



NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

**Traffic Light Controller
With VHDL**

**Graduation Project
COM-400**

Student: Adem KILIÇ

Supervisor: Mr.Mehmet Özakman

Nicosia-2007

ACKNOWLEDGEMENTS

First, I feel proud to pay my special regards to my project adviser 'Mehmet Özalkan'. He never disappointed me in any affair. He delivered me too much information and did his best of efforts to make me able to complete my project.

Second, I would like to thank Assoc. Pr. Dr. Rahib Abiyev for teaching programming methods in C languages, so it has been possible to work on the project via a widely used programming language.

Last, more over I want to pay special regards to my parents and especially my wife Nurten Çiftçi Kılıç and my daughter Alyanur Kılıç who are enduring these all expenses and supporting me in all events. I am nothing without their supports. They also encouraged me in crises. I shall never forget their sacrifices for my education so that I can enjoy my successful life as they are expecting. They may get peaceful life in Heaven. At the end I am again thankful to those all persons who helped me or even encouraged me to complete me, my project. My all efforts to complete this project might be fruitful.

To the best of my knowledge, I want to honor those all persons who have supported me or helped me in my project. I also pay my special thanks to my all friends who have helped me in my project and gave me their precious time to complete my project.

ABSTRACT

Today is electronic design VHDL is a behavioral language to describe design without going to complex of the electronic circuit diagram. We have done our design using VHDL and verify the design using simulation tools. We wrote the VHDL code to the synthesis tools which generate detailed electronic circuits.

We use VHDL language in the project. First we have defined the specification of the inputs and outputs of the traffic light controller than the function of the project. We have divided the project into the smaller modules which have been verified its function. Then we connected of these functions together under top level module to complete the design. We used a test bench to verify function of design. We used the synthesis tool to generate the detailed design, and then we converted the specification. Also VHDL entity is constructed to define the inputs and the outputs, after that we constructed VHDL architecture to define each module of the project.

The synthesizer that generates the design taking the consideration of the Xilinx FPGA device that we specified at the top level design. Each programmable chips have own characteristic. The same VHDL code is portable and can be synthesized to different FPGA programmable devices. We used Xilinx development system and Xilinx programmable devices because each vendor's tools are developed its on programmable devices.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
INTRODUCTION	1
CHAPTER ONE: INTRODUCTION TO TRAFFIC LIGHT	
CONTROLLER DESIGN	2
1.1. General System Requirement	2
1.2. Developing a Block Diagram of the System	3
1.3. State Diagram	3
1.4. Description of the State Diagram	4
CHAPTER TWO: TRAFFIC LIGHT CONTROLLER	
DESIGN & SIMULATION	5
2.1. ISE General Information	5
2.1.1. Xilinx ISE Overview	5
2.1.2. Design Entry	5
2.1.3. Synthesis	5
2.1.4. Implementation	5
2.1.5. Verification	6
2.1.6. Device Configuration	6
2.1.7. Architecture Support	6
2.1.8. Operating System Support	7
2.2. Using Project Navigator	7
2.2.1. Project Navigator Overview	7
2.2.2. Project Navigator Main Window	7
2.2.3. Using the Sources Window	9
2.2.4. Using the Processes Window	10
2.2.5. Process Types	10
2.2.6. Process Status	11
2.2.7. Running Processes	11

2.2.8. Setting Process Properties	13
2.2.9. Using the Workspace	13
2.2.10. Using the Transcript Window	14
2.2.11. Using the Toolbars	14
2.3. Creating a Project	14
2.4. Creating Traffic Light Project on ISE	18
2.5. Creating a Source File	20
2.6. Creating Counter.vhd VHDL Module	24
2.7. Creating a Test Bench Waveform	26
2.8. Creating Counter.vhd Test Bench Waveform	26
2.9. Creating a State Machine Diagram	31
2.10. Creating Top-Level VHDL Design	37
CHAPTER THREE: TRAFFIC LIGHT CONTROLLER	
IN VHDL	40
3.1. Very High Speed Integrated Circuit HDL	40
3.1.1. History of VHDL	40
3.1.2. VHDL – Application Field	42
3.1.3. VHDL Language and Syntax	42
3.1.4. VHDL Structural Elements	43
3.1.5. Declaration of VHDL Objects	44
3.1.6. Entity	44
3.1.7. Architecture	45
3.1.8. Architecture Structure	46
3.1.9. Process	47
3.1.10. Signals	47
3.2. Sequential Statements in VHDL	48
3.2.1. IF Statement	49
3.2.2. CASE Statement	49
3.2.3. FOR Loops	50
3.2.4. Loop Syntax	50
3.2.5. WAIT Statement	50
3.2.6. WAIT Statements and Behavioral Modeling	51

3.2.7. Variables	51
3.2.8. Variables and Signals	51
3.2.9. Use of Variables	52
3.3. VHDL Codes of Counter.vhd	52
3.4. VHDL Codes of STAT_MAC.vhd	53
3.5. VHDL Codes of top.vhd	57
CHAPTER FOUR: TRAFFIC LIGHT CONTROLLER	
SYNTHESIS	60
4.1. XST Design Flow Overview	60
4.2. XST Input and Output Files	61
4.3. XST Detailed Design Flow	62
4.3.1. HDL Parsing	62
4.3.2. HDL Synthesis	62
4.3.3. Low Level Optimization	63
CONCLUSION	65
REFERENCES	66

INTRODUCTION

The term “digit” comes from the Latin word “digitus” which means finger. Human beings can most easily learn counting by using their fingers, that’s why the word digit is used for denoting number.

The term digital is derived from the way computers perform operations, by counting digits. For many years, applications of digital electronics were confined to computer systems. Today, digital technology is applied wide range of areas in addition to computers. Such applications as television, communications systems, radar, navigation and guidance systems, military systems, medical instrumentation, industrial process control, and consumer electronics use digital techniques. Over the years digital technology has progressed from vacuum-tube circuits to discrete transistor to complex integrated circuits, some of which contain millions of transistors.

The objective of this project is controlling of a traffic light for safe traffic flow in where it is stand by using VHDL tool. The project consists of introduction, four chapters, and conclusion.

Chapter One presents specification and requirements of the traffic light controller depending on basic minimal diagram.

Chapter Two presents the steps how the project “Traffic Light Controller System” is constructed on ISE software environment tool. It also presents the architectural construction of the components and the description of the functions of each component.

Chapter Three describes the VHDL (Very High Speed Integrated Circuit Hardware Description Language). It also gives brief information about history and development of VHDL and how declaration of its object and sequential statements are used. Additionally, it consists of the complete VHDL codes of the project.

Chapter Four describes how a prepared project is synthesis in Integrated Software Environment.

Finally, the conclusion section presents how the described hardware (in VHDL) is constructed on the device with the complete code within the project.

CHAPTER ONE: INTRODUCTION TO TRAFFIC LIGHT CONTROLLER DESIGN

The aim of this project is to define the process of designing a traffic light controlling application using VHDL and implementing on Xilinx Spartan-3E device. Firstly, we construct the system requirements and a state diagram to define the sequence of operation.

1.1. General System Requirement

A digital controller is necessitated to control a traffic light at the intersection of a predefined time intervals. We should be define a timer that count 0000 to 1111 and a reset to move the initial state (RED) whatever position of the state in the system.

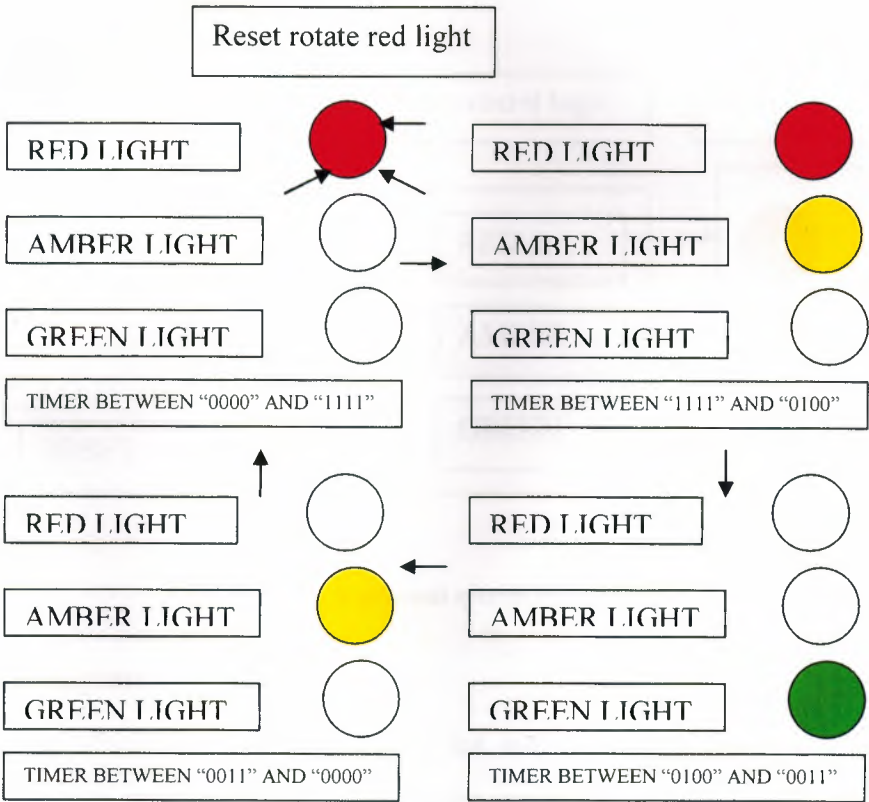


Figure 1. 1 Requirements for the traffic light sequence.

1.2. Developing a Block Diagram of the System

From the requirements we can develop a block diagram of the system. First, we consider that the system must control three different lights. These are the red, yellow, and green lights.

Using the minimal system block diagram we can begin to feel in the details. The system has four details as indicated in the below figure 1.2. So a logic circuit is needed to control the sequence of the states. Also circuits are needed to generate the proper time intervals of 1111 ns and 0000 ns in binary base that are required in the system and to generate a clock signal for cycling the system. The time intervals are inputs to the sequential logic because the sequencing of states is a function of these variables. Logic circuits are also needed to determine which of the four states of the system is in at any given time, to generate the proper outputs to lights.

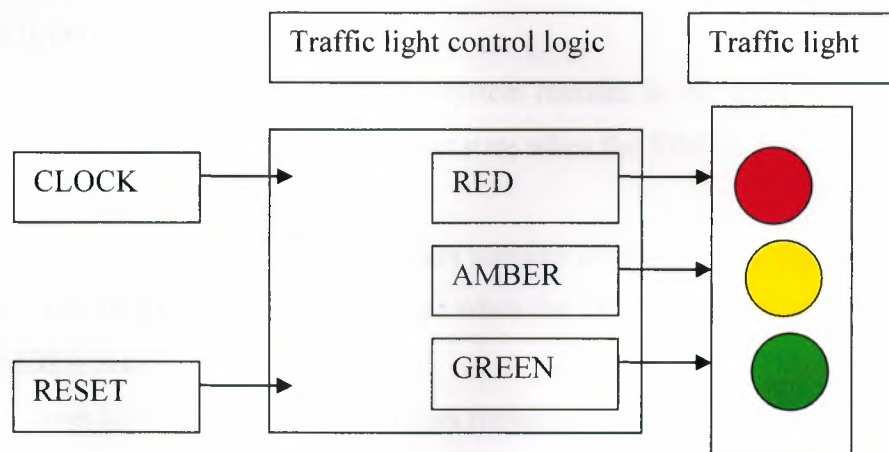


Figure 1. 2. A minimal system block diagram.

1.3. State Diagram

A state diagram graphically shows the sequence of states in a system and the conditions for each state and for transitions from one state to the next.

Before traditional state diagram can be developed, the variables that determined how the system sequences through its states must be defined these variables and their symbols are listed as follows:

- 0000 to 1111 timer is on =TIMER
- Red state carrier =RESET
- RD for red light output
- AMB for amber light output
- GRN for green light output

1.4. Description of the State Diagram

A state diagram is shown in figure CIZ. Each of the four states is labeled according to color name abbreviation as indicated by the ellipses. Each of the arrows going from one state to the next indicates a state transition under the condition defined by the equation associated variables. Also TIMER is associated with four bit up-counter

- RED State:

The street light is red this system remains in the state for a period from 0000 to 1111. The system goes to the next state when the TIMER is equal to 1111.

- REDAMB State:

The street light is red and amber the system remains in this state for a period from 1111 to 0100. The system goes to the next state when the TIMER is equal to 0100.

- GREEN State:

The street light is green and the system remains in this state for a period from 0100 to 0011. The system goes to the next state when the TIMER is equal to 0011.

- AMBER State:

The street light is amber and the system remains in this state from 0011 to 0000. The system goes to the next state when the TIMER is equal to 0000.

CHAPTER TWO: TRAFFIC LIGHT CONTROLLER DESIGN & SIMULATION

2.1. ISE General Information

2.1.1. Xilinx ISE Overview

The Integrated Software Environment (ISE™) is the Xilinx® design software suite that allows us to take our design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes our 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, we create our source files based on our design objectives. We can create our top-level design file using a Hardware Description Language (HDL), such as VHDL, Verilog, or ABEL, or using a schematic. We can use multiple formats for the lower-level source files in our design. If we are working with a synthesized EDIF or NGC/NGO file, we can skip design entry and synthesis and start with the implementation process.

2.1.3. Synthesis

After design entry and optional simulation, we 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, we 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, we can run the implementation process in one step, or we can run each of the implementation processes separately. Implementation processes vary depending on whether we are targeting a Field Programmable Gate Array (FPGA) or a Complex Programmable Logic Device (CPLD).

2.1.5. Verification

We can verify the functionality of our design at several points in the design flow. We can use simulator software to verify the functionality and timing of our design or a portion of our 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 us to create and verify complex functions in a relatively small amount of time. We can also run in-circuit verification after programming the device.

2.1.6. Device Configuration

After generating a programming file, we configure our device. During configuration, we generate configuration files and download the programming files from a host computer to a Xilinx device. Xilinx ISE Overview Architecture Support

2.1.7. Architecture Support

The ISE™ software supports the following device families.

Table 2. 1 Supported devices by ISE.

FPGAs	CPLDs
Spartan™-II	Cool Runner™ XPLA3
Spartan-IIe	Cool Runner-II
Spartan-3	XC9500™
Spartan-3E	XC9500XL
Spartan-3L	XC9500XV
Virtex™	
Virtex-E	
Virtex-II	
Virtex-II Pro	
Virtex-II Pro X	
Virtex-4	
Virtex-5 LX	

2.1.8. Operating System Support

The ISE™ software is supported on the following operating systems.

Table 2. 2 Supported operating systems by ISE.

Operating System	Versions
Windows®	Windows XP® Professional
	Windows 2000® Professional
Solaris®	Solaris 8
	Solaris 9
Linux	Red Hat® Enterprise WS 3.0 32-bit/64-bit
	Red Hat Enterprise WS 4.0 32-bit/64-bit

2.2. Using Project Navigator

2.2.1. Project Navigator Overview

Project Navigator organizes our 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 our Xilinx FPGA and CPLD designs, which allows us to do the following:

1. Add and create design source files, which appear in the Sources window
2. Modify the source files in the Workspace
3. Run processes on the source files in the Processes window
4. View output from the processes in the Transcript window

2.2.2. Project Navigator Main Window

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

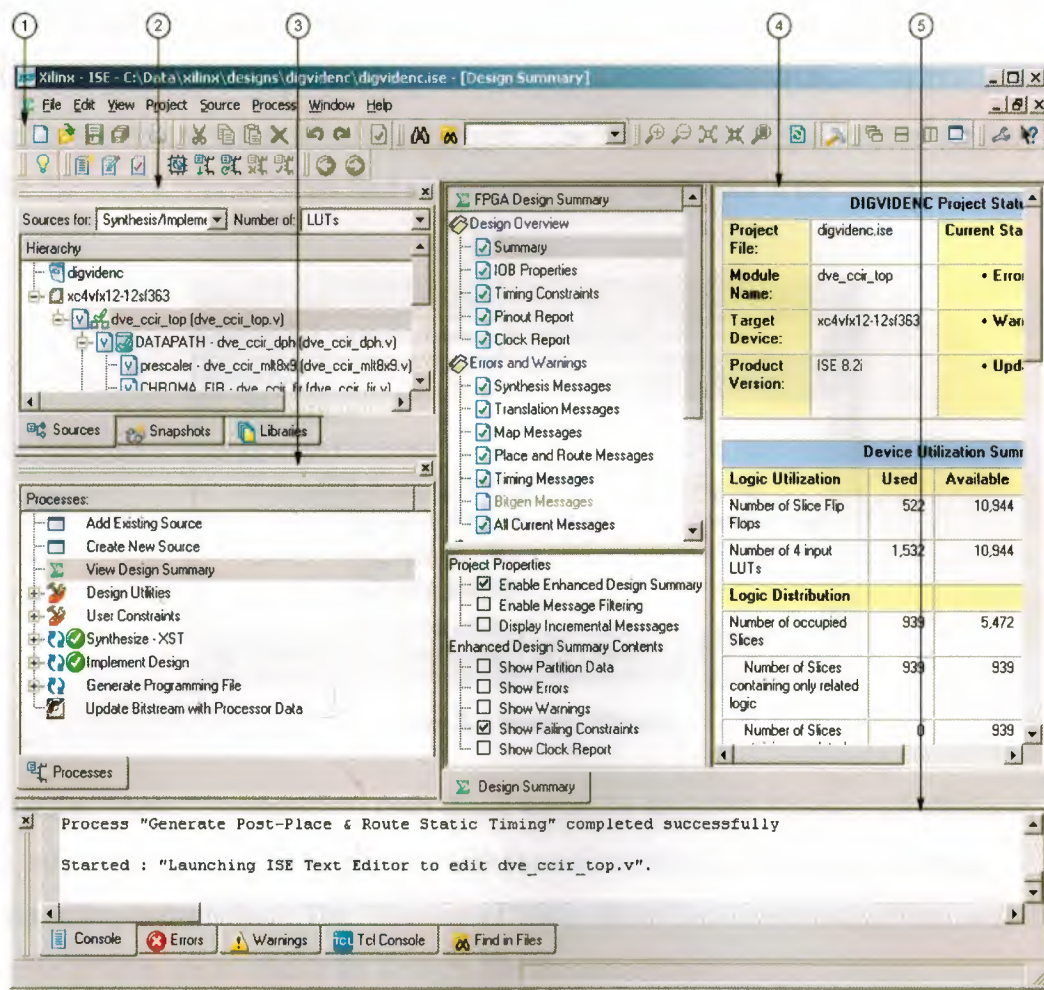


Figure 2. 1 Project Navigator Main Window

- 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 our 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 we create and add to our project, as shown in the following figure.

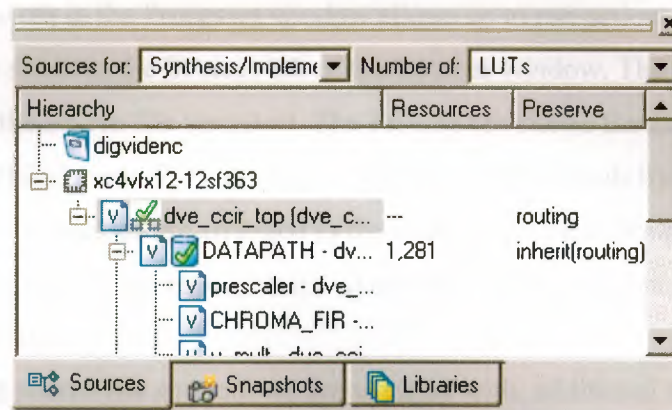


Figure 2. 2 Source Window

The Design View ("Sources for") drop-down list at the top of the Sources tab allows us to view only those source files associated with the selected Design View (for instance, Synthesis/Implementation). The "Number of" drop-down list, Resources column, and Preserve column are available for designs that use Partitions.

The Sources tab shows the hierarchy of our design. We 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 we select determines the processes available in the Processes window. We can double-click a source file to open it for editing in the Workspace. For information on the different 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.

Depending on the source file and tool we 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

- Schematic Editor: Symbols tab
- Technology Viewer: Design tab
- Timing Analyzer: Timing tab

2.2.4. Using the Processes Window

The Processes tab in the Processes window allows us to run actions or "processes" on the source file we select in the Sources tab of the Sources window. The processes change according to the source file we select. The Process tab shows the available processes in a hierarchical view. We 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 we are working with, additional tabs are available in the Processes window:

- Always available: Processes tab
- Floorplan Editor: Design Objects tab, Implemented - Selection tab
- ISE Simulator: Hierarchy Browser tab
- Schematic Editor: Options tab
- Timing Analyzer: Timing Objects tab

2.2.5. Process Types

The following types of processes are available as we work on our design:

- Tasks 

When we run a task process, the ISE software runs in "batch mode," that is, the software processes our 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 instance, the Synthesis Report or Map Report. When we run a report process, the report appears in the Workspace.

- Tools 


When we run a tools process, the related tool launches in standalone mode or appears in the Workspace where we can view or modify our design source files. 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 we work on our design, we may make changes that require some or all of the processes to be rerun. For example, if we edit a source file, it may require that the Synthesis process and all subsequent process be rerun. Project Navigator keeps track of the changes we 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, we 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 we made design changes, which require that the process be rerun. If this icon is next to a report process, we 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, we can do any of the following:

- Double-click the process

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

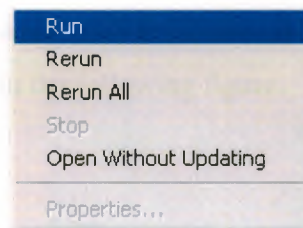





Figure 2. 3 Run Comment

- 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 we run a process, Project Navigator automatically processes our design as follows:

- Automatically runs lower-level processes

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

- Automatically runs preceding processes

When we run a process, Project Navigator runs any preceding processes that are required, thereby "pulling" our design through the design flow. For example, to pull our design through the entire flow, double-click Generate Programming File.

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

If we 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 we change our UCF file, the Synthesize process remains up to date, but the Translate process becomes out of date. If we run the Map process, Project Navigator runs Translate but does not run Synthesize.

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, we can right-click while positioned over the process and select Properties from the popup menu, as shown in the following figure.

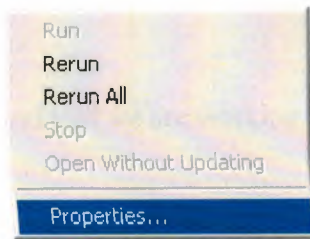


Figure 2. 4 Properties Comment

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

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

2.2.9. Using the Workspace

When we open a project source file, we open the Language Templates, or run certain processes, such as viewing reports or logs, the corresponding file or view appears in the Workspace. We can open multiple files or views at one time. Tabs at the bottom of the Workspace show the names for each file or view. A tab is clicked to bring it to the front. To open a file or view in a standalone window outside of the Project Navigator Workspace, the Float toolbar button is used. To dock a floating window, the Dock toolbar button is used.

- Float 
- Dock 

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

2.2.10. Using the Transcript Window

The Console tab of the Transcript window shows output messages from the processes we run. If a line number appears as part of the message, we 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 we are working with, additional tabs are available in the Transcript window:

- Always available: Console tab, Errors tab, Warnings tab, Tcl Console tab, Find in Files tab.
- 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. To execute a command a toolbar button click once on. To see a short popup description of a toolbar button, the mouse pointer is holding 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, the Help toolbar button  is clicked, and then the toolbar button is clicked for which we want Help. For more information on getting Help, we should see Using Xilinx Help.

2.3. Creating a Project

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

To Create a Project

1. Select File > New Project.

2. In the New Project Wizard Create New Project page, steps are as follows:

- In the Project Name field, we enter a name for the project. It follows the naming conventions described in File Naming Conventions.
- In the Project Location field, we enter the directory name or browse to the directory.
- In the Top-Level Source Type drop-down list, we select one of the following:

- HDL

We select this option if our 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

We select this option if our 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 our design to structural HDL before implementation; therefore, we *must* specify a synthesis tool when working with schematic projects, as described in step 5.

3. Click Next.

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

- Product Category
- Family

To target a Spartan-3L™ device, Spartan-3™ should be selected as the family. When creating an EDIF project, the device family information is read from our EDIF project file, and changing the device family is *not* recommended.

- Device

To target a Spartan-3L device, device that ends in L should be selected, such as xc3s2000L.

- Package
- Speed
- Top-Level Source Type

This is automatically set.

- Synthesis Tool

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

- 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.

- Simulator

We select one of the following simulators and the HDL language for simulation. The language we select determines the default language in which to generate simulation netlists and other generated files that affect simulation. We can also select the language in which to generate files by setting process properties as described in Setting Process Properties.

- ISE Simulator (Xilinx®, Inc.)

This simulator allows us to run integrated simulation processes as part of our ISE design flow.

- ModelSim (Mentor Graphics®, Inc.)

We can run integrated simulation processes as part of our ISE design flow using any of the following ModelSim® editions: ModelSim Xilinx Edition (MXE), ModelSim MXE Starter, ModelSim PE, or ModelSim SE™.

- Others

We select this option if we do not have ISE Simulator or ModelSim installed or if we want to run simulation outside of Project Navigator. This

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

- **Enable Enhanced Design Summary**

We select this option to show the number of errors and warnings for each of the Detailed Reports in the Design Summary.

- **Enable Message Filtering**

We select this option to show the number of messages we filtered in the Design Summary. We must enable this option, filter messages, and then run the software to show the number of filtered messages.

- **Display Incremental Messages**

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

5. If we are creating an HDL or schematic project, we click Next, and optionally, we create a new source file for our project in the Create New Source page. We can only create one new source file while creating a new project. We can create additional new sources after our project is created.
6. We click Next, and optionally, we add existing source files to our project in the Add Existing Sources page.
7. We click Next to display the Project Summary page.
8. We click Finish to create the project.

If we prefer, we 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, we should deselect the Use new project wizard option in the ISE General Options page of the Preferences dialog box.

What to Expect:

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

design from design entry through implementation to programming the targeted Xilinx device. For information on changing project properties, we see Changing Project, Source, and Snapshot Properties.

What to Do Next:

We can perform any of the following:

- To create and add source files to our project.
- To add existing source files to our project.
- To run processes on our source files.

2.4. Creating Traffic Light Project on ISE

First of all, we click the shortcut icon of the Xilinx – ISE to open Project Navigator Main Window. Then, to create a new project, we select File > New Project.

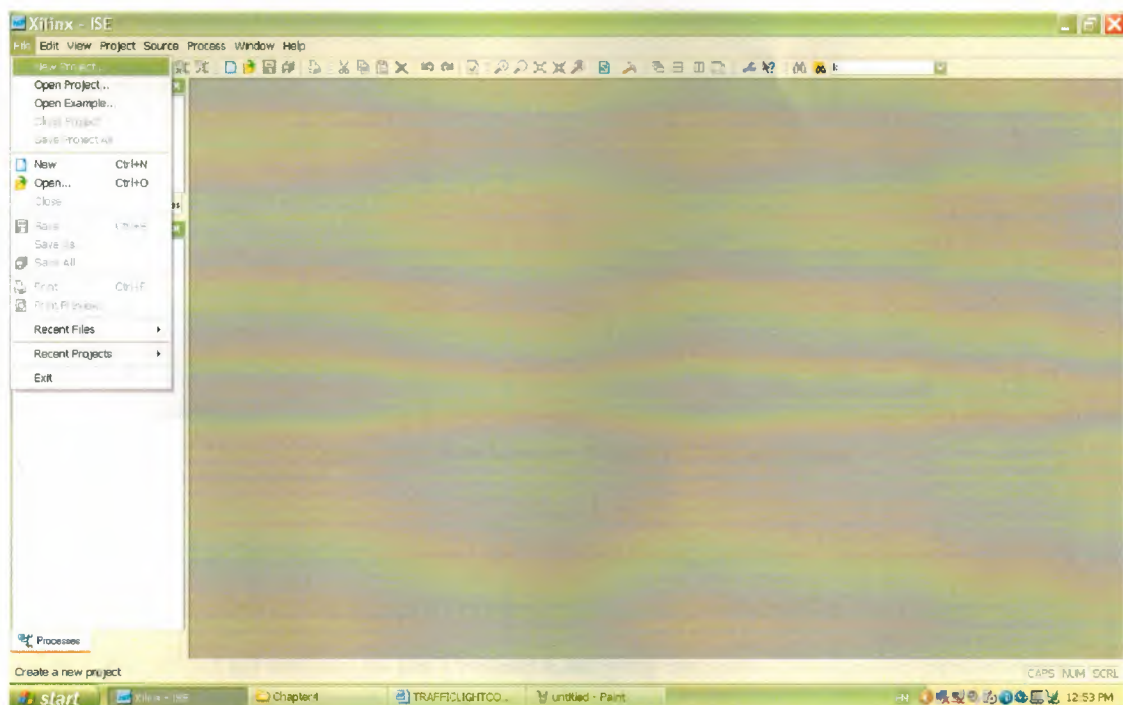


Figure 2. 5. Project Navigator Main Window

In the New Project Wizard we can create New Project page, doing the following:

- In the Project Name field, we enter project name “Traffic_Lights_Controller”.

- In the Project Location field, we enter the directory name “C:\VHDLProjects\Traffic_Lights_Controller\” to the directory.
- In the Top-Level Source Type drop-down list, we select HDL in which our top level design file is a VHDL, Verilog, or ABEL (for CPLDs). An HDL Project can include lower levels models of different files types.

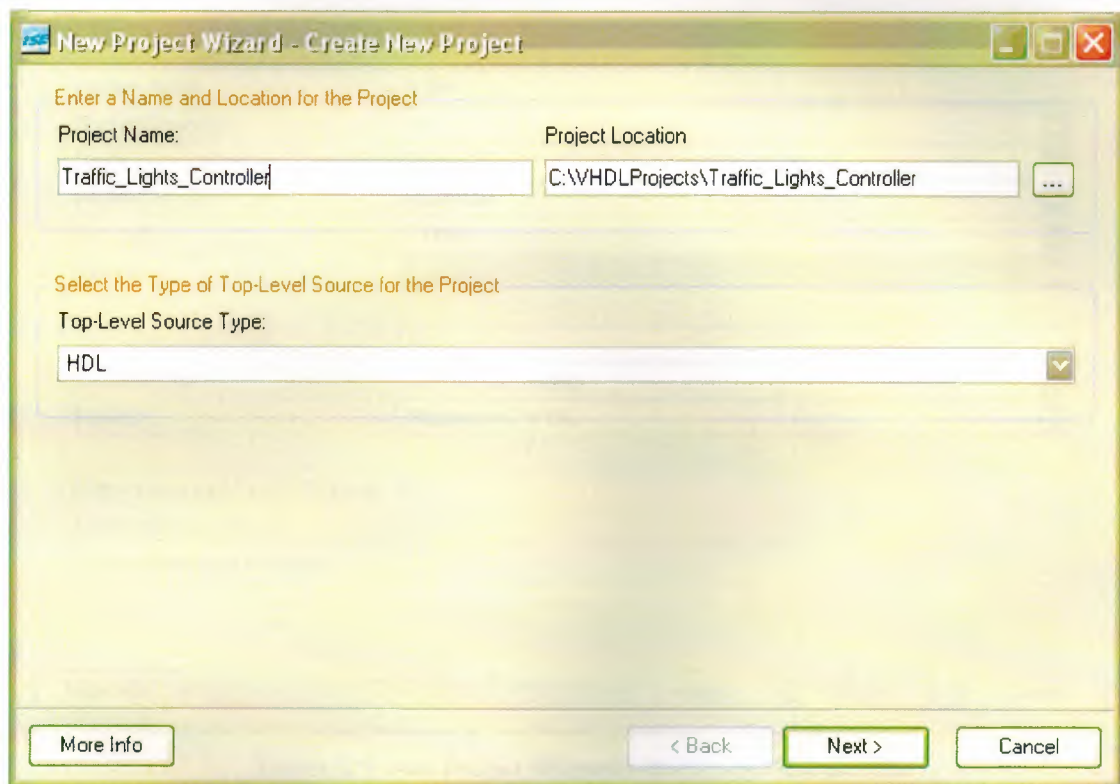


Figure 2. 6. New Project Window – Project Name.

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

- We select all section in the Product Category section.
- We select Spartan-3E family in the Family section.
- We select XC3S100E device in the Device section.
- We select TQ144 package in the Package section.
- We construct speed -4 in the Speed section.
- HDL is automatically selected in the Top-Level Source Type section.

- XST (VHDL/Verilog) is selected in the Synthesis Tool section.
- We select ModelSim-XE VHDL simulator in the Simulator section.

Then we click Next and not add any new source or existing source in the wizard that is optional. After finishing Project Navigator, it creates the project file Traffic Lights Controller ise, in the directory we specified.

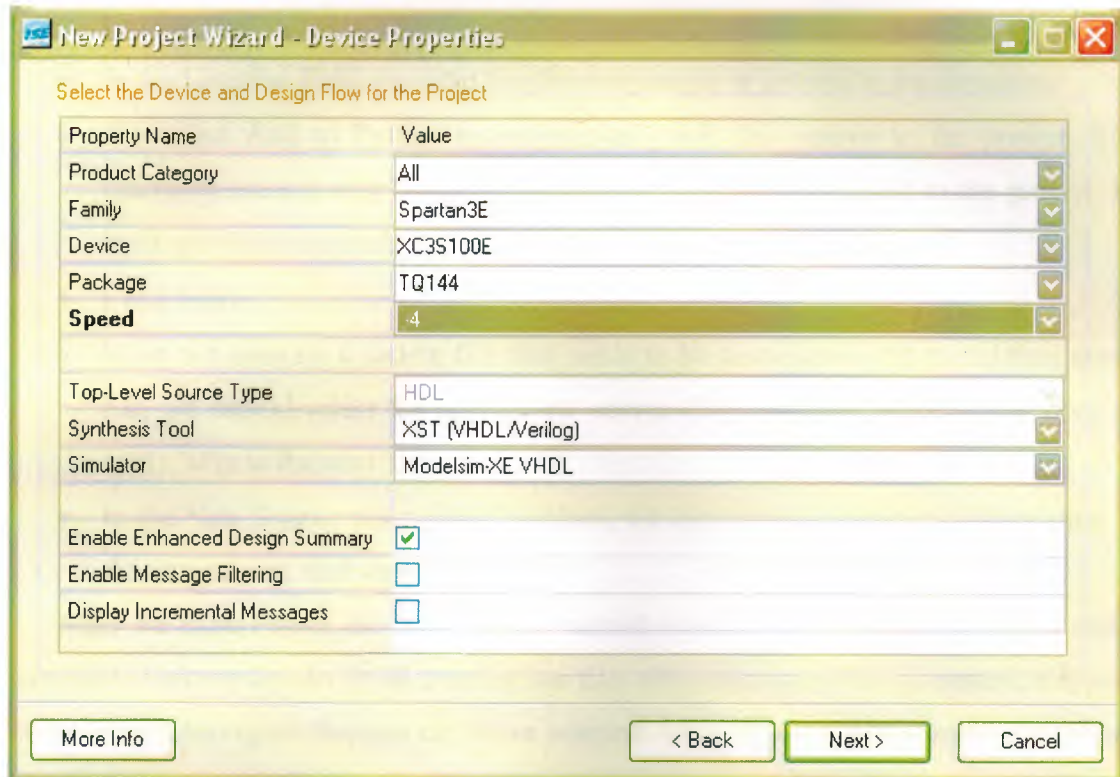


Figure 2. 7. New Project Window – Device and Design Flow

2.5. Creating a Source File

A source file is any file that contains information about a design. Project Navigator provides a wizard to help us create new source files for our project. If we are targeting a Spartan-3A or Virtex-5 device, we can use the New Source Wizard to pre-assign package pins for an empty project. For details, Pre-Assigning Package Pins in the New Source Wizard.

What to Do First, We open a project in Project Navigator.

To Create a Source File:






- We select Project > New Source.

- In the New Source Wizard, we select the type of source we want to create. Different source types are available depending on our project properties (top-level module type, device type, synthesis tool, and language). Some source types launch additional tools to help us create the file, as described in Source File Types.
- We enter a name for the new source file in the File Name field. Then we follow the naming conventions described in File Naming Conventions.
- In the Location field, we enter the directory name or browse to the directory.
- We select Add to Project to automatically add this source to the project. State machines created with StateCAD cannot be automatically added to the project. We must add them manually.
- Click Next.
- If we are creating a source file that needs to be associated with an existing source file, we should select the appropriate source file, and click Next. If this does not apply, skip to the next step.
- In the New Source Information window, we can read the summary information for the new source, and we click Finish.

After we click Finish, the New Source wizard closes. In some cases, a related tool is launched in which we can finish creating our file. After the source file is created, it appears in the Project Navigator Sources tab. If we selected Add to Project when creating the source file, the file is automatically added to the project.

Sources File Types:

The following table shows the source file types that appear in the Project Navigator Sources tab. Available source types vary depending on our project properties (top-level module type, device type, synthesis tool, and language). The last column describes what to expect when creating the file with the New Source wizard and, if applicable, includes the tool launched when using the New Source wizard or when editing the file from Project Navigator.

File Type	Extension	Icon	Description	New Source Wizard Behavior/Tool Launched
Project	.ise		Contains process property settings, status, and information for managing the ISE™ project.	N/A
Schematic	.sch		Contains a schematic design.	Opens the schematic file in the Project Navigator Workspace. For details, see the Schematic Overview.
State diagram	.dia		Contains a state diagram file.	Launches StateCAD in which we can define your state diagram. For details, see Working with State Machines.
Test Bench Waveform	.tbw		Contains a graphical representation of a test bench that can be converted to an HDL test bench or test fixture.	Prompts you to associate the file with a source and opens the Test Bench Waveform Editor in the Project Navigator Workspace with the signals populated. For details, see the ISE Simulator Help. This file is for use with the Xilinx® Test Bench Waveform Editor only.
Undefined	N/A		Contains an instantiated module that has not been	N/A




			added to the ISE project but is referenced by a source file in the ISE project.	
User Document	.doc, .txt, .wri		Contains user information that is not implemented with the project, for example, supporting documentation.	N/A Must be added to the project.
VHDL Module	.vhd		Contains VHDL design code.	Opens the file in the text editor we specify in the Editor Options page of the Preferences dialog box.
VHDL Test Bench	.vhd		Defines the stimulus to the ports of an HDL file.	Prompts we to associate the file with a VHDL source and then opens a skeleton test bench file in the text editor we specify in the Editor Options page of the Preferences dialog box.

Figure 2. 8. Source File Type

2.6. Creating Counter.vhd VHDL Module

To create a new source file we click Project > New Source. In the New Source Wizard, we select the type of source is VHDL Module and its name given as Counter to create 4 bit counter.

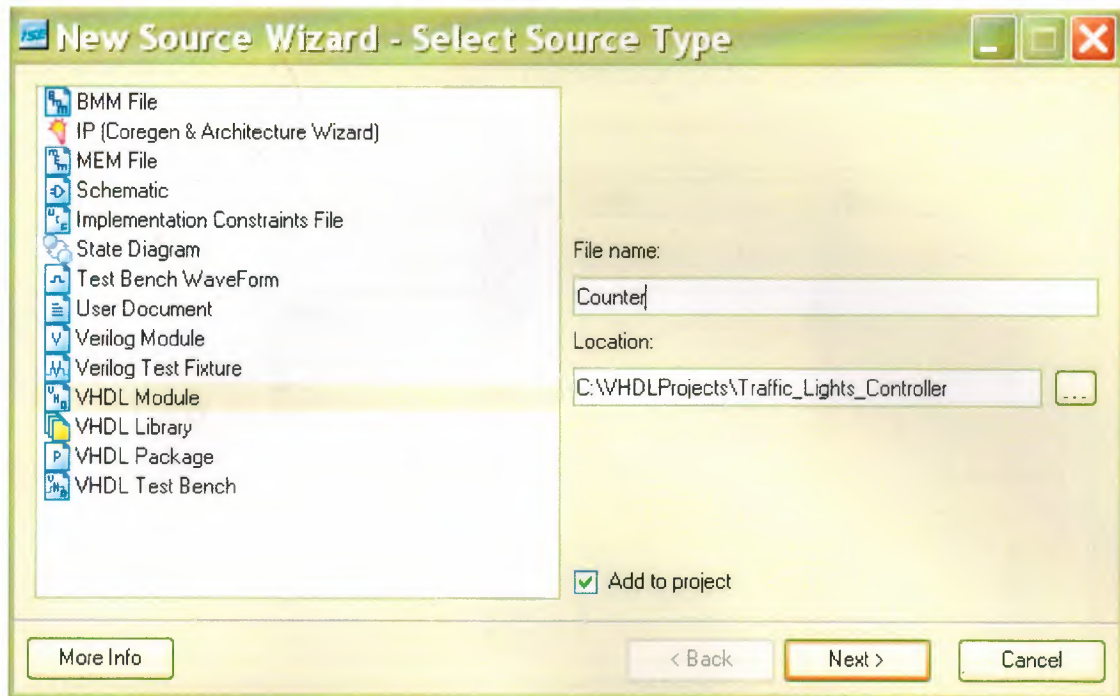


Figure 2. 9. New Source Window

After Next button clicking it pass to the define part and we declare three ports: “clock”, “reset”, and “count.” The clock and reset ports direction signed “in” also count “inout” with 4 bit bus. After clicking Next the source file generated automatically and added to the project.

A typical VHDL module consists of library declarations, an entity, and architecture. The library declarations are needed to tell the compiler which packages are required. The entity declares all ports associated with the design. Count (3 up to 0) means that count is a 4-bit logic vector. This design has two inputs –clock and –reset also one output, a 4 bit bus called “count.” The actual functional description of the design appears after the begin statement in the architecture. The function of this design is to increment a signal “count” when clock equal to 1 and there is an event on the clock. This is resolved into a positive edge. The reset is synchronous as is evaluated before the clock action. The area still within the architecture but before the begin statement is where declarations reside.



Figure 2. 10. VHDL Source Window

If we notice that a file called “counter.vhd” has been added to the project in the Sources in Project Window of the Project Navigator. And we double click on counter.vhd source and we added to process source on it. These codes are shown below.

```
process (CLOCK,RESET)
begin
    if RESET = '1' then
        COUNT <= "0000";
    elsif CLOCK = '1' and CLOCK'event then
        COUNT <= COUNT + 1;
    end if;
end process;
```

We define a clock process with parameters CLOCK and RESET in which whenever reset is active COUNT is cleared. On the other hand, if CLOCK rising edge event and CLOCK active are occurred together that are initiate to counter to up one by one.

2.7. Creating a Test Bench Waveform

The ISE Simulator tools provide a Waveform Editor in which we can graphically create test benches or test fixtures. Using the Waveform Editor window, we can specify stimulus, expected outputs and test bench length, using waveforms and menus.

By editing waveforms graphically, using mouse-clicks and menus, we can see our design stimulus and expected simulation results in a familiar waveform view. We do not need any knowledge of HDL or language scripting to verify that our design will behave as we intended. We can save our waveforms and test bench properties into a Test Bench Waveform (.tbw) file that is added to our ISE project. We can then use this TBW file to drive our design simulation, in the same way we would use an HDL test bench.

At any point in our Test Bench Waveform design, we can choose to view the equivalent HDL test bench. We can also write out the equivalent HDL test bench or test fixture and add it to our project. Therefore if we wish, we may begin our test bench creation graphically, and after the initial test bench framework is written, we can choose to continue test bench development in HDL, outside of the Waveform Editor.

2.8. Creating a counter_tb Test Bench Waveform

Before simulate a VHDL file, we must first create a testbench. Again from the project menu, we select new source and we indicate the source type as Test Bench Waveform that is file name given "counter_tb."

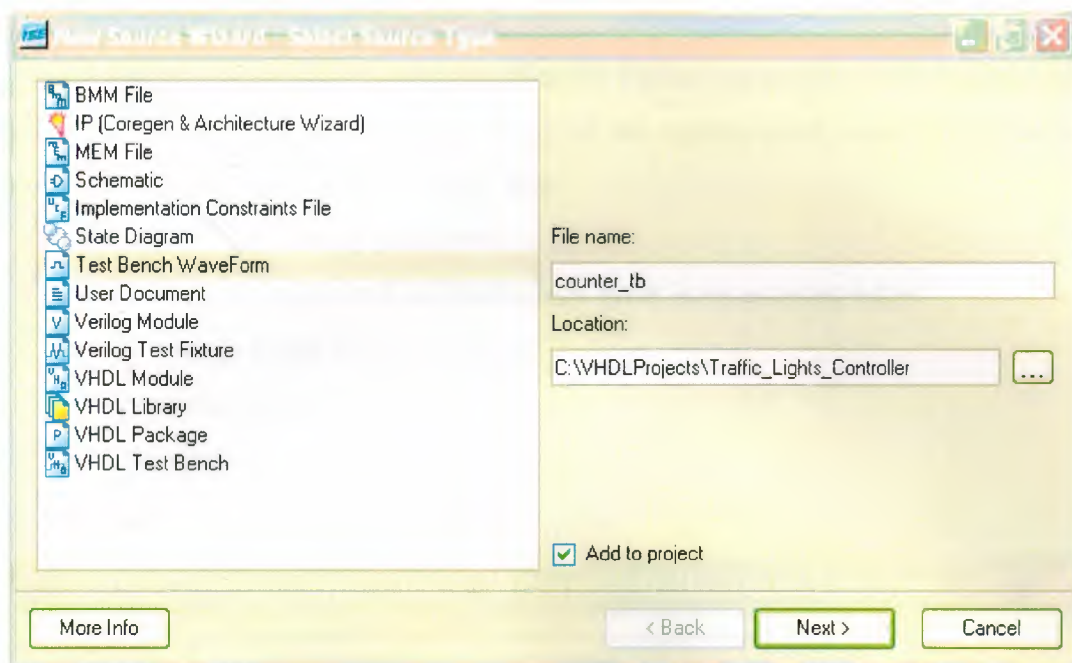


Figure 2. 11. Adding Test Bench WaveForm

After clicking the Next button the testbench is going to simulate the counter module, so when asked which source we want to associate the source with, selected “Counter” and click Next button.

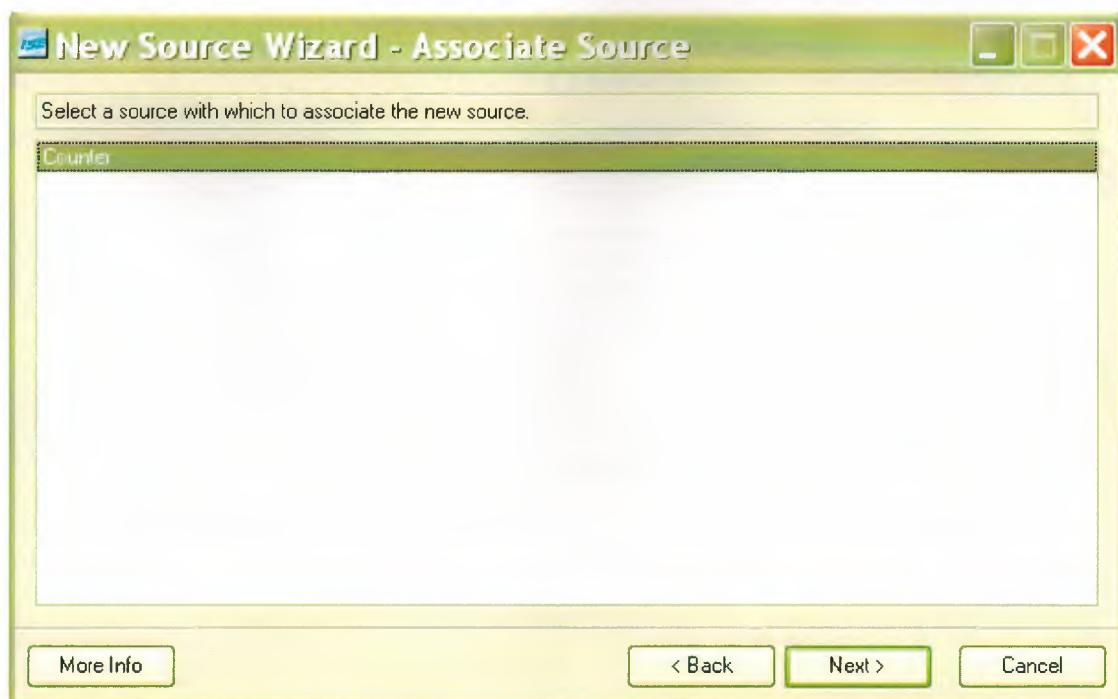


Figure 2. 12. Associate Source Window

We review the information and click the Finish button. The HDL Bench tool now reads in the design. We set the frequency of the system clock, setup requirements, and output delays in the “Initialize Timing” box.

We set initialize timing as follows:

- We select single clock in which clock event is set as rising edge.
- We select Clock high time 100 ns.
- We select Clock low time 100 ns.
- We set input setup time 15 ns.
- We set output valid delay 15 ns.

Initial Timing and Clock Wizard - Initialize Timing

Maximum output delay

Minimum input setup

Clock high for

Clock low for

Clock Timing Information

Inputs are assigned at "Input Setup Time" and outputs are checked at "Output Valid Delay".

☒ Rising Edge ☐ Falling Edge ☐ Dual Edge (DDR or DET)

Clock High Time: 100 ns

Clock Low Time: 100 ns

Input Setup Time: 15 ns

Output Valid Delay: 15 ns

Offset: 0 ns

Clock Information

☒ Single Clock ☐ Multiple Clocks ☐ Combinatorial (or internal clock)

clock

Combinatorial Timing Information

Inputs are assigned, outputs are decoded then checked. A delay between inputs and outputs avoids assignment/checking conflicts.

Check Outputs: 50 ns After Inputs are Assigned

Assign Inputs: 50 ns After Outputs are Checked

Global Signals

☐ PRLD (CPLD) ☐ GSR (FPGA)

High for Initial: 100 ns

Initial Length of Test Bench: 1000 ns

Time Scale: ns

☐ Add Asynchronous Signal Support

More Info < Back Finish Cancel

Figure 2. 13. Initial Timing and Clock Wizard

If we note that the blue cells are for entering input stimulus and the yellow cells are for entering expected response. When entering a stimulus, we are clicking the left mouse button on the cell will cycle through the available values for that cell.

We open a pattern text field and button by double clicking on a signal's cell or a single clicking on a bus cell. From this pattern window, we can enter a value in the text field or we can click on the pattern button to open a pattern wizard.

We enter the input stimulus as follows:

- We set the reset cell below CLK cycle 1 to a value of "1."
- We set the reset cell below CLK cycle 2 to a value of "0."

Then, we click the yellow COUNT [3:0] cell under CLK cycle 1 and again click Pattern button to launch the Pattern Wizard and set parameters as follows:

- Pattern Type is set count up.
- Number of cycle is set 16.
- Radix is selected binary.
- Initial value is set 0000.
- Terminal value is set 1111.
- Increment by is set 1.
- Count Every is set 1.

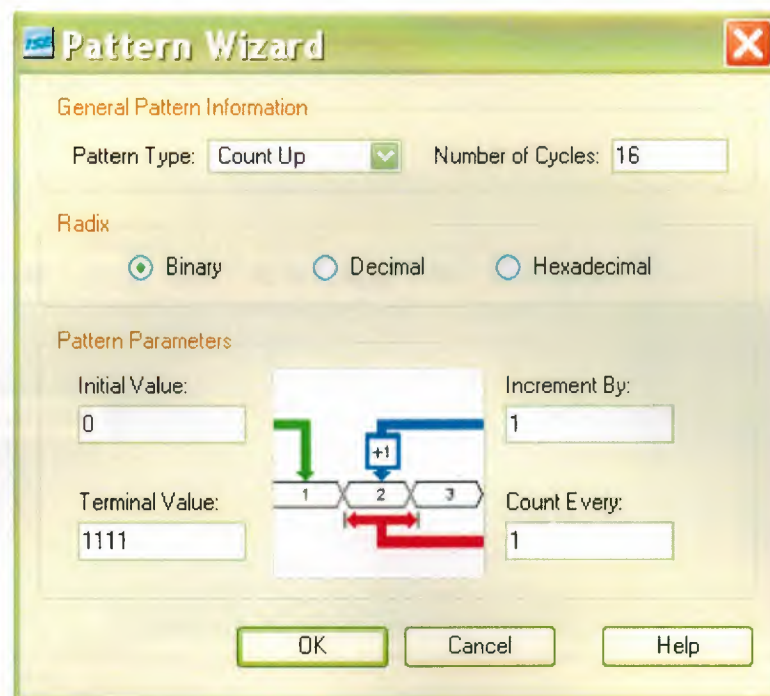


Figure 2. 14. Pattern Wizard Window.

After all we see that it's waveform look like Figure 4 – 10.

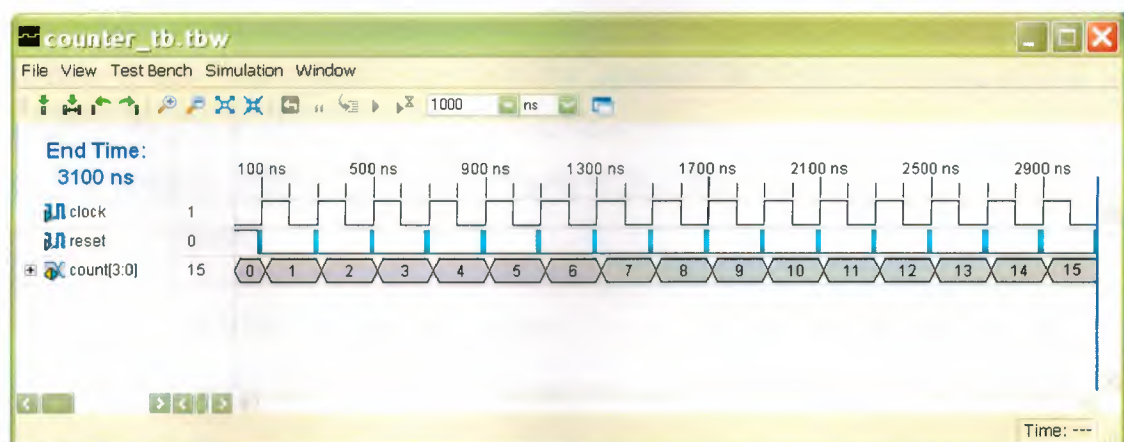


Figure 2. 15. Waveform Window

Then we click File > Save to save the waveform and closed the HDL Benchner tool. Now that the testbench is created, we can simulate the design.

We select "counter_tb.tbw" in the ISE source window and we expand the ModelSim simulator by clicking in the process window. Then right click on Simulate



Behavioral VHDL model we can select properties in the simulation run time field, type “all” and pressed OK.

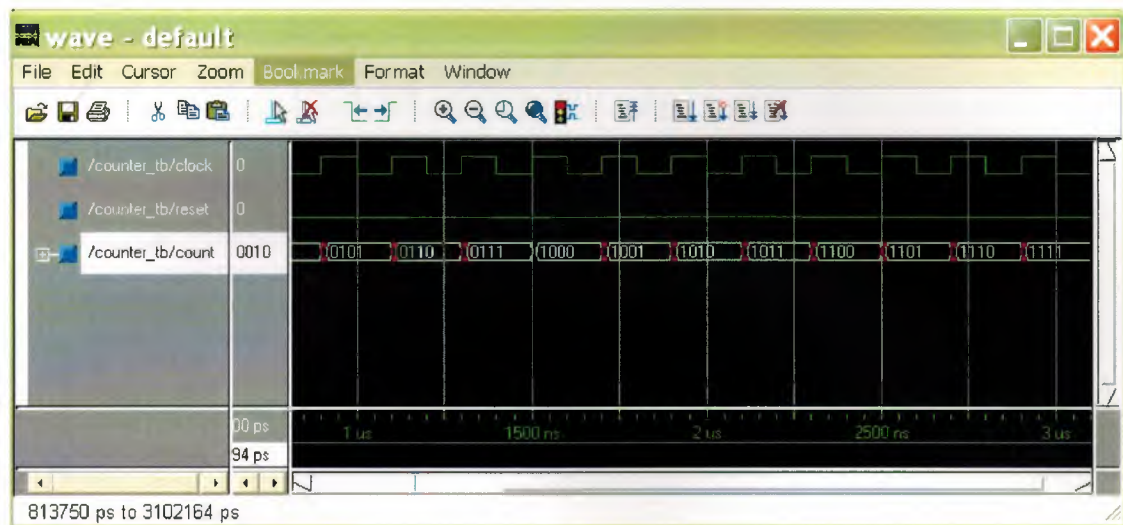


Figure 2. 16. Wave Window

ISE software automates the simulation process by creating and launching a simulation macro file (a “.do” file, or a “.fdo” file). This creates the design library, compiles the design and testbench source files, and calls a user editable “.do” file called “counter_tb.udo.” it also invokes the simulator, opens all the viewing windows, adds all the signals to the list window, and runs the simulation for the time specified by the simulation run time property. After all we see that figure 4- 11. Then, we use File > Exit to close the ModelSim simulator.

2.9. Creating a State Machine Diagram

For our traffic light design, the counter acts as a timer that determines the transitions of a state machine.

The state machine will run through four states, each state controlling a combination of the three lights.

- State 1: Red Light
- State 2: Red and Amber Light
- State 3: Green Light
- State 4: Amber Light

To invoke the state machine editor, we select new source from the project menu. Highlight State Diagram and its name given as “stat_mac.dia.” clicked next button, then the finish button.

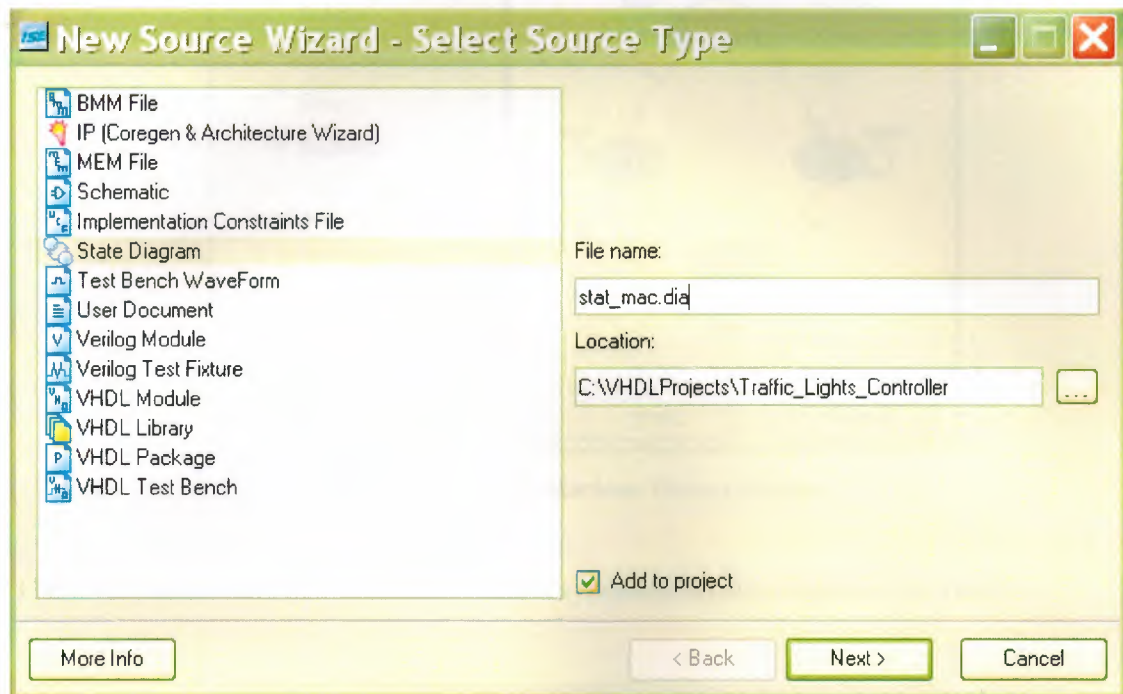


Figure 2. 17. New Source Window

To open the state machine wizard we clicked the Draw State Machines button in the main toolbar. We set the number of sates to “4” and hit next button.

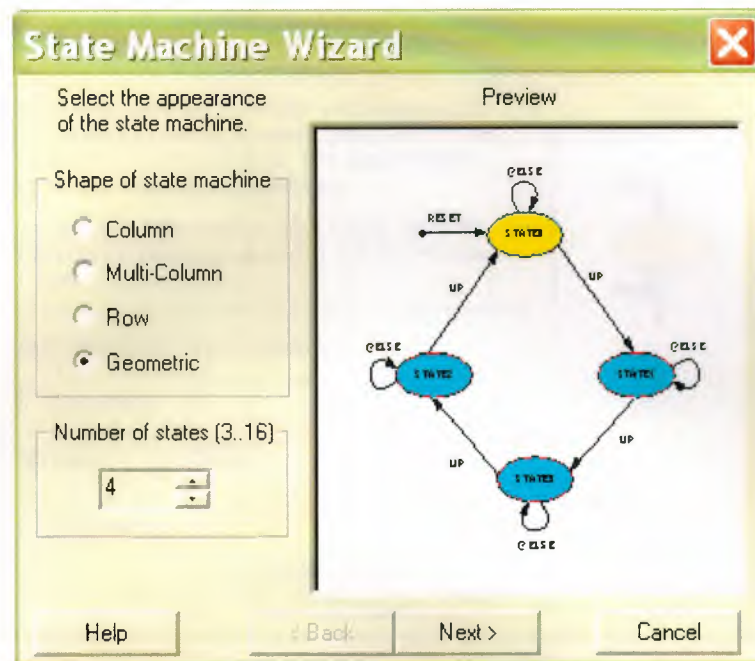


Figure 2.18. State Machine Wizard Window

Then, we press to the next button to build a synchronous state machine for reset.

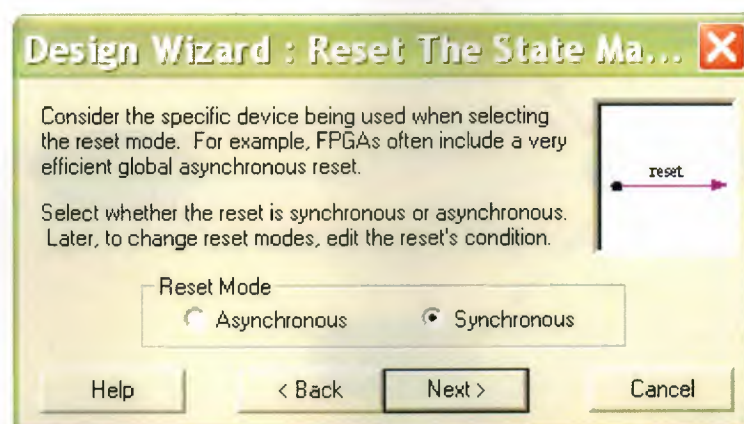


Figure 2.19. Reset State Machine Wizard

We setup transition box depending on the conditions and also we substitute type "TIMER" in the next field and finish it. We hit state CAD floor, all state appear on it. It shown in figure 2.17.



Figure 2.20. Setup Transition Window

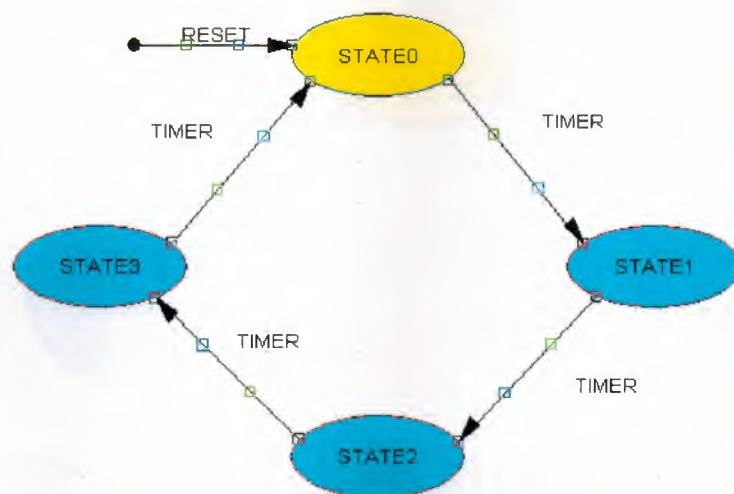


Figure 2.21. State Diagram Scheme

Our design has three outputs named RD, AMB, and GRN in the logic wizard; we declare these outputs in the DOUT field. At this stage we rename all state and declared outputs as follows:

- State 0 renamed as RED with outputs RD = 1, AMB = 0, and GRN = 1.
- State 1 renamed as REDAMB with outputs RD = 1, AMB = 0, and GRN = 1.
- State 2 renamed as GREEN with outputs RD = 1, AMB = 0, and GRN = 1.

- State 3 renamed as AMBER with outputs RD = 1, AMB = 0, and GRN = 1.

After this operation we double click on transition line between state "RED" and state "REDAMB." In the edit condition window, we set a transition to occur when timer is 1111 by editing the condition field to TIMER = "1111." We repeated for all other states.

Transition REDAMB to GREEN, TIMER = "0100", GREEN to AMBER, TIMER = "0011", AMBER to RED, TIMER = "0000." Hence, the traffic light completes a RED, REDAMB, GREEN, and AMBER once every three cycles of the counter.

Finally, we declare the vector TIMER by clicking on the button on the left hand side of the toolbar. After dropping the marker on the page, double clicked on it, we enter the name "TIMER" with a width of 4 bits (Range 3:0). We click OK button it shows that in figure 2-18.

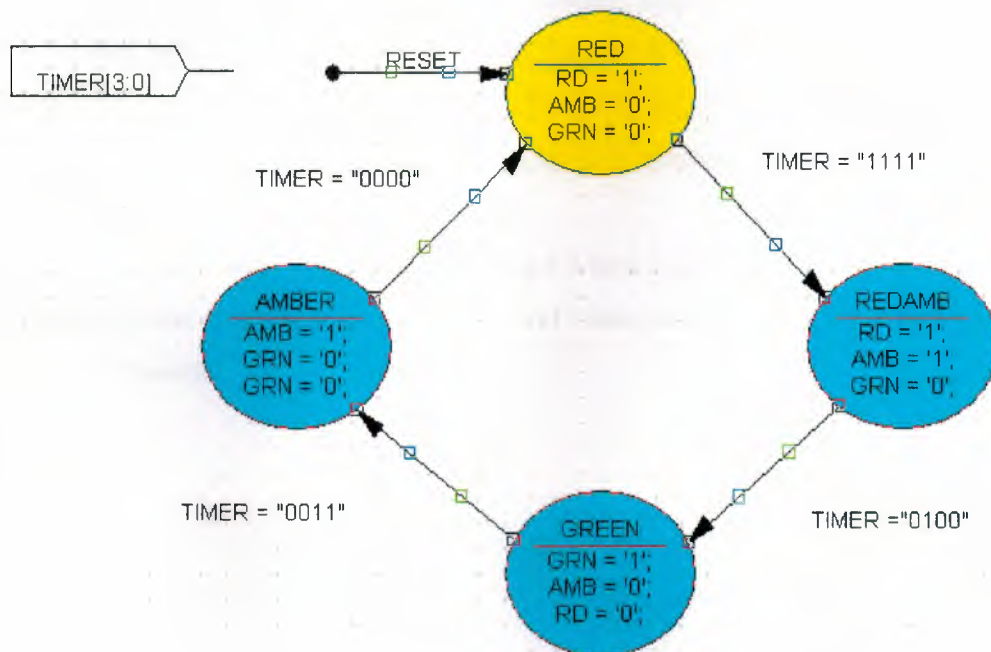


Figure 2. 22. State Machine Drawing

Also we can see all variables on the state machine clicking variables section. It is shown figure 2- 19.

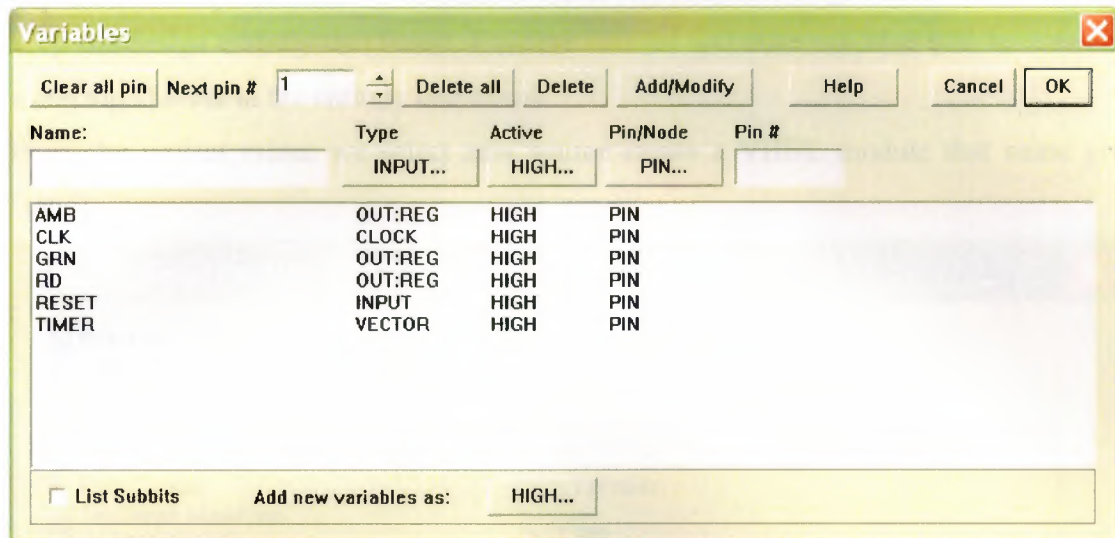


Figure 2. 23. Variables Window

Then we click on the Generate HDL button on the top toolbar. The results window should read "Compiled perfectly." We close the dialog box and the generated HDL Browser Window saving with StateCAD. The state machine can now be added to the ISE Project.

In the next step, we go to the Project Menu and select Add Source in the Project Navigator. In the existing source box, we find "stat_mac.dia." if it not added automatically to the ISE Project.

2.10. Creating Top – Level VHDL Design

At this point in the flow, two modules in the design are connected together by a top level file. Some designers like to create a top level schematic diagram, while others like to keep the design entirely text based. First the counter and state machine will be connected using top.vhd file in the entirely text based.

From the project menu, we select new source create a VHDL module that name given “top.”

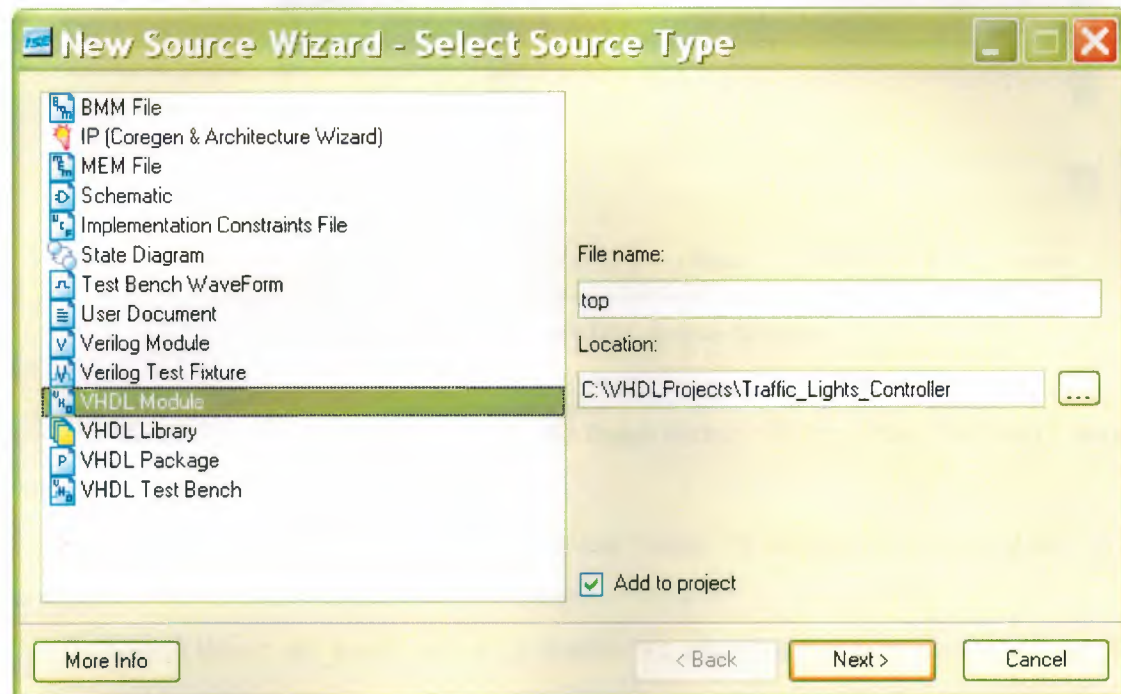


Figure 2. 24. New Source Window

Then, we click on the next button and fill out the define VHDL Source dialog box, as shown in figure 2.20.



Figure 2. 25. Define VHDL Source Window

We click on the next button, then the finish button. Our new file, “top.vhd,” should look like Figure 4- 19.

In the top.vhd, we declare a signal called “timer” by adding the following line in the component declarations inside the architecture:

Signal timer: std_logic_vector (3 downto 0); after that we connect the counter and state machine instantiated modules. When we save “top.vhd,” the source windows automatically rearrange depending on the instantiated modules.

Now we can simulate the entry design. First we add a new testbench waveform source, associated with the module “top.” In the waveform diagram, we enter the input stimulus as follows:

- We set RESET cell below CLK cycle 1 to a value of “1.”
- We click the RESET cell below CLK cycle 2 to reset if low
- We scroll to the 64th clock cycle right clicking and selecting set end of testbench.

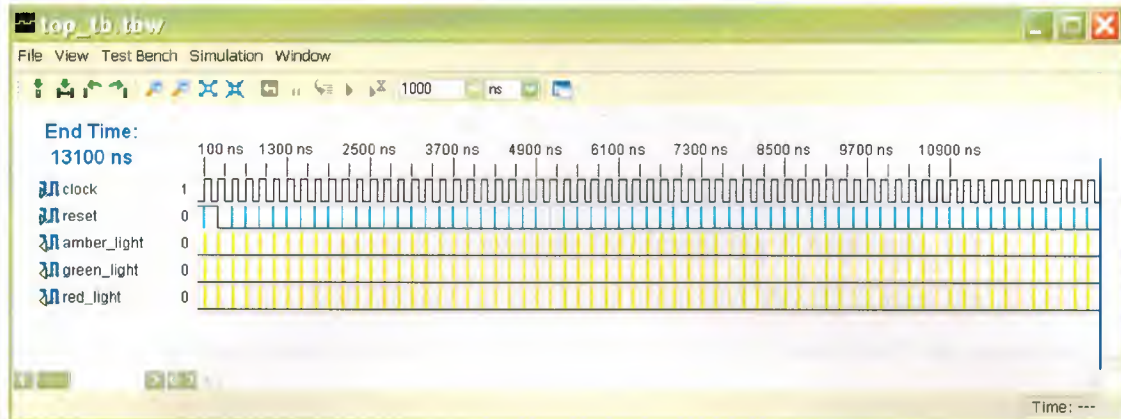


Figure 2.26. Waveform Diagram

Then we click File > Save to save the waveform and closed the HDL Benchner tool. Now that the testbench is created, we can simulate the design.

We select "top_tb.tbw" in the ISE source window and we expand the ModelSim simulator by clicking in the process window. Then right click on Simulate Behavioral VHDL model we can select properties in the simulation run time field, type "all" and pressed OK.

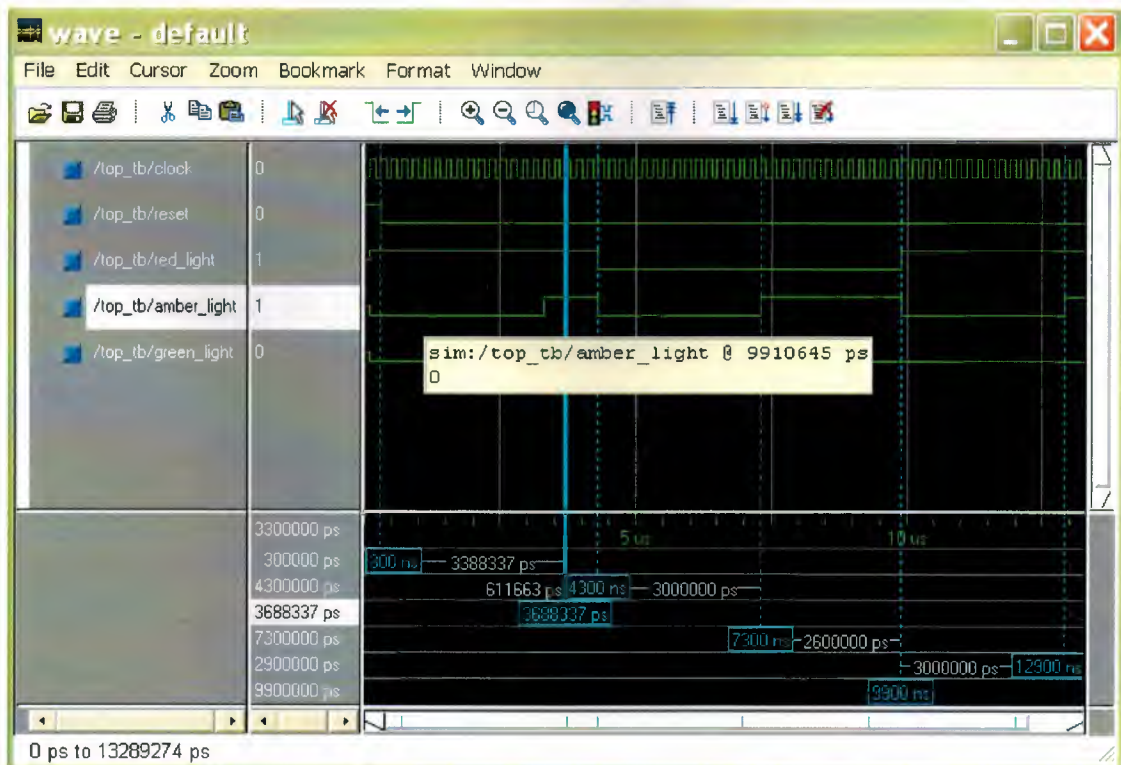


Figure 2.27. Waveform Window

CHAPTER THREE: TRAFFIC LIGHT CONTROLLER IN VHDL

3.1. Very High Speed Integrated Circuit Hardware Description Language

VHDL development was initiated originally from the American Department of Defense (DoD). They requested a language for describing a hardware, which had to be readable for machines and humans at the same time and strictly forces the developer to write structured and comprehensible code, so that the source code itself can serve as a kind of specification document. Most important was the concept of concurrency to cope with the parallelism of digital hardware. Sequential statements to model very complex functions in a compact form were also allowed.

In 1987, VHDL was standardized by the American Institute of Electrical and Electronics Engineers (IEEE) for the first time with the first official update in 1993. Apart from the file handling procedures these two versions of the standard are compatible. The standard of the language is described in the Language Reference Manual (LRM).

A new and difficult stage was entered with the effort to upgrade VHDL with analogue and mixed-signal language elements. The upgrade is called VHDL-AMS (analogue- mixed- signal) and it is a superset of VHDL. The digital mechanisms and methods have not been altered by the extension.

For the time being, only simulation is feasible for the analogue part because analogue synthesis is a very complex problem affected by many boundary conditions. The mixed signal simulation has to deal with the problem of synchronizing the digital- and analogue simulators, which has not been solved adequately, yet.

3.1.1. History of VHDL

The acronym VHDL stands for the VHSIC Hardware Description Language. The VHSIC, refers to the Very High Speed Integrated Circuit program. The Department of Defence sponsored this program. During the program, the increasing complexity of digital systems that were made possible by continuous advanced in semiconductor and packaging technologies was found to have a fundamental impact on the economics of the design of military and space electronic systems.

The idea of being able to simulate "documents" was so obviously attractive in which logic simulators were developed that could read the VHDL files. The next step was the development of logic synthesis tools that read the VHDL, and output a definition of the physical implementation of the circuit. Modern synthesis tools can extract RAM, counter, and arithmetic blocks out of the code, and implement them according to what the user specifies. Thus, the same VHDL code could be synthesized differently for lowest cost, highest power efficiency, highest speed, or other requirements.

VHDL has a syntax that is essentially a subset of the Ada programming language, along with an added set of constructs to handle the parallelism inherent in hardware designs. VHDL is strongly-typed and case insensitive. A team of DoD contractors was awarded the contract to develop the language and the first version was released in 1985. the language was subsequently transferred to the IEEE for standardization, after which representatives from industry, government and academe were involved in its further development. The language was ratified and become IEEE 1076-1987 standards, after five years with the addition of new features, forms the 1076-1993 version of the language. Unless otherwise stated, simulation and synthesis compilers will generally support both versions and will provide mechanisms to control which version of the language is being supported.

VHDL is a language that describes digital systems. A simulator will use its descriptions to simulate the behaviour of the system without having to actually construct it. Alternatively, synthesis compilers can utilize such as a description to create descriptions of the digital hardware for implementing the system. Although VHDL has been investigated for its use in describing and simulating analog systems, the language is used predominantly in the design of digital electronic systems. VHDL is a language which is permanently extended and revised. The original standard itself needed more than 16 years from the initial concept to the final, official IEEE standard. When the document passed the committee it was agreed that the standard should be revised every 5 years. The first revision phase resulted in the updated standard of the year 1993.

Independently of this revision agreement, additional effort is made to standardize "extensions" of the pure language reference. These extensions cover for examples packages (std_logic_1164, numeric_bit, numeric_std ...) containing widely needed data types and

subprograms, or the definition of special VHDL subsets like the synthesis subset IEEE 1076.6.

3.1.2. VHDL - Application Field

VHDL is used mainly for the development of Application Specific Integrated Circuits (ASICs). Tools for the automatic transformation of VHDL code into a gate-level netlist were developed already at an early point of time. This transformation is called synthesis and is an integral part of current design flows.

For the use with Field Programmable Gate Arrays (FPGAs) several problems exist. In the first step, Boolean equations are derived from the VHDL description, no matter, whether an ASIC or a FPGA is the target technology. But now, this Boolean code has to be partitioned into the configurable logic blocks (CLB) of the FPGA. This is more difficult than the mapping onto an ASIC library. Another big problem is the routing of the CLBs as the available resources for interconnections are the bottleneck of current FPGAs. While synthesis tools cope pretty well with complex designs, they obtain usually only suboptimal results. Therefore, VHDL is hardly used for the design of low complexity Programmable Logic Devices (PLDs).

VHDL can be applied to model system behavior independently from the target technology. This is either useful to provide standard solutions, e.g. for micro controllers, error correction (de-)coders, etc, or behavioral models of microprocessors and RAM devices are used to simulate a new device in its target environment. An ongoing field of research is the hardware/software co-design. The most interesting question is which part of the system should be implemented in software and which part in hardware. The decisive constraints are the costs and the resulting performance.

3.1.3. VHDL Language and Syntax

VHDL is generally case insensitive which means that lower case and upper case letters are not distinguished. This can be exploited to define own rules for formatting the VHDL source code. VHDL keyword could for example be written in lower case letters and self defined identifiers in upper case letters. This convention is valid for the following slides.

Statements are terminated in VHDL with a semicolon. That means as many line breaks or other constructs as wanted can be inserted or left out. Only the semicolons are considered by the VHDL compiler. List is normally separated by commas. Signal assignments are notated with the composite assignment operator '<='.

Self defined identifier as defined by the VHDL 87 standard may contain letters, numbers and underscores and must begin with a letter. Further no VHDL keywords may be used. The VHDL 93 standard allows defining identifiers more flexible as the next slide will show.

3.1.4. VHDL Structural Elements

- Entity: Interface
- Architecture: Implementation, behavior, function
- Configuration: Model chaining, structure, hierarchy
- Process: Concurrency, event controlled
- Package: Modular design, standard solution, data types, constants
- Library: Compilation, object code

The main units in VHDL are entities, architectures, configurations and packages (together with package bodies). While an entity describes an interface consisting of the port list most of the time, architecture contains the description of the function of the corresponding module. In general, a configuration is used for simulation purposes, only. In fact, the configuration is the only simulatable object in VHDL as it explicitly selects the entity/architecture pairs to build the complete model. Packages hold the definition of commonly used data types, constants and subprograms. By referencing a package, its content can be accessed and used.

Another important construct is the process. While statements in VHDL are generally concurrent in nature, this construct allows for a sequential execution of the assignments. The process itself, when viewed as a whole object, is concurrent. In reality, the process code is not always executed. Instead, it waits for certain events to occur and is suspended most of the time.

A library in VHDL is the logical name of a collection of compiled VHDL units (object code). This logical name has to be mapped by the corresponding simulation or synthesis tool to a physical path on the file system of the computer.

3.1.5. Declaration of VHDL Objects

A subprogram is similar to a function in C and can be called many times in a VHDL design. It can be declared in the declarative part of an entity, architecture, process or even another subprogram and in packages. As a subprogram is thought to be used in several places (architectures) it is useful to declare it in a package, always.

Components are necessary to include entity/architecture pairs in the architecture of the next higher hierarchy level. These components can only be declared in architecture or a package. This is useful, if an entity/architecture pair might be used in several architectures as only one declaration is necessary in this case.

Configurations, themselves, are complete VHDL design units. But it is possible to declare configuration statements in the declarative part of architecture. This possibility is only rarely used, however, as it is better to create an independent configuration for the whole model.

Constants and data types can be declared within all available objects.

Port declarations are allowed in entities, only. They list those architecture signals that are available as interface to other modules. Additional internal signals can be declared in architectures, processes, subprograms and packages. Please note that signals can not be declared in functions, a special type of a subprogram.

Generally, variables can only be declared in processes and subprograms. In VHDL'93, global variables are defined which can be declared in entities, architectures and packages.

3.1.6. Entity

On the following pages, a full-adder consisting of two half-adders and an OR gate will be created step by step. We confine ourselves to a purely structural design, i.e. we are using gate level descriptions and do not need any synthesis tools. The idea is to demonstrate the interaction of the different VHDL objects in a straightforward manner.

The interface between a module and its environment is described within the entity

declaration which is initiated by the keyword 'entity'. It is followed by a user-defined, (hopefully) descriptive name, in this case: half-adder. The interface description is placed between the keyword 'is' and the termination of the entity statement which consists of the keyword 'end' and the name of the entity. In the new VHDL'93 standard the keyword 'entity' may be repeated after the keyword 'end' for consistency reasons.

The input and output signal names and their data types are defined in the port statement which is initiated by the keyword 'port'. The list of ports is enclosed in a '(' ')' pair. For each list element the port name(s) is given first, followed by a ':', the port mode and the data type. Within the list, the ';' symbol is used to separate elements, not to terminate a statement. Consequently, the last list element is not followed by a ';

Several ports with the same mode and data type can be declared by a single port statement when the port names are separated by ','. The port mode defines the data flow (in: input, i.e. the signal influences the module behavior; out: output, i.e. the signal value is generated by the module) while the data type determines the value range for the signals during simulation.

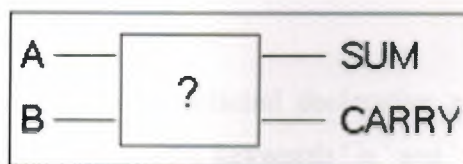


Figure 3.7 Entity

3.1.7. Architecture

The architecture contains the implementation for an entity which may be either a behavioral description (behavioral level or, if synthesizable, RT level) or a structural netlist or a mixture of those alternatives.

Architecture is strictly linked to a certain entity. An entity, however, may very well have several architectures underneath, e.g. different implementations of the same algorithm or different abstraction levels. Architectures of the same entity have to be named differently in order to be distinguishable. The name is placed after the keyword 'architecture' which initiates an architecture statement. 'RTL' was chosen in this case.

It is followed by the keyword 'of' and the name of entity that is used as interface ('HALFADDER'). The architecture header is terminated by the keyword 'is', like in entity

statements. In this case, however, the keyword ' begin ' must be placed somewhere before the statement is terminated. This is done the same way as in entity statements: The keyword ' end ', followed by the architecture name. Once again, the keyword ' architecture ' may be repeated after the keyword ' end ' in VHDL'93.

As the VHDL code is synthesizable, RTL was chosen as architecture name. In case of this simple function, however, there is no difference to behavioural (algorithmic) description. We will use 'BEHAVE', 'RTL', 'GATE', 'STRUCT' and 'TEST' to indicate the abstraction level and the implemented behavior, respectively. The name 'EXAMPLE' will be used whenever the architecture shows the application of new VHDL elements and is not associated with a specific entity.

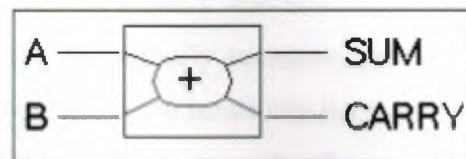


Figure 3.18 Architecture

3.1.8. Architecture Structure

Each architect is split into an optional declarative part and the definition part. The declarative part is located between the keywords ' is ' and ' begin '. New objects that are needed only within the architecture constants, data-types, signals, subprograms, etc. can be declared here.

The definition part is initiated by the keyword ' begin ' and holds concurrent statements. These can be simple signal assignments, process statements, which group together sequential statements, and component instantiations. Concurrency means that the order in which they appear in the VHDL code is not important. The signal SUM, for example, gets always the result of (3 + 7), independently of the location of the two assignments to the signals DIGIT_A and DIGIT_B.

Signal assignments are carried out by the signal assignment operator ' <= '. The symbol represents the data flow, i.e. the target signal whose value shall be updated is placed on the left side of the operator. The right side holds an expression that evaluates to the new signal value. The data types on the left and on the right side have to be identical.

Please remember that the signals that are used in this example were defined implicitly by the port declaration of the entity.

3.1.9. Process

Because the statements within architecture operate concurrently another VHDL construct is necessary to achieve sequential behavior. A process, as a whole, is treated concurrently like any other statement in architecture and contains statements that are executed one after another like in conventional programming languages. In fact it is possible to use the process statement as the only concurrent VHDL statement.

The execution of a process is triggered by events. Either the possible event sources are listed in the sensitivity list or explicit wait statements are used to control the flow of execution. These two options are mutually exclusive, i.e. no wait statements are allowed in a process with sensitivity list. While the sensitivity list is usually ignored by synthesis tools, a VHDL simulator will invoke the process code whenever the value of at least one of the listed signals changes. Consequently, all signals that are read in a purely combinational process i.e. that influence the behavior, have to be mentioned in the sensitivity list if the simulation is to produce the same results as the synthesized hardware. Of course the same is true for clocked processes, yet new register values are to be calculated with every active clock edge, only. Therefore the sensitivity list contains the clock signal and asynchronous control signals (e.g. reset).

A process statement starts with an optional label and a ':' symbol, followed by the 'process' keyword. The sensitivity list is also optional and is enclosed in a '(' ')' pair. Similar to the architecture statement, a declarative part exists between the header code and the keyword 'begin'. The sequential statements are enclosed between 'begin' and 'end process'. The keyword 'process' has to be repeated! If a label was chosen for the process, it has to be repeated in the end statement, as well.

3.1.10. Signals

Each signal has a predetermined data type which limits the amount of possible values for this signal. Synthesizable data types offer only a limited number of values, i.e. it is possible to map these values to a certain number of wires. Only the most basic data types are already predefined in VHDL, like bit, bit vectors and integer.

The user can define his own data types which might become necessary to enhance the accuracy of the model (tri-state drivers, for example, may be set to high impedance instead of a low or high voltage level), for better and to allow for automatic error detection.

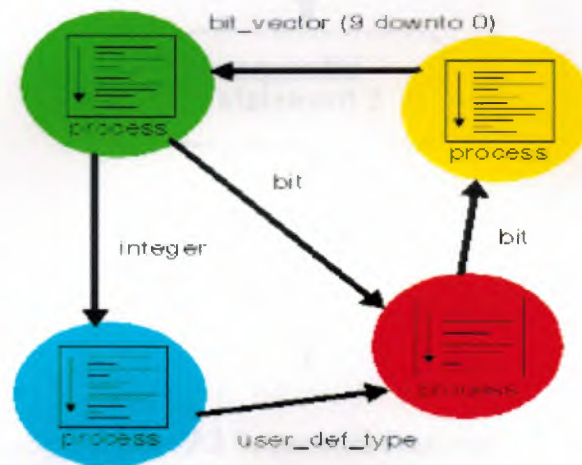


Figure 3. 1. Signals

3.2. Sequential Statements in VHDL

All statements in processes or subprograms are processed sequentially, i.e. one after another. Like in ordinary programming languages there exist a variety of constructs to control the flow of execution. The 'if' clause is probably the most obvious and most frequently used. The IF condition must evaluate to a Boolean value (true or false). After the first IF condition, any number of ELSIF conditions may follow. Overlaps may occur within different conditions. An ELSE branch, which combines all cases that have not been covered before, can optionally be inserted last. The IF statement is terminated with END IF. The first IF condition has top priority: if this condition is fulfilled, the corresponding statements will be carried out and the rest of the IF - END IF block will be skipped.

The example code shows two different implementations of equivalent behavior. The signal assignment to the signal Z in the first line of the left process (architecture EXAMPLE1) is called a default assignment, as its effects will only be visible if it is not overwritten by another assignment to Z. Note that the two conditions of the if and elsif part overlap, because $X="1111"$ is also true when $X>"1000"$. As a result of the priority mechanism of this if construct, Z will receive the value of B if $X="1111"$.

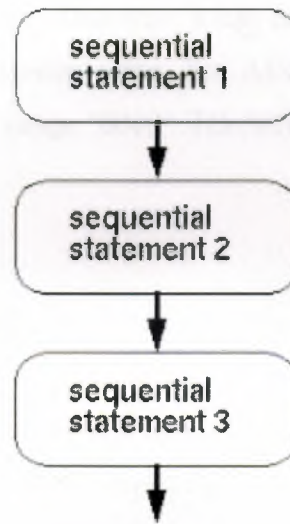


Figure 3.2 Sequential Statements

3.2.1. IF Statement

The if condition must evaluate to a Boolean value ('true' or 'false'). After the first if condition, any number of elsif conditions may follow. Overlaps may occur within different conditions. An else branch, which combines all cases that have not been covered before, can optionally be inserted last. The if statement is terminated with 'end if'.

The first if condition has top priority: if this condition is fulfilled, the corresponding statements will be carried out and the rest of the 'if - end if' block will be skipped.

3.2.2. CASE Statement

While the priority of each branch is set by means of the query's order in the IF case, all branches are equal in priority when using a CASE statement. Therefore it is obvious that there must not be any overlaps. On the other hand, all possible values of the CASE EXPRESSION must be covered. For covering all remaining, i.e. not yet covered, cases, the keyword 'others' may be used.

The type of the EXPRESSION in the head of the CASE statement has to match the type of the query values. Single values of EXPRESSION can be grouped together with the '|' symbol, if the consecutive action is the same. Value ranges allow covering even more choice options with relatively simple VHDL code.

Ranges can be defined for data types with a fixed order, only, e.g. user defined enumerated types or integer values. This way, it can be decided whether one value is less than, equal to or greater than another value. For ARRAY types (e.g. a BIT_VECTOR) there is no such order, i.e. the range "0000" TO "0100" is undefined and therefore not admissible.

3.2.3. FOR Loops

Loops operate in the usual way, i.e. they are used to execute the same some VHDL code a couple of times. Loop labels may be used to enhance readability, especially when loops are nested or the code block executed within the loop is rather long. The loop variable is the only object in VHDL which is implicitly defined. The loop variable can not be declared externally and is only visible within the loop. Its value is read only, i.e. the number of cycles is fixed when the execution of the for loop begins.

If a for loop is to be synthesized, the range of the loop variable must not depend on signal or variable values (i.e., it has to be locally static). By means of the range assignment, both the direction and the range of the loop variable is determined. If a variable number of cycles is needed, the while statement will have to be used. While loops are executed as long as condition evaluates to a 'true' value. Therefore this construct is usually not synthesizable.

3.2.4. Loop Syntax

The loop label is optional. By defining the range the direction as well as the possible values of the loop variable is fixed. The loop variable is only accessible within the loop. For synthesis the loop range has to be locally static and must not depend on signal or variable values. Loops are not generally synthesizable.

3.2.5. WAIT Statement

As mentioned before, processes may be coded in two flavors. If the sensitivity list is omitted, another method will be needed to stop process execution. Wait statements put the process execution on hold until the specified condition is full-filled. If no condition is given, the process will never be reactivated again. Wait statements must not be combined with a sensitivity list, independent from the application field.

3.2.6. WAIT Statements and Behavioral Modeling

Wait constructs, in general, are an excellent tool for describing timing specifications. For example it is easy to implement a bus protocol for simulation. The timing specification can directly be translated to simulatable VHDL code. But keep in mind that this behavioral modeling can only be used for simulation purposes as it is definitely not synthesizable.

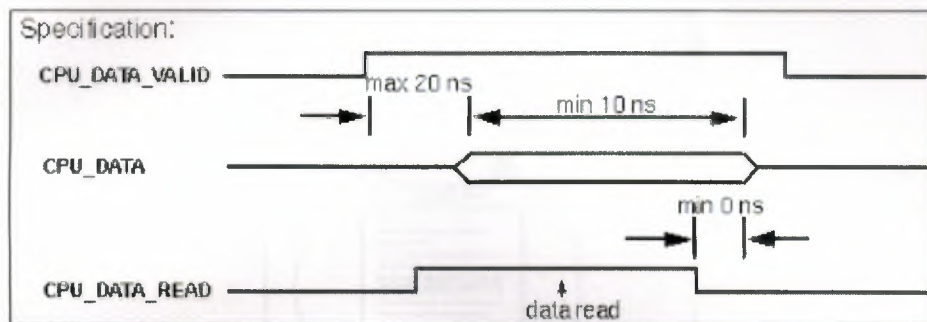


Figure 3.3 Wait Statement Model

3.2.7. Variables

Variables can only be defined in a process and they are only accessible within this process.

Variables and signals show a fundamentally different behavior. In a process, the last signal assignment to a signal is carried out when the process execution is suspended. Value assignments to variables, however, are carried out immediately. To distinguish between a signal and a variable assignment different symbols are used: ' \leq ' indicates a signal assignment and ' $:=$ ' indicates a variable assignment.

3.2.8. Variables and Signals

The two processes shown in the example implement different behavior as both outputs X and Y will be set to the result of $B+C$ when signals are used instead of variables. Please note that the intermediate signals have to be added to the sensitivity list, as they are read during process execution.

3.2.9. Use of Variables

Variables are especially suited for the implementation of algorithms. Usually, the signal values are copied into variables before the algorithm is carried out. The result is assigned to a signal again afterwards. Variables keep their value from one process call to the next, i.e. if a variable is read before a value has been assigned, the variable will have to show storage behavior. That means it will have to be synthesized to a latch or flip-flop respectively.

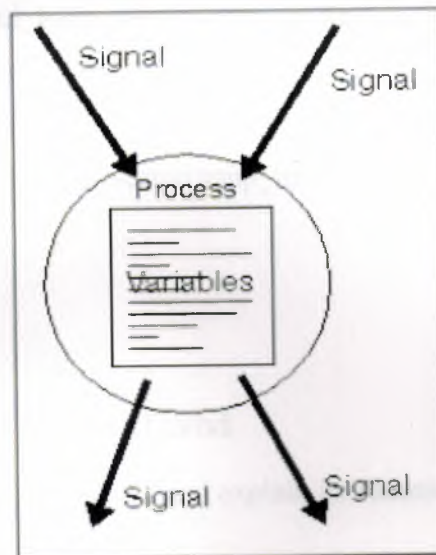


Figure 3.13 Use of Variables

3.3. VHDL Codes of Counter.vhd

We show all codes of counter.vhd and explain its statements.

Counter.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Counter is
    Port ( clock : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          count : inout STD_LOGIC_VECTOR (3 downto 0));
```

```

end Counter;
architecture Behavioral of Counter is
begin
process(CLOCK,RESET)
variable temp_int : integer :=0;
begin
    if RESET = '1' then
        COUNT <= conv_std_logic_vector(temp_int,4);
    elsif CLOCK = '1' and CLOCK'event then
        temp_int:= temp_int + 1;
        COUNT<=conv_std_logic_vector(temp_int,4);
    end if;
end process;
end Behavioral;

```

3.4. VHDL Codes of STAT_MAC.vhd

We show all codes of STAT_MAC.vhd and explain its statements.

STAT_MAC.vhd

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY SHELL_STAT_MAC IS

PORT (CLK,RESET,TIMER0,TIMER1,TIMER2,TIMER3: IN std_logic;
 AMB,GRN,RD : OUT std_logic);

END;

ARCHITECTURE BEHAVIOR OF SHELL_STAT_MAC IS

TYPE type_sreg IS (AMBER,GREEN,RED,REDAMB);

SIGNAL sreg, next_sreg : type_sreg;

SIGNAL next_AMB,next_GRN,next_RD : std_logic;

BEGIN

PROCESS (CLK, next_sreg, next_AMB, next_GRN, next_RD)

BEGIN


```

IF CLK='1' AND CLK'event THEN
    sreg <= next_sreg;
    AMB <= next_AMB;
    GRN <= next_GRN;
    RD <= next_RD;
END IF;
END PROCESS;

PROCESS (sreg,RESET,TIMER0,TIMER1,TIMER2,TIMER3)
BEGIN
    next_AMB <= '0'; next_GRN <= '0'; next_RD <= '0';

    next_sreg<=AMBER;
    IF ( RESET='1' ) THEN
        next_sreg<=RED;
        next_GRN<='0';
        next_AMB<='0';
        next_RD<='1';
    ELSE
        IF NOT ( (sreg=AMBER) OR (sreg=GREEN) OR (sreg=RED) OR
(sreg=REDAMB)) THEN

            next_sreg<=RED;
            next_RD<='1';
            next_AMB<='0';
            next_GRN<='0';
        ELSE
            CASE sreg IS
                WHEN AMBER =>
                    IF ( std_logic_vector'(TIMER3, TIMER2,
TIMER1, TIMER0)) =
                        std_logic_vector'("0000") THEN

```

```

        next_sreg<=RED;
        next_RD<='1';
        next_AMB<='0';
        next_GRN<='0';
    ELSE
        next_sreg<=AMBER;
        next_AMB<='1';
        next_GRN<='0';
        next_RD<='0';
    END IF;
WHEN GREEN =>
    IF ( std_logic_vector(TIMER3, TIMER2,
TIMER1, TIMER0)) =
        std_logic_vector("0011") THEN
        next_sreg<=AMBER;
        next_AMB<='1';
        next_GRN<='0';
        next_RD<='0';
    ELSE
        next_sreg<=GREEN;
        next_GRN<='1';
        next_AMB<='0';
        next_RD<='0';
    END IF;
WHEN RED =>
    IF ( std_logic_vector(TIMER3, TIMER2,
TIMER1, TIMER0)) =
        std_logic_vector("1111") THEN
        next_sreg<=REDAMB;
        next_RD<='1';
        next_AMB<='1';

```

```

        next_GRN<='0';
    ELSE
        next_sreg<=RED;
        next_RD<='1';
        next_AMB<='0';
        next_GRN<='0';
    END IF;
    WHEN REDAMB =>
        IF ( std_logic_vector(TIMER3, TIMER2,
TIMER1, TIMER0)) =
            std_logic_vector("0100") THEN
                next_sreg<=GREEN;
                next_GRN<='1';
                next_AMB<='0';
                next_RD<='0';
            ELSE
                next_sreg<=REDAMB;
                next_RD<='1';
                next_AMB<='1';
                next_GRN<='0';
            END IF;
        WHEN OTHERS =>
            END CASE;
    END IF;
END IF;
END PROCESS;
END BEHAVIOR;
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY STAT_MAC IS

```



```

    PORT (TIMER : IN std_logic_vector (3 DOWNTO 0);
          CLK,RESET: IN std_logic;
          AMB,GRN,RD : OUT std_logic);
END;

ARCHITECTURE BEHAVIOR OF STAT_MAC IS
    COMPONENT SHELL_STAT_MAC
        PORT (CLK,RESET,TIMER0,TIMER1,TIMER2,TIMER3: IN std_logic;
              AMB,GRN,RD : OUT std_logic);
    END COMPONENT;
BEGIN
    SHELL1_STAT_MAC : SHELL_STAT_MAC PORT MAP
(CLK=>CLK,RESET=>RESET,TIMER0=>

    TIMER(0),TIMER1=>TIMER(1),TIMER2=>TIMER(2),TIMER3=>TIMER(3),A
    MB=>AMB,GRN=>GRN
        ,RD=>RD);
END BEHAVIOR;

```

3.5. VHDL Codes of top.vhd

We show all codes of top.vhd and explain its statements.

top.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

entity top is

```

    Port ( clock : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          red_light : out STD_LOGIC;

```

```

        amber_light : out STD_LOGIC;
        green_light : out STD_LOGIC);
end top;

```

architecture Behavioral of top is

```

signal timer :std_logic_vector (3 downto 0 );

```

component counter

```

port(
    CLOCK: in std_logic;
        RESET: in std_logic;
        COUNT: inout std_logic_vector(3 downto 0)
    );
end component;

```

component stat_mac

```

port(
    TIMER: in std_logic_vector(3 downto 0);
        CLK: in std_logic;
        RESET: in std_logic;
        AMB: out std_logic;
        GRN: out std_logic;
        RD: out std_logic
    );
end component;

```

begin

```

    Inst_counter: counter PORT MAP(
        CLOCK => clock,
        RESET => reset,

```

```

COUNT=>timer
);

Inst_stat_mac: stat_mac PORT MAP(
TIMER => timer,
CLK => clock,
RESET => reset,
AMB => amber_light,
GRN => green_light,
RD => red_light
);
end Behavioral;

```



CHAPTER FOUR: TRAFFIC LIGHT CONTROLLER SYNTHESIS

4.1. XST Design Flow Overview

After design entry and optional simulation, we run synthesis. In the Sources tab, we select Synthesis/Implementation from the Design View drop-down list, and we select the top module. In the Processes tab, with double-click Synthesize.

The ISE™ software includes Xilinx® Synthesis Technology (XST), which synthesizes VHDL, Verilog, or mixed language designs to create Xilinx-specific netlist files known as NGC files. Unlike output from other vendors, which consists of an EDIF file with an associated NCF file, NGC files contain *both* logical design data and constraints. XST places the NGC file in our project directory and the file is accepted as input to the Translate (NGD Build) step of the Implement Design process. To specify XST as our synthesis tool, we must set the Synthesis Tool Project Property to XST.

The following figure shows the flow of files through the XST software.

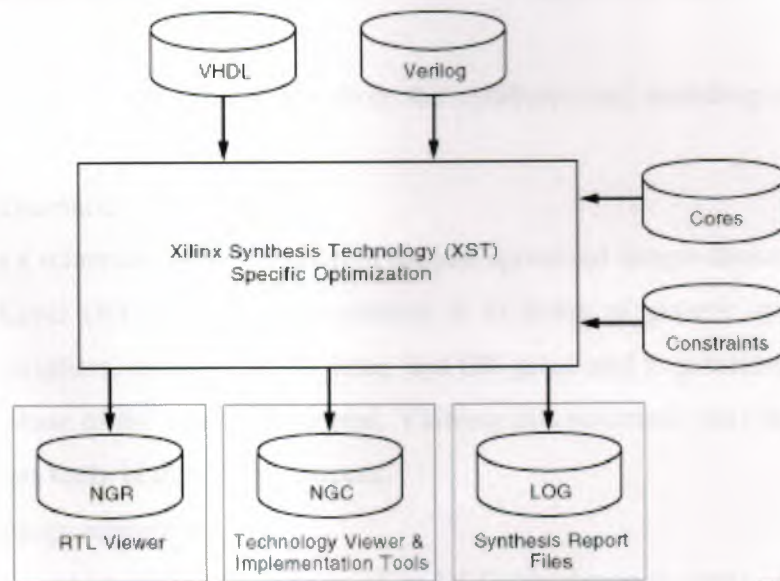


Figure 4. 1. XST Design Flow

4.2. XST Input and Output Files

XST supports extensive VHDL and Verilog subsets from the following standards:

- VHDL: IEEE 1076-1987, IEEE 1076-1993, including IEEE standard and Synopsys®
- Verilog: IEEE 1364-1995, IEEE 1364-2001.

In addition to a VHDL or Verilog design description, XST can also accept the following files as input:

- XCF

Xilinx constraints file in which we can specify synthesis, timing, and specific implementation constraints that can be propagated to the NGC file.

- Core files

These files can be in either NGC or EDIF format. XST does not modify cores. It uses them to inform area and timing optimization. Cores are supported for FPGAs only, not CPLDs.

In addition to NGC files, XST also generates the following files as output:

- Synthesis Report

This report contains the results from the synthesis run, including area and timing estimation.

- RTL schematic

This is a schematic representation of the pre-optimized design shown at the Register Transfer Level (RTL). This representation is in terms of generic symbols, such as adders, multipliers, counters, AND gates, and OR gates, and is generated after the HDL synthesis phase of the synthesis process. Viewing this schematic may help we discover design issues early in the design process.

- Technology schematic:

This is a schematic representation of an NGC file shown in terms of logic elements optimized to the target architecture or "technology," for example, in terms of LUTs, carry logic, I/O buffers, and other technology-specific components. It is generated after the optimization and technology targeting phase of the synthesis process. Viewing this schematic allows we to see a technology-level representation of our HDL optimized for a specific Xilinx architecture, which may help we discover design issues early in the

design process. When the design is run in Incremental Synthesis mode, XST generates multiple NGC and NGR files, which each represent a single user design partition.

4.3 XST Detailed Design Flow

The following figure shows each of the steps that take place during XST synthesis. The following sections describe each step in detail.

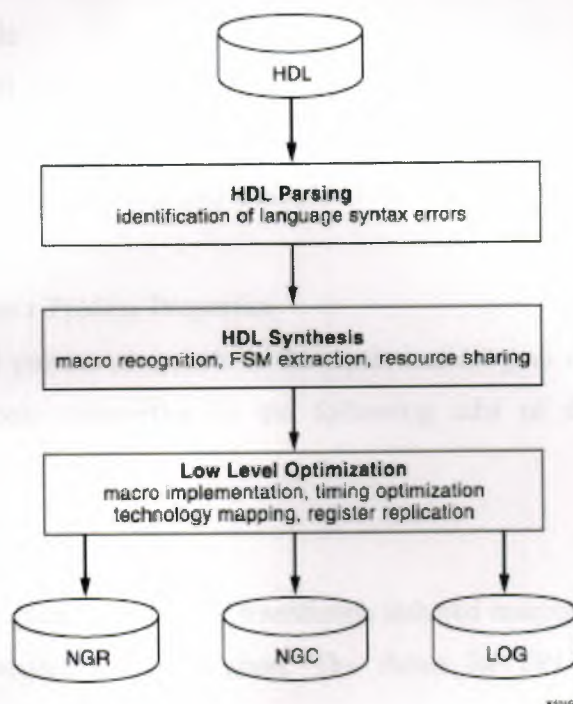


Figure 4. 2. XST Detailed Design Flow

4.3.1. HDL Parsing

During HDL parsing, XST checks whether your HDL code is correct and reports any syntax errors.

4.3.2. HDL Synthesis

During HDL synthesis, XST analyzes the HDL code and attempts to infer specific design building blocks or macros (such as MUXes, RAMs, adders, and subtractors) for which it can create efficient technology implementations. To reduce the amount of inferred

macros, XST performs a resource sharing check. This usually leads to a reduction of the area as well as an increase in the clock frequency.

Finite state machine (FSM) recognition is also part of the HDL synthesis step. XST recognizes FSMs independent of the modeling style used. To create the most efficient implementation, XST uses the target optimization goal, whether area or speed, to determine which of several FSM encoding algorithms to use. We can control the HDL synthesis step using constraints and enter constraints using any of the following methods:

- HDL source file

We enter VHDL attributes or Verilog metacomments.

- XCF

We enter global parameters and module-level constraints in the Xilinx constraints (XCF) file.

- Project Navigator Process Properties

We set global parameters, such as the optimization goal or effort level. We can modify the synthesis properties in the following tabs of the Synthesize Process Properties dialog box:

4.3.3 Low Level Optimization

During low level optimization, XST transforms inferred macros and general glue logic into a technology-specific implementation. The flows for FPGAs and CPLDs differ significantly at this stage as follows:

- FPGA Flow

The FPGA flow is timing-driven and can be controlled using constraints, such as PERIOD and OFFSET. During low level optimization, XST infers specific components, such as the following:

- Carry logic (MUXCY, XORCY, MULT_AND)
- RAM (block or distributed)
- Shift Register LUTs (SRL16, SRL16E, SRLC16, SRLC16E)
- Clock Buffers (IBUFG, BUFGP)
- Multiplexers (MUXF5, MUXF6, MUXF7, MUXF8)

The use of technology-specific features may come from a macro implementation mechanism or from general logic mapping. Due to mapping complexity issues, not all available FPGA features may be used. The FPGA synthesis flow supports advanced design and optimization techniques, such as Register Balancing, Incremental Synthesis, and Modular Design.

- **CPLD Flow**

The CPLD flow is not timing driven. We cannot specify the frequency of a clock or of an offset value. The goal of the CPLD flow is to reduce the number of logic levels. During low level optimization, XST generates a netlist that contains elements such as 'AND' and 'OR' gates. The CPLD Fitter then determines how to fit these equations to the targeted device. XST supports a special optimization mode, called Equation Shaping, in which XST optimizes and reduces the Boolean equations to sizes accepted by device macrocells. This forces the CPLD Fitter to retain the equation modifications through the KEEP and COLLAPSE constraints in the NGC file.

CONCLUSION

Today, most devices that people use for the routine tasks of their daily lives amongst some of them regulate our activities such as traffic lights are produced by using hardware design languages. One of the most widely used is VHDL that can be used for a wide variety of applications. Under these circumstances, VHDL is effective hardware description tool by the means of specification, design, synthesis and implementation for a project.

In the first section, specification and requirements of the traffic light controller depending on basic minimal diagram. In the second section the steps how the project "Traffic Light Controller System" is constructed on ISE software environment tool. It also presents the architectural construction of the components and the description of the functions of each component. Third section describes the VHDL (Very High Speed Integrated Circuit Hardware Description Language). It also gives brief information about history and development of VHDL and how declaration of its object and sequential statements are used. Additionally, it consists of the complete VHDL codes of the project. The last one describes how a prepared project is synthesis in Integrated Software Environment.

The objective of this project is to describe the process of designing a traffic light controlling application using VHDL software tool with ISE environment and implementing on Xilinx Spartan-3E device.

At the beginning, we are inspired by controlling a hardware using hardware description language. To do so, we had to understand this description language prior to try to use it. Before we worked on small example modules on ISE examples and then we improved our project specifications. Then we split up to 3 modules to minimize complexity of the project. First modules counter counts up to 16. Second module describes the traffic light status and function depending on the interval. Last module is used to combine first and second module.

Traffic light controller can be further improved for multiple junctions to insure safe traffic flow .Even though; in the future the drivers who breach the red lights might be detected by the sensors placed on the traffic light.

REFERENCES

[1] Thomas L. Floyd, Digital Fundamentals: Pearson Education, Inc. Prentice Hall PTR Upper Saddle River, New Jersey 07458, 2006.

[2] Richard S. Sandige, Digital Design Essentials: Prentice Hall, Inc. Upper Saddle River, New Jersey 07458, 2002.