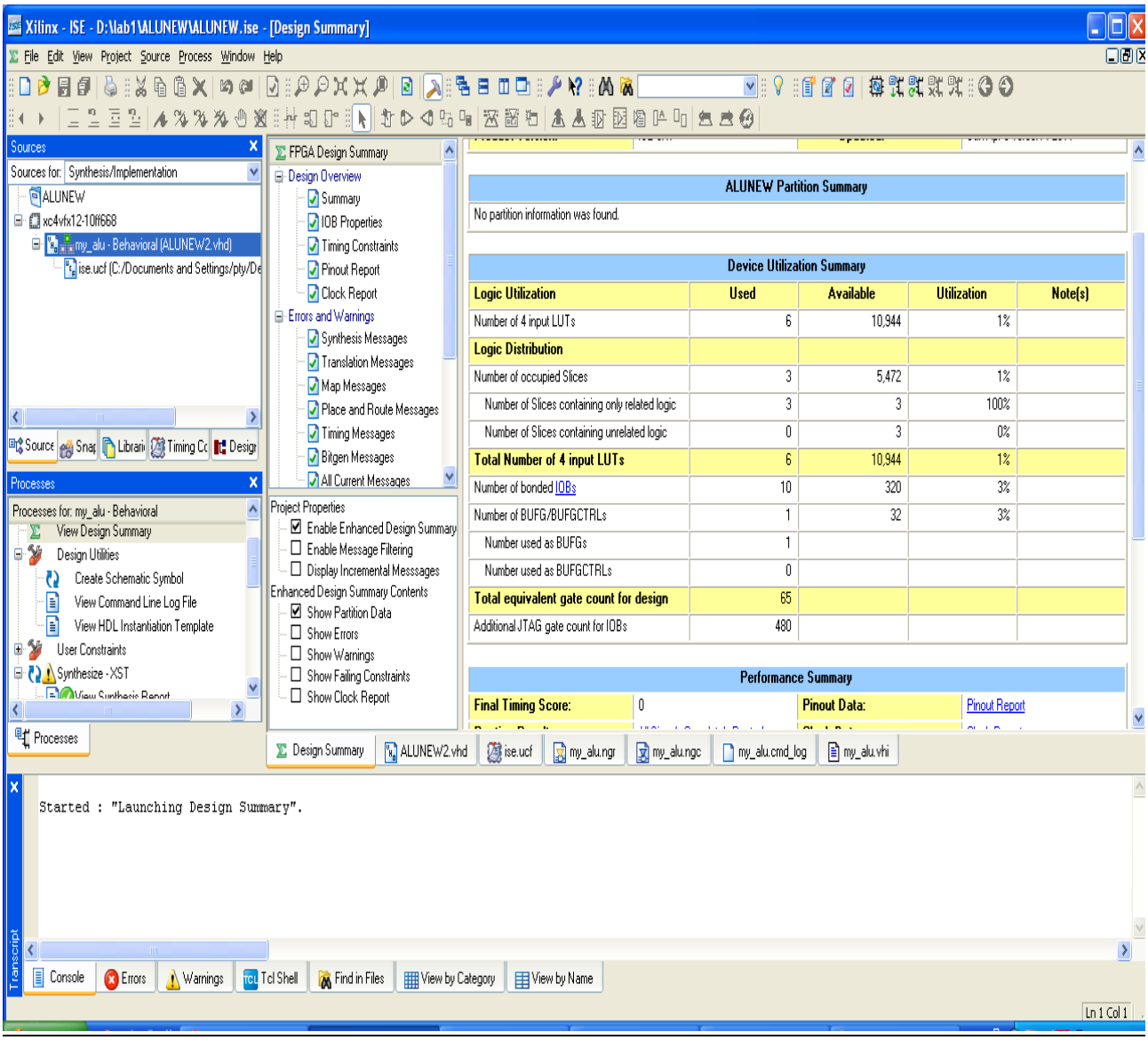




8. When others

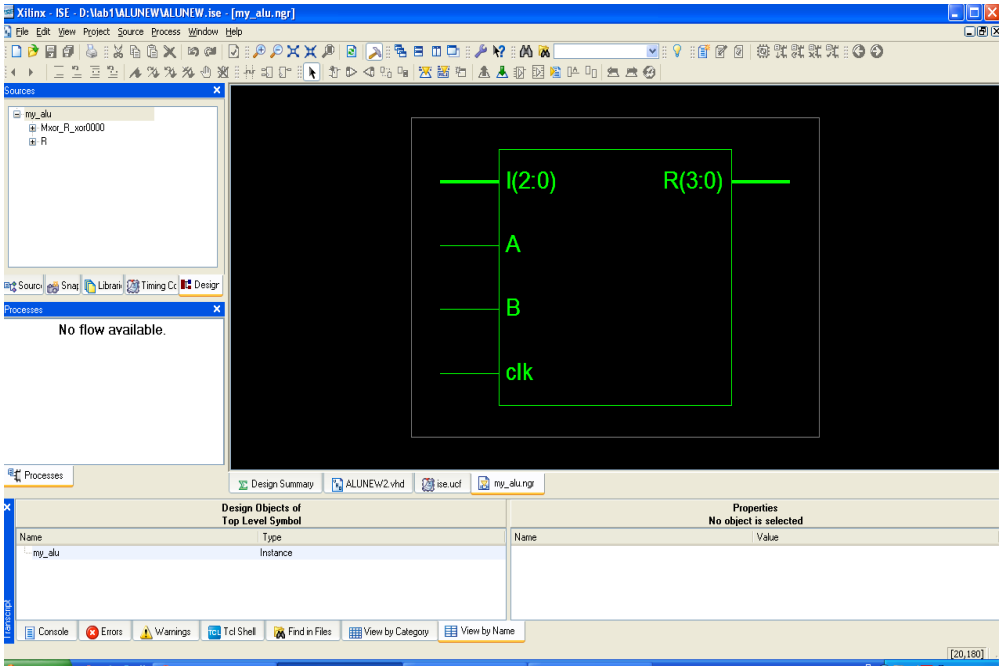


4.2 DESIGN SUMMARY:

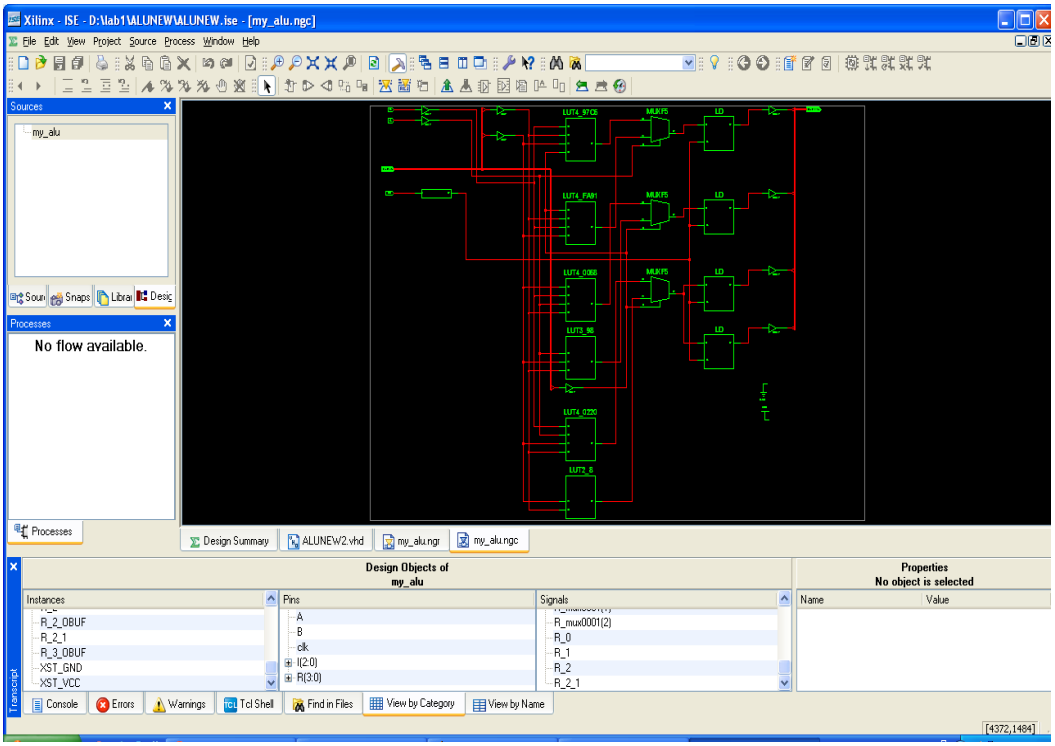


4.3 SCHEMATICS

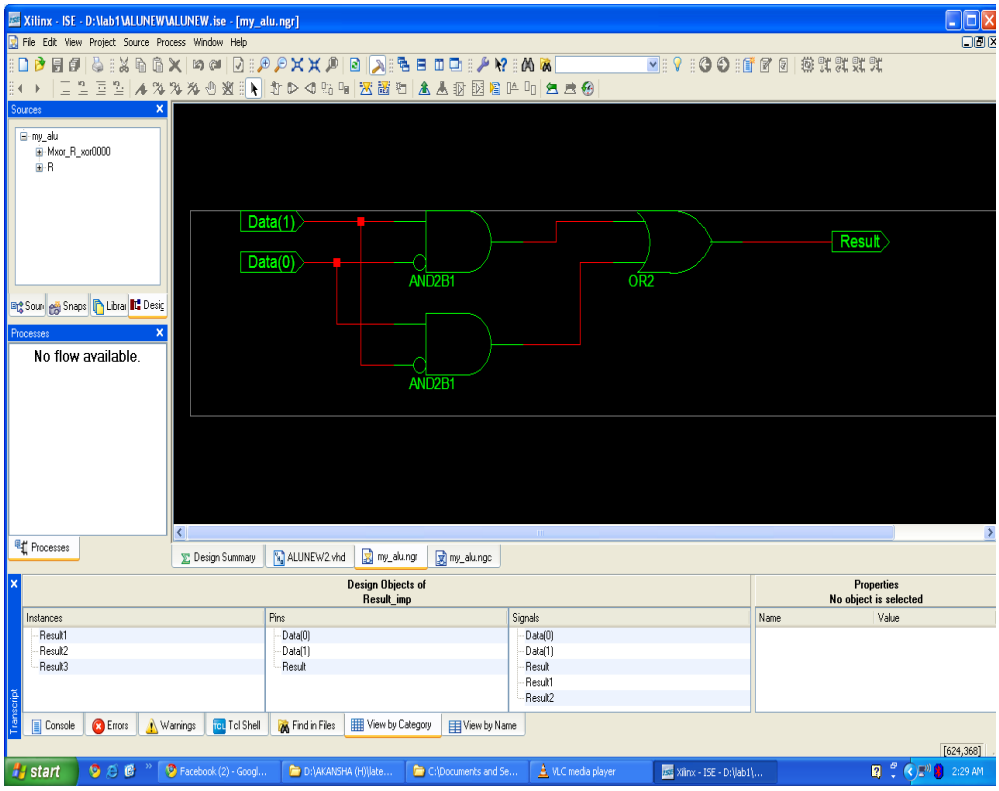
1.) RTL SCHEMATICS :



2.) SCHEMATICS



3.) STRUCTURE:



TECHNICAL DIFFICULTY

We were using the FPGA kit vertex ML403 which had added certain restrictions to the project. The push buttons that were used to provide the inputs from the user were five in number whereas we required twenty one; as we were inputting two 8 bits data from user and five bits for the selective instructions. Moreover, there were only four LEDs used for the display. Hence we had to restrict our program to two one bit inputs and three bits selective instructions.

CHAPTER V: FUTURE ASPECTS

In computing, an Arithmetic Logic Unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The ALU can be inside high speed computers and even supercomputers .It will find its requirement in the field of nano-technology. Commercially it would be very useful in the smart mobile phones and calculating devices. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs. Hence the ALU interfaced with various devices will find its application in almost every electronic device.

APPENDIX A TRUTH TABLES

FOR ORIGINAL CODE

OPCODE	OPERATIONS
00000	A+B
00001	A-B
00010	B-A
00011	A and B
00100	A or B
00101	A xor B
00110	A nand B
00111	A nor B
01000	A xor B
01001	A =B
01010	A <B
01011	A>=B
01100	A<=B
01101	A>=B
01110	A=A+1
01111	A=A-1
10000	B=B+1
10001	B=B-1
10010	A/B
10011	A mod B
10100	A rem B
10101	A ** 2
10110	B ** 2
10111	not A(0)
11000	not B(0)

Table2 – Opcode & Operations (for orginal code)

FOR IMPLEMENTED CODE

OPCODE	OPERATION
000	A+B
001	A-B
010	not A
011	A=B?
100	A=A+1
101	A mod B
110	A ** 2
111	1111

Table3 – Opcode & Operations (for implemented code)

APPENDIX B ORIGINAL & IMPLEMENTED CODE

ORIGINAL CODE

```
-- Company:
-- Engineer:
--
-- Create Date:    16:25:01 03/29/2011
-- Design Name:
-- Module Name:    hi - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use IEEE.NUMERIC_BIT.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity my_alu is
  Port (
        clk:in std_logic;
        A : in  std_logic_vector(7 downto 0);
        B : in  std_logic_vector (7 downto 0);
```

```
        Inst : in std_logic_vector (4 downto 0);
        R : out std_logic_vector(7 downto 0));

end my_alu;

architecture Behavioral of my_alu is

begin
process(a,b,clk)
   variable temp : integer ;
begin
   if clk='1' then
case Inst is
when "00000" => --Add (R=A+B)
temp := conv_integer(A)+ conv_integer(B);
R <= conv_std_logic_vector(temp,8);

when "00001" =>   --Subtract (R=A-B)
temp := conv_integer(A) - conv_integer(B);
R <= conv_std_logic_vector(temp,8);

when "00010" => --Subtract (R=B-A)
temp := conv_integer(B)- conv_integer(A);
R <= conv_std_logic_vector(temp,8);

when "00011" => -- (R=A AND B)
R <=A AND B;
when "00100" => -- (R=A OR B)
R <= A OR B;
when "00101" => -- (R=A XOR B)
R <= A XOR B;
when "00110" => -- (R=A NAND B)
R <= A NAND B;
when "00111" => -- (R=A NOR B)
R <= A NOR B;
when "01000" => -- (R=A XNOR B)
R <= A XNOR B;

when "01001" => -- (R=A equal to B)
```

```
if  A = B
--then temp:= '1';
then R <=conv_std_logic_vector('1' ,8);
else R <=conv_std_logic_vector('0' ,8);
end if;

when "01010" => -- (R=A less than B)
if A < B
--then temp:= '1';
then R <=conv_std_logic_vector('1' ,8);
else R <=conv_std_logic_vector('0' ,8);
end if;

when "01011" => -- (R=A greater than B)
if A > B
then R <=conv_std_logic_vector('1' ,8);
else R <=conv_std_logic_vector('0' ,8);
end if;

when "01100" => -- (R=A less than or equal to B)
if A <= B
then R <=conv_std_logic_vector('1' ,8);
else R <=conv_std_logic_vector('0' ,8);
end if;

when "01101" => -- (R=A greater than or equal to B)
if A >= B
then R <=conv_std_logic_vector('1' ,8);
else R <=conv_std_logic_vector('0' ,8);
end if;

when "01110" => -- (R=A incremented by 1)
temp := (conv_integer(A))+ 1;
R <=conv_std_logic_vector(temp,8);

when "01111" => -- (R=A decremented by 1)
temp := (conv_integer(A))- 1;
R <=conv_std_logic_vector(temp,8);
```

```
when "10000" => -- (R=B incremented by 1)
temp := (conv_integer(B))+ 1;
R <=conv_std_logic_vector(temp,8);

when "10001" => -- (R=B decremented by 1)
temp := (conv_integer(B))- 1;
R <=conv_std_logic_vector(temp,8);

when "10010" => -- (R=A divided by B)
temp := conv_integer(A) / conv_integer(B);
R<= conv_std_logic_vector(temp,8);

when "10011" => -- (R=A modulus to B)
temp := conv_integer(A) mod conv_integer(B);
R<= conv_std_logic_vector(temp,8);

when "10100" => -- (R=A remainder B)
temp := conv_integer(A) rem conv_integer(B);
R<= conv_std_logic_vector(temp,8);

when "10101" => -- (R=exponent of A)
temp := conv_integer(A)**(2);
R<= conv_std_logic_vector(temp, 8);

when "10110" => -- (R=exponent of B)
temp := conv_integer(B)**(2);
R<= conv_std_logic_vector(temp, 8);

when "10111" => -- (R=not of A0)
if (A(0)='0')
then R <=conv_std_logic_vector('1',8);
else R <=conv_std_logic_vector('0',8);
end if;

when "11000" => -- (R=not of B0)
if (B(0)='0')
then R <=conv_std_logic_vector('1',8);
else R <=conv_std_logic_vector('0',8);
end if;
```

```
when others=>
   R<="11111111";

end case;
end if;
end process;
end Behavioral;
```

IMPLEMENTED CODE

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    12:27:44 04/01/2011
-- Design Name:
-- Module Name:    ALUNEW2 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity my_alu is
  Port ( clk:in std_logic;
                A : in  std_logic;
                B : in  std_logic;
                 I : in std_logic_vector (2 downto 0);
                 R : out  std_logic_vector (3 downto 0));
       end my_alu;

architecture Behavioral of my_alu is
begin
process(A, B, clk)
   variable temp : integer ;
begin
   if (clk'event and clk='1' )then
case I is

when "000" => --Add (R=A+B)
temp := conv_integer(A)+ conv_integer(B);
R <= transport conv_std_logic_vector(temp,4) after 10 ns ;

when "001" =>   --Subtract (R=A-B)
temp := conv_integer(A) - conv_integer(B);
R <= conv_std_logic_vector(temp,4);

when "010" => -- (R=not of A0)
if (A='0')
then R <=conv_std_logic_vector('1',4);
else R <=conv_std_logic_vector('0',4);
end if;

when "011" => -- (R=A equal to B)
if  A = B
--then temp:= '1';
then R <=conv_std_logic_vector('1' ,4);
else R <=conv_std_logic_vector('0' ,4);
end if;

when "100" => -- (R=A incremented by 1)
```

```
temp := (conv_integer(A))+ 1;
R <=conv_std_logic_vector(temp,4);

when "101" => -- (R=A modulus to B)
temp := conv_integer(A) mod 10;
R<= conv_std_logic_vector(temp,4);

when "110" => -- (R=remainder of A)
temp := conv_integer(A)rem 10;
R<= conv_std_logic_vector(temp, 4);

when others=>
R<="1111";

end case;
end if;
end process;
end Behavioral;
```

APPENDIX C UCF FILE USED IN PROJECT

```
Net clk LOC=AE14;
Net clk IOSTANDARD = LVCMOS33;
Net clk TNM_NET = clk;
#TIMESPEC TS_clk = PERIOD clk 10000 ps;

Net R<0> LOC=G5;                #led-0(lsb)
Net R<0> IOSTANDARD = LVCMOS25;
Net R<0> PULLUP;
Net R<0> SLEW = SLOW;
Net R<0> DRIVE = 2;
Net R<0> TIG;

Net R<1> LOC=G6;                #led-0(lsb)
Net R<1> IOSTANDARD = LVCMOS25;
Net R<1> PULLUP;
Net R<1> SLEW = SLOW;
Net R<1> DRIVE = 2;
Net R<1> TIG;

Net R<2> LOC=A11;               #led-0(lsb)
Net R<2> IOSTANDARD = LVCMOS25;
Net R<2> PULLUP;
Net R<2> SLEW = SLOW;
Net R<2> DRIVE = 2;
Net R<2> TIG;

Net R<3> LOC=A12;               #led-0(lsb)
Net R<3> IOSTANDARD = LVCMOS25;
Net R<3> PULLUP;
Net R<3> SLEW = SLOW;
Net R<3> DRIVE = 2;
Net R<3> TIG;

# push buttons\par
Net A LOC=E2;   # north
```

```
Net A IOSTANDARD = LVCMOS25;
Net A PULLUP;
Net A SLEW = SLOW;
Net A DRIVE = 2;
Net A TIG;

Net I<0> LOC=E10;   # east
Net I<0> IOSTANDARD = LVCMOS25;
Net I<0> PULLUP;
Net I<0> SLEW = SLOW;
Net I<0> DRIVE = 2;
Net I<0> TIG;

Net I<1> LOC=A5;   # south
Net I<1> IOSTANDARD = LVCMOS25;
Net I<1> PULLUP;
Net I<1> SLEW = SLOW;
Net I<1> DRIVE = 2;
Net I<1> TIG;

Net I<2> LOC=F9;   # west\m<2>
Net I<2> IOSTANDARD = LVCMOS25;
Net I<2> PULLUP;
Net I<2> SLEW = SLOW;
Net I<2> DRIVE = 2;
Net I<2> TIG;

Net B LOC=C6;  #Center
Net B IOSTANDARD = LVCMOS25;
Net B PULLUP;
Net B SLEW = SLOW;
Net B DRIVE = 2;
Net B TIG;
```

APPENDIX D SUCCESSFUL REPORTS

SYNTHESIS REPORT

Reading design: my_alu.prj

```
=========================================================================
*                     HDL Compilation                     *
=========================================================================
Compiling vhdl file "D:/lab1/ALUNEW/ALUNEW2.vhd" in Library work.
Architecture behavioral of Entity my_alu is up to date.
=========================================================================
*                  Design Hierarchy Analysis                  *
=========================================================================
Analyzing hierarchy for entity <my_alu> in library <work> (architecture <behavioral>).
=========================================================================
*                     HDL Analysis                     *
=========================================================================
Analyzing Entity <my_alu> in library <work> (Architecture <behavioral>).
WARNING:Xst:819 - "D:/lab1/ALUNEW/ALUNEW2.vhd" line 43: The following signals are missing in the process sensitivity list:
   I.
Entity <my_alu> analyzed. Unit <my_alu> generated.
=========================================================================
*                     HDL Synthesis                     *
=========================================================================
Performing bidirectional port resolution...
Synthesizing Unit <my_alu>.
    Related source file is "D:/lab1/ALUNEW/ALUNEW2.vhd".
WARNING:Xst:646 - Signal <temp> is assigned but never used.
WARNING:Xst:737 - Found 4-bit latch for signal <R>.
```

```
    Found 1-bit adder carry out for signal <R$addsub0000> created at line 49.
    Found 1-bit adder carry out for signal <R$addsub0001> created at line 75.
    Found 4-bit 8-to-1 multiplexer for signal <R$mux0001> created at line 47.
    Found 1-bit subtractor for signal <R$sub0000> created at line 54.
    Found 1-bit xor2 for signal <R$xor0000> created at line 66.
    Summary:
        inferred   3 Adder/Subtractor(s).
        inferred   4 Multiplexer(s).
Unit <my_alu> synthesized.
=========================================================================
====
HDL Synthesis Report
Macro Statistics
# Adders/Subtractors                   : 3
 1-bit adder carry out                 : 2
 1-bit subtractor                      : 1
# Latches                              : 1
 4-bit latch                           : 1
# Multiplexers                         : 1
 4-bit 8-to-1 multiplexer              : 1
# Xors                                 : 1
 1-bit xor2                            : 1
=========================================================================
====
=========================================================================
====
*                  Advanced HDL Synthesis                  *
=========================================================================
====
Loading device for application Rf_Device from file '4vfx12.nph' in environment C:\Xilinx91i.
=========================================================================
====
Advanced HDL Synthesis Report
Macro Statistics
# Adders/Subtractors                   : 3
 1-bit adder carry out                 : 2
 1-bit subtractor                      : 1
# Latches                              : 1
 4-bit latch                           : 1
```

```
# Multiplexers                          : 1
 4-bit 8-to-1 multiplexer              : 1
# Xors                                  : 1
 1-bit xor2                            : 1
=========================================================================
=========================================================================
*                  Low Level Synthesis                  *
=========================================================================
INFO:Xst:2261 - The FF/Latch <R_2> in Unit <my_alu> is equivalent to the following FF/Latch, which will be removed : <R_3>
Optimizing unit <my_alu> ...
Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block my_alu, actual ratio is 0.
Latch R_2 has been replicated 1 time(s) to handle iob=true attribute.
Final Macro Processing ...
=========================================================================
Final Register Report
Found no macro
=========================================================================
=========================================================================
*                  Partition Report                  *
=========================================================================
Partition Implementation Status
-------------------------------
  No Partitions were found in this design.
-------------------------------
=========================================================================
*                  Final Report                  *
=========================================================================
Clock Information:
```

```
-----------------
---------------------------------+----------------------+-------+
Clock Signal                     | Clock buffer(FF name)  | Load  |
---------------------------------+----------------------+-------+
clk                              | BUFGP                | 4     |
---------------------------------+----------------------+-------+
Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -10
   Minimum period: No path found
   Minimum input arrival time before clock: 2.342ns
   Maximum output required time after clock: 4.790ns
   Maximum combinational path delay: No path found
=========================================================================
====
Process "Synthesize" completed successfully
```

IMPLEMENTATION REPORT

```
NotUpToDate:generated file list is cmd
ngdbuild -ise "D:/lab1/ALUNEW/ALUNEW.ise" -intstyle ise -dd _ngo -nt
timestamp -uc "C:/Documents and Settings/pty/Desktop/final
project/ise.ucf" -p xc4vfx12-ff668-10 "my_alu.ngc" my_alu.ngd is cmd
Command Line: C:\Xilinx91i\bin\nt\ngdbuild.exe -ise D:/lab1/ALUNEW/ALUNEW.ise
-intstyle ise -dd _ngo -nt timestamp -uc C:/Documents and
Settings/pty/Desktop/final project/ise.ucf -p xc4vfx12-ff668-10 my_alu.ngc
my_alu.ngd
Reading NGO file "D:/lab1/ALUNEW/my_alu.ngc" ...
Applying constraints in "C:/Documents and Settings/pty/Desktop/final
project/ise.ucf" to the design...
Checking timing specifications ...
```

```
Checking Partitions ...
Checking expanded design ...


WARNING:NgdBuild:486 - Attribute "SLEW" is not allowed on symbol "A" of type
  "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "DRIVE" is not allowed on symbol "A" of type
  "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "SLEW" is not allowed on symbol "B" of type
  "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "DRIVE" is not allowed on symbol "B" of type
  "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "SLEW" is not allowed on symbol "I<2>.PAD" of
  type "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "DRIVE" is not allowed on symbol "I<2>.PAD" of
  type "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "SLEW" is not allowed on symbol "I<1>" of type
  "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "DRIVE" is not allowed on symbol "I<1>" of type
  "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "SLEW" is not allowed on symbol "I<0>" of type
  "IPAD".  This attribute will be ignored.
WARNING:NgdBuild:486 - Attribute "DRIVE" is not allowed on symbol "I<0>" of type
  "IPAD".  This attribute will be ignored.


Partition Implementation Status
 No Partitions were found in this design.
-------------------------------
NGDBUILD Design Results Summary:
 Number of errors:     0
 Number of warnings:  10
Writing NGD file "my_alu.ngd" ...
Writing NGDBUILD log file "my_alu.bld"...
NGDBUILD done.
Process "Translate" completed successfully
Using target part "4vfx12ff668-10".
Mapping design into LUTs...
Running directed packing...
Running delay-based LUT packing...
```

Running related packing...
Design Summary:
Number of errors: 0
Number of warnings: 6
Logic Utilization: Number of 4 input LUTs: 6 out of 10,944 1%
Logic Distribution:
Number of occupied Slices: 3 out of 5,472 1%
Number of Slices containing only related logic: 3 out of 3 100%
Number of Slices containing unrelated logic: 0 out of 3 0%
*See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs: 6 out of 10,944 1%
Number of bonded IOBs: 10 out of 320 3%
Number of BUFG/BUFGCTRLs: 1 out of 32 3%
Number used as BUFGs: 1
Number used as BUFGCTRLs: 0
Total equivalent gate count for design: 65
Additional JTAG gate count for IOBs: 480
Peak Memory Usage: 211 MB
Total REAL time to MAP completion: 16 secs
Total CPU time to MAP completion: 7 secs

NOTES:
Related logic is defined as being logic that shares connectivity - e.g. two
LUTs are "related" if they share common inputs. When assembling slices,
Map gives priority to combine logic that is related. Doing so results in the best timing performance.
Unrelated logic shares no connectivity. Map will only begin packing unrelated logic into a slice once 99% of the slices are occupied through related logic packing.
Note that once logic distribution reaches the 99% level through related logic packing, this does not mean the device is completely utilized.
Unrelated logic packing will then begin, continuing until all usable LUTs and FFs are occupied. Depending on your timing budget, increased levels of unrelated logic packing may adversely affect the overall timing performance of your design.
Mapping completed.
See MAP report file "my_alu_map.mrp" for details.
Process "Map" completed successfully
Constraints file: my_alu.pcf.
Loading device for application Rf_Device from file '4vfx12.nph' in environment C:\Xilinx91i.
"my_alu" is an NCD, version 3.1, device xc4vfx12, package ff668, speed -10

Initializing temperature to 85.000 Celsius. (default - Range: -40.000
to 100.000 Celsius)
Initializing voltage to 1.140 Volts. (default - Range: 1.140 to 1.260 Volts)
Device speed data version: "PRODUCTION 1.63 2006-11-06".
Device Utilization Summary:
  Number of BUFGs   1 out of 32   3%
  Number of External IOBs   10 out of 320   3%
    Number of LOCed IOBs   10 out of 10   100%
  Number of OLOGICs   4 out of 320   1%
  Number of Slices   3 out of 5472   1%
    Number of SLICEMs   0 out of 2736   0%
Overall effort level (-ol):   Standard
Placer effort level (-pl):   High
Placer cost table entry (-t): 1
Router effort level (-rl):   Standard
Starting initial Timing Analysis. REAL time: 17 secs
Finished initial Timing Analysis. REAL time: 17 secs
Starting Placer
Phase 1.1
Phase 1.1 (Checksum:9896d9) REAL time: 29 secs
Phase 2.7
Phase 2.7 (Checksum:1312cfe) REAL time: 29 secs
Phase 3.31
Phase 3.31 (Checksum:1c9c37d) REAL time: 29 secs
Phase 4.2
Phase 4.2 (Checksum:26259fc) REAL time: 29 secs
Phase 5.30
Phase 5.30 (Checksum:2faf07b) REAL time: 29 secs
Phase 6.3
Phase 6.3 (Checksum:39386fa) REAL time: 29 secs
Phase 7.5
Phase 7.5 (Checksum:42c1d79) REAL time: 29 secs
Phase 8.8
.Phase 8.8 (Checksum:99399f) REAL time: 29 secs
Phase 9.5
Phase 9.5 (Checksum:55d4a77) REAL time: 29 secs
Phase 11.18
Phase 11.18 (Checksum:68e7775) REAL time: 29 secs
Phase 12.27
Phase 12.27 (Checksum:7270df4) REAL time: 29 secs

```
Phase 13.5
Phase 13.5 (Checksum:7bfa473) REAL time: 29 secs
REAL time consumed by placer: 29 secs
CPU  time consumed by placer: 9 secs
Writing design to file my_alu.ncd
Total REAL time to Placer completion: 30 secs
Total CPU time to Placer completion: 9 secs
Starting Router
Phase 1: 56 unrouted;       REAL time: 31 secs
Phase 2: 32 unrouted;       REAL time: 31 secs
Phase 3: 6 unrouted;       REAL time: 31 secs
Phase 4: 6 unrouted; (0)      REAL time: 31 secs
Phase 5: 6 unrouted; (0)      REAL time: 31 secs
Phase 6: 6 unrouted; (0)      REAL time: 31 secs
Phase 7: 0 unrouted; (0)      REAL time: 31 secs
Phase 8: 0 unrouted; (0)      REAL time: 31 secs
Phase 9: 0 unrouted; (0)      REAL time: 31 secs
Total REAL time to Router completion: 31 secs
Total CPU time to Router completion: 9 secs
Partition Implementation Status
-------------------------------
 No Partitions were found in this design.
-------------------------------
Generating "PAR" statistics.

**************************
Generating Clock Report
**************************
+---------------------+--------------+------+------+------------+-------------+
|        Clock Net    |   Resource   |Locked|Fanout|Net Skew(ns)|Max Delay(ns)|
+---------------------+--------------+------+------+------------+-------------+
|          clk_BUFGP | BUFGCTRL_X0Y0| No   |    4 |  0.037     |  2.514      |
+---------------------+--------------+------+------+------------+-------------+
* Net Skew is the difference between the minimum and maximum routing
only delays for the net. Note this is different from Clock Skew which
is reported in TRCE timing report. Clock Skew is the difference between
the minimum and maximum path delays which includes logic delays.
Timing Score: 0
Generating Pad Report.
All signals are completely routed.
```

Total REAL time to PAR completion: 33 secs
Total CPU time to PAR completion: 11 secs
Peak Memory Usage: 236 MB
Placement: Completed - No errors found.
Routing: Completed - No errors found.
Timing: Completed - No errors found.
Number of error messages: 0
Number of warning messages: 0
Number of info messages: 0
Writing design to file my_alu.ncd
PAR done
Process "Place & Route" completed successful
Started : "Generate Post-Place & Route Static Timing".
Loading device for application Rf_Device from file '4vfx12.nph' in environment
C:\Xilinx91i.
  "my_alu" is an NCD, version 3.1, device xc4vfx12, package ff668, speed -10
Analysis completed Sun Apr 03 18:07:17 2011

Generating Report ...
Number of warnings: 0
Total time: 24 secs
Process "Generate Post-Place & Route Static Timing" completed successfully
Started : "Generate Programming File".
WARNING:PhysDesignRules:781 - PULLUP on an active net. PULLUP of comp R<0>/R<0>
  is set but the tri state is not configured.
WARNING:PhysDesignRules:781 - PULLUP on an active net. PULLUP of comp R<1>/R<1>
  is set but the tri state is not configured.
WARNING:PhysDesignRules:781 - PULLUP on an active net. PULLUP of comp R<2>/R<2>
  is set but the tri state is not configured.
WARNING:PhysDesignRules:781 - PULLUP on an active net. PULLUP of comp R<3>/R<3>
  is set but the tri state is not configured.
Process "Generate Programming File" completed successfully

REFERENCES

BOOKS:

1.) Circuit Design with VHDL by Volnei A. Pedroni

2.) VHDL Primer by J. Bhaskar.

3.) The Student's Guide to VHDL (Systems on Silicon) by Peter J Ashenden

4.) VHDL Programming By Example by Douglas L. Perry

PDFS

1.) Xilinx ML 401/402/403 Evaluation Platform User Guide.

2.) ISE 9.1i Quick Start Tutorial.

WEBSITES:

1.) http://www.sm.luth.se/csee/courses/smd/098/lab2.pdf

2.) http://www.vlsibank.com/sessionspage.asp?titl_id=30020

3.) http://cegt201.bradley.edu/projects/proj2006/vhdlmcpr/

4.) http://cegt201.bradley.edu/projects/proj2006/vhdlmcpr/

5.) http://www.xilinx.com/support/sw_manuals/xilinx82/

6.) http://toolbox.xilinx.com/docsan/xilinx8/help/iseguide/iseguide.htm

7.) http://toolbox.xilinx.com/docsan/xilinx8/help/iseguide/html/ise_fpga_design_flow_overview.htm

8.) http://www.ee.iitb.ac.in/vlsi/resources/resource/xilinx8.2_final_copy.pdf