Laboratory Manual

Digital System Design
CSE-308L

Prepared By

Engr. Anees ullah
Department of Computer System Engineering
NWFP University of Engineering and Technology
Peshawar
aneesullaah@yahoo.com

January 2010

Preface

This DSD Lab Manual is designed for the 6th semester students of Department of Computer System Engineering (DCSE).

Following are the prerequisites for the labs.
• Digital Logic Design
• Computer Architecture and Organization
• C/C++ Programming
• Basics of Digital Signal Processing

Digital Logic Design is a must pre-requisite. Knowledge of Computer organization and Architecture will be a big bonus. Basic knowledge of C/C++ programming language is assumed. Similarly some understanding of digital signal processing is must.

Following Software tools will be used
• ModelSim 5.7f
• Xilinx ISE 9.2i
• KCPSM3 assembler
• pBlazIDE simulator

Following Hardware Platform will be used
• Xilinx Spartan 3 FPGA Starter Kit

The labs will be carried out by first simulating the objective in ModelSim and then Synthesizing with Xilinx ISE. For labs on PicoBlaze Microcontrollers all the above four software tools will be used. After Synthesis students will implement the lab on Xilinx Spartan 3 Starter kit. The code for the labs is not given in the manual but the students are guided to perform the procedures, in order to encourage them using their own thoughts and ideas in implementing the labs. Students will observe strict discipline in following all the labs and will submit a DSD related project at the end of the semester.

LAB No 1
INTRODUCTION TO MODELSIM AND GATE LEVEL MODELING

Objectives:

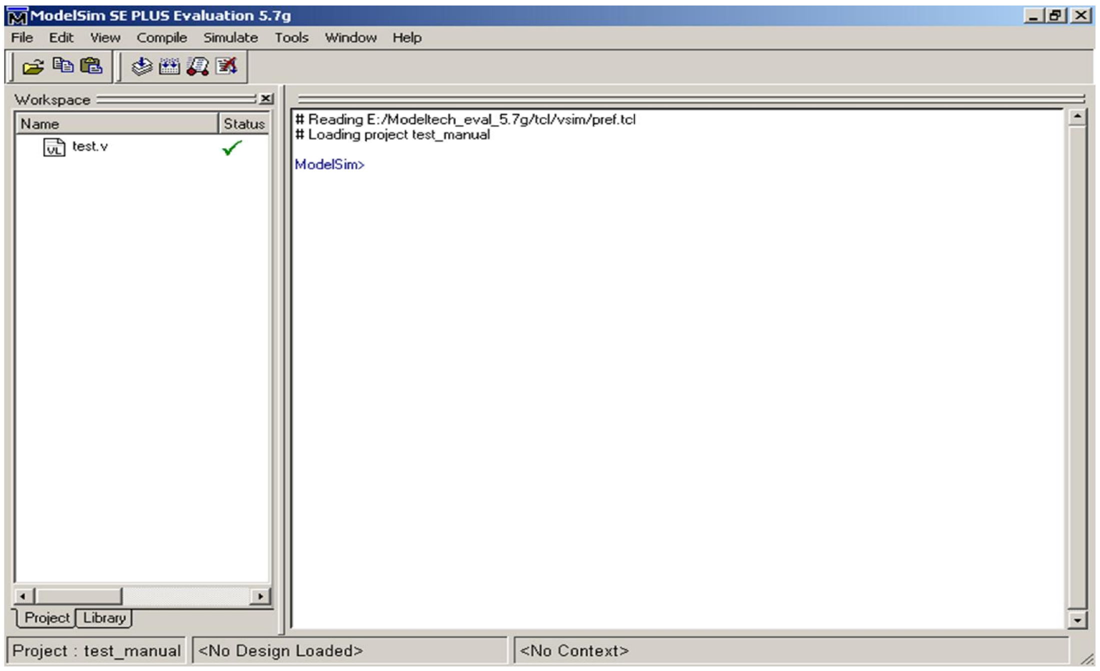
Introduction to MODELSIM

Software used:

MODELSIM

MODELSIM:

MODELSIM is a simulator which can be used for the simulations of both VHDL and Verilog HDL. It has the following interface.


ModelSim SE PLUS Evaluation 5.7g
File Edit View Compile Simulate Tools Window Help
Workspace
Name
Status
test.v
Project
Library

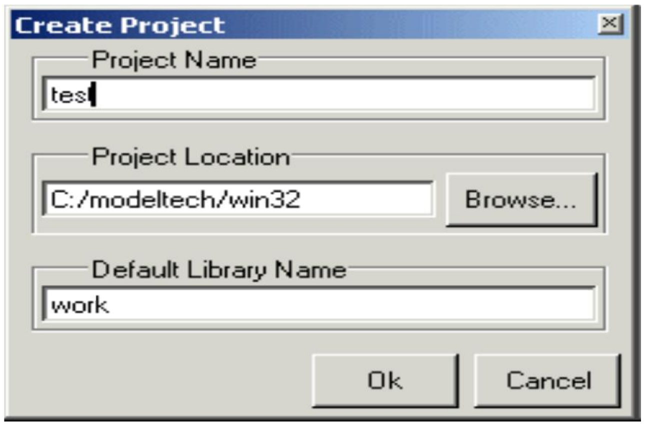
```
# Reading E:/Modeltech_eval_5.7g/tcl/vsim/pref.tcl
# Loading project test_manual

ModelSim>
```

Project : test_manual <No Design Loaded> <No Context>

How to use MODELSIM?

Step 1
Creating a new project

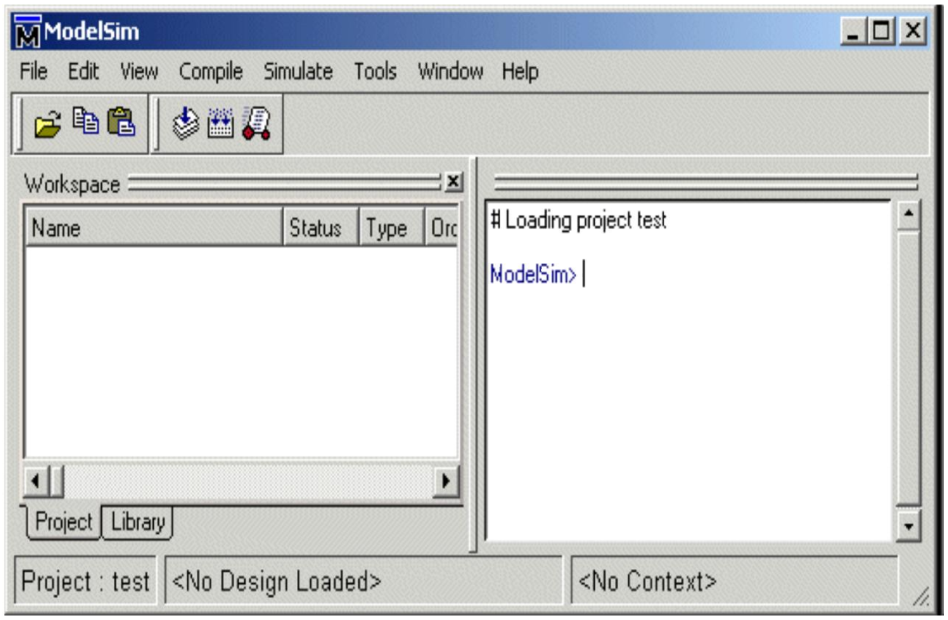
Select **File > New > Project** (Main window) to create a new project. This opens the **Create Project** dialog. The dialog includes these options:



- **Project Name**
  The name of the new project.
- **Project Location**
  The directory in which the *.mpf* file will be created.
- **Default Library Name**
  The name of the working library.

You can generally leave the **Default Library Name** set to "work." The name you specify will be used to create a working library subdirectory within the Project Location.

After selecting OK, you will see a blank Project tab in the workspace area of the Main window and the **Add Items to the Project** dialog.



```
# Loading project test

ModelSim>
```

The name of the current project is shown at the bottom left corner of the Main window.

Step 2
Adding items to the project

The **Add Items to the Project** dialog includes these options:

Create New File

Create a new VHDL, Verilog, Tcl, or text file using the Source window. See below for details.

- **Add Existing File**

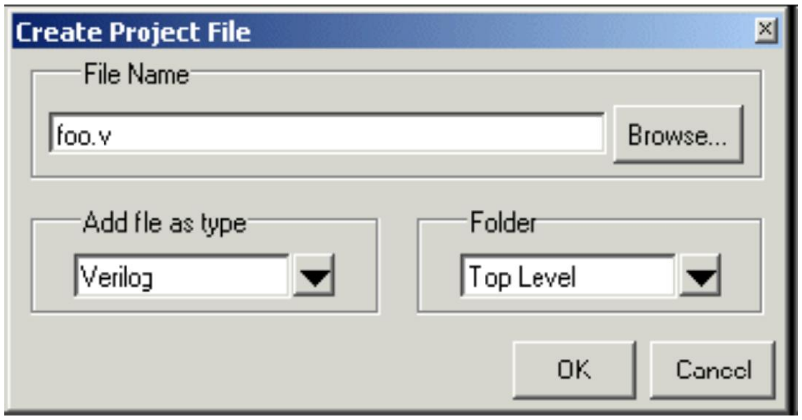
  Add an existing file. See below for details.
- **Create Simulation**

  Create a Simulation Configuration that specifies source files and simulator options. See
- **Create New Folder**

  Create an organization folder.

Create New File

The **Create New File** command lets you create a new VHDL, Verilog, Tcl, or text file using the Source window. You can also access this command by selecting **File > Add to Project > New File** (Main window) or right-clicking (2nd button in Windows; 3rd button in UNIX) in the Project tab and selecting **Add to Project > New File**.


Create Project File

File Name: foo.v

Add file as type: Verilog

Folder: Top Level

The **Create Project File** dialog includes these options:

- **File Name**

  The name of the new file
- **Add file as type**

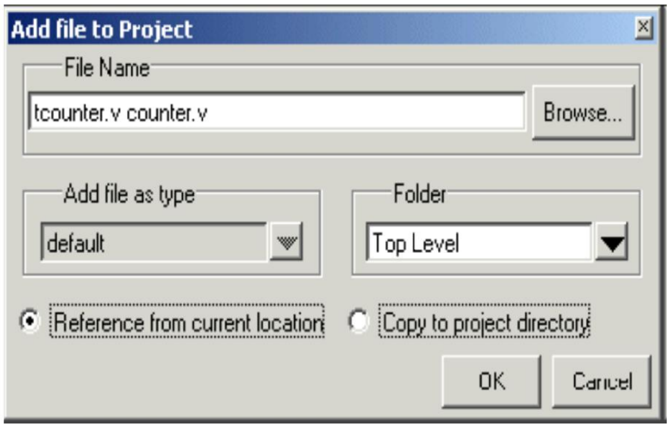
  Add the type of the new file. Select VHDL, Verilog, TCL, or text.
- **Folder**

  The organization folder in which you want the new file placed. You must first create folders in order to access them here

When you select OK, the Source window opens with an empty file, and the file is listed in the Project tab of the Main window workspace.

Add Existing File

You can also access this command by selecting **File > Add to Project > Existing File** (Main window) or by right-clicking (2nd button in Windows; 3rd button in UNIX) in the Project tab and selecting **Add to Project > Existing File**.


Add file to Project

File Name: tcounter.v counter.v

Add file as type: default

Folder: Top Level

[✓ Reference from current location] Copy to project directory

The **Add file to Project** dialog includes these options:

- **File Name**

The name of the file to add. You can add multiple files at one time.

- **Add file as type**

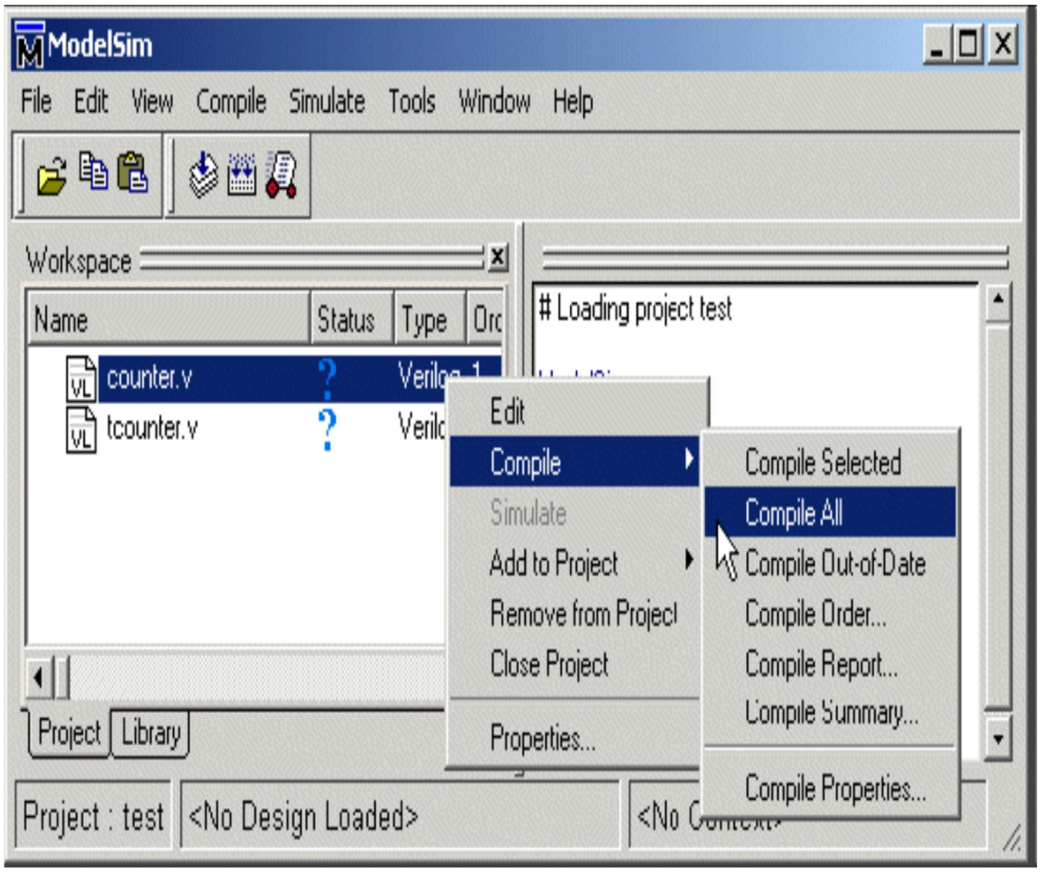
The type of the file. "Default" assigns type based on the file extension (e.g., *.v* is type Verilog).

- **Folder**

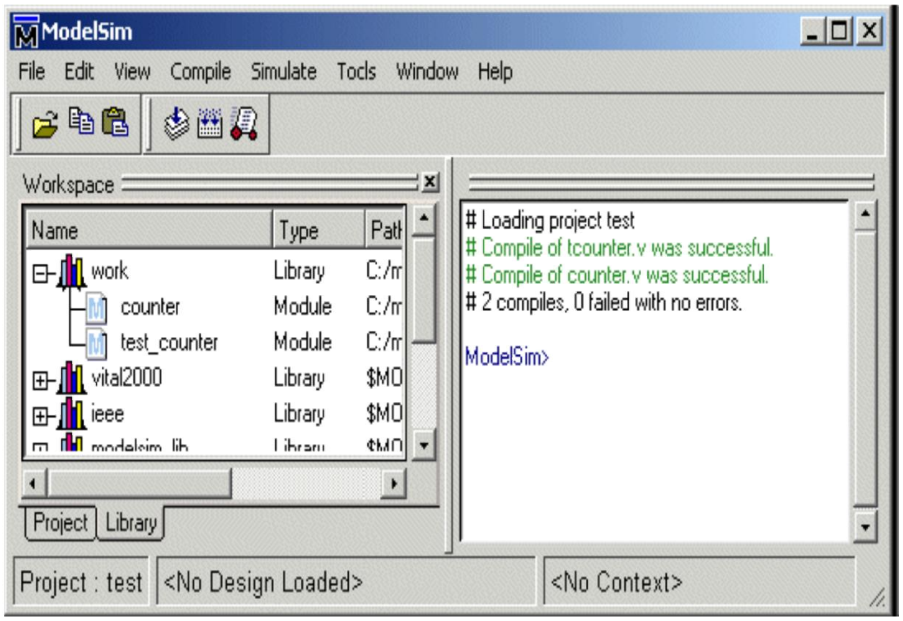
The organization folder in which you want the file placed. You must first create folders in order to access them here. Choose whether to reference the file from its current location or to copy it into the project directory. When you select OK, the file(s) is listed in the Project tab of the Main window workspace.

Step 3
Compiling the files

The question marks next to the files in the Project tab denote either the files haven't been compiled into the project or the source has changed since the last compile. To compile the files, select **Compile > Compile All** (Main window) or right click in the Project tab and select **Compile > Compile All**.



Once compilation is finished, click the Library tab, expand library *work* by clicking the "+", and you'll see the two compiled design units.



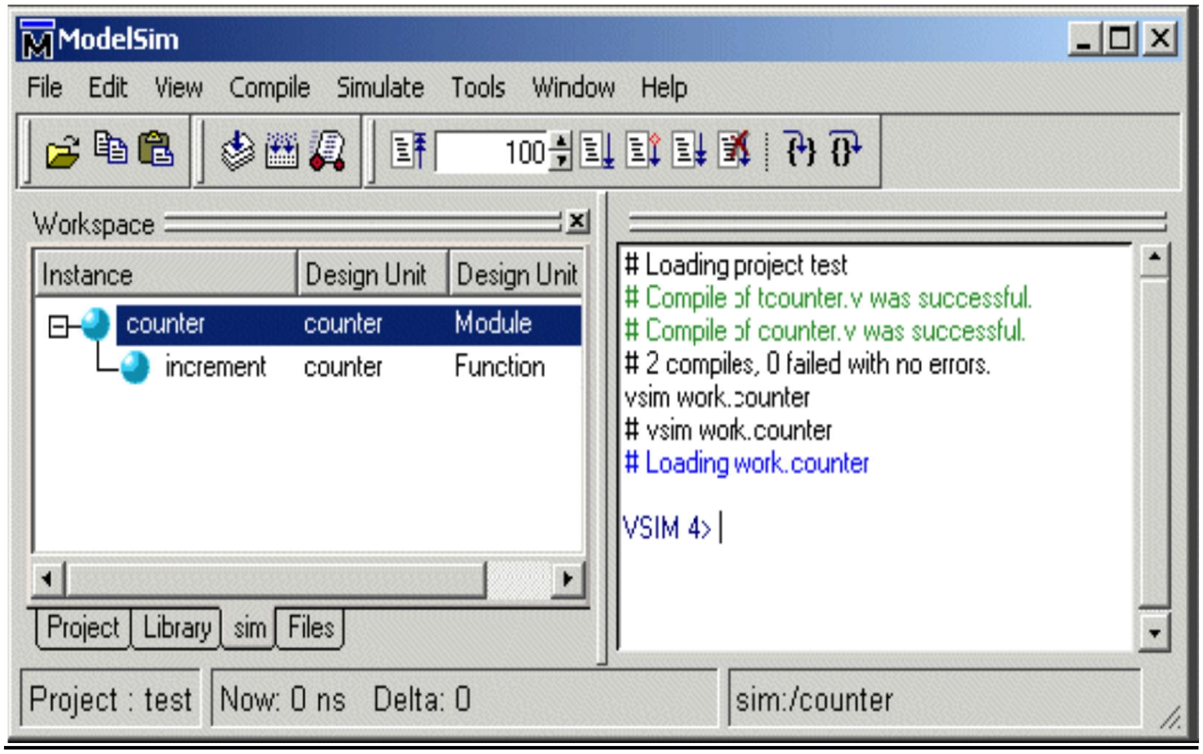
Name	Type	Path
work	Library	C:/m
counter	Module	C:/m
test_counter	Module	C:/m
vital2000	Library	$MO
ieee	Library	$MO
modelsim_lib	Library	$MO

```
# Loading project test
# Compile of tcounter.v was successful.
# Compile of counter.v was successful.
# 2 compiles, 0 failed with no errors.

ModelSim>
```

Project : test <No Design Loaded> <No Context>

Step 4
Simulating a design

To simulate one of the designs, either double-click the name or right-click the name and select Simulate. A new tab appears showing the structure of the active simulation. At this point you are ready to run the simulation and analyze your results. You often do this by adding signals to the Wave window and running the simulation for a given period of time.



ModelSim
File Edit View Compile Simulate Tools Window Help
100
Workspace
Project Library sim Files

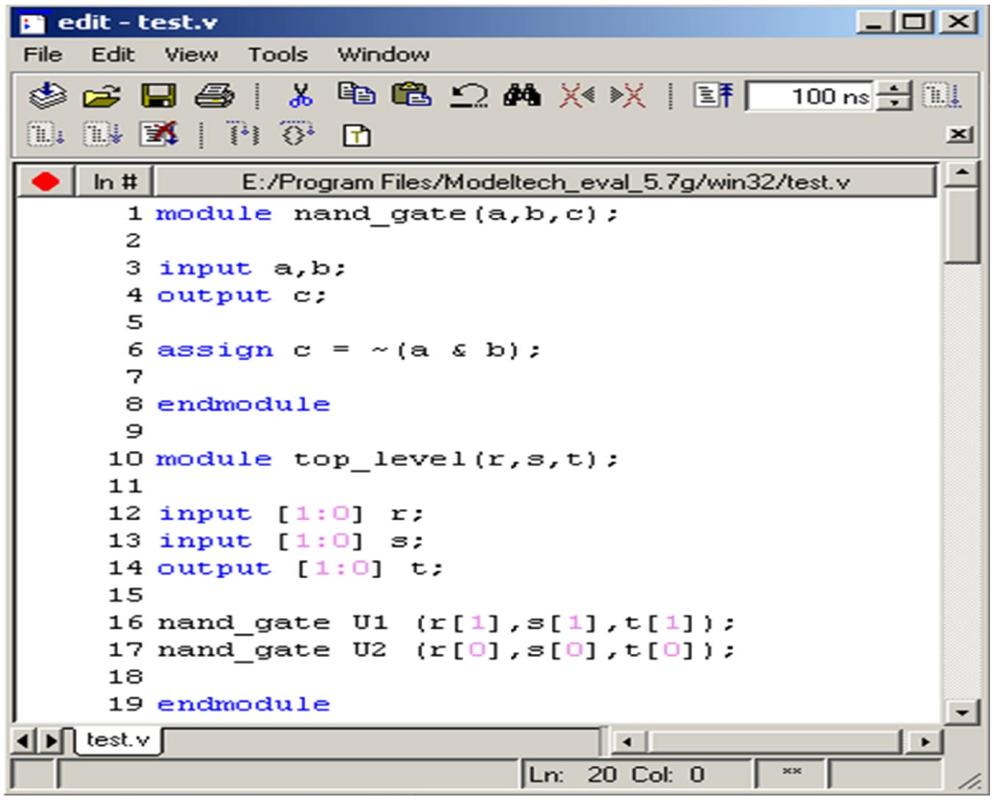
Instance	Design Unit	Design Unit
counter	counter	Module
increment	counter	Function

```
# Loading project test
# Compile of tcounter.v was successful.
# Compile of counter.v was successful.
# 2 compiles, 0 failed with no errors.
vsim work.counter
# vsim work.counter
# Loading work.counter

VSIM 4>
```

Project : test Now: 0 ns Delta: 0 sim:/counter

Program:

Example for the module instantiation


edit - test.v
File Edit View Tools Window
100 ns
ln #
E:/Program Files/Modeltech_eval_5.7g/win32/test.v
test.v
Ln: 20 Col: 0

```
module nand_gate(a,b,c);

input a,b;
output c;

assign c = ~(a & b);

endmodule

module top_level(r,s,t);

input [1:0] r;
input [1:0] s;
output [1:0] t;

nand_gate U1 (r[1],s[1],t[1]);
nand_gate U2 (r[0],s[0],t[0]);

endmodule
```

Lab tasks:

1- Implement a buffer at the gate level.
2- Implement an inverter at the gate level.
3- Implement an OR gate using a NAND gates.
4- Implement the following equation where z is output and x1, x2, x3, x4, and x5 are inputs of the circuit.

$$z = (y1 + y2)'$$
$$y1 = x1.x2$$
$$y2 = (x3.x4.x5)'$$

LAB 2
INTRODUCTION TO XILINX ISE AND S3BOARD

Objectives:
Introduction to FPGA and Xilinx

Software used:
XILINX

FPGA:
FPGAs are programmable digital logic circuits. It can be programmed to do almost any digital function. There are at least 5 companies making FPGAs in the world. Xilinx is the biggest name in the FPGA world.
The FPGA kits available in our labs are SPARTAN-3 STARTER KIT BOARD.

SPARTAN-3 STARTER KIT BOARD:
The features of this kit are
200,000-gate Xilinx Spartan-3 XC3S200 FPGA in a 256-ball thin Ball Grid Array package (XC3S200FT256)
4,320 logic cell equivalents
Twelve 18K-bit block RAMs (216K bits)
Twelve 18x18 hardware multipliers
Four Digital Clock Managers (DCMs)
Up to 173 user-defined I/O signals
2Mbit Xilinx XCF02S Platform Flash, in-system programmable configuration PROM
1Mbit non-volatile data or application code storage available after FPGA configuration
Jumper options allow FPGA application to read PROM data or FPGA configuration from other sources
□1M-byte of Fast Asynchronous SRAM (bottom side of board, see Figure 1-3)
□Two 256Kx16 ISSI IS61LV25616AL-10T 10 ns SRAMs
Configurable memory architecture
Single 256Kx32 SRAM array, ideal for MicroBlaze code images
Two independent 256Kx16 SRAM arrays
Individual chip select per device
Individual byte enables
□3-bit, 8-color VGA display port
□9-pin RS-232 Serial Port
DB9 9-pin female connector (DCE connector)
Maxim MAX3232 RS-232 transceiver/translator
Uses straight-through serial cable to connect to computer or workstation serial port
Second RS-232 transmit and receive channel available on board test points
PS/2-style mouse/keyboard port
Four-character, seven-segment LED display
Eight slide switches

Eight individual LED outputs
Four momentary-contact push button switches
50 MHz crystal oscillator clock source (bottom side of board, see Figure 1-3)
Socket for an auxiliary crystal oscillator clock source
FPGA configuration mode selected via jumper settings
Push button switch to force FPGA reconfiguration (FPGA configuration happens automatically at power-on)
LED indicates when FPGA is successfully configured
Three 40-pin expansion connection ports to extend and enhance the Spartan-3 Starter Kit Board
FPGA serial configuration interface signals available on the A2 and B1 connectors PROG_B, DONE, INIT_B, CCLK, DONE
JTAG port for low-cost download cable
Digilent JTAG download/debugging cable connects to PC parallel port
JTAG download/debug port compatible with the Xilinx Parallel Cable IV and MultiPRO Desktop Tool
AC power adapter input for included international unregulated +5V power supply
Power-on indicator LED
On-board 3.3V , 2.5V , and 1.2V regulators

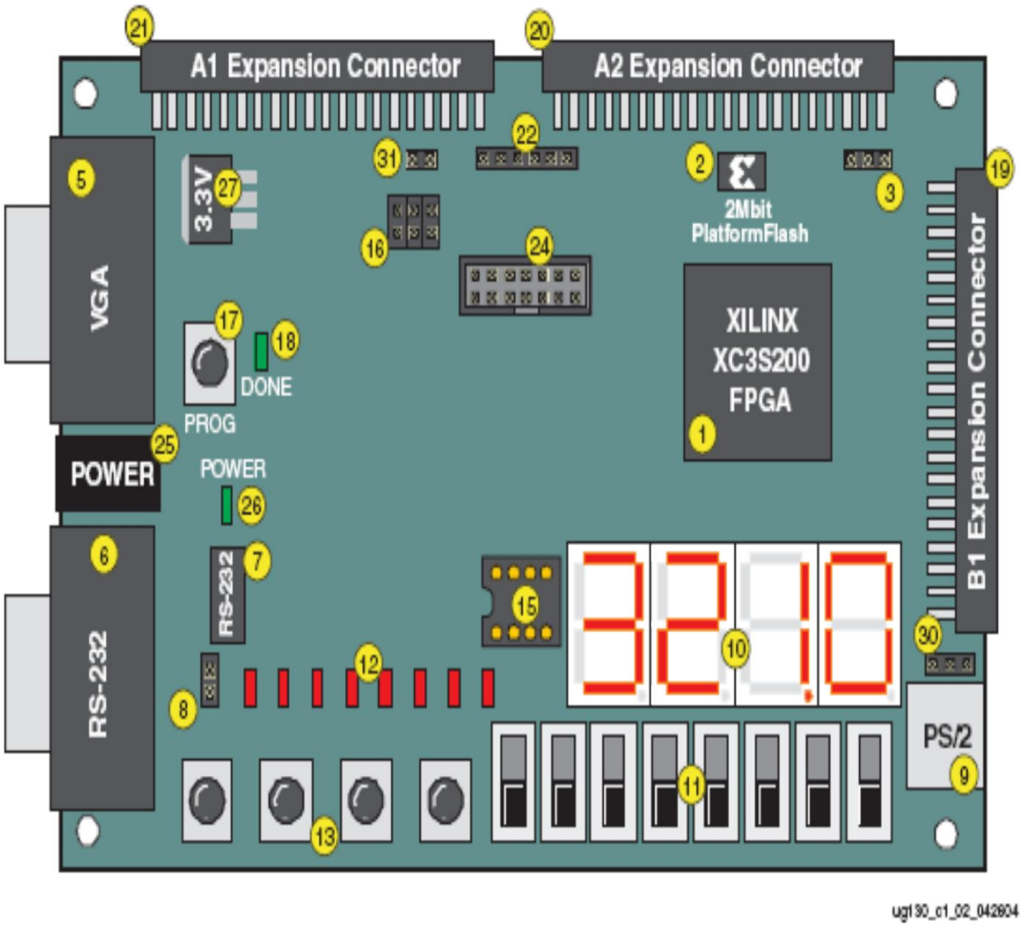


Figure 1-2: Xilinx Spartan-3 Starter Kit Board (Top Side)

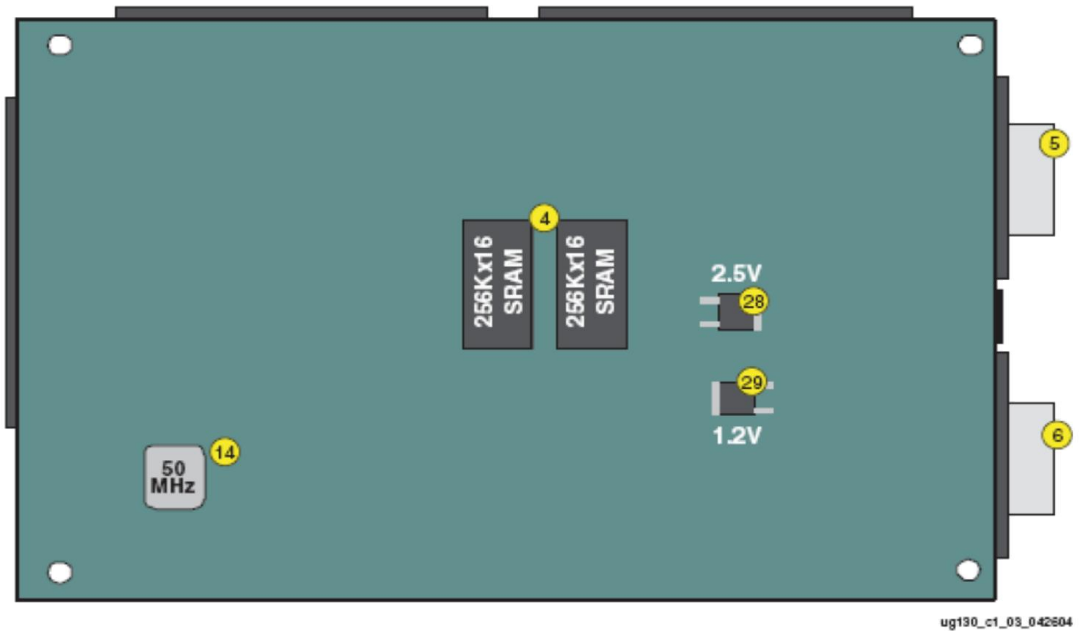


Figure 1-3: **Xilinx Spartan-3 Starter Kit Board (Bottom Side)**

<u>XILINX:</u>

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

Design Entry

Design entry is the first step in the ISE design flow. During design entry, you create your source files based on your design objectives. You can create your top-level design file using a Hardware Description Language (HDL), such as VHDL, Verilog, or ABEL, or using a schematic. You can use multiple formats for the lower-level source files in your design.

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

Synthesis

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

Implementation

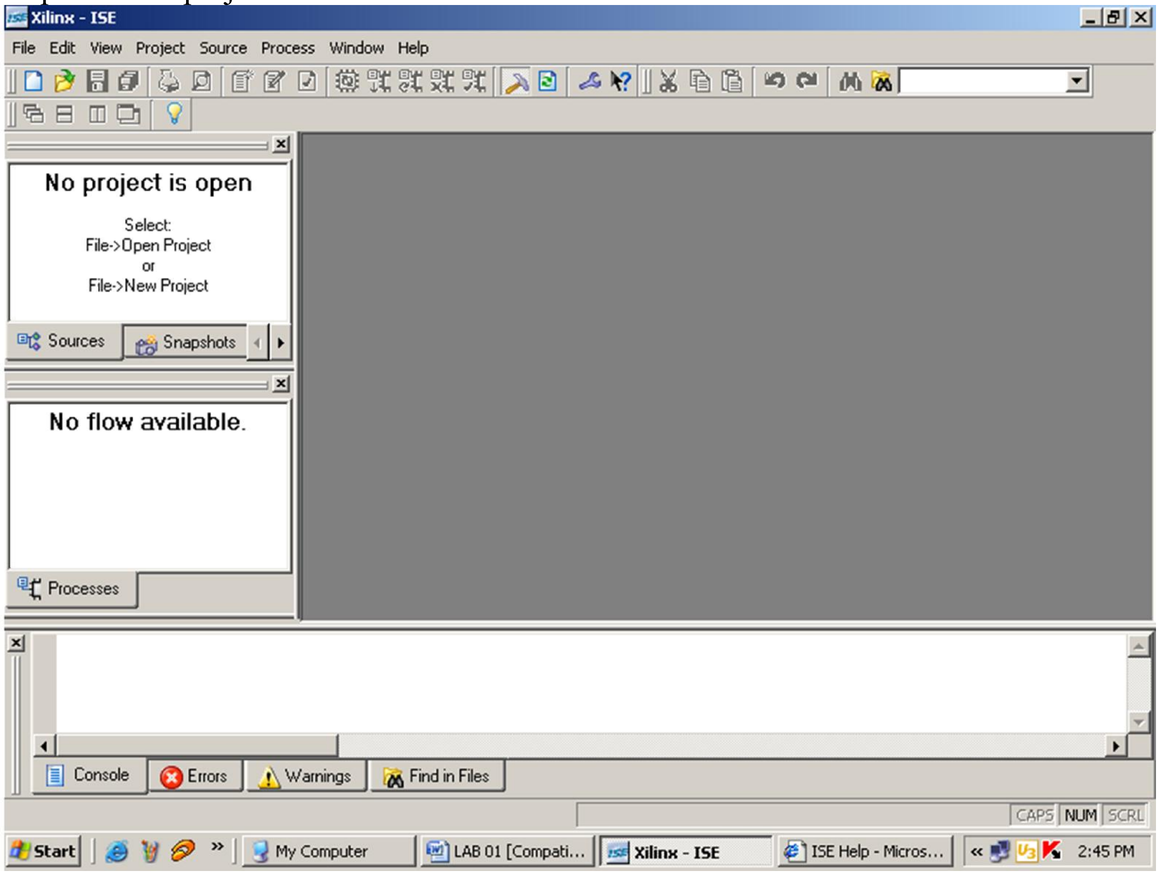
After synthesis, you run design implementation, which converts the logical design into a physical file format that can be downloaded to the selected target device. From Project Navigator, you can run the implementation process in one step, or you can run each of the implementation processes separately.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

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

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

How to make new Project:

Steps for new project:



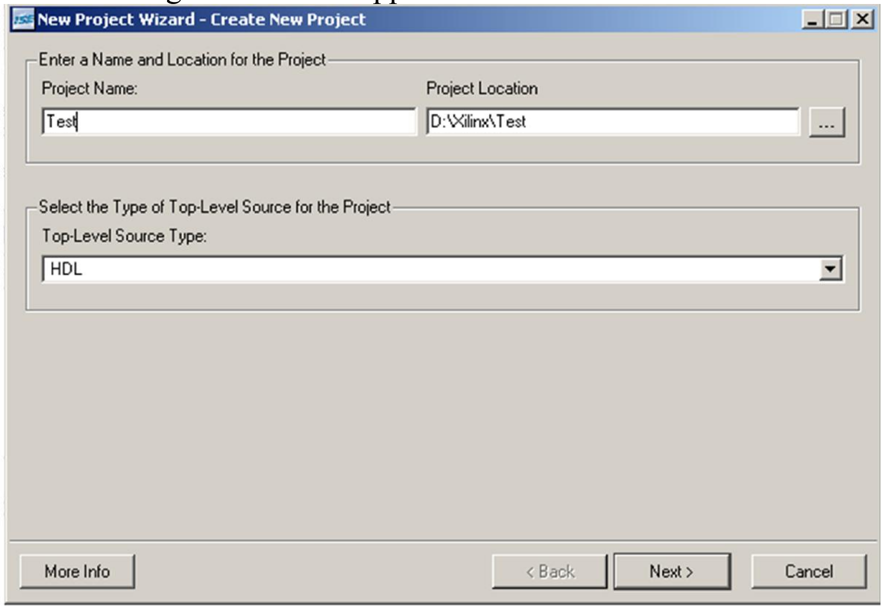
No project is open

Select:
File->Open Project
or
File->New Project

No flow available.

File→New Project

The following window will appear



Enter a Name and Location for the Project

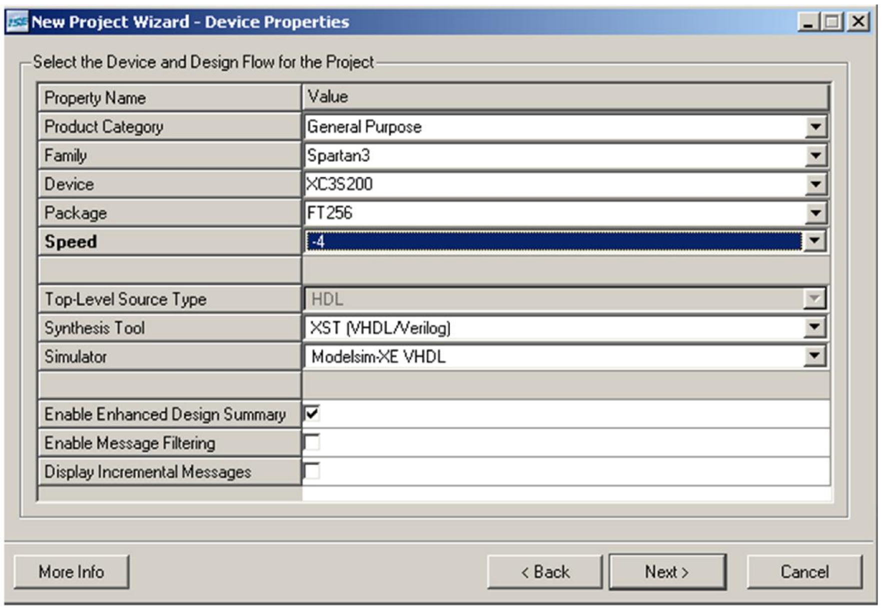
Project Name: Test

Project Location: D:\Xilinx\Test

Select the Type of Top-Level Source for the Project

Top-Level Source Type: HDL

Enter the Project Name and Project Location and Press Next>
The following window will appear

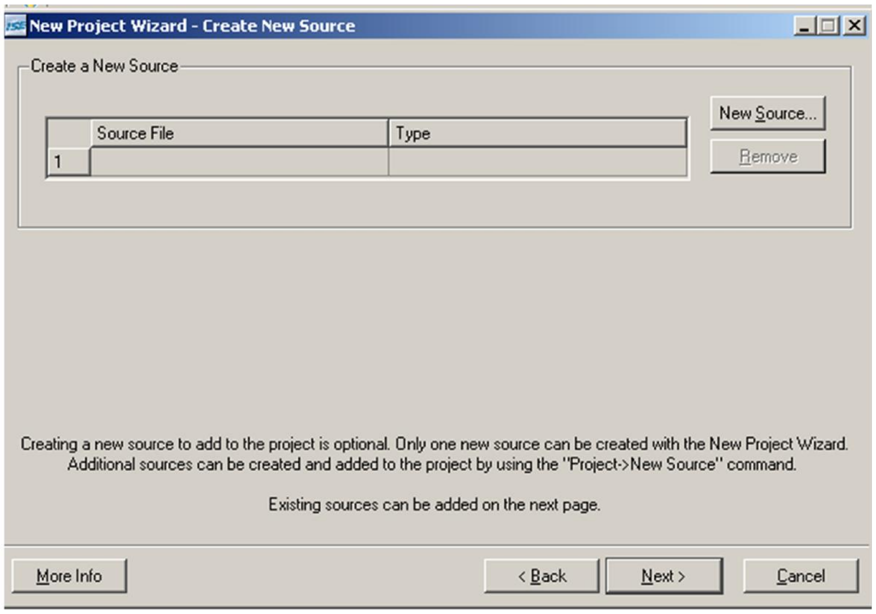


Select the Device and Design Flow for the Project

Property Name	Value
Product Category	General Purpose
Family	Spartan3
Device	XC3S200
Package	FT256
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-XE VHDL
Enable Enhanced Design Summary	✓
Enable Message Filtering	☐
Display Incremental Messages	☐

Select the options from the PULL DOWN lists as show above, and Press Next>

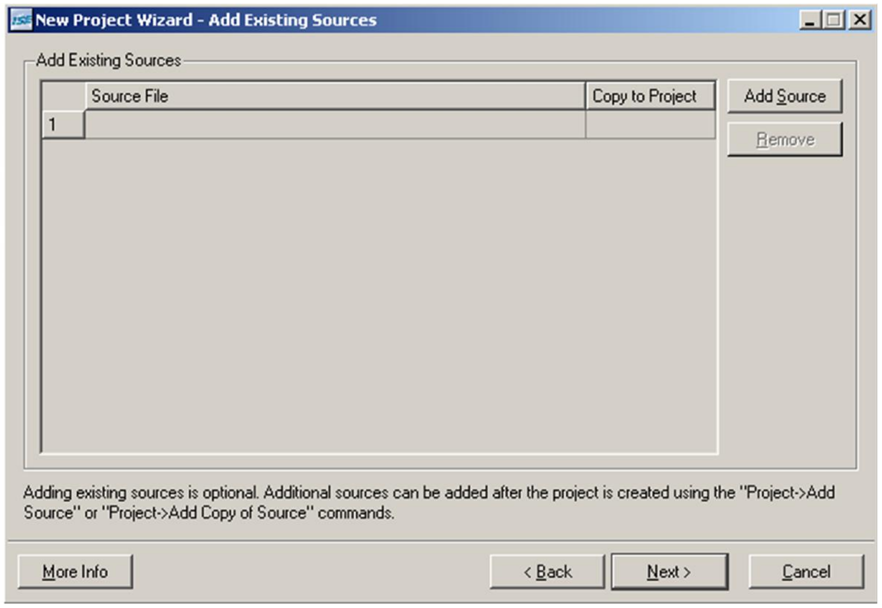
The following window will appear



Creating a new source to add to the project is optional. Only one new source can be created with the New Project Wizard. Additional sources can be created and added to the project by using the "Project->New Source" command.

Existing sources can be added on the next page.

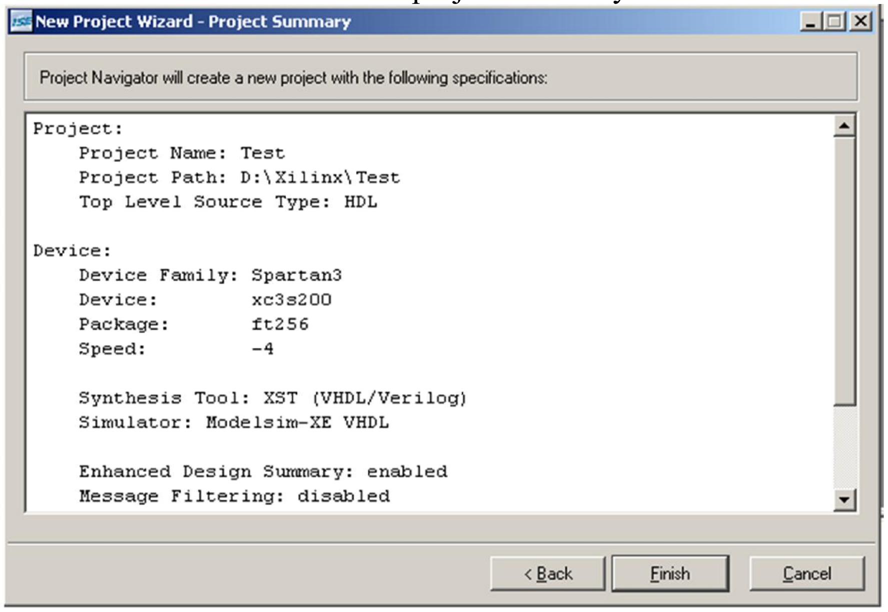
Now if you want to create a source file, then click New Source otherwise click Next> to add the existing Source file.
The following window will appear after pressing the Next>



Adding existing sources is optional. Additional sources can be added after the project is created using the "Project->Add Source" or "Project->Add Copy of Source" commands.

Click Add Source to add the source files and then press Next>

The next window will show the project summary as shown below.

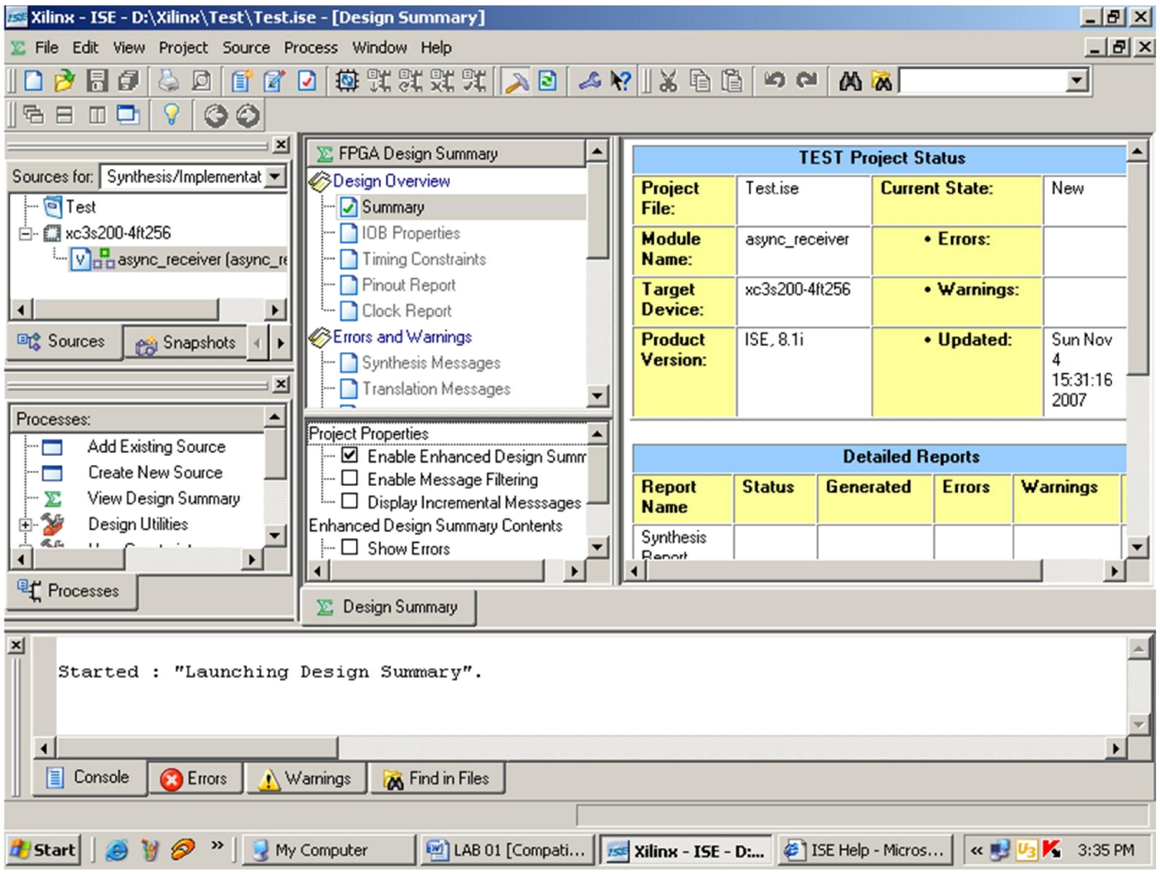


New Project Wizard - Project Summary

Project Navigator will create a new project with the following specifications:

```
Project:
    Project Name: Test
    Project Path: D:\Xilinx\Test
    Top Level Source Type: HDL

Device:
    Device Family: Spartan3
    Device:        xc3s200
    Package:       ft256
    Speed:         -4

    Synthesis Tool: XST (VHDL/Verilog)
    Simulator: Modelsim-XE VHDL

    Enhanced Design Summary: enabled
    Message Filtering: disabled
```

And press Finish.



Xilinx - ISE - D:\Xilinx\Test\Test.ise - [Design Summary]

Project Properties
- [✓] Enable Enhanced Design Summ
- [ ] Enable Message Filtering
- [ ] Display Incremental Messsages

Enhanced Design Summary Contents
- [ ] Show Errors

TEST Project Status			
Project File:	Test.ise	**Current State:**	New
Module Name:	async_receiver	**• Errors:**	
Target Device:	xc3s200-4ft256	**• Warnings:**	
Product Version:	ISE, 8.1i	**• Updated:**	Sun Nov 4 15:31:16 2007

Detailed Reports

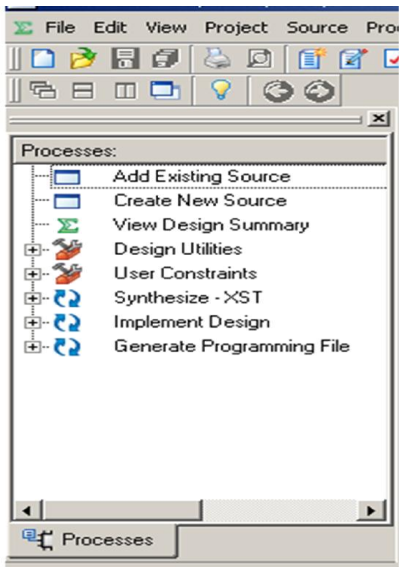
Report Name	Status	Generated	Errors	Warnings
Synthesis Report				

```
Started : "Launching Design Summary".
```

The code can be synthesized by the following steps.

1. First of all the USER DEFINED CONSTRAINT file is created and added in the project
2. Code is then synthesized
3. Design is implemented
4. Programming file is then generated which can be downloaded into the FPGA

The Process windows has all the above options



Lab Task:

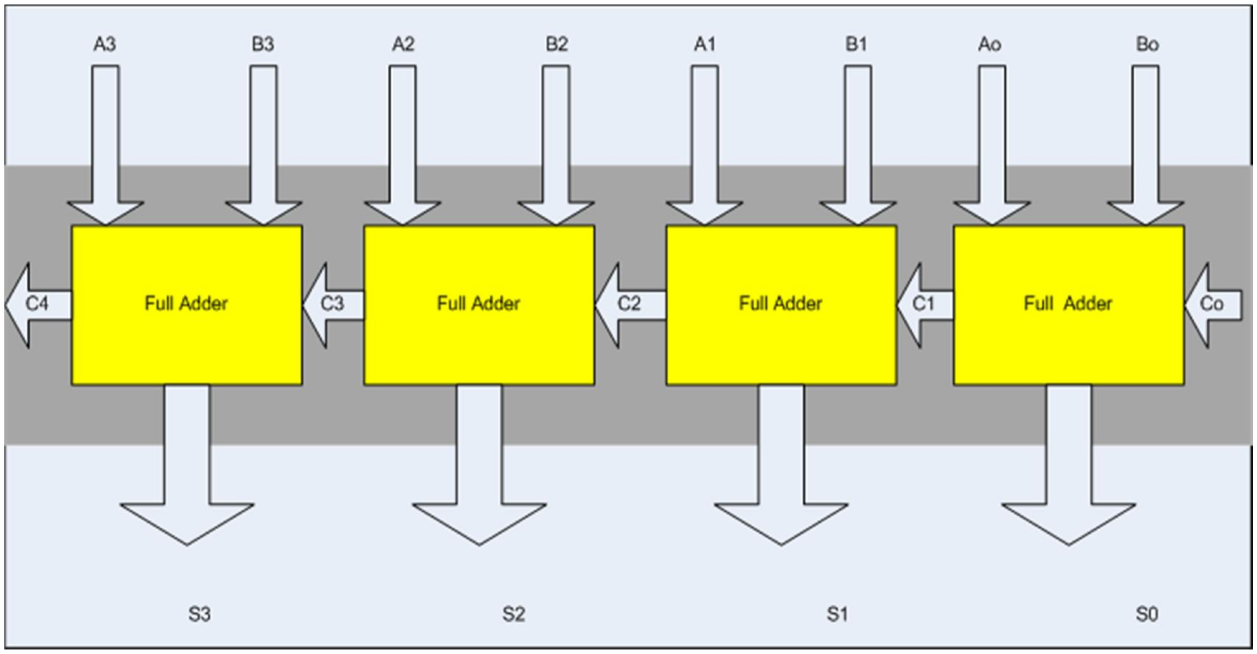
1- Develop a program to control on Board LED using On board available switch.
2- Develop a program that implements a 2x1 multiplexer on the board. Connect the inputs to switches and output to led. View the Synthesized Circuit and the Maximum Combinational path delay (Critical path).

Lab 3

4 bit Ripple Carry Adder

Objectives:
This lab will enable students to learn to top down and bottom up design methodologies.

Block Diagram: Following is the block diagram of a 4 bit Ripple Carry Adder.



I/O Connection: Ground "Co" Permanently and connect S0-S3 with four LEDs also connect C4 to another LED.

Lab Task:
The following steps should be performed while designing a 4 bit RCA adder
ModelSim:

1- First implement a Full adder using data flow/gate level modeling.
2- Simulate the Full adder with a test bench.
3- Instantiate the Full adder four times and connect the circuit as shown.
4- Now again write a test bench and simulate the 4 bit RCA.

Xilinx:

1. Make new project in Xilinx and add the files that you simulated in ModelSim.
2. Add User Constraint File inputs should be locked with the switches, C0 should be permanently "0" while S0-S4 and C4 with LEDS

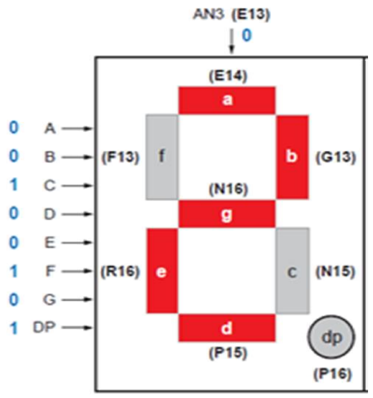
Lab 4

BCD to Seven Segment Decoder

Objective: To implement a BCD to Seven Segment Decoder on S3BOARD.

Block Diagram: In This lab you are required to take a BCD input from the user and display that number on the seven segment display. Following diagram shows the 7 bit code for displaying "2" on the seven segment display all the input are active low signals. Note that enable signal should be held low in order to turn on the particular seven segment display. There are 4 seven segment displays on the S3BOARD. In later labs you will learn how to use time multiplexing techniques to turn on all the four seven segment displays as the input A,B,C,D,E,F,G, Dp are shared by all the four seven segment displays.

I/O Connections: your module will have 4 bit input and 8 bit output. Connect input to switches and output to seven segment display.



Lab Task:

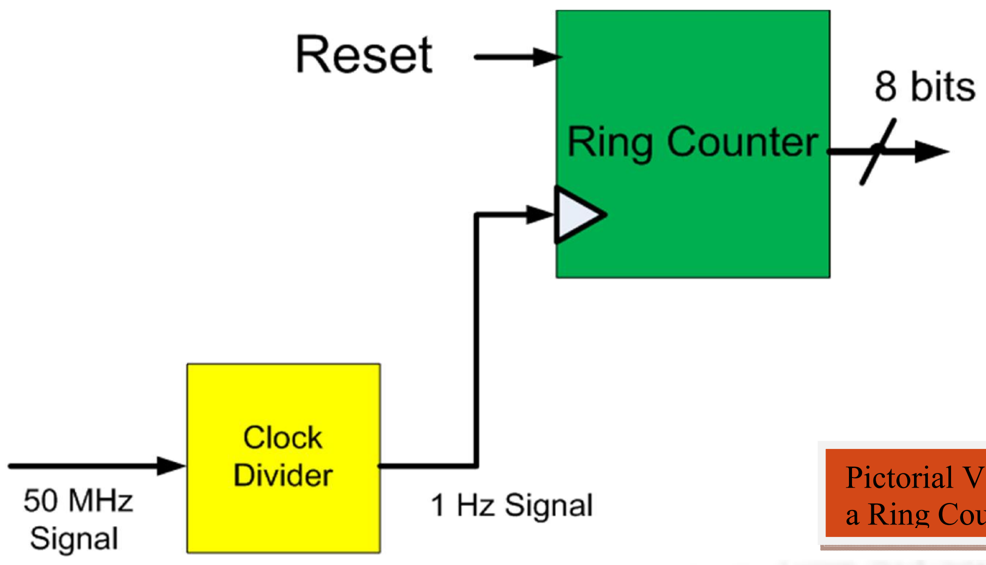
1- Using switches enter a BCD number and show the resulting number on the seven segment display.
2- Connect the output of your lab assignment 1 S0-S4 to the seven segment display. Note that number above 1001 are not valid BCD numbers. In this situation keep the seven segment display off and just the dp on.

Lab 5

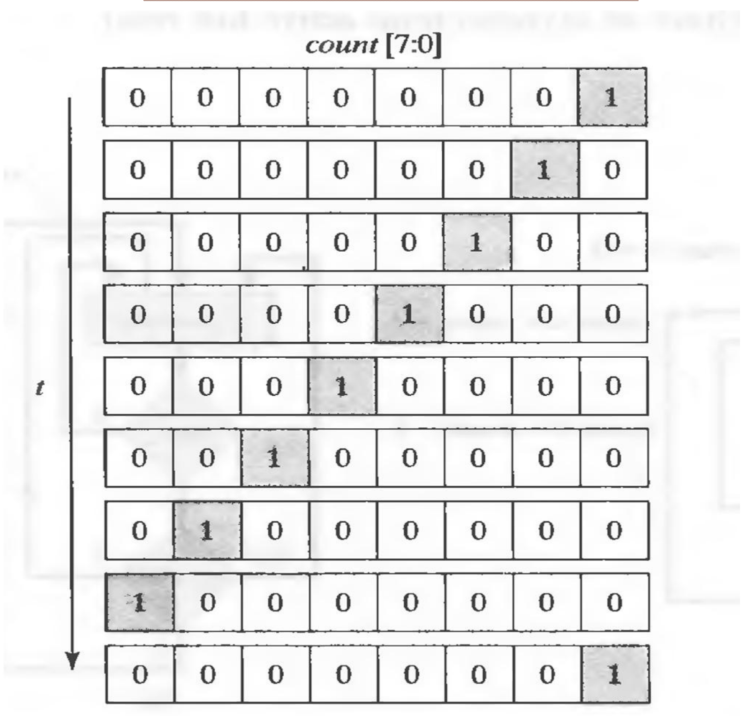
Implementation of a 8 bit Ring Counter

Objective: To Implement an 8 Bit Ring Counter on Spartan 3 FPGA starter kit.

Block Diagram: The Spartan 3 starter kit has a clock source of 50 MHz . If we use it in applications like counters, the counter will count at an incredibly fast speed and we will not be able to see the output. Your task is to divide the 50 Mhz frequency into a 1 Hz frequency. The Module Clock divider is responsible for it. The functional detail of the 8 bit Ring Counter is shown in the following figures.



Pictorial View of the Operation of a Ring Counter



count [7:0]

0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1

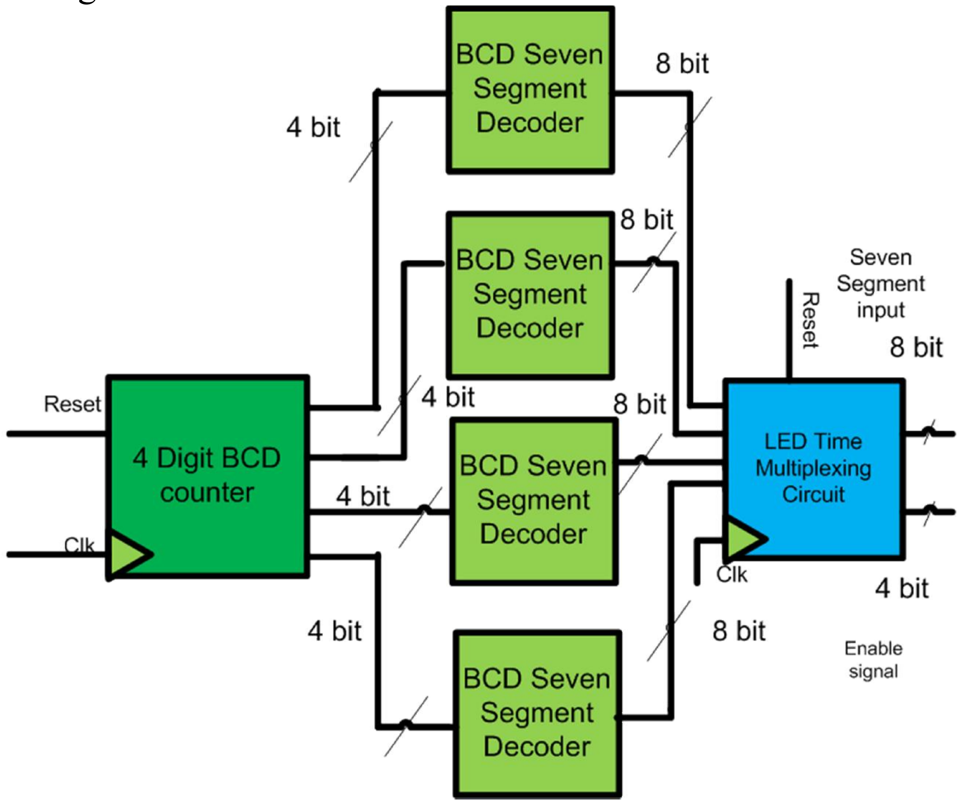
I/O Connection:
The output of the Ring Counter should be connected to 8 LEDs on Spartan 3 Starter Kit. The clock input is connected to pin "T9" on the board. The LEDs should turn on and off one by one from left to Right with an interval of 1 second.

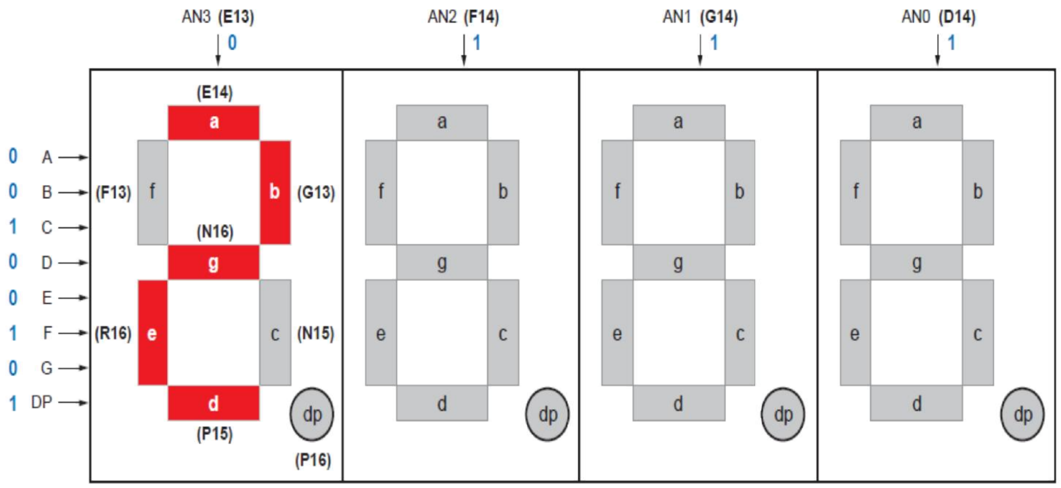
Lab No 6

4 digit BCD Counter on Multiplexed Seven Segment Display

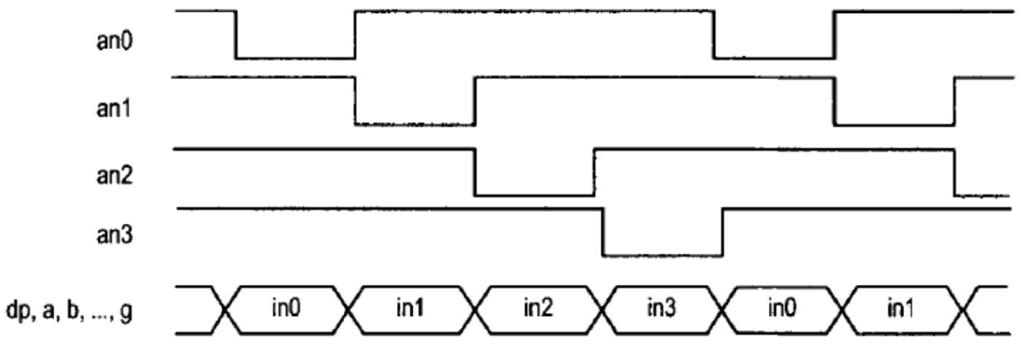
Objective: Learn to use time multiplexed 4 digit Seven Segment display

Block Diagram: The S3 board has four seven-segment LED displays, each containing seven bars and one small round dot. To reduce the use of FPGA's I/0 pins, the S3 board uses a time-multiplexing sharing scheme. In this scheme, the four displays have their individual enable signals but share eight common signals to light the segments. All signals are active low (i.e., enabled when a signal is 0). The schematic of displaying a "2" on the leftmost LED is shown in Figure.
Note that the enable signal (i.e., an) is "0111". This configuration clearly can enable only one display at a time. We can time-multiplex the four LED patterns by enabling the four displays in turn, as shown in the simplified timing diagram. If the refreshing rate of the enable signal is fast enough, the human eye cannot distinguish the on and off intervals of the LEDs and perceives that all four displays are lit simultaneously. This scheme reduces the number of I/O pins from 32 to 12 (i.e., eight LED segments plus four enable signals) but requires a time-multiplexing circuit.


BCD Seven Segment Decoder
BCD Seven Segment Decoder
BCD Seven Segment Decoder
BCD Seven Segment Decoder
4 Digit BCD counter
Reset
Clk
4 bit
4 bit
4 bit
4 bit
8 bit
8 bit
8 bit
8 bit
LED Time Multiplexing Circuit
Reset
Clk
Seven Segment input
8 bit
4 bit
Enable signal

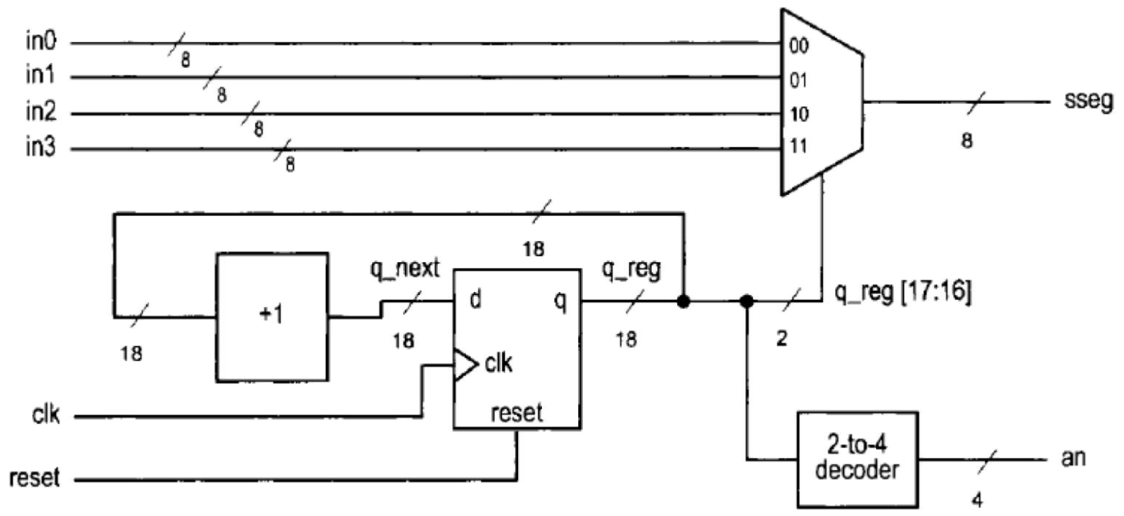


Timing Diagram of LED Multiplexign:



LED time Multiplexing:

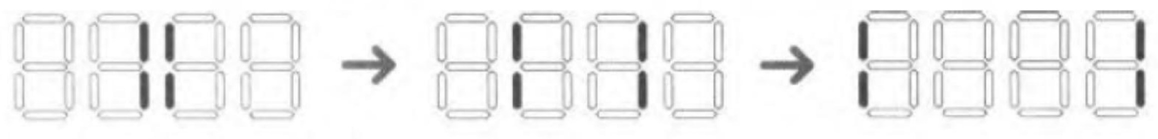
The refresh rate of the enable signal has to be fast enough to fool our eyes but should be slow enough so that the LEDs can be turned on and off completely. The rate around the range 1000 Hz should work properly. In our design, we use an 18-bit binary counter for this purpose. The two MSBs are decoded to generate the enable signal and are used as the selection signal for multiplexing. The refreshing rate of an individual bit, such as an0 becomes ($50 \times 10^6 / 2^{16}$) which is about 800 Hz.



Lab Tasks:

1- Implement a BCD counter that runs from 0000 to 9999 and shows each BCD digit on the seven segment display.

2- Implement the following sequence Such that the sequence repeats at 72 Hz.

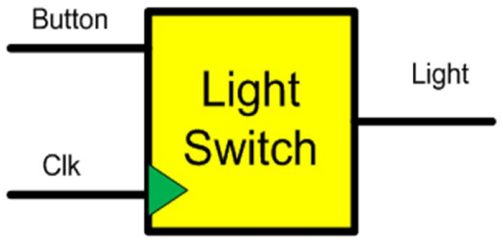


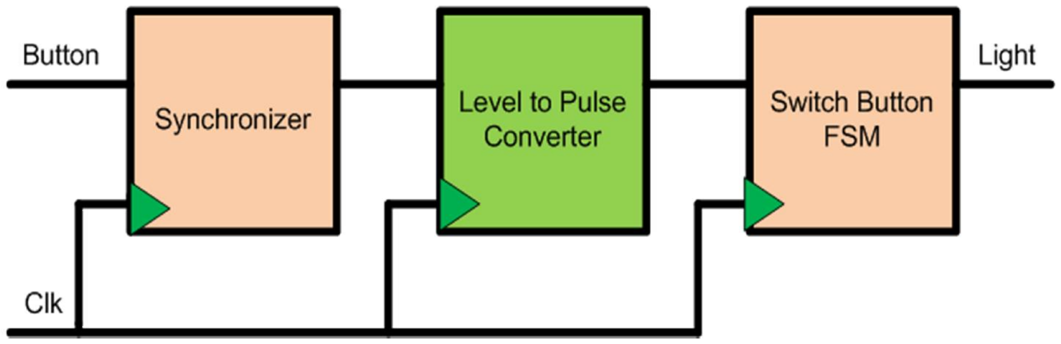
Lab No 7
Design of a Light switch

Objective: Build a light switch controller such that when the push button is pressed the light if “off” turns “on” and if “on” turns “off”.

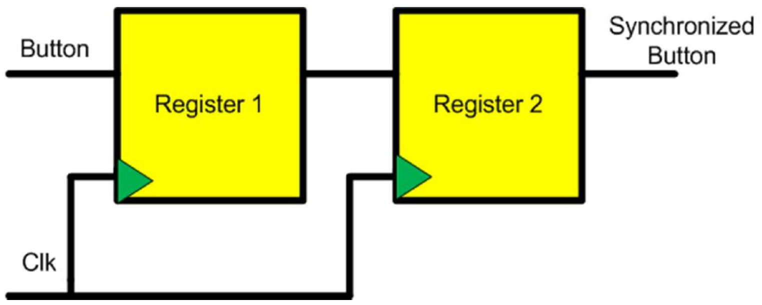
Block Diagram: The top level block diagram is shown in the following figure. The second diagram is a more detailed view of the top level diagram. Synchronizer circuit is used to avoid the metastability problem which arises when synchronous digital system are fed with an asynchronous input. The level to pulse converter is there to convert level input from a push button into a pulse that is high for only one clock cycle. The switch button FSM is actually implementing the state machine for the requirement i-e when the push button is pressed the light if “off” turns “on” and if “on” turns “off”.

I/O connection: Connect the button input to a push button on the S3BOARD and light output to LED.

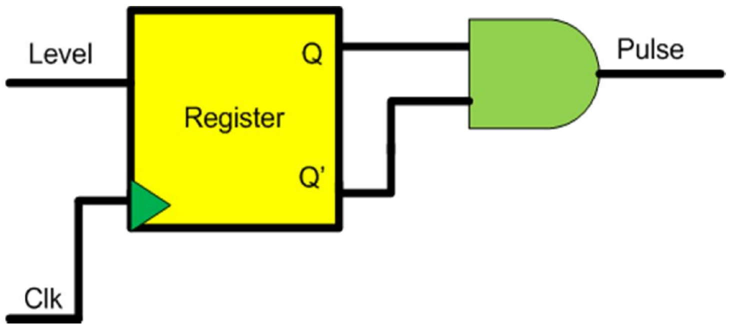




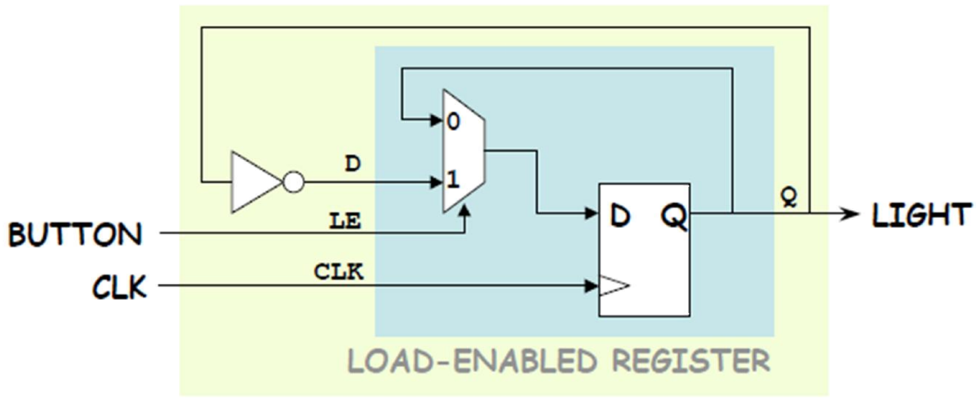
Synchronizer Circuit:



Level to Pulse FSM:



Switch Button FSM:

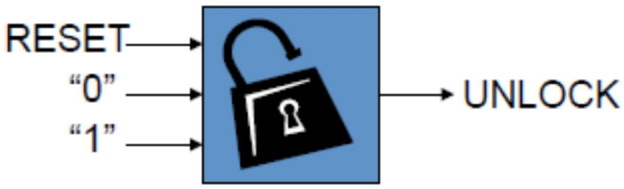


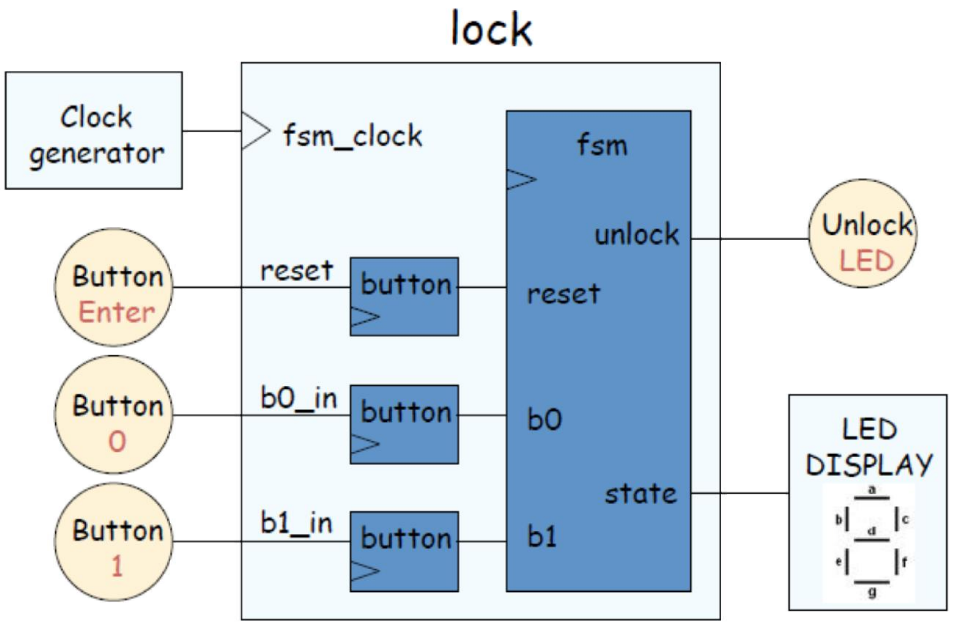
Lab 8

A Digital Lock

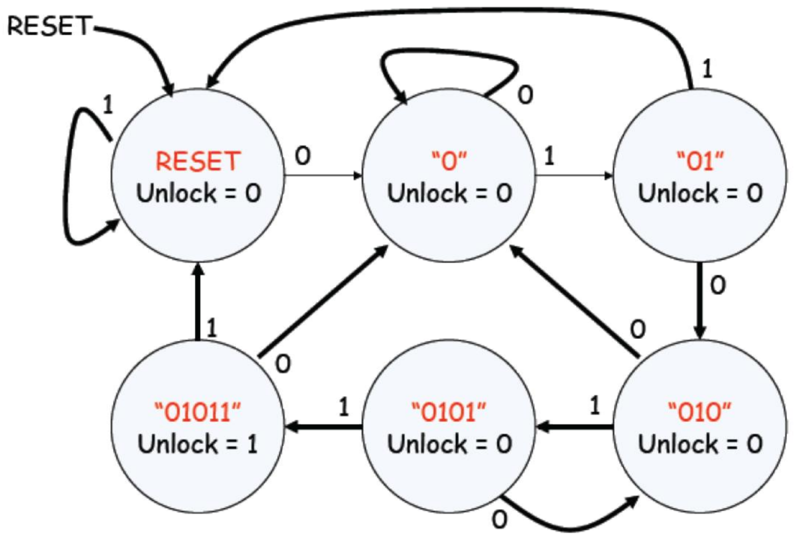
Objective: Build an electronic combination lock with reset button, two number buttons (0 and 1), and an unlock output. The combination should be “01011”

Block Diagram: A combinational digital lock has three input buttons for Reset, entering a “0” and entering a “1” and output button UNLOCK and state which shows in which state the machine is currently in. The state output is connected to the seven segment display on the S3BOARD. The Module button in the below diagram is an abstraction of the synchronizer and level to pulse converter from the previous lab. The state transition diagram is given in the following figures.





State Transition Diagram:



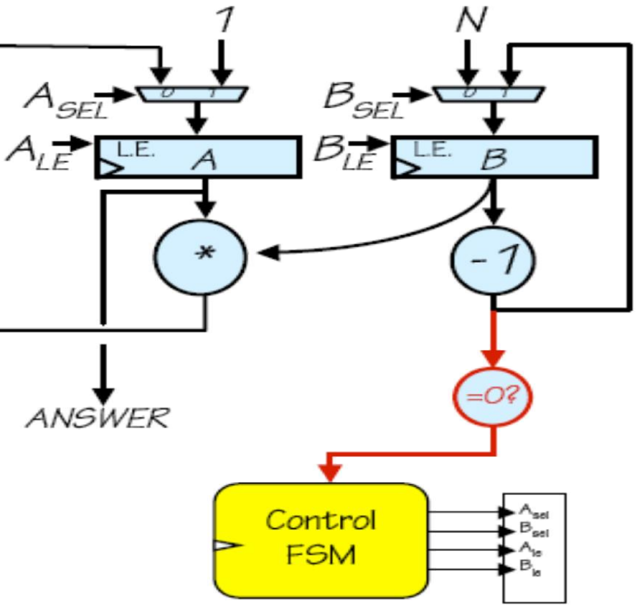
Lab Tasks:

1- Change the functionality of the lock such that it unlocks on the sequence of 11011.

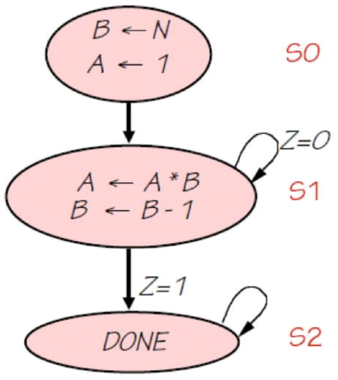
Lab No 9
A Taste of Data path + Control Design Example: Factorial Circuit

Objective : To implement a circuit that calculates factorial of a number.

Block Diagram: The datapath for calculating the factorial of an "N" bit number is given below. The datapath is controlled by a Finite State Machine (FSM). FSM generates signals Asel, Ale, Bsel and Ble at the correct times to operate the datapath. Input to FSM is a signal "Z" when Z=0 means the operation of the machine is complete and ANSWER can be read. The STG is also given in the following diagram.



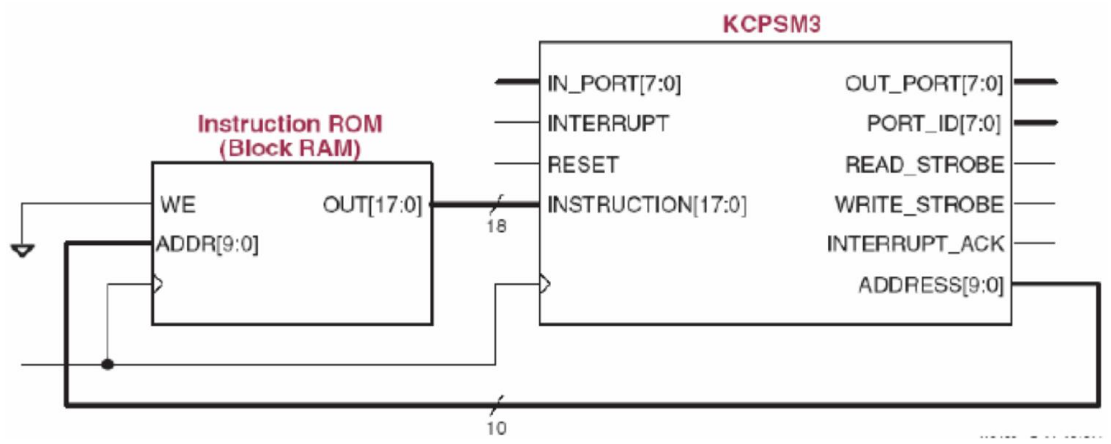
State Transition Diagram:



I/O Connection:
Connect the input N to switches so automatically N=8 bits and connect the ANSWER to 8 LEDs. Don't give input greater than 5 else the ANSWER will not 8 bit wide.

Lab No 10
Introduction to PicoBlaze Microcontroller

Objective: To learn how to use Xilinx PicoBlaze Microcontroller.
Block Diagram: The block diagram of PicoBlaze Microcontoller is given below. All the instruction resides in the instruction memory. Development tools are provided by Xilinx. Assembler, Simulator and source codes will be provided to you. In this lab you will write an assembly code to take input from the switches on the S3BOARD read it through PicoBlaze and Mirror it onto the LEDs.



Key Feature Set:

• 16 byte-wide general-purpose data registers
• 1K instructions of programmable on-chip program store, automatically loaded during FPGA configuration
• Byte-wide Arithmetic Logic Unit (ALU) with CARRY and ZERO indicator flags
• 64-byte internal scratchpad RAM
• 256 input and 256 output ports for easy expansion and enhancement
• Automatic 31-location CALL/RETURN stack
• Predictable performance, always two clock cycles per instruction, up to 200 MHz or 100 MIPS in a Virtex-4™ FPGA and 88 MHz or 44 MIPS in a Spartan-3 FPGA
• Fast interrupt response; worst-case 5 clock cycles
• Assembler, instruction-set simulator support

Following files are provided to you

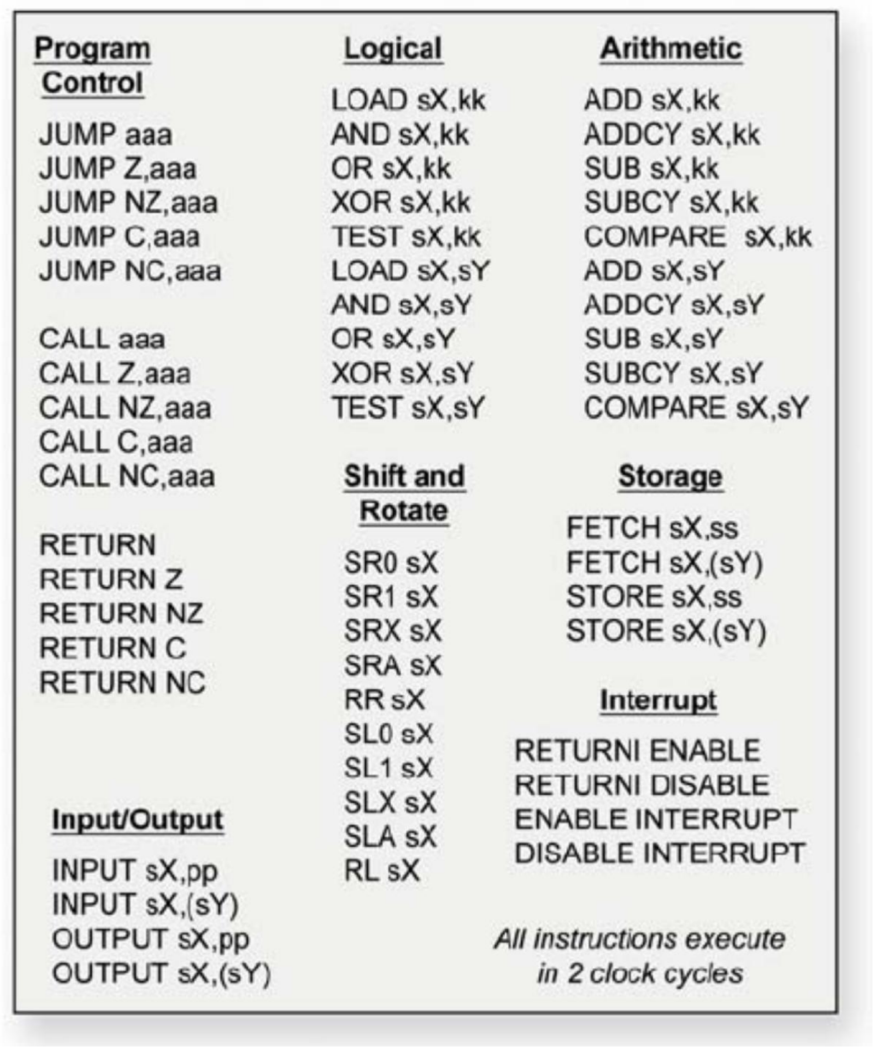
1- Kcpsm3.v This file contains the code for picoblaze microcontroller.

2- embedded_kcpsm3.v This file contains the instantiation of kcpsm3.v file and instruction rom instantiation. Note that instruction rom is automatically generated for you by the assembler from your input assembly code. If you assembly code file names is My_ROM.psm assembler will generate My_ROM.v file which is your instruction memory.

Steps to Follow:

1- First write assembly code in a notepad file save it with file type ".psm" lets say code.psm
2- Then open DOS prompt and come to the directory where your assembler KCPSM3 exists.
3- Now write at the DOS Prompt KCPSM3 code.psm the assembler will start assembling your file in the process it will generate about 9 files with the same name as your assembly code file but with different extensions. A file with extension ".v" will also be generated in this case "code.v" this is your verilog module for instruction memory.
4- Make project in Xilinx Synthesizer and add the files KCPSM3.v, embedded_kcpsm3.v, and code.v (generated by assembler) Note: you have to change the name of module instantiated inside embedded_kcpsm3.v to the module generated by assembler.
5- Now develop the constraint file for design. In this case, connect the input port of picoblaze to switches and output port to leds.
6- Now start synthesis.

PicoBlaze Instruction Set*



Program Control

JUMP aaa
JUMP Z,aaa
JUMP NZ,aaa
JUMP C,aaa
JUMP NC,aaa

CALL aaa
CALL Z,aaa
CALL NZ,aaa
CALL C,aaa
CALL NC,aaa

RETURN
RETURN Z
RETURN NZ
RETURN C
RETURN NC

Input/Output

INPUT sX,pp
INPUT sX,(sY)
OUTPUT sX,pp
OUTPUT sX,(sY)

Logical

LOAD sX,kk
AND sX,kk
OR sX,kk
XOR sX,kk
TEST sX,kk
LOAD sX,sY
AND sX,sY
OR sX,sY
XOR sX,sY
TEST sX,sY

Shift and Rotate

SR0 sX
SR1 sX
SRX sX
SRA sX
RR sX
SL0 sX
SL1 sX
SLX sX
SLA sX
RL sX

Arithmetic

ADD sX,kk
ADDCY sX,kk
SUB sX,kk
SUBCY sX,kk
COMPARE sX,kk
ADD sX,sY
ADDCY sX,sY
SUB sX,sY
SUBCY sX,sY
COMPARE sX,sY

Storage

FETCH sX,ss
FETCH sX,(sY)
STORE sX,ss
STORE sX,(sY)

Interrupt

RETURNI ENABLE
RETURNI DISABLE
ENABLE INTERRUPT
DISABLE INTERRUPT

All instructions execute in 2 clock cycles