NEAR EAST UNIVERSITY LIBRARY 1988 - LEFKOŞA

NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

Department of Computer Engineering

DUAL PORT RAM

Graduation Project
COM400

Student: Taha Yaseen (20034260)

Supervisor: Mr.Mehmet Ozakman

Nicosia - 2007

ACKNOWLEDMENT

"I am waiting this time since along time, so firstly I would like to thank Allah for giving me the opportunity to be reached my objective.

More over, I want to pay my best regards to my dear father who is responsible for all my expenses and supporting me to educate as much as possible, as well as I thank my dear mother for being in touch with me and praying to god for my succession, I also thank my dear brother and sisters for every thing and I would love to thank my love Handegul Erdogan for supporting me in studying, I will never forget their sacrifices for my education in all my life time.

I would like also to thank all my educators who they are: Okan DONANGIL, Rahib B. ABIYEV, Umit ILHAN, Adnan KHASHMAN and Jemal FATHI... I shall never forget good intention and inexhaustible labor of them.

As well as, I feel proud to pay my best regards to my project advisor who is "Mehmet Kadir OZAKMAN" it was nice time I spent with him because I got much knowledge and I interested while getting the knowledge from him, he never gives up of explaining or teaching something and he is always available when ever I need him while preparing my project.

Absolutely ''NEAR EAST UNIVERSITY'' has a big role of supporting me to educate, it contains various cultures and nationalities and in another hand its beauty, natural view and its strategy are considered which make me glad to be at it.

Eventually, I give my greeting to all persons who supported me and who worked for my project, also I want to thank all my friends especially Mahmoud Masoud for giving me his precious time to complete my project".

ABSTRACT

Today's technology uses high level behavioral languages such as VHDL to do hardware electronic design, I have selected my project in VHDL to learn the current technology and the methods to do hardware design, my project is Dual Port Ram where I can to write in a ram using address A and read from this ram from locations at the same time using address A and address B, the contents of address A shows the output A and the contents of address B shows the output B.

Today's technology we don't go and buy a Dual Port Ram as a device you just write a VHDL for it, (there are other languages like ,Verilog) and the synthesizer generates the design and Implements the design in FPGA which stands for (Field Programmable Gate Array) as I shown in my design description.
As I can from this implementation I did not have to do the detail electronic design all I did is to write a VHDL code to describe Dual Port Ram and the ISE tools did the rest.

Introduction

What is VHDL and what is its connection with brain building?

VHDL stands for (Very High Speed Integrated Circuit) Hardware Description Language.

Before we can answer why VHDL is thought to be highly relevant To the field of brain building, we first need to have some idea of what VHDL is.
An HDL is a high level language (similar to C, Pascal, Fortran, etc) Used to specify the design of electronic circuits.

With modern programmable hardware (i.e. configuring bit strings can be sent into a programmable chip to tell the chip how to wire itself up (i.e. to configure itself).

Modern hardware compilers can take a high level description of an Electronic circuit (e.g. written in an HDL such as VHDL, or Verilog, Or ABEL, etc), and translate it into configuring bit strings, which are Then used to configure a programmable chip (e.g. Xilinx's Virtex chip).

Thanks to Moore's Law, the number of programmable logic gates (e.g. AND gates, NAND gates, etc) in today's chips are now in the millions.

With such electronic capacities on a single chip, it is now possible to Place whole electronic systems on a chip.
This has advantages and disadvantages. The advantage is that Electronics become more sophisticated and powerful and cheaper.

The disadvantage is that electronics becomes harder to design.

Earlier versions of HDLs operated more at the gate level of Description (e.g. "connect the output of gate A to the input of Gate B").

But, as chips increased in their logic gate count, the above rather low Level of description became increasingly impractical due to the huge Number of gates on a single chip.

To cope with this problem, HDLs are taking an ever more behavioral Level of description. Electronic designers nowadays give a behavioral Or functional description of what they want their circuit to perform, And the HDL compiler does the rest.
E.g. instead of saying "connect this gate to that gate", one says

"Multiply these two numbers and store the result in this buffer".

The HDL compiler then translates the latter statement into the Corresponding circuitry that performs the required function.

Nowadays, it is almost as easy to program hardware as to program software!

This is not strictly true, since to be able to use an HDL well, one needs to understand the principles of digital electronic design (e.g. multiplexors, flip-flops, buffers, counters, etc).

But, increasingly, hardware design is becoming more like programming in a high level software language, like "C".

We will have a lot more to say about programming in an HDL.
But we now know enough about the basic idea of an HDL to answer The question of the relevance of HDLs to brain building.

If one wants to build artificial brains with hundreds/thousands and more of evolved neural net circuit modules, then the speed of Evolution of those modules and the speed of the neural signaling of The interconnected brain comprised of those evolved modules, is Paramount.

It is well known that hardware speeds are typically hundreds to Thousands of times faster than software speeds on the same task.

As Moore's law creates chips with millions and later billions of Logic gates on a single chip, it will become increasingly possible to Put artificial brain technology into them.

Today's programmable chips contain about ten million gates (107) This is already enough to start putting tens of modules together to Build simple artificial brains in a single chip.

By placing dozens of chips on an electronic board (not cheap!) Then the size of the brain scales linearly with the number of chips.

People who want to be trained in the principles of brain building Technology therefore needs to know how to put their brain designs Into hardware, so that they can both evolve their component Modules and run them once they are interconnected.

	ACKNOWLEDGMENTS	i
	ABSTRACT	ii
	INTRODUCTION	iii
	TABLE OF CONTENTS	v
1.	**INTRODUCTION TO HDL**	1
	1.2 Hardware Description Languages	1
	1.3Advantages of Using HDLs to Design FPGA Devices	1
	1.3.1 Top-Down Approach for Large Projects	2
	1.3.2 Functional Simulation Early in the Design Flow	2
	1.3.3 Synthesis of HDL Code to Gates	2
	1.3.4 Early Testing of Various Design Implementations	3
	1.3.5 Reuse of RTL Code	3
	1.4 Designing FPGA Devices with HDLs	3
	1.4.1 Designing FPGA Devices with Verilog	4
	1.4.2 Designing FPGA Devices with VHDL	5
	1.4.3 Designing FPGA Devices with Synthesis Tools	5
	1.4.4 Using FPGA System Features	5
	1.4.5 Designing Hierarchy	6
	1.4.6 Specifying Speed Requirements	7
2.	**INTEGRATED SOFTWARE ENVIRONMENT**	8
	2.1 ISE General Information	8
	2.1.1 Xilinx ISE Overview	8
	2.1.2 Design Entry	8
	2.1.3 Synthesis	8
	2.1.4 Implementation	8
	2.1.5 verification	9
	2.1.6 Device Configuration	9
	2.2 The Project Navigator	9
	2.2.1 Project Navigator Overview	9
	2.2.2 Project Navigator Main Window	10
	2.2.3 Using the Sources Window	12
	2.2.4 Using the Processes Window	13
	2.2.5 Process Types	14
	2.2.6 Process Status	14
	2.2.7 Running Processes	15
	2.2.8 Setting Process Properties	17
	2.2.9 Using the Workspace	18
	2.2.10 Using the Transcript Window	19
	2.2.11 Using the Toolbars	19
	2.3 Creating a Project	20
	2.3.1 to Create a Project	20
	2.3.2 What to Expect	27
	2.4 working with projects source file	27
	2.4.1 Creating a Source File	27
	2.4.2 Adding a Source File to a Project	28
	2.4.3 Adding a Copy of a Source File to a Project	29
	2.4.4 Editing a Source File	30

	2.4.5 Removing Files from a Project	31
	2.5 Running and Stopping Processes	31
3.	**My project (dual port ram)**	34
	3.1 Overview	34
	3.2 Design process is shown below:	34
	3.2.1 Requirement	35
	3.2.2 Specification	35
	3.2.3 The inputs and outputs of dual port ram	36
	3.2.4 The function of VHDL code	37
	3.2.4.1 Process (clock write)	37
	3.2.4.2 Process (clock read)	37
	3.2.5 VHDL code	37
	3.3 Creating My project	40
	3.3.1 How to Create my project	40
	3.3.2 Creating an HDL Source	42
	3.3.2.1Creating a VHDL Source	43
	3.3.2.2 Using Language Templates (VHDL)	46
	3.3.2.3 Final Editing of VHDL Source	46
	3.4 Synthesize	51
	3.4.1 View synthesize report	51
	3.4.2 View RTL Schematic	64
	3.4.3 check syntax	65
	3.5 Writing the test bech	65
	3.5.1 Verifying Functionality using Behavioral Simulation	65
	3.5.2 Final Editing of My VHDL Source	67
	3.5.3 The simulation	74
	3.6 Implementing the design	77
	CONCLUSION	79
	REFERENCES	80

1. INTRODUCTION TO HDL

1.1 Overview

This chapter provides a general overview of designing Field Programmable Gate Arrays
(FPGA devices) with Hardware Description Languages (HDLs). This chapter includes the following sections.

- Hardware Description Languages
- Advantages of Using HDLs to Design FPGA Devices
- Designing FPGA Devices with HDLs

1.2 Hardware Description Languages

Designers use Hardware Description Languages (HDLs) to describe the behavior and Structure of system and circuit designs. This chapter includes:

- A general overview of designing FPGA devices with HDLs
- System requirements and installation instructions for designs available from the web
- A brief description of why FPGA devices are superior to ASIC devices for your design needs

Understanding FPGA architecture allows you to create HDL code that effectively uses
FPGA system features. To learn more about designing FPGA devices with HDL:

- Enroll in training classes offered by Xilinx® and by the vendors of synthesis software.
- Review the sample HDL designs in the later chapters of this Guide.
- Download design examples from Xilinx Support.
- Take advantage of the many other resources offered by Xilinx, including documentation, tutorials, Tech Tips, service packs, a telephone hotline, and an answers database. See "Additional Resources" in the Preface of this Guide.

1.3 Advantages of Using HDLs to Design FPGA Devices

Using HDLs to design high-density FPGA devices has the following advantages:

- . "Top-Down Approach for Large Projects.
- Functional Simulation Early in the Design Flow.
- Synthesis of HDL Code to Gates.
- Early Testing of Various Design Implementations.
- Reuse of RTL Code.

1.3.1 Top-Down Approach for Large Projects

Designers use HDLs to create complex designs. The top-down approach to system design supported by HDLs is advantageous for large projects that require many designers working together. After they determine the overall design plan, designers can work independently on separate sections of the code.

1.3.2 Functional Simulation Early in the Design Flow

You can verify the functionality of your design early in the design flow by simulating the HDL description. Testing your design decisions before the design is implemented at the RTL or gate level allows you to make any necessary changes early in the design process.

1.3.3 Synthesis of HDL Code to Gates

You can synthesize your hardware description to target the FPGA implementation. This step:

- Decreases design time by allowing a higher-level specification of the design rather than specifying the design from the FPGA base elements.
- Generally reduces the number of errors that can occur during a manual translation of a hardware description to a schematic design.
- Allows you to apply the automation techniques used by the synthesis tool (such as machine encoding styles and automatic I/O insertion) during the optimization of your design to the original HDL code. This results in greater optimization and efficiency.

1.3.4 Early Testing of Various Design Implementations

HDLs allow you to test different implementations of your design early in the design flow. Use the synthesis tool to perform the logic synthesis and optimization into gates.

Additionally, Xilinx FPGA devices allow you to implement your design at your computer, since the synthesis time is short; you have more time to explore different architectural possibilities at the Register Transfer Level (RTL). You can reprogram Xilinx FPGA devices to test several implementations of your design.

1.3.5 Reuse of RTL Code

You can retarget RTL code to new FPGA architectures with a minimum of recoding.

1.4 Designing FPGA Devices with HDLs

If you are used to schematic design entry, you may find it difficult at first to create HDL designs. You must make the transition from graphical concepts, such as block diagrams, state machines, flow diagrams, and truth tables, to abstract representations of design components. Ease this transition by not losing sight of your overall design plan as you code in HDL.

To effectively use an HDL, you must understand the:

- Syntax of the language
- Synthesis and simulator software
- Architecture of your target device
- Implementation tools

This section gives you some design hints to help you create FPGA devices with HDLs.

1.4.1 Designing FPGA Devices with Verilog

Verilog is popular for synthesis designs because:

- Verilog is less verbose than traditional VHDL.
- Verilog is standardized as IEEE-STD-1364-95 and IEEE-STD-1364-2001.

Since Verilog was not originally intended as an input to synthesis, many Verilog constructs are not supported by synthesis software. The Verilog coding examples in this Guide were tested and synthesized with current, commonly-used FPGA synthesis software. The coding strategies presented in the remaining chapters of this Guide can help you create HDL descriptions that can be synthesized.

System Verilog is a new emerging standard for both synthesis and simulation. It is currently unknown if, or when, this standard will be adopted and supported by the various design tools.

Whether or not you plan to use this new standard, Xilinx recommends that you:

- Review the standard to make sure that your current Verilog code can be readily carried forward as the new standard evolves.
- Review any new keywords specified by the standard.
- Avoid using the new keywords in your current Verilog code.

1.4.2 Designing FPGA Devices with VHDL

VHSIC Hardware Description Language (VHDL) is a hardware description language for designing Integrated Circuits (ICs). It was not originally intended as an input to synthesis, and many VHDL constructs are not supported by synthesis software. However, the high level of abstraction of VHDL makes it easy to describe the system-level components and test benches that are not synthesized. In addition, the various synthesis tools use different subsets of the VHDL language. The examples in this Guide work with most commonly used FPGA synthesis software. The coding strategies presented in the remaining chapters of this Guide can help you create HDL descriptions that can be synthesized.

1.4.3 Designing FPGA Devices with Synthesis Tools

Most of the commonly-used FPGA synthesis tools have special optimization algorithms for Xilinx FPGA devices. Constraints and compiling options perform differently depending on the target device. Some commands and constraints in ASIC synthesis tools do not apply to FPGA devices. If you use them, they may adversely impact your results.

You should understand how your synthesis tool processes designs before you create FPGA designs. Most FPGA synthesis vendors include information in their guides specifically for Xilinx FPGA devices.

1.4.4 Using FPGA System Features

To improve device performance, area utilization, and power characteristics, creates HDL code that uses such FPGA system features as DCM, multipliers, shift registers, and memory. For a description of these and other features, see the FPGA data sheet and userguide. The choice of the size (width and depth) and functional characteristics need to be taken into account by understanding the target FPGA

resources and making the proper system choices to best target the underlying architecture.

1.4.5 Designing Hierarchy

HDLs give added flexibility in describing the design. However, not all HDL code is optimized the same. How and where the functionality is described can have dramatic effects on end optimization. For example:

- Certain techniques may unnecessarily increase the design size and power while decreasing performance.
- Other techniques can result in more optimal designs in terms of any or all of those same metrics.

This Guide will help instruct you in techniques for optional FPGA design methodologies.

Design hierarchy is important in both the implementation of an FPGA and during interactive changes. Some synthesizers maintain the hierarchical boundaries unless you group modules together. Modules should have registered outputs so their boundaries are not an impediment to optimization. Otherwise, modules should be as large as possible within the limitations of your synthesis tool.

The "5,000 gates per module" rule is no longer valid, and can interfere with optimization. Check with your synthesis vendor for the preferred module size. As a last resort, use the grouping commands of your synthesizer, if available.The size and content of the modules influence synthesis results and design implementation.

This Guide describes how to create effective design hierarchy.

1.4.6 Specifying Speed Requirements

To meet timing requirements, you should understand how to set timing constraints in both the synthesis tool and the placement and routing tool. If you specify the desired timing at the beginning, the tools can maximize not only performance, but also area, power, and tool runtime. This generally results in a design that better matches the desired performance. It may also result in a design that is smaller, and which consumes less power and requires less time processing in the tools

2. INTEGRATED SOFTWARE ENVIRONMENT

2.1 ISE General Information

2.1.1 Xilinx ISE Overview

The Integrated Software Environment (ISE™) is the Xilinx® design software suite that allows you to take your design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes your design through the following steps in the ISE design flow.

2.1.2 Design Entry

Design entry is the first step in the ISE design flow. During design entry, you create your source files based on your design objectives.

You can create your top-level design file using a Hardware Description Language (HDL), such as VHDL, Verilog, or ABEL, or using a schematic. You can use multiple formats for the lower-level source files in your design.

Note If you are working with a synthesized EDIF or NGC/NGO file, you can skip design entry and synthesis and start with the implementation process.

2.1.3 Synthesis

After design entry and optional simulation, you run synthesis. During this step, VHDL, Verilog, or mixed language designs become netlist files that are accepted as input to the implementation step.

2.1.4 Implementation

After synthesis, you run design implementation, which converts the logical design into a physical file format that can be downloaded to the selected target device. From Project Navigator, you can run the implementation process in one step, or you can run each of the implementation processes separately. Implementation processes vary depending on whether you are targeting a Field Programmable Gate Array (FPGA) or a Complex Programmable Logic Device (CPLD).

2.1.5 Verification

You can verify the functionality of your design at several points in the design flow. You can use simulator software to verify the functionality and timing of your design or a portion of your design.

The simulator interprets VHDL or Verilog code into circuit functionality and displays logical results of the described HDL to determine correct circuit operation. Simulation allows you to create and verify complex functions in a relatively small amount of time. You can also run in-circuit verification after programming your device.

2.1.6 Device Configuration

After generating a programming file, you configure your device. During configuration, you generate configuration files and download the programming files from a host computer to a Xilinx device.

2.2 The Project Navigator

2.2.1 Project Navigator Overview

Project Navigator organizes your design files and runs processes to move the design from design entry through implementation to programming the targeted Xilinx® device. Project Navigator is the high-level manager for your Xilinx FPGA and CPLD designs, which allows you to do the following:

- Add and create design source files, which appear in the Sources window
- Modify your source files in the Workspace
- Run processes on your source files in the Processes window
- View output from the processes in the Transcript window

Note optionally, you can run processes from a script you create or from a command line prompt. However, it is recommended that you first become familiar with the basic use of the Xilinx Integrated Software Environment (ISE™) software and with project management, as described in the following sections.

2.2.2 Project Navigator Main Window

The following figure 2.1 shows the Project Navigator main window, which allows you to manage your design starting with design entry through device configuration.

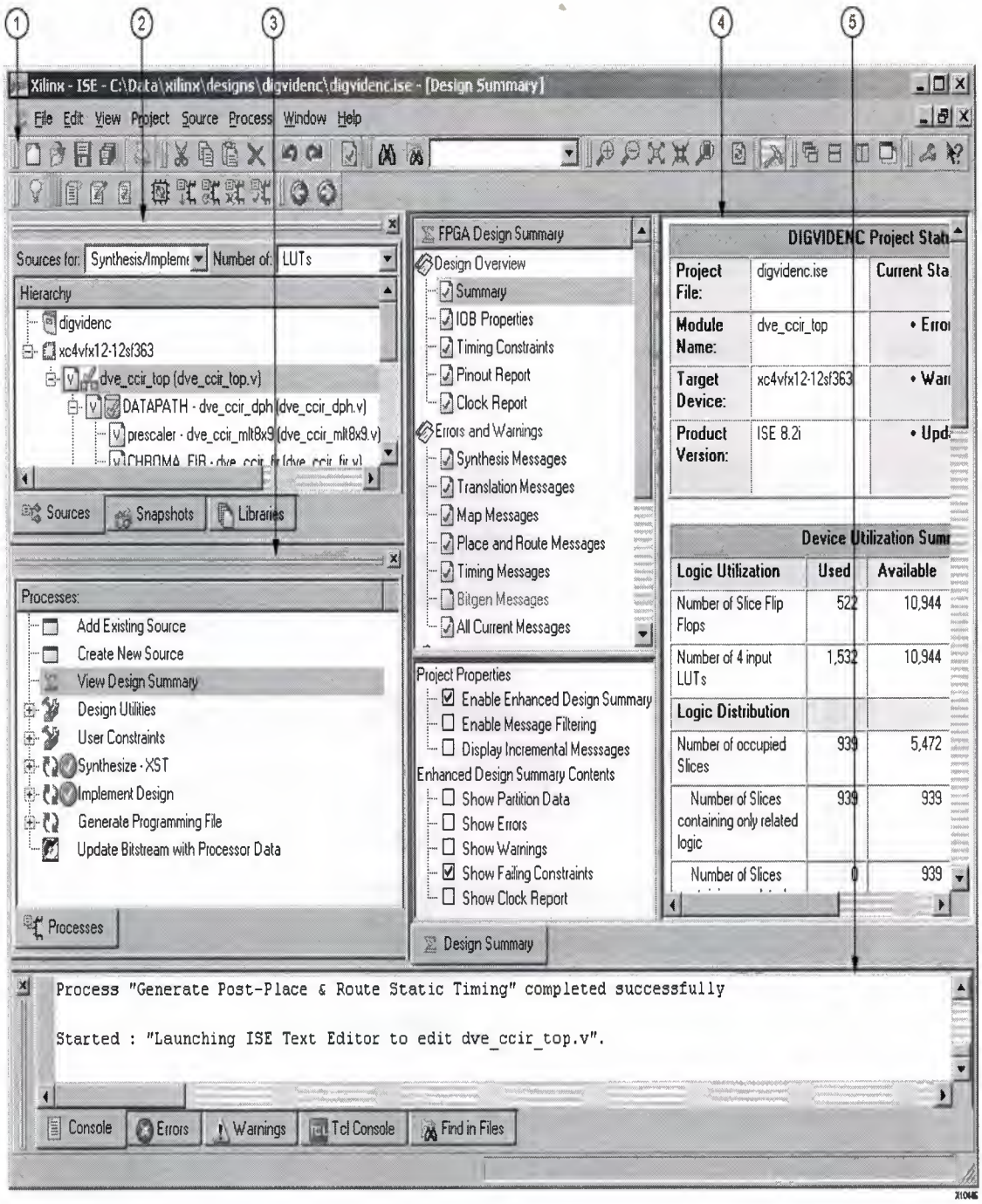

Figure 2.2.2 project window

1. Toolbar
2. Sources window
3. Processes window
4. Workspace

5. Transcript window

2.2.3 Using the Sources Window

The first step in implementing your design for a Xilinx® FPGA or CPLD is to assemble the design source files into a project. The Sources tab in the Sources window shows the source files you create and add to your project, as shown in figure2.2.

For information on creating projects and source files, see Creating a Project and Creating a Source File.

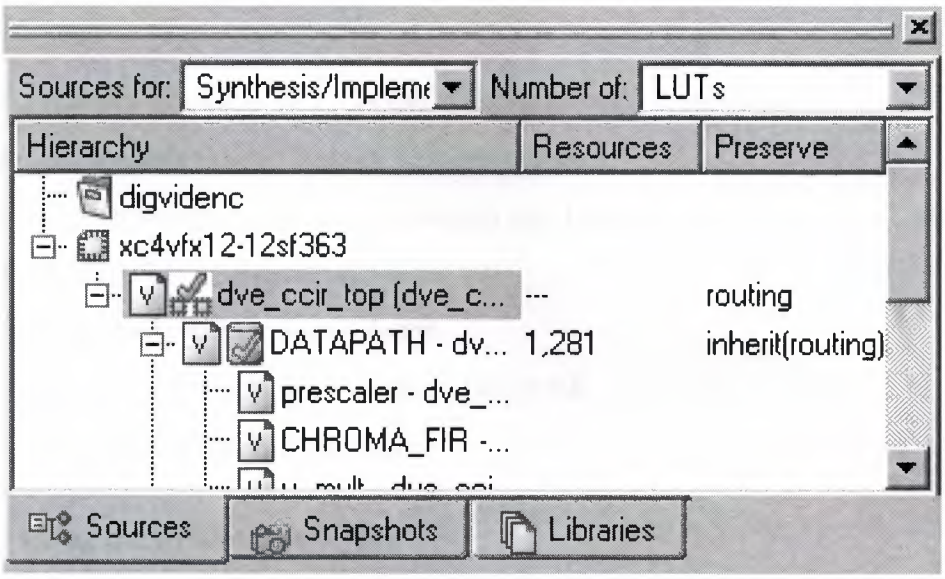

Figure 2.2 source window

The Design View ("Sources for") drop-down list at the top of the Sources tab allows you to view only those source files associated with the selected Design View (for example, Synthesis/Implementation).

For details, see using the Design Views. The "Number of" drop-down list, Resources column, and Preserve column are available for designs that use Partitions. For details, see Using Partitions.

The Sources tab shows the hierarchy of your design. You can collapse and expand the levels by clicking the plus (+) or minus (-) icons. Each source file appears next to an icon that shows its file type. The file you select determines the

processes available in the Processes window. You can double-click a source file to open it for editing in the Workspace. For information on the different file types, see Source File Types.

You can change the project properties, such as the device family to target, the top-level module type, the synthesis tool, the simulator, and the generated simulation language. For information, see Changing Project, Source, and Snapshot Properties.

Depending on the source file and tool you are working with, additional tabs are available in the Sources window:

- Always available: Sources tab, Snapshots tab, Libraries tab
- Constraints Editor: Timing Constraints tab
- Floorplan Editor: Translated Netlist tab, Implemented Objects tab
- iMPACT: Configuration Modes tab
- Schematic Editor: Symbols tab
- RTL and Technology Viewers: Design tab
- Timing Analyzer: Timing tab

2.2.4 Using the Processes Window

The Processes tab in the Processes window allows you to run actions or "processes" on the source file you select in the Sources tab of the Sources window. The processes change according to the source file you select.

The Process tab shows the available processes in a hierarchical view. You can collapse and expand the levels by clicking the plus (+) or minus (-) icons. Processes are arranged in the order of a typical design flow: project creation, design entry, constraints management, synthesis, implementation, and programming file creation.

Depending on the source file and tool you are working with, additional tabs are available in the Processes window:

- Always available: Processes tab

- Floorplan Editor: Design Objects tab, Implemented - Selection tab
- iMPACT: Configuration Operations tab
- ISE Simulator: Hierarchy Browser tab
- Schematic Editor: Options tab
- Timing Analyzer: Timing Objects tabz

2.2.5 Process Types

The following types of processes are available as you work on your design:

- Tasks

When you run a task process, the ISE software runs in "batch mode," that is, the software processes your source file but does not open any additional software tools in the Workspace. Output from the processes appears in the Transcript window.

- Reports

Most tasks include report sub-processes, which generate a summary or status report, for example, the Synthesis Report or Map Report. When you run a report process, the report appears in the Workspace.

- Tools

When you run a tools process, the related tool launches in standalone mode or appears in the Workspace where you can view or modify your design source files.

Note The icons for tools processes vary depending on the tool. For example, the Timing Analyzer icon is shown above.

2.2.6 Process Status

As you work on your design, you may make changes that require some or all of the processes to be rerun. For example, if you edit a source file, it may require that the Synthesis process and all subsequent process be rerun. Project Navigator

keeps track of the changes you make and shows the status of each process with the following status icons:

- Running

This icon shows that the process is running.

- Up-to-date

This icon shows that the process ran successfully with no errors or warnings and does not need to be rerun. If the icon is next to a report process, the report is up-to-date; however, associated tasks may have warnings or errors. If this occurs, you can read the report to determine the cause of the warnings or errors.

- Warnings reported

This icon shows that the process ran successfully but that warnings were encountered.

- Errors reported

This icon shows that the process ran but encountered an error.

- Out-of-Date

This icon shows that you made design changes, which require that the process be rerun. If this icon is next to a report process, you can rerun the associated task process to create an up-to-date version of the report.

- No icon

If there is no icon, this shows that the process was never run.

2.2.7 Running Processes

To run a process, you can do any of the following:

- Double-click the process

- Right-click while positioned over the process, and select **Run** from the popup menu, as shown in figure 2.3

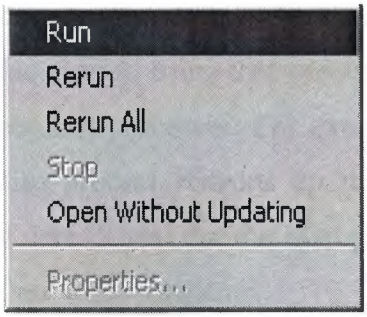

Figure 2.2.7 Run process.

- Select the process, and then click the Run toolbar button .
- To run the Implement Design process and all preceding processes on the top module for the design, select Process > Implement Top Module, or click the Implement Top Module toolbar button .

When you run a process, Project Navigator automatically processes your design as follows:

- Automatically runs lower-level processes

When you run a high-level process, Project Navigator runs associated lower-level processes or sub-processes. For example, if you run Implement Design for your FPGA design, all of the following sub-processes run: Translate Map, and Place & Route.

- Automatically runs preceding processes

When you run a process, Project Navigator runs any preceding processes that are required, thereby "pulling" your design through the design flow. For

example, to pull your design through the entire flow, double-click Generate Programming File.

- Automatically runs related processes for out-of-date processes

  If you run an out-of-date process, Project Navigator runs that process and any related processes required to bring that process up to date. It does not necessarily run all preceding processes. For example if you change your UCF file, the Synthesize process remains up to date, but the Translate process becomes out of date. If you run the Map process, Project Navigator runs Translate but does not run Synthesize.

Note For more information on running processes, including additional Process menu commands, see running and Stopping Processes.

2.2.8 Setting Process Properties

Most processes have a set of properties associated with them. Properties control specific options, which correspond to command line options. When properties are available for a process, you can right-click while positioned over the process and select Properties from the popup menu, as shown in the following figure 2.2.8

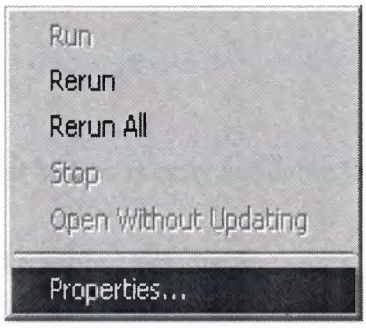

Figure2.2.8 popup menu for a process

When you select Properties, a Process Properties dialog box appears, with standard properties that you can set. The Process Properties dialog box differs depending on the process you select.

After you become familiar with the standard properties, you can set additional, advanced properties in the Process Properties dialog box; however, setting these options is not recommended if you are just getting started with using the ISE software. When you enable the advanced properties, both standard and advanced properties appear in the Process Properties dialog box. More Info

Note For more information on process properties, see Setting Process Properties. To set command line options using process properties, see Setting Command Line Options using Process Properties.

2.2.9 Using the Workspace

When you open a project source file, open the Language Templates, or run certain processes, such as viewing reports or logs, the corresponding file or view appears in the Workspace. You can open multiple files or views at one time. Tabs at the bottom of the Workspace show the names for each file or view. Click a tab to bring it to the front.

To open a file or view in a standalone window outside of the Project Navigator Workspace, use the Float toolbar button. To dock a floating window, use the Dock toolbar button.

- Float
- Dock

Note The Dock toolbar button is only available from the floating window.

For more information, see Arranging Windows.

2.2.10 Using the Transcript Window

The Console tab of the Transcript window shows output messages from the processes you run. When the following icons appear next to a message, you can right-click the message and select Goto Answer Record to open the Xilinx website and show any related Answer Records. If a line number appears as part of the message, you can right-click the message and select Goto Source to open the source file with the appropriate line number highlighted.

- Warning
- Error

Depending on the source file and tool you are working with, additional tabs are available in the Transcript window:

- Always available: Console tab, Errors tab, Warnings tab, Tcl Shell tab, Find in Files tab

  Note The Errors and Warnings tabs show a filtered version of the output messages in the Console tab, that is, only the errors and only the warnings are shown.

- ISE Simulator: Simulation Console tab
- RTL and Technology Viewers: View by Name tab, View by Category tab

2.2.11 Using the Toolbars

Toolbars provide convenient access to frequently used commands. Click once on a toolbar button to execute a command. To see a short popup description of a toolbar button, hold the mouse pointer over the button for about two seconds. A longer description appears in the status bar at the bottom of the main window.

For Help on a toolbar button, click the Help toolbar button, and then click the toolbar button for which you want Help. For more information on getting Help, see Using Xilinx Help.

2.3 Creating a Project

Project Navigator allows you to manage your FPGA and CPLD designs using an ISE™ project, which contains all the files related to your design. First, you must create a project and then add source files. With your project open in Project Navigator, you can view and run processes on all the files in your design. Project Navigator provides a wizard to help you create a new project, as follows.

2.3.1 to Create a Project

1. Select **File > New Project**.
2. In the New Project Wizard Create New Project page, do the following:
    a. In the Project Name field, enter a name for the project. Follow the naming conventions described in File Naming Conventions.
    b. In the Project Location field, enter the directory name or browse to the directory.
    c. In the Top-Level Source Type drop-down list, select one of the following:

        - **HDL**

        Select this option if your top-level design file is a VHDL, Verilog, or ABEL (for CPLDs) 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, as described in step 5.

- **EDIF**

Select this option if you converted your design to this file type, for example, using a synthesis tool. Using this file type allows you to skip the Project Navigator synthesis process and to start with the implementation processes.

- **NGC/NGO**

Select this option if you converted your design to this file type, for example, using a synthesis tool. Using this file type allows you to skip the Project Navigator synthesis process and start with the implementation processes.

3. Click **Next**.
4. If you are creating an HDL or schematic project, skip to the next step. If you are creating an EDIF or NGC/NGO project, do the following in the Import EDIF/NGC Project page:
    a. In the Input Design field, enter the name of the input design file, or browse to the file and select it.

b. Select **Copy the input design to the project directory** to copy your file to the project directory. If you do not select this option, your file is accessed from the remote location.

c. In the Constraint File field, enter the name of the constraints file, or browse to the file and select it.

d. Select **Copy the constraints file to the project directory** to copy your file to the project directory. If you do not select this option, your file is accessed from the remote location.

e. Click **Next**.

5. In the Device Properties page, set the following options. These settings affect other project options, such as the types of processes that are available for your design.

- Product Category
- Family

Note To target a Spartan-3L™ device, select Spartan-3™ as the family. When creating an EDIF project, the device family information is read from your EDIF project file, and changing the device family is not recommended.

- Device

Note To target a Spartan-3L device, select a device that ends in l, such as xc3s2000l.

- Package
- Speed
- Top-Level Source Type

Note This is automatically set.

- Synthesis Tool

Select one of the following synthesis tools and the HDL language for your project. VHDL/Verilog is a mixed language flow. If you plan to run behavioral simulation, your simulator must support multiple language simulation.

1. **XST** (Xilinx® Synthesis Technology)

   XST is available with ISE Foundation™ software installations. It 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. For more information, see XST Synthesis Overview.

- **Synplify** and **Synplify Pro** (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. For more information, see Using Synplify or Synplify Pro for Synthesis.

- **Precision** (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. For more information, see Using Precision for Synthesis.

Note When creating an EDIF or NGC/NGO project, this option is not applicable. A partner synthesis tool is only available as an option if the software was installed on your computer. If a synthesis tool was installed, but it does not appear as an option, set the path to the

synthesis software in the Integrated Tools Options page of the Preferences dialog box. How?

- Simulator

Select one of the following simulators and the HDL language for simulation.

1. **ISE Simulator** (Xilinx®, Inc.)

   This simulator allows you to run integrated simulation processes as part of your ISE design flow. For more information, see the ISE Simulator Help.

- **ModelSim** (Mentor Graphics®, Inc.)

  You can run integrated simulation processes as part of your ISE design flow using any of the following ModelSim® editions: ModelSim Xilinx Edition (MXE), ModelSim MXE Starter, ModelSim PE, or ModelSim SE™. For more information on ModelSim, including the differences between each edition, see Using the ModelSim Simulator.

- **NC-Sim** (Cadence®, Inc.)

  The NC-Sim simulator is not integrated with ISE and must be run standalone. For more information, see the documentation provided with the simulator.

- **VCS** (Synopsys®, Inc.)

  The VCS® simulator is not integrated with ISE and must be run standalone. For more information, see the documentation provided with the simulator.

- **Other**

  Select this option if you do not have ISE Simulator or ModelSim installed or if you want to run simulation outside of Project Navigator.

  This instructs Project Navigator to disable the integrated simulation processes for your project.

Note If you are using a simulator that is not integrated with ISE, you must still specify your simulator. This ensures that all generated files are written in the correct format.

- Preferred Language

Select one of the following to set your preferred language. The Preferred Language project property controls the default setting for process properties that generate HDL output. If the Synthesis Tool and/or Simulator options are set to a single-language tool, the default language for generated HDL output files will be automatically chosen appropriately. 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 will be created.

Note You can also select the language in which to generate files by setting process properties as described in Setting Process Properties.

- **Verilog**

  Select this option if both Synthesis Tool and Simulation are set to mixed-language and you want the default language to be Verilog.

- **VHDL**

Select this option if both Synthesis Tool and Simulation are set to mixed-language and you want the default language to be VHDL.

- **N/A**

  This option will appear if both Synthesis Tool and Simulation are set to a single language.

- **Enable Enhanced Design Summary**

Select this option to show the number of errors and warnings for each of the Detailed Reports in the Design Summary. For information, see Using the Design Summary for FPGAs.

- **Enable Message Filtering**

Select this option to show the number of messages you filtered in the Design Summary. You must enable this option, filter messages, and then run the software to show the number of filtered messages.

- **Display Incremental Messages**

Select this option to show the number of new messages for the most recent software run in the Design Summary. You must enable this option and then run the software to show the number of new messages.

6. If you are creating an EDIF or NGC/NGO project, skip to step 8. If you are creating an HDL or schematic project, click **Next**, and optionally, create a new source file for your project in the Create New Source page.

Note You can only create one new source file while creating a new project. You can create additional new sources after your project is created. How?

7. Click **Next**, and optionally, add existing source files to your project in the Add Existing Sources page.
8. Click **Next** to display the Project Summary page.
9. Click **Finish** to create the project.

Note If you prefer, you can create a project using the New Project dialog box instead of the New Project Wizard, as described above. To use the New Project dialog box, deselect the **Use new project wizard** option in the ISE General Options page of the Preferences dialog box. How?

2.3.2 What to Expect

Project Navigator creates the project file, project_name.ise, in the directory you specified. All source files related to the project appear in the Project Navigator Sources tab. Project Navigator manages your project based on the project properties (top-level module type, device type, synthesis tool, and language) you selected when you created the project. It organizes all the parts of your design and keeps track of the processes necessary to move the design from design entry through implementation to programming the targeted Xilinx device.

2.4 working with projects source file

2.4.1 Creating a Source File

A source file is any file that contains information about a design. Project Navigator provides a wizard to help you create new source files for your project.

- **What to Do First**

Open a project in Project Navigator. How?

- **To Create a Source File**

1. Select **Project > New Source**.

Note Alternatively, you can double-click **Create New Source** in the Processes tab.

2. In the New Source Wizard, select the type of source you want to create.

Different source types are available depending on your project properties (top-level module type, device type, synthesis tool, and language). Some source types launch additional tools to help you create the file, as described in Source File Types.

3. Enter a name for the new source file in the File Name field. Follow the naming conventions described in File Naming Conventions.

4. In the Location field, enter the directory name or browse to the directory.
5. Select **Add to Project** to automatically add this source to the project.

6. Click **Next**.
7. If you are creating a source file that needs to be associated with an existing source file, select the appropriate source file, and click **Next**. If this does not apply, skip to the next step.
8. In the New Source Information window, read the summary information for the new source, and click **Finish**.

2.4.2 Adding a Source File to a Project

Project Navigator allows you to add an existing source file to a project. The source file can reside in the project directory or in a remote directory. If you generated your source file using the New Source wizard and selected **Add to Project**, you do not need to add the source file to your project; it is automatically part of your project.

Note If you want to copy a source file from a remote directory to your project directory and add it to your project, use the Add Copy of Source command instead, as described in Adding a Copy of a Source File to a Project. If you are working with CORE Generator™ or Architecture Wizard IP, use the Add Copy of

Source command to copy the IP core and associated files that reside in a remote directory to your local project directory. The files will not simulate or implement correctly if you add them as remote source files.

- **What to Do First**

Open a project in Project Navigator. How?

- **To Add a Source File to a Project**

1. Select **Project > Add Source**.

   Note Alternatively, you can double-click **Add Existing Source** in the Processes tab.

2. In the Add Existing Sources dialog box, browse to the source file and select it.
3. Click **Open**.
4. In the Adding Source Files dialog box, select the Design View in which you want the source file to appear. More Info

   Note If you want to change the Design View association after the source file has been added, select the source file in the Sources tab, and then select **Source > Properties**.

5. Click **OK**.

2.4.3 Adding a Copy of a Source File to a Project

Project Navigator allows you to copy a source file from a remote directory to your project directory and then, add it to your project as follows.

Note If you want to leave the source file in the remote directory and add it to your project, see Adding a Source File to a Project.

- **What to Do First**

Open a project in Project Navigator. How?

- **To Add a Copy of a Source File to a Project**

1. Select **Project > Add Copy of Source**.
2. In the Add Existing Sources dialog box, browse to the source file and select it.
3. Click **Open**.
4. In the Adding Source Files dialog box, select the Design View in which you want the source file to appear. More Info

   Note If you want to change the Design View association after the source file has been added, select the source file in the Sources tab, and then select **Source > Properties**.

5. Click **OK**.

2.4.4 Editing a Source File

After you create a source file, you can edit it using the ISE™ software.

- **What to Do First**

Open a project in Project Navigator. How?

- **To Edit a Source File**

1. In the Sources tab, select a Design View from the drop-down list. More Info
2. Double-click the source file.
3. Edit the file in the tool that appears.

Each source type launches a different tool to help you edit the file, as described in Source File Types. See the Help provided with each tool for detailed information.

2.4.5 Removing Files from a Project

You can remove files from a project that you no longer need. The file is removed from the project, but is not deleted from your disk.

Caution! When you remove snapshots, the snapshot directory is deleted from the disk. For details, see Working with Snapshots.

- **What to Do First**

Open a project in Project Navigator. How?

- **To Remove a Source File from a Project**

1. In the Sources tab, select a Design View from the drop-down list. More Info
2. Select the file to remove.
3. Select **Source > Remove**, or press the **Delete** key on the keyboard.
4. Click **Yes** to remove the file from your project.

2.5 Running and Stopping Processes

In the Processes tab, you can run processes on your selected source file. You can run a task, generate a report, or launch a tool. You can also stop a process while it is running.

- **What to Do First**

Open a project in Project Navigator. How?

- **To Run a Process**

1. In the Sources tab, select a Design View from the drop-down list. More Info
2. Select the source file to process.

Note the source file you select affects the processes that appear in the Processes tab; only the processes that apply to the selected source are shown.

3. In the Processes tab, select a process.
4. From the Process menu, select one of the following commands:

    - Run to run the selected process and any preceding processes that are out of date.

Note alternatively, you can double-click the process to run it.

- Rerun to force a run on the selected up-to-date process.

- Rerun All to force a run on the selected up-to-date process and all processes that precede the selected process.

- Open without Updating to open a file for an out-of-date task or to open an out-of-date report for investigative purposes.

Note you can also right-click a process and select one of these commands.

- To Stop a Process

To stop the currently running process, select Process > Stop.

Note Stopping a process is not always immediate; some processes may proceed until a suitable stopping point is reached.

- **What to Expect**

One of the following status icons appears next to the process in the Processes tab:

- Running

This icon shows that the process is running.

- Up-to-date

This icon shows that the process ran successfully with no errors or warnings and does not need to be rerun.

- Warnings reported

This icon shows that the process ran successfully but that warnings were encountered.

- Errors reported

This icon shows that the process ran but encountered an error.

- Out-of-Date

This icon shows that you made design changes, which require that the process be rerun. If this icon is next to a report process, you can rerun the process to create an up-to-date version of the report.

- No icon

If there is no icon, this shows that the process was never run.

3. My project (dual port ram)

3.1 Overview

In this chapter explains the design process, the design steps and how to implement them.

3.2 Design process is shown below:

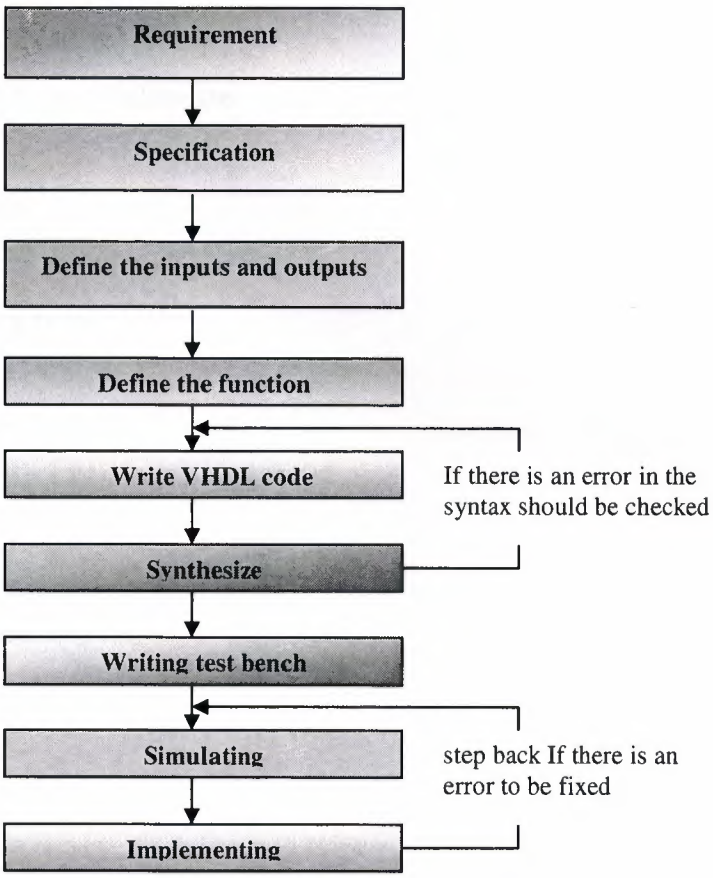

Figure3.2 design process

3.2.1 Requirement

- Dual port ram

3.2.2 Specification

- 512*64
- 512 locations
- 64 bits in each location
- The output has to be synchronize.
- Clock
- Read/write port
- Read port
- Clock frequency is 100 MHz

3.2.3 The inputs and outputs of dual port ram

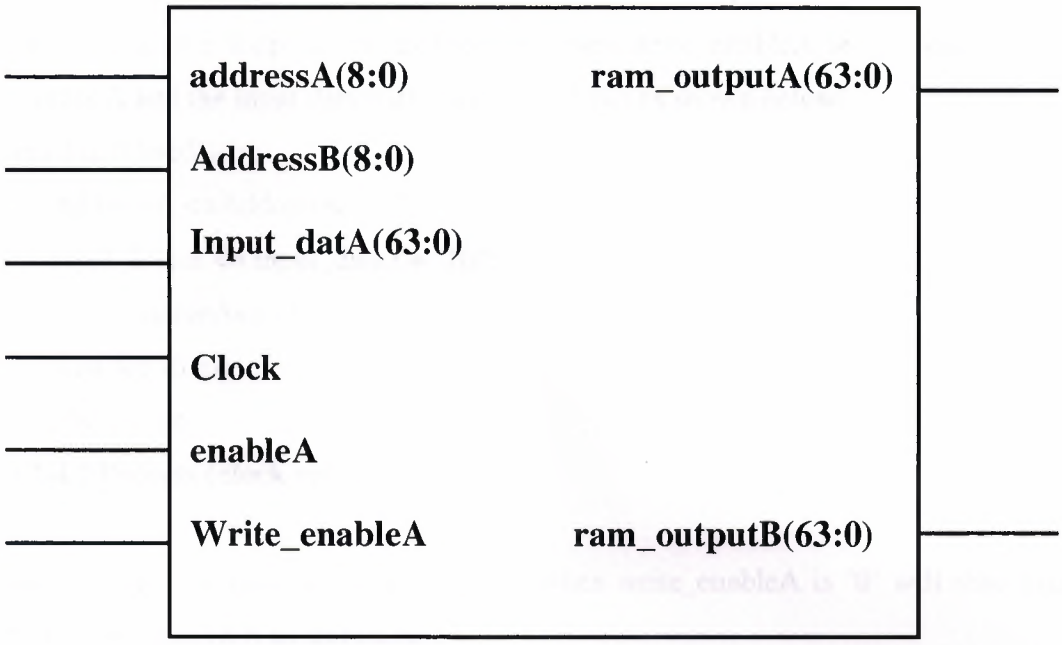

Figure 3.3 the inputs and out puts

3.2.4 The function of VHDL code

3.2.4.1 Process (clock write)

I have written a loop in the architecture when write_enableA is '1' will write in address A and the input data will increase two bits as shown below:

```
For I in 0 to 63 loop
    AddressA <=AddressA + '1';
    input_Dataa <= input_dataa + "10";
    write_enableA<='1';
    wait for 10 ns;
    END loop;
```

3.2.4.2 Process (clock read)

there is another loop in the architecture when write_enableA is '0' will read from both addresses A&B as shown below:

```
for i in 0 to 62 loop
    AddressA <=AddressA + '1';
    AddressB <=AddressB + '1';
    input_Dataa <= input_dataa + "10";
    wait for 10 ns;
    End loop;
```

3.2.5 VHDL code

The entity section of VHDL used to implement the inputs and outputs then the architecture section used to define the function

```
-----------------------------------------------------------------
-- 512*64 DPRAM module
```

```
--
-- KEYWORD: array, concurrent processes, generic, conv_integer
-----------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-----------------------------------------------------------

entity DPRAM is
generic(
        width:          integer:=64;

        depth:          integer:=512;

        addr:           integer:=9);



port(   Clock:          in std_logic;
        enableA:        in std_logic;
  write_enableA :       in std_logic;

        AddressA:       in std_logic_vector(addr-1 downto 0);
        AddressB:       in std_logic_vector(addr-1 downto 0);
     input_dataA:       in std_logic_vector(width-1 downto 0);
```

```
    ram_outputA:                    out std_logic_vector(width-1 downto 0);
    ram_outputB:                    out std_logic_vector(width-1 downto 0)
);

end DPRAM;

-----------------------------------------------------------

architecture behav of DPRAM is

-- use array to define the bunch of internal temparary signals

  type ram_type is array (0 to depth-1) of
                          std_logic_vector(width-1 downto 0);
  signal tmp_ram: ram_type;

begin


      -- process(Clock, Read)
         process(Clock)

begin
 if (Clock'event and Clock='1') then
 if (enableA ='1') then

 if (write_enableA ='1') then
 tmp_ram(conv_integer(addressA)) <= input_dataA;

end if;
 ram_outputA <= tmp_ram(conv_integer(addressA));
```

```
 ram_outputB <= tmp_ram(conv_integer(addressB));
end if;
end if;

end process;


end behav;
-----------------------------------------------------------------
```

3.3 Creating My project

3.3.1 How to create my project

Create a new ISE project which will target the FPGA device on the Spartan-3 Startup Kit demo board

To create my project:

1. Select **File → New Project...** The New Project Wizard appears.
2. Type **DPRAM** in the Project Name field.
3. Enter or browse to a location (directory path) for the new project. A tutorial Subdirectory is created automatically.
4. Verify that **HDL** is selected from the Top-Level Source Type list.
5. Click **Next** to move to the device properties page.

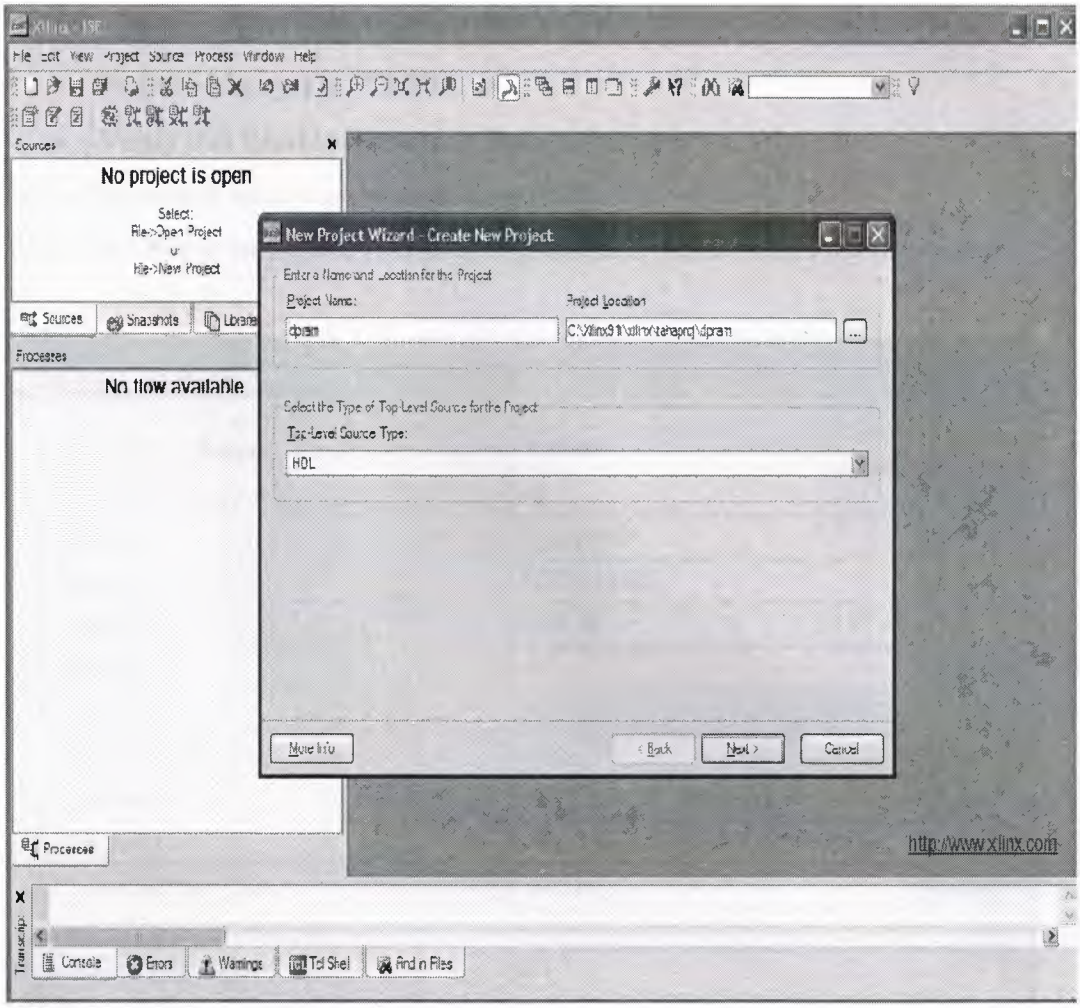

Figure 3.3.1 Create a New project

6. Fill in the properties in the table as shown below:

- Product Category: **All**
- Family: **Spartan3**
- Device: **XC3S1500l**
- Package: **FT676**
- Speed Grade: **-4**
- Top-Level Source Type: **HDL**

- Synthesis Tool: **XST (VHDL)**
- Simulator: **ISE Simulator (VHDL)**
- Preferred Language: **VHDL**
- Verify that **Enable Enhanced Design Summary** is selected.

Leave the default values in the remaining fields.

When the table is complete, your project properties will look like the following:

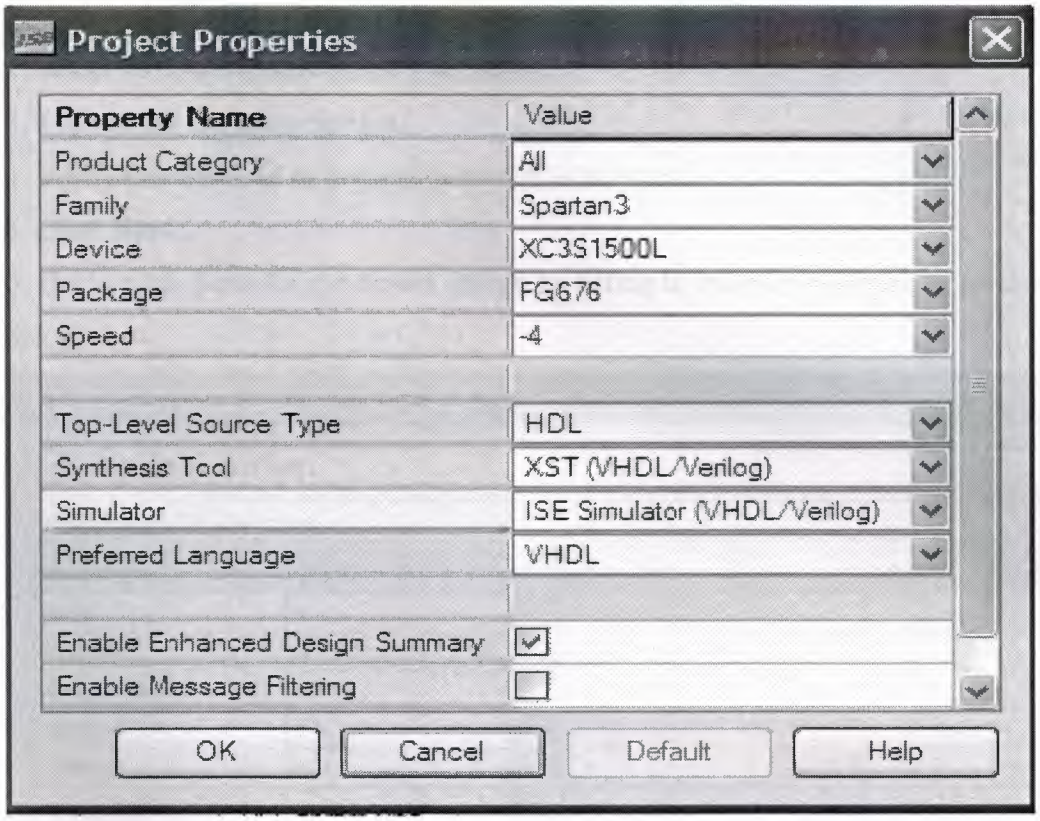

Figure 3.3.1 Project Device Properties

7. Click **Next** to proceed to the Create New Source window in the New Project Wizard. At the end of the next section, my project will be complete.

3.3.2 Creating an HDL Source

In this section, will create the top-level HDL file for my design. Determine the language that i wish to use for the tutorial. Then, continue either to the "Creating a VHDL Source" section.

3.3.2.1Creating a VHDL Source

Create a VHDL source file for the project as follows:

1. Click the **New Source** button in the New Project Wizard.
2. Select **VHDL Module** as the source type.
3. Type in the file name **dpram**.
4. Verify that the **Add to project** checkbox is selected.
5. Click **Next**.
6. Declare the ports for the dpram design by filling in the port information as shown below:

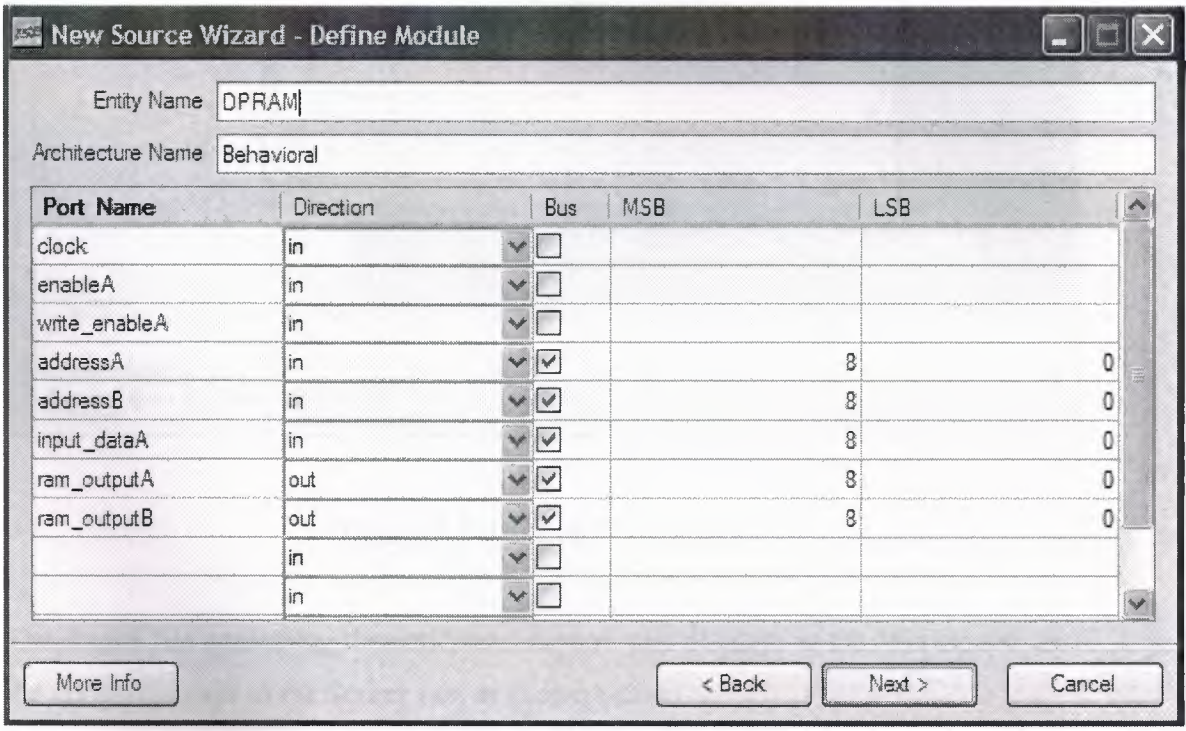

Figure3.3.2.1 Define Module

7. Click **Next**, and then **Finish** in the New Source Wizard - Summary dialog box to complete the new source file template

8. Click **Next**, then **Next**, then **Finish**.

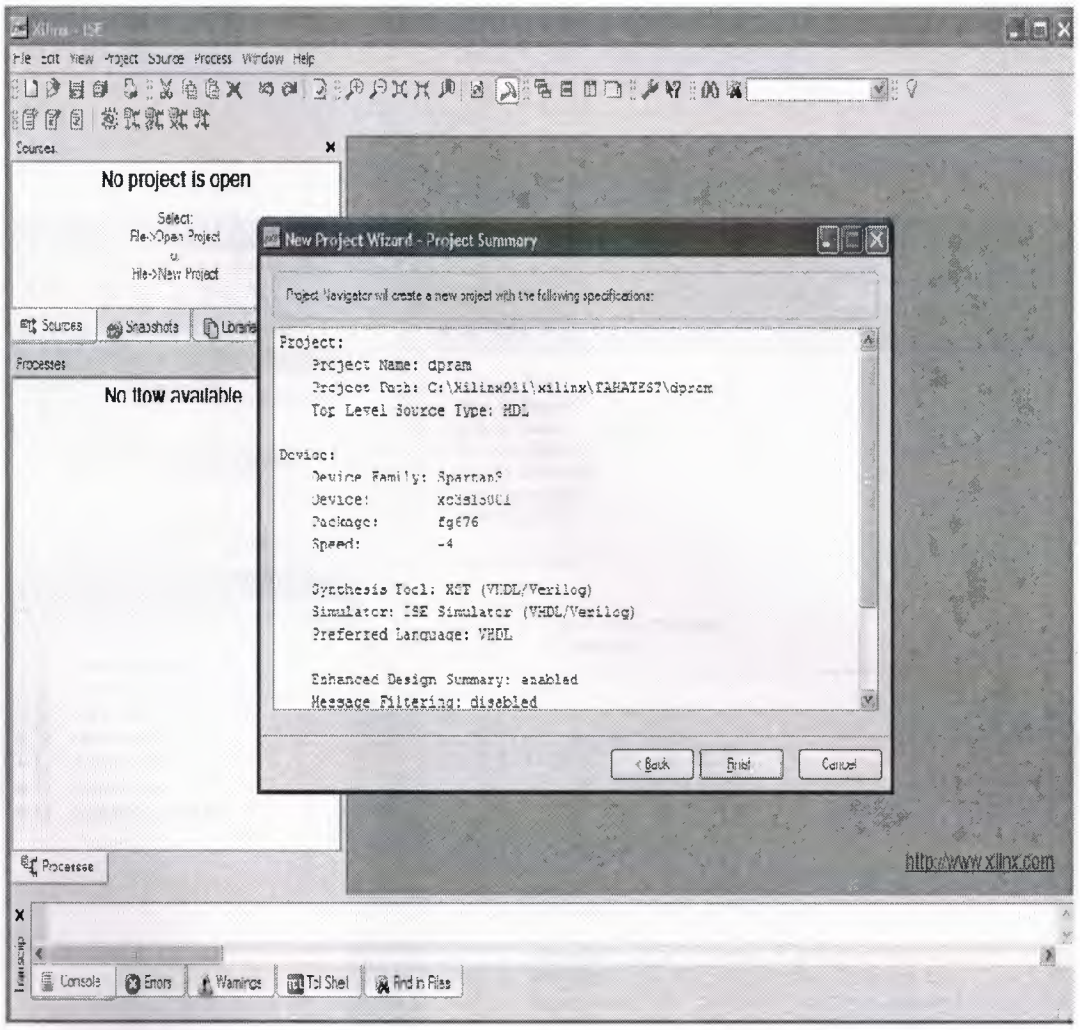

Figure3.3 .2.1 project summary

The source file containing the entity/architecture pair displays in the Workspace, and the dpram displays in the Source tab, as shown below:

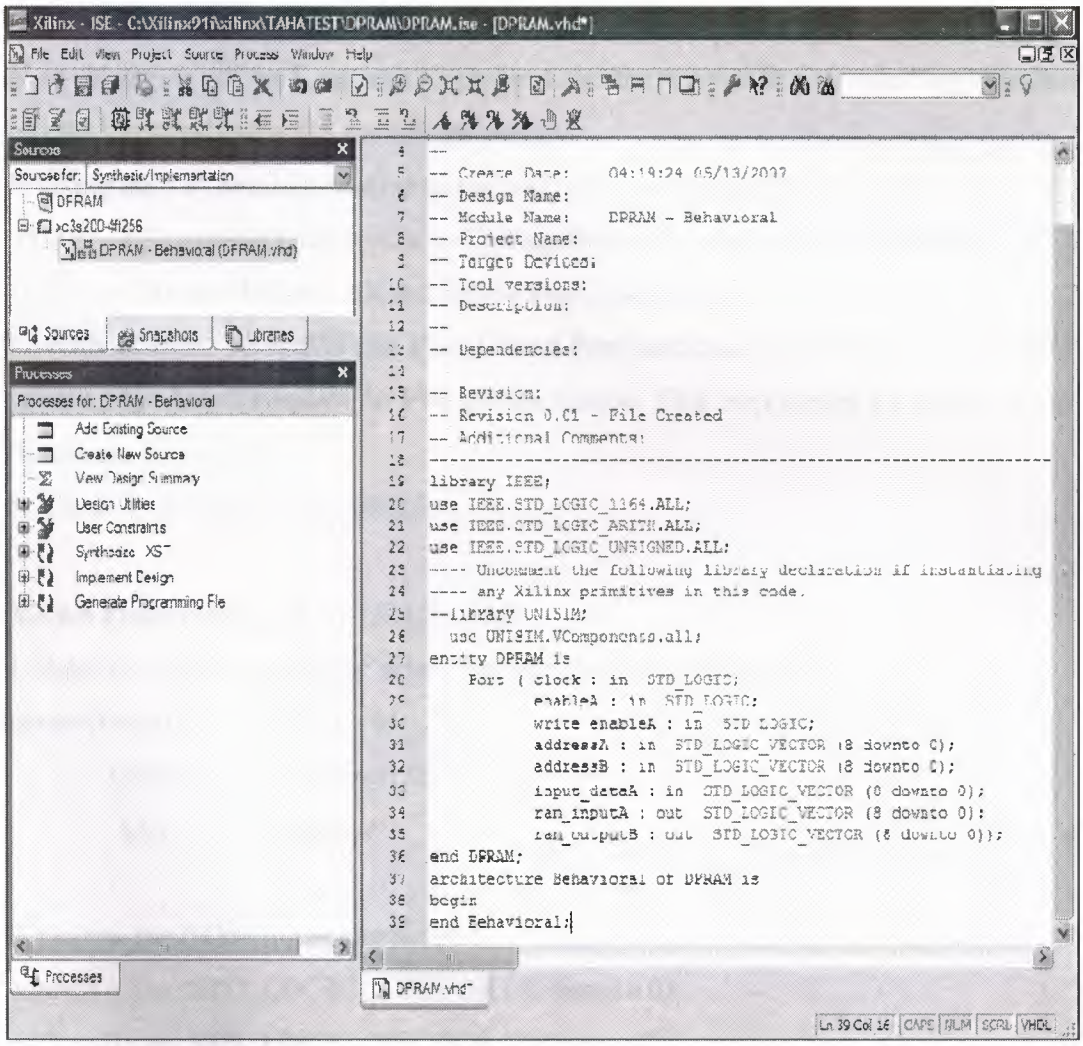

Figure 3.3.2.1 New Project in ISE

3.3.2.2 Using Language Templates (VHDL)

The next step in creating the new source is to add the behavioral description for the counter. To do this you will use a simple counter code example from the ISE Language Templates and customize it for the counter design.

1. Place the cursor just below the begin statement within the counter architecture.
2. Open the Language Templates by selecting **Edit →Language Templates...**

Note: You can tile the Language Templates and the counter file by selecting **Window →Tile Vertically** to make them both visible.

3. Using the "+" symbol, browse to the following code example:

VHDL → Synthesis Constructs →Coding Examples →Ram →Block Ram → Dual port Ram→1Clock, 1Read\Write Port,1Read Port

4. With **1Clock, 1Read\Write Port,1Read Port** selected, select **Edit→Use in File**, or select the **Use Template in File** toolbar button. This step copies the template into the dpram source file.
5. Close the Language Templates.

3.3.2.3 Final Editing of VHDL Source

1. Add the generic parameter below the entity and above the port:

```
generic(width:    integer:=64;
        depth:    integer:=512;
        addr:     integer:=9);
```

2. change the following occurrences

```
addressA : in  STD_LOGIC_VECTOR ( 8 downto 0);
addressB : in  STD_LOGIC_VECTOR (8 downto 0);
input_dataA : in  STD_LOGIC_VECTOR (8 downto 0);
ram_outputA : in  STD_LOGIC_VECTOR (8 downto 0);
ram_outputB : in  STD_LOGIC_VECTOR (8 downto 0);
```

To:

```
addressA : in  STD_LOGIC_VECTOR (addr-1 downto 0);
addressB : in  STD_LOGIC_VECTOR (addr-1 downto 0);
input_dataA : in  STD_LOGIC_VECTOR (width-1 downto 0);
ram_outputA : in  STD_LOGIC_VECTOR (width-1 downto 0);
ram_outputB : in  STD_LOGIC_VECTOR (width-1 downto 0);
```

3. Add the following signal declaration to handle the feedback of the dpram output below the architecture declaration and above the first begin statement:

```
type ram_type is array (0 to depth-1) of
std_logic_vector(width-1 downto 0);
signal tmp_ram: ram_type;
```

4. Customize the source file for the counter design by replacing the port and signal name place holders with the actual ones as follows:

- replace all occurrences of < clock > with CLOCK
- replace all occurrences of < enableA > with enableA
- replace all occurrences of < write_enableA > with write_enableA
- replace all occurrences of < addressA > with addressA
- replace all occurrences of < addressB > with addressB
- replace all occurrences of < input_dataA> with input_dataA
- replace all occurrences of < ram_outputA> with ram_outputA
- replace all occurrences of < ram_outputB> with ram_outputB
- replace all occurrences of < ram_name> with tmp_ram

5. Save the file by selecting **File →Save**.

When I am finished, the dpram source file will look like the following:

```
-- 512*64 DPRAM module

-- KEYWORD: array, concurrent processes, generic, conv_integer
-----------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-----------------------------------------------------------


entity DPRAM is

        generic(
                width:          integer:=64;

                depth:          integer:=512;

                addr:           integer:=9);

        port(   Clock:          in std_logic;
```

```
                    enableA:            in std_logic;

                    write_enableA :     in std_logic;



                    AddressA:    in std_logic_vector(addr-1 downto 0);
                    AddressB:    in std_logic_vector(addr-1 downto 0);



                    input_dataA: in std_logic_vector(width-1 downto 0);


                    ram_outputA: out std_logic_vector(width-1 downto 0);
    ram_outputB:    out std_logic_vector(width-1 downto 0)
);

end DPRAM;

-------------------------------------------------------------

architecture behav of DPRAM is

-- use array to define the bunch of internal temparary signals

        type ram_type is array (0 to depth-1) of
        std_logic_vector (width-1 downto 0);
        signal tmp_ram: ram_type;
```

```
begin

        -- Read Functional Section
        -- process(Clock, Read)

                process(Clock)

begin
                if (Clock'event and Clock='1') then

                if (enableA ='1') then


                if (write_enableA ='1') then
                tmp_ram(conv_integer(addressA)) <= input_dataA;


    end if;

                ram_outputA <= tmp_ram(conv_integer(addressA));
                ram_outputB <= tmp_ram(conv_integer(addressB));


      end if;


      end if;

end process;

end behav;
```

3.4 Synthesize

from the process window click on '+' which is for synthesize XST

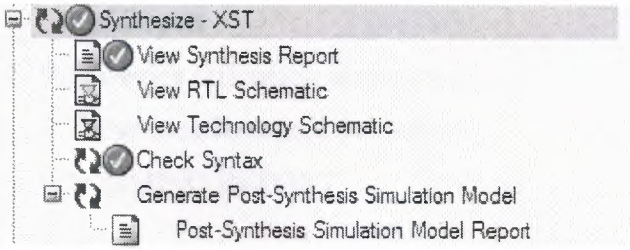

Figure 3.4 Synthesize in Process Window

3.4.1 View synthesize report

Release 9.1i - xst J.30

Copyright (c) 1995-2007 Xilinx, Inc. All rights reserved.

--> Parameter TMPDIR set to ./xst/projnav.tmp

CPU : 0.00 / 2.97 s | Elapsed : 0.00 / 3.00 s

--> Parameter xsthdpdir set to ./xst

CPU : 0.00 / 2.98 s | Elapsed : 0.00 / 3.00 s

--> Reading design: DPRAM.prj

TABLE OF CONTENTS

1) Synthesis Options Summary
2) HDL Compilation
3) Design Hierarchy Analysis
4) HDL Analysis
5) HDL Synthesis
   5.1) HDL Synthesis Report
6) Advanced HDL Synthesis

6.1) Advanced HDL Synthesis Report

7) Low Level Synthesis

8) Partition Report

9) Final Report

9.1) Device utilization summary

9.2) Partition Resource Summary

9.3) TIMING REPORT

```
=========================================================================
*                    Synthesis Options Summary                          *
=========================================================================
---- Source Parameters
Input File Name                    : "DPRAM.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "DPRAM"
Output Format                      : NGC
Target Device                      :  xc3s1500l-4-fg676

---- Source Options
Top Module Name                    : DPRAM
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
```

NEAR EAST UNIVERSITY LIBRARY 1988 - LEFKOŞA

RAM Style : Auto
ROM Extraction : Yes
Mux Style : Auto
Decoder Extraction : YES
Priority Encoder Extraction : YES
Shift Register Extraction : YES
Logical Shifter Extraction : YES
XOR Collapsing : YES
ROM Style : Auto
Mux Extraction : YES
Resource Sharing : YES
Asynchronous To Synchronous : NO
Multiplier Style : auto
Automatic Register Balancing : No

---- Target Options
Add IO Buffers : YES
Global Maximum Fanout : 500
Add Generic Clock Buffer(BUFG) : 8
Register Duplication : YES
Slice Packing : YES
Optimize Instantiated Primitives : NO
Use Clock Enable : Yes
Use Synchronous Set : Yes
Use Synchronous Reset : Yes
Pack IO Registers into IOBs : auto
Equivalent register Removal : YES

---- General Options
Optimization Goal : Speed
Optimization Effort : 1

```
Library Search Order          : DPRAM.lso
Keep Hierarchy                : NO
RTL Output                    : Yes
Global Optimization           : AllClockNets
Read Cores                    : YES
Write Timing Constraints      : NO
Cross Clock Analysis          : NO
Hierarchy Separator           : /
Bus Delimiter                 : <>
Case Specifier                : maintain
Slice Utilization Ratio       : 100
BRAM Utilization Ratio        : 100
Verilog 2001                  : YES
Auto BRAM Packing             : NO
Slice Utilization Ratio Delta : 5

=========================================================================

*                    HDL Compilation                    *
=========================================================================
Compiling vhdl file "C:/Xilinx91i/xilinx/tahaproj/dpram/dpram.vhd" in Library work.
Entity <dpram> compiled.
Entity <dpram> (Architecture <behav>) compiled.

=========================================================================
```

```
*                    Design Hierarchy Analysis                    *
=================================================================
============
Analyzing hierarchy for entity <DPRAM> in library <work> (architecture <behav>)
with generics.
        addr = 9
        depth = 512
        width = 64


=================================================================
============
*                         HDL Analysis                         *
=================================================================
============
Analyzing generic Entity <DPRAM> in library <work> (Architecture <behav>).
        width = 64
        depth = 512
        addr = 9
Entity <DPRAM> analyzed. Unit <DPRAM> generated.


=================================================================
============
*                        HDL Synthesis                         *
=================================================================
============

Performing bidirectional port resolution...

Synthesizing Unit <DPRAM>.
```

```
    Related source file is "C:/Xilinx91i/xilinx/tahaproj/dpram/dpram.vhd".
    Found 512x64-bit dual-port RAM <Mram_tmp_ram> for signal <tmp_ram>.
    Found 64-bit register for signal <ram_outputA>.
    Found 64-bit register for signal <ram_outputB>.
    Summary:
        inferred   1 RAM(s).
        inferred 128 D-type flip-flop(s).
Unit <DPRAM> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# RAMs                                  : 1
 512x64-bit dual-port RAM               : 1
# Registers                             : 2
 64-bit register                        : 2

=========================================================================

=========================================================================
*                      Advanced HDL Synthesis                           *
=========================================================================

Loading device for application Rf_Device from file '3s1500l.nph' in environment
C:\Xilinx91i.
```

INFO:Xst:2691 - Unit <DPRAM> : The RAM <Mram_tmp_ram> will be implemented as a BLOCK RAM, absorbing the following register(s): <ram_outputA> <ram_outputB>.

```
-----------------------------------------------------------------------
| ram_type        | Block                                   |        |
-----------------------------------------------------------------------
| Port A                                                    |
|    aspect ratio  | 512-word x 64-bit                      |        |
|    mode          | read-first                             |        |
|    clkA          | connected to signal <Clock>            | rise   |
|    enA           | connected to signal <enableA>          | high   |
|    weA           | connected to signal <write_enableA>    | high   |
|    addrA         | connected to signal <AddressA>         |        |
|    diA           | connected to signal <input_dataA>      |        |
|    doA           | connected to signal <ram_outputA>      |        |
-----------------------------------------------------------------------
| Port B                                                    |
|    aspect ratio  | 512-word x 64-bit                      |        |
|    mode          | write-first                            |        |
|    clkB          | connected to signal <Clock>            | rise   |
|    enB           | connected to signal <enableA>          | high   |
|    addrB         | connected to signal <AddressB>         |        |
|    doB           | connected to signal <ram_outputB>      |        |
-----------------------------------------------------------------------
```

==========

Advanced HDL Synthesis Report

Macro Statistics

RAMs : 1

512x64-bit dual-port block RAM : 1

===

===

* Low Level Synthesis *

===

Optimizing unit <DPRAM> ...

Mapping all equations...

Building and optimizing final netlist ...

Found area constraint ratio of 100 (+ 5) on block DPRAM, actual ratio is 0.

Final Macro Processing ...

===

Final Register Report

Found no macro

===

===

* Partition Report *

```
=========================================================================
=============

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
=============
*                    Final Report                    *
=========================================================================
=============
Final Results
RTL Top Level Output File Name     : DPRAM.ngr
Top Level Output File Name         : DPRAM
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 213

Cell Usage :
# BELS                             : 1
#      GND                         : 1
# RAMS                             : 2
#      RAMB16_S36_S36              : 2
# Clock Buffers                    : 1
```

```
#      BUFGP                    : 1
# IO Buffers                    : 212
#      IBUF                     : 84
#      OBUF                     : 128
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1500lfg676-4

 Number of Slices:                    0  out of  13312     0%
 Number of IOs:                     213
 Number of bonded IOBs:               213  out of    487    43%
 Number of BRAMs:                     2  out of     32     6%
 Number of GCLKs:                     1  out of      8    12%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT
```

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

Clock Signal	Clock buffer(FF name)	Load
Clock	BUFGP	2

Asynchronous Control Signals Information:

No asynchronous control signals found in this design

Timing Summary:

Speed Grade: -4

Minimum period: No path found
Minimum input arrival time before clock: 2.222ns
Maximum output required time after clock: 6.457ns
Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

```
=========================================================================
============
Timing constraint: Default OFFSET IN BEFORE for Clock 'Clock'
  Total number of paths / destination ports: 106 / 106
-------------------------------------------------------------------------
Offset:              2.222ns (Levels of Logic = 1)
  Source:            enableA (PAD)
  Destination:       inst_Mram_mem (RAM)
  Destination Clock: Clock rising

  Data Path: enableA to inst_Mram_mem
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             4   0.821   0.917  enableA_IBUF (enableA_IBUF)
     RAMB16_S36_S36:ENA        0.484          inst_Mram_mem
    ----------------------------------------
    Total                     2.222ns (1.305ns logic, 0.917ns route)
                                      (58.7% logic, 41.3% route)

=========================================================================
============
Timing constraint: Default OFFSET OUT AFTER for Clock 'Clock'
  Total number of paths / destination ports: 128 / 128
-------------------------------------------------------------------------
Offset:              6.457ns (Levels of Logic = 1)
  Source:            inst_Mram_mem1 (RAM)
  Destination:       ram_outputA<63> (PAD)
  Source Clock:      Clock rising

  Data Path: inst_Mram_mem1 to ram_outputA<63>
```

```
                    Gate     Net
   Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
   ----------------------------------------  ------------
    RAMB16_S36_S36:CLKA->DOA27        1     0.012     0.801   inst_Mram_mem1
(ram_outputA_63_OBUF)
    OBUF:I->O                 5.644         ram_outputA_63_OBUF (ram_outputA<63>)
   ----------------------------------------
   Total                    6.457ns (5.656ns logic, 0.801ns route)
                                (87.6% logic, 12.4% route)

=========================================================================
============
CPU : 11.56 / 14.75 s | Elapsed : 11.00 / 14.00 s

-->

Total memory usage is 159180 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :  0 (   0 filtered)
Number of infos    :    1 (   0 filtered)
```

3.4.2 View RTL Schematic

Figure 3.4.2 inputs&outputs

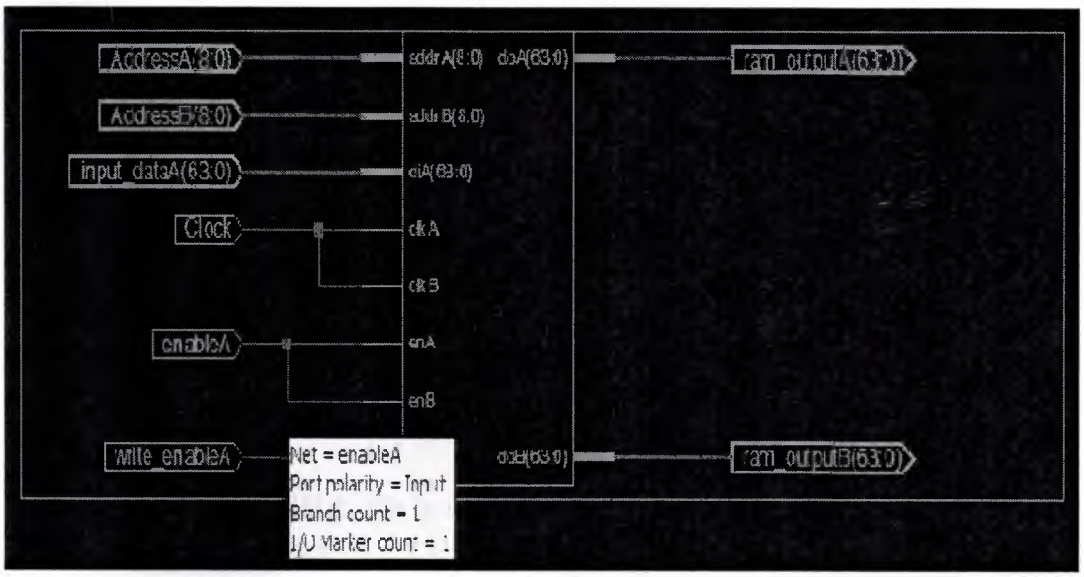

Figure 3.4.2 RTL schematic

3.4.3 check syntax

Started : "Check Syntax for DPRAM".

```
=================================================================
============
*                    HDL Compilation                    *
=================================================================
============
```

Compiling vhdl file "C:/Xilinx91i/xilinx/tahaproj/dpram/dpram.vhd" in Library work.
Architecture behav of Entity dpram is up to date.

Process "Check Syntax" completed successfully

3.5 Writing the test bech

3.5.1 Verifying Functionality using Behavioral Simulation

Creating my test bench containing input stimulus you can use to verify the Functionality of the DPRAM module. The test bench is a graphical view of a test bench.

Create my test bench as follows:

1. Select the DPRAM **HDL** file in the Sources window.
2. Create a new test bench source by selecting **Project →New Source**.

3. In the New Source Wizard, select **Test Bench** as the source type, and type **DPRAM _tb** in the File Name field. As shown below:

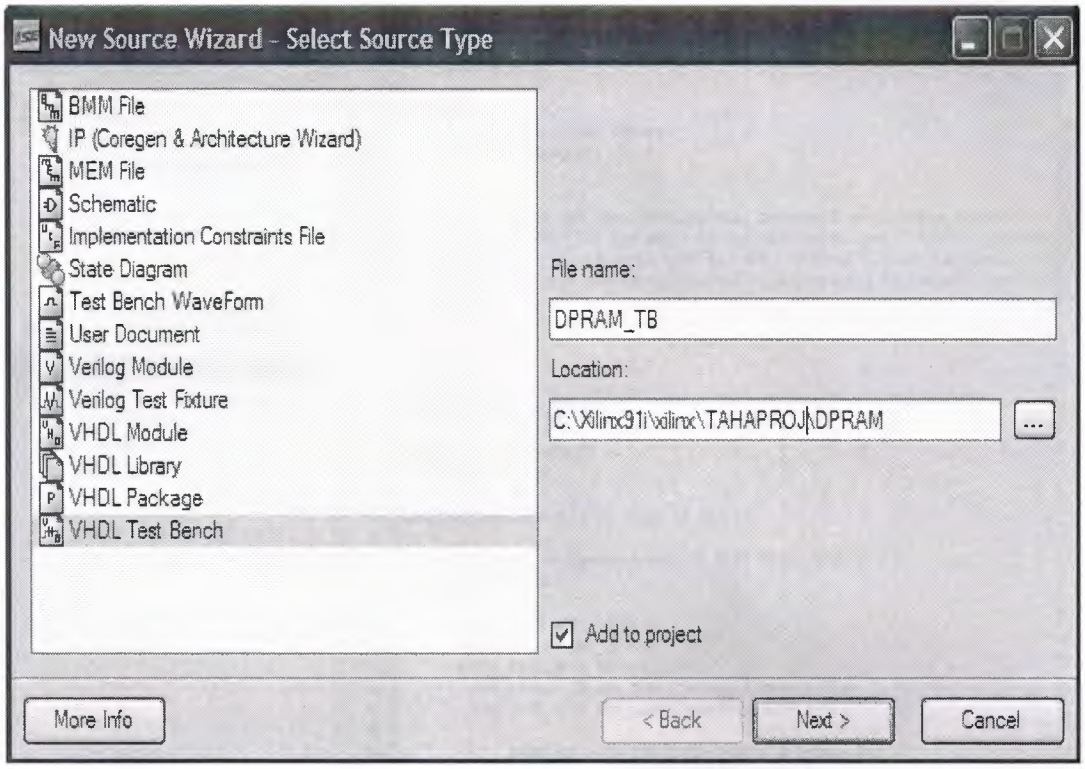

Figure 3.5.1 Select Source Type

4. Click **Next**.

5. The Associated Source page shows that you are associating the test bench with the source file DPRAM. Click **Next**.

6. The Summary page shows that the source will be added to the project, and it displays the source directory, type and name. Click **Finish**.

The source file containing the entity/architecture pair displays in the Workspace, and the dpram displays in the Source tab, as shown below:

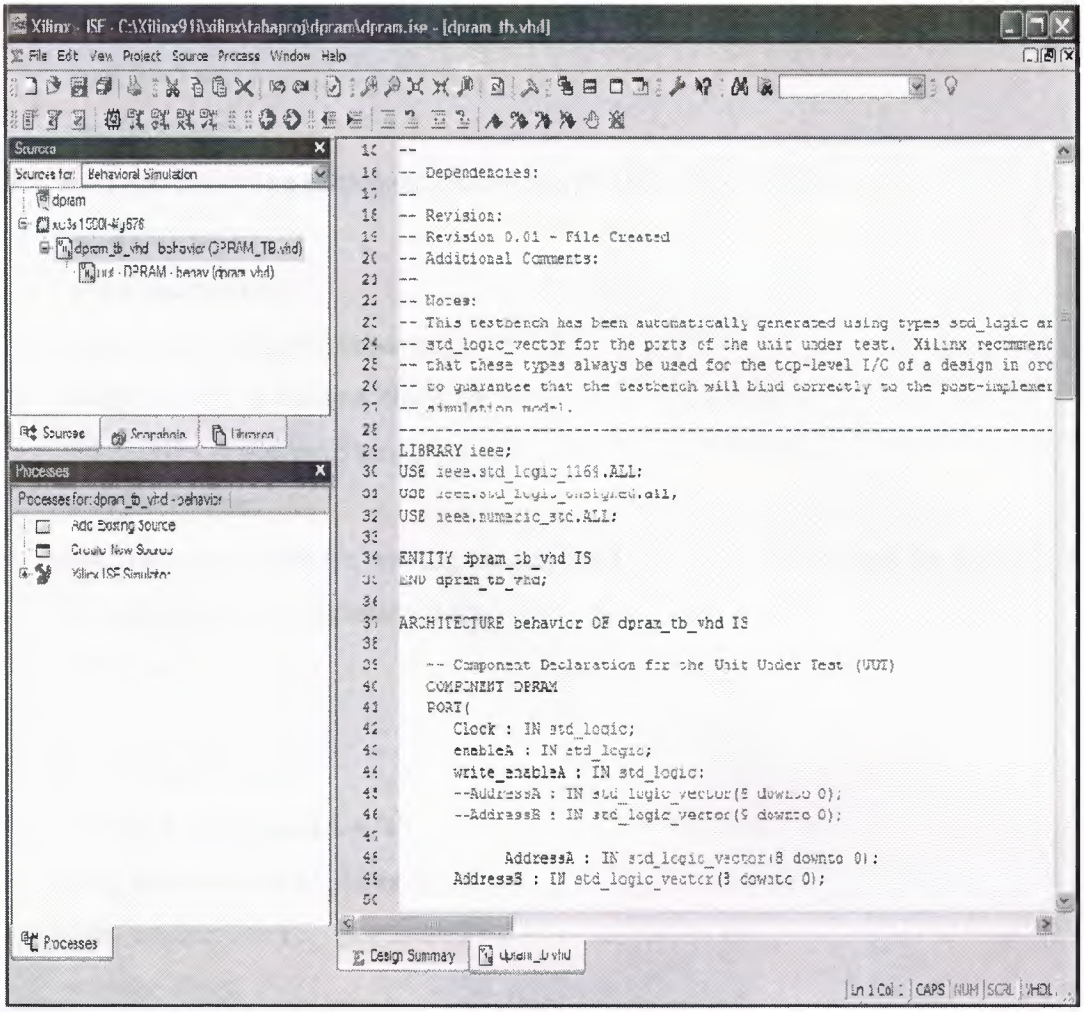

Figure 3.5.2: New Project in ISE

3.5.2 Final Editing of VHDL Source

1. Add the process&clock below the Instantiate the Unit Under Test (UUT) and above tb: process

```
clk_p: process
begin
clock <= '0';
wait for 5 ns;
```

```
clock <= '1';
wait for 5 ns;
end process;
```

2. Add the following statements below the tb: PROCESS BEGIN:

```
enableA <='1';
write_enableA<='1';
nput_dataa<=(input_dataa'range => '0');
AddressA <=( AddressA'range => '0');
AddressB <=(addressB'range => '0');
wait for 100 ns;
```

3. make two loops, one for writing in addressA just, and another one for reading, it can read from both addresses A&b:

```
--first loop is:

for i in 0 to 63 loop
AddressA <=AddressA + '1';
input_Dataa <= input_dataa + "10";
write_enableA<='1';
 wait for 10 ns;
END loop;
```

- Here I Add the following to make addressA zero after finishing form first loop and wait to start reading with addressB:

```
AddressA <=( AddressA'range => '0');
write_enableA<='0';
wait for 20 ns;

--Second loop is:
for i in 0 to 62 loop
AddressA <=AddressA + '1';
```

```
AddressB <=AddressB + '1';
input_Dataa <= input_dataa + "10";
wait for 10 ns;
End loop;
```

- . then I have to wait for ever after finishing loops to stop looping otherwise it is going to keep in looping to infinity so I write:

```
wait;
```

- . eventually end the process& end the test bench:

```
end process;
End behavior
```

When I am finished, the dpram source file will look like the following:

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   17:57:11 04/26/2007
-- Design Name:   DPRAM
-- Module Name:   C:/Xilinx91i/xilinx/tahaproj/dpram/dpram_tb.vhd
-- Project Name:  dpram
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: DPRAM
--
-- Dependencies:
--
```

```
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY dpram_tb_vhd IS
END dpram_tb_vhd;

ARCHITECTURE behavior OF dpram_tb_vhd IS

	-- Component Declaration for the Unit Under Test (UUT)
	COMPONENT DPRAM
	PORT(
		Clock : IN std_logic;
		enableA : IN std_logic;
		write_enableA : IN std_logic;
		--AddressA : IN std_logic_vector(9 downto 0);
		--AddressB : IN std_logic_vector(9 downto 0);
```

```
                    AddressA : IN std_logic_vector(8 downto 0);
            AddressB : IN std_logic_vector(8 downto 0);

            --input_dataA : IN std_logic_vector(127 downto 0);
    --      ram_outputA : OUT std_logic_vector(127 downto 0);
    --      ram_outputB : OUT std_logic_vector(127 downto 0)

                    input_dataA : IN std_logic_vector(63 downto
0);
            ram_outputA : OUT std_logic_vector(63 downto 0);
            ram_outputB : OUT std_logic_vector(63 downto 0)

            );
    END COMPONENT;

    --Inputs
    SIGNAL Clock : std_logic := '0';
    SIGNAL enableA : std_logic := '0';
    SIGNAL write_enableA : std_logic := '0';

    --SIGNAL AddressA : std_logic_vector(9 downto 0) := (others=>'0');
    --SIGNAL AddressB : std_logic_vector(9 downto 0) := (others=>'0');
    --SIGNAL input_dataA : std_logic_vector(127 downto 0) :=
(others=>'0');

    --Outputs
--  SIGNAL ram_outputA : std_logic_vector(127 downto 0);
--  SIGNAL ram_outputB : std_logic_vector(127 downto 0);
```

```
SIGNAL AddressA :  std_logic_vector(8 downto 0) := (others=>'0');
SIGNAL AddressB :  std_logic_vector(8 downto 0) := (others=>'0');
SIGNAL input_dataA :std_logic_vector(63 downto 0) := (others=>'0');

--Outputs
SIGNAL ram_outputA :  std_logic_vector(63 downto 0);
SIGNAL ram_outputB :  std_logic_vector(63 downto 0);


BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: DPRAM PORT MAP(
        Clock => Clock,
        enableA => enableA,
        write_enableA => write_enableA,
        AddressA => AddressA,
        AddressB => AddressB,
        input_dataA => input_dataA,
        ram_outputA => ram_outputA,
        ram_outputB => ram_outputB
);
clk_p: process
begin
clock <= '0';
wait for 5 ns;
clock <= '1';
wait for 5 ns;
end process;
```

```
tb: PROCESS
BEGIN                    -- Wait 100 ns for global reset to finish
        wait for 100 ns;

        enableA <='1';
        --wait for 100 ns;
        write_enableA<='1';

        input_dataa<=(input_dataa'range => '0');
        -- Place stimulus here
        AddressA <=( AddressA'range => '0');
        AddressB <=(addressB'range => '0');
        wait for 100 ns;



--for i in 0 to 127 loop
For I in 0 to 63 loop
    AddressA <=AddressA + '1';
    input_Dataa <= input_dataa + "10";
        write_enableA<='1';


        --wait; -- will wait forever
    Wait for 10 ns;

        END loop;

AddressA <=( AddressA'range => '0');
write_enableA<='0';
wait for 20 ns;
```

```
--for i in 0 to 126 loop
For I in 0 to 62 loop
    AddressA <=AddressA + '1';
    AddressB <=AddressB + '1';
    input_Dataa <= input_dataa + "10";

    Wait for 10 ns;

    END loop;
            wait; -- will wait forever


            End process;
            End behavior;
```

3.5.3 The simulation:

A VHDL based simulation typically uses the VHDL language to describe the stimulus, as well as the device which is being designed. The code that defines the stimulus is generally called the " testbench", and having a portable stimulus description gives greater flexibility.

To simulate:

- source window should be on behavioral simulation
- click at '+' sign of Xilinx ISE simulator at the process window then double click on Simulate Behavioral Model

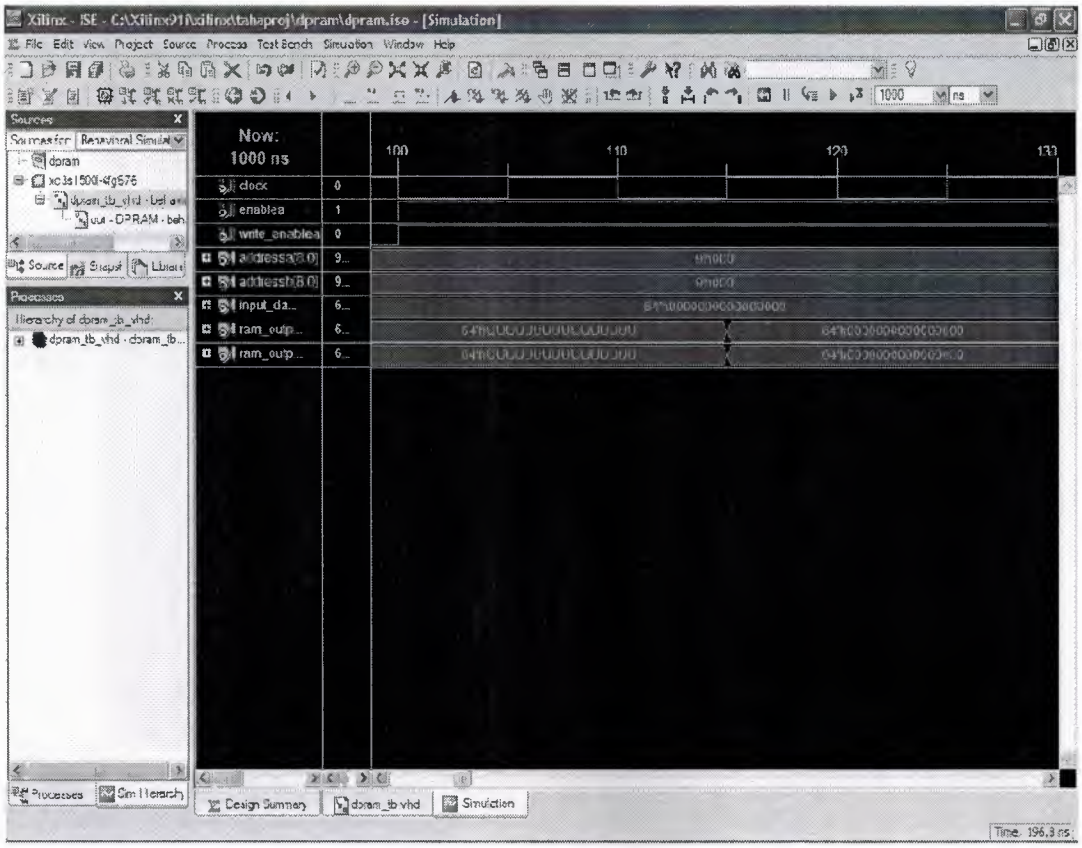

Figure3.5.3 high simulation result

in this simulation write_enableA is 1 so that shows when it is writing in address A while increasing 1 bit.

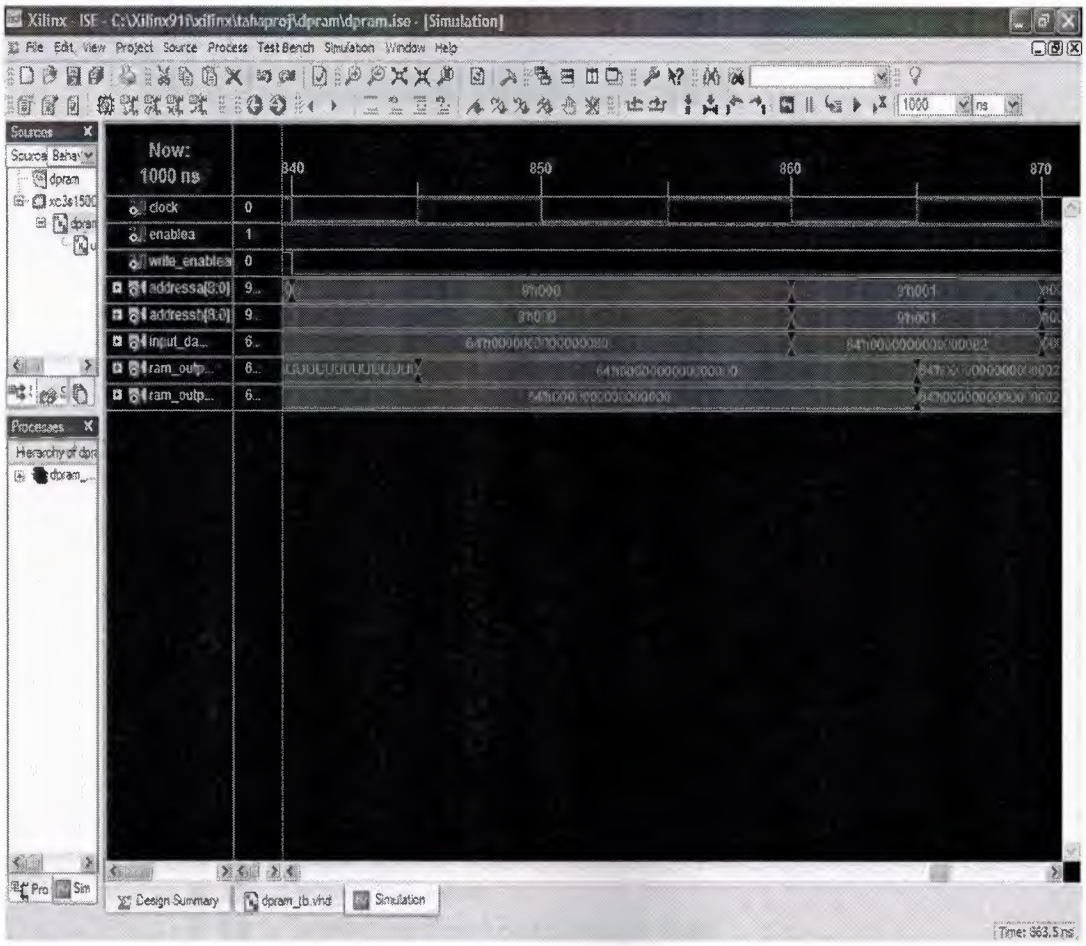

Figure 3.5.3 low Simulation result

This simulation shows that the writing has been stopped and started to read after 20 ns from both address A and address B at the same time.

3.6 Implementing my design

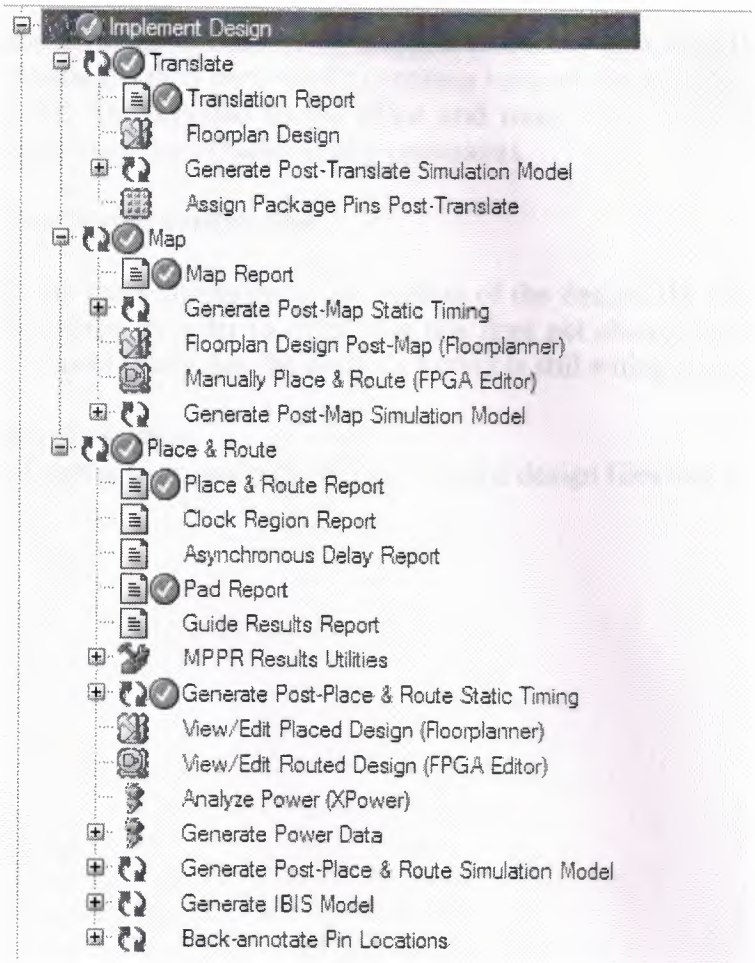

Figure 3.7 Implementing

Once we have a logical net list then we can think about translating this into a physical description. This process is cold implementation.
The implementation step first involves mapping, the placement and routing, and finally the programming of the target device.

- **Mapping**

Mapping is the process of packing the logical description generated by the synthesis tool into the different resources of the device, such as Configurable Logif Blocs (CLBs) or functional Blocks (FBs);

- **Place And Route (PAR)**

Placement involves allocation of mapped resources into specific locations on the device. Routing is then performed to connect between these building blocks. If timing requirements are supplied to the place and route tools, then the routing will be performed to rty to meet these timing constraints.

- **Post layout verification**

Now that we have this physical description of the design, the timing will have been changed slightly. In order to check that this does not change the functionality of the dasign, we must verify that the design's timing is stiil within specification.

- **Programming**

The final step is to program the device with the design files that you have created.

CONCLUSION

In this project, it has been designed a Dual Port Ram by using VHDL as a language; by using Xilinx ISE (Integrated Software Environment) as the development tool. First of all the specifications are written from the requirements. Then I/O are defined, and the function. VHDL codes are being completed and synthesized. Eventually the test bench is written to see the output where the inputs are defined. This gave the expected outputs which show that the project is successful.

REFERENCES

Form the DVD which I got from my supervisor which includes

- Xilinx 9.1i ISE
- Software manuals
- Tutorials
- language templates
- Application notes from (www.Xilinx. Com)

Form my TEACHER

- the soft copy
- hard copy
- VHDL examples
- Lap notes