NEAR EAST UNIVERSITY

Faculty Of Engineering

Department Of Computer Engineering

FIFO (First In First Out)

Graduation Project
COM- 400

Student: Esra Tuğba Oğuzhanoğlu(20030624)

Supervisor: Mehmet Kadir Özakman

Nicosia - 2008

ACKNOWLEDGMENTS

Firstly, I would like to thank to my supervisor Mr Mehmet Kadir Özakman for his great advise and recomendation for finishing my project properly also, teaching and guiding me in others lectures

Secondly, I am greatly indepted to my family for their endless support from my starting day in my educational life until today. I will never forget the things that my parents did for me during my educational life.

Finally, I would like thank all my teachers in Near East University, including faculty of engineering. Specially to my Dean Mr.Rahib Abiyev and Advisor Mr.Kaan Uyar.

LIST OF ABBREVIATIONS

HDL	Hardware Description Language
FIFO	First In First Out
VHSIC	Very High Speed Integrated Circuit
ASIC	Application Specific Integrated Circuits
RAM	Random Access Memory
RTL	Register Transfer Level
FPGAQ	Field Programmable Gate Arrays
CLB	Configurable Logic Devices
PLD	Programmable Logic Devices
IEEE	The Institute of Electrical and Electronics Engineers (read eyetriple-e)
IC	Intagrated Circuits
Ada	Name of Porgramming Language

ABSTRACT

Today's technology uses high level behavioral languages such as VDHL to do hardware electronic design. The project is selected in VHDL to learn the current technology and the methods to do hardware design. The name of the project is FIFO in computer science FIFO are use in queue structure. The fist data to be added to the queue will be the first data to be read. So the process proceeds sequentially in the same order.

Today's technology we don't go and buy a FIFO as a device we just write a VHDL for it and the development tools (Xilinx -ISE) generates the design and Implements the design in FPGA which stands for (Field Programmmable Gate Array) as I shown in my design description.

As I can from this implementation I did not have to do the detail electronic design all I did is to write a VHDL code to describe FIFO and the ISE tools did the rest.

INTRODUCTION

The aim of this project is to design simulate and generate the programming code for a 512*36 FIFO to implement into Virtex-II FPGA device.

The Virtex-II FPGA series provides dedicated on-chip blocks of 18 Kbit True Dual-Port synchronous RAM for use in FIFO applications. My project describes a way to create a common-clock (synchronous) version of a 511 * 36 FIFO, with the depth and width being adjustable within the VHDL code.

The project consists of introduction, two chapters and conclusion.

Chapter one presents how to make a project by using ISE and the software properties which are used in the project.

Chapter two describes the development of FIFO and how it works.

Finally, the conclusion section presents the important results obtained within the project.

Table of Contents

1. VHDL GENERAL INFORMATION

1.1 What is VHDL

The "V" stands for VHSIC (Very High Speed Integrated Circuit), and the "HDL" stands for "Hardware Description Language'.

1.2 Application Areas

1.2.1 Electronic Design Process

Let us now look at the stages involved in designing an electronic system, and see which are the main application areas of VHDL.

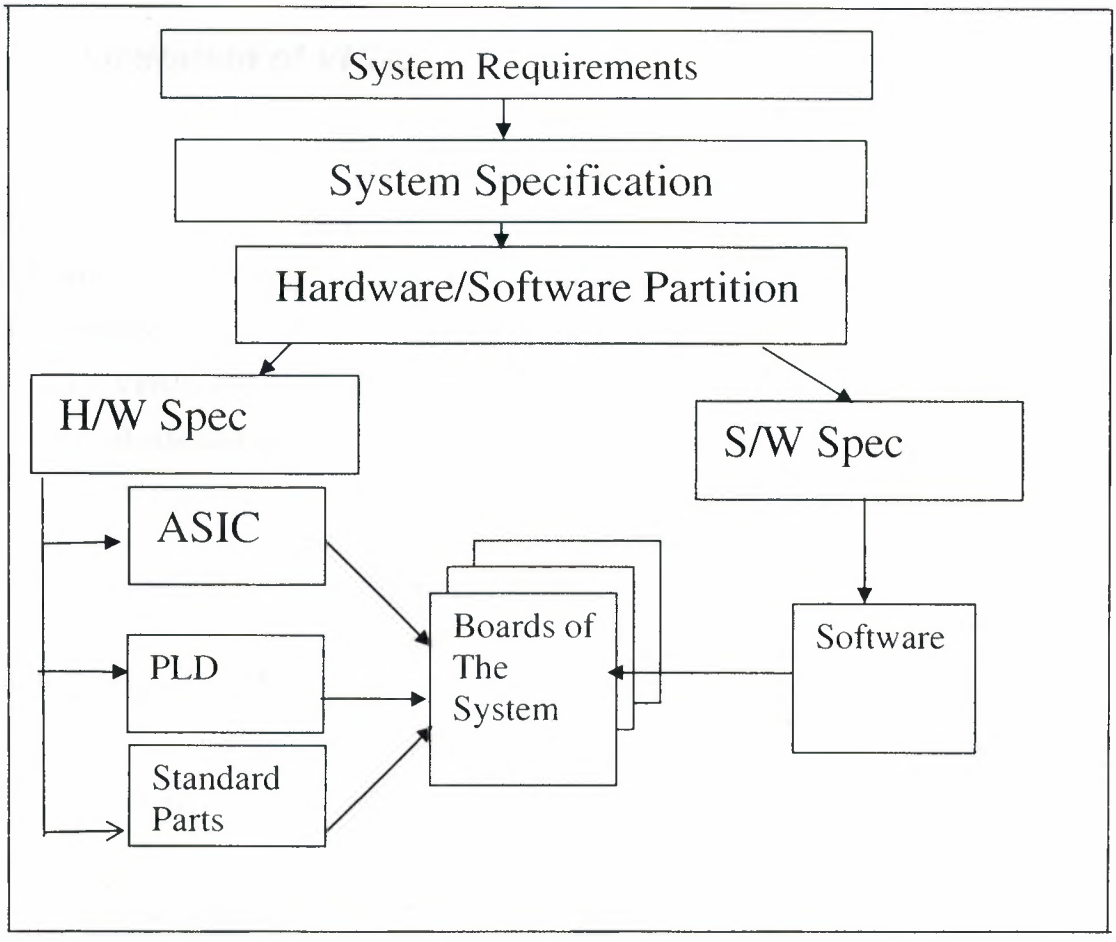

Figure 1- 1 Electronic Design Process

This Diagram shows the complete electronic design Process, from the requirements for the system, through hardware and software partitioning, down to the specification and implementation of the hardware and software parts of the completed system.

1.2.2. Hardware Implementation

In the early 1990s, VHDL was being used primarily for complex ASIC design, using synthesis tools to automatically create and optimize the implementation. Then the use of VHDL with synthesis has moved into the area of programmable logic design.

Modeling Specifications:

There is also an increase in the use of VHDL for modeling specifications, both of hardware part of the system, and the complete system itself.

1.3 Limitation of VHDL

VHDL is primarily a digital design language. It currently has very limited capabilities in the analog area, and there is a lot of work going on to standardize an analog version of the language. The 1076 standard defines a language and its syntax, without describing any styles of using it on a design project.

That VHDL code may need to be slightly modified before it can be used with different synthesis tool set than it was originally written for.

1.4 Levels of Abstraction

The different styles of writing VHDL code are to do with a concept known as abstraction. Abstraction defines how much detail about the design is specified in a particular description of it.

Let's look at the 4 main levels of abstraction to illustrate the principle.

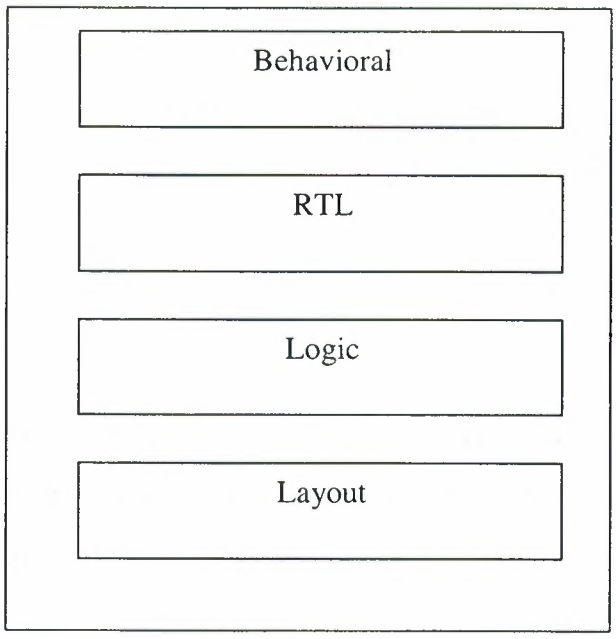

Figure 1- 2 Levels of Abstraction

Layout Level:

The lowest level of abstraction is the layout level. This specifies information about the actual layout of the design on silicon, and may also specify detail timing information and analog effects.

Logic Level:

Above the layout level is the logic level, where we interconnect logic gates and registers. Layout information is ignored, and the design contains information about the function, architecture, technology and detailed timing.

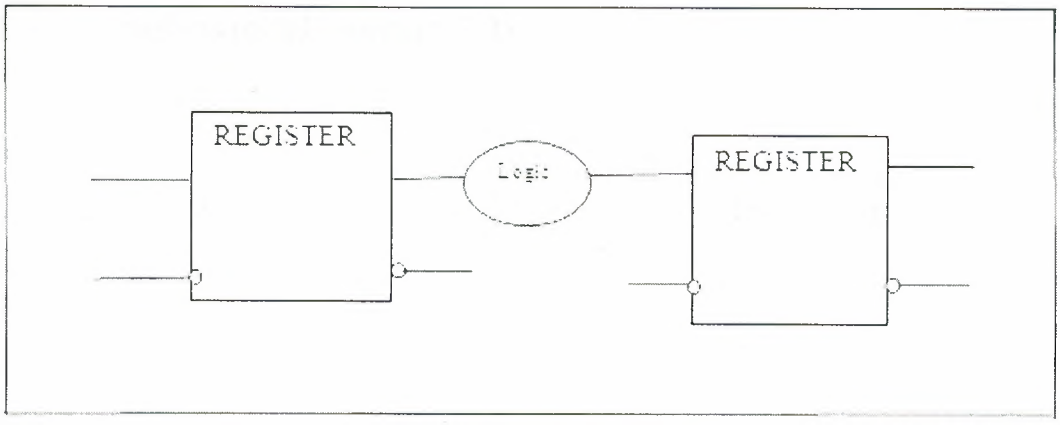

Figure 1- 3 Register Transfer Level

Register Transfer Level:

At the Register Transfer Level we Use VHDL in a strict style that defines every register in the design, and the logic between them. The design still contains architecture information but not the details of the details of the technology. Absolute timing delays are not specified.

Behavioral:

Above the RTL, we have the behavioral level. This level uses VHDL to describe the function of a design, without specifying the architecture of registers. Behavioral code can contain as much timing information as the designer requires representing his function.

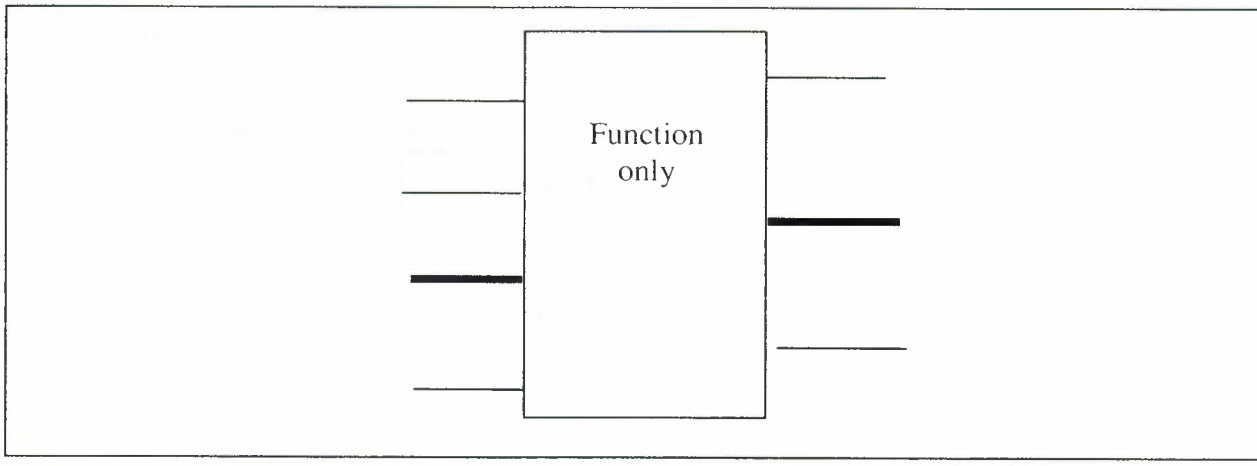

Figure 1- 4 Behavioral

1.5 Behavioral Versus RTL

So we see that there are at least two distinct styles of using VHDL: Behavioral, and RTL.

At RTL style the designer has control over the architecture of the registers in his design.

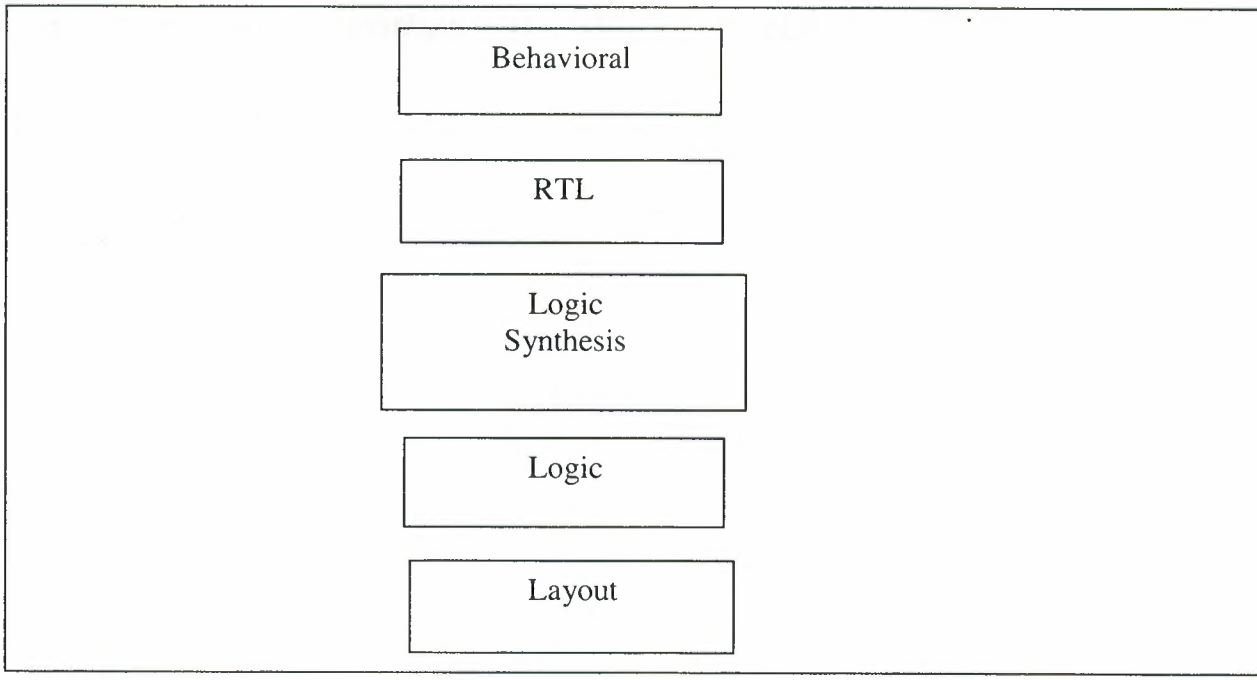

Figure 1- 5 Logic Synthesis

At Behavioral style the synthesis tools generates the architecture.

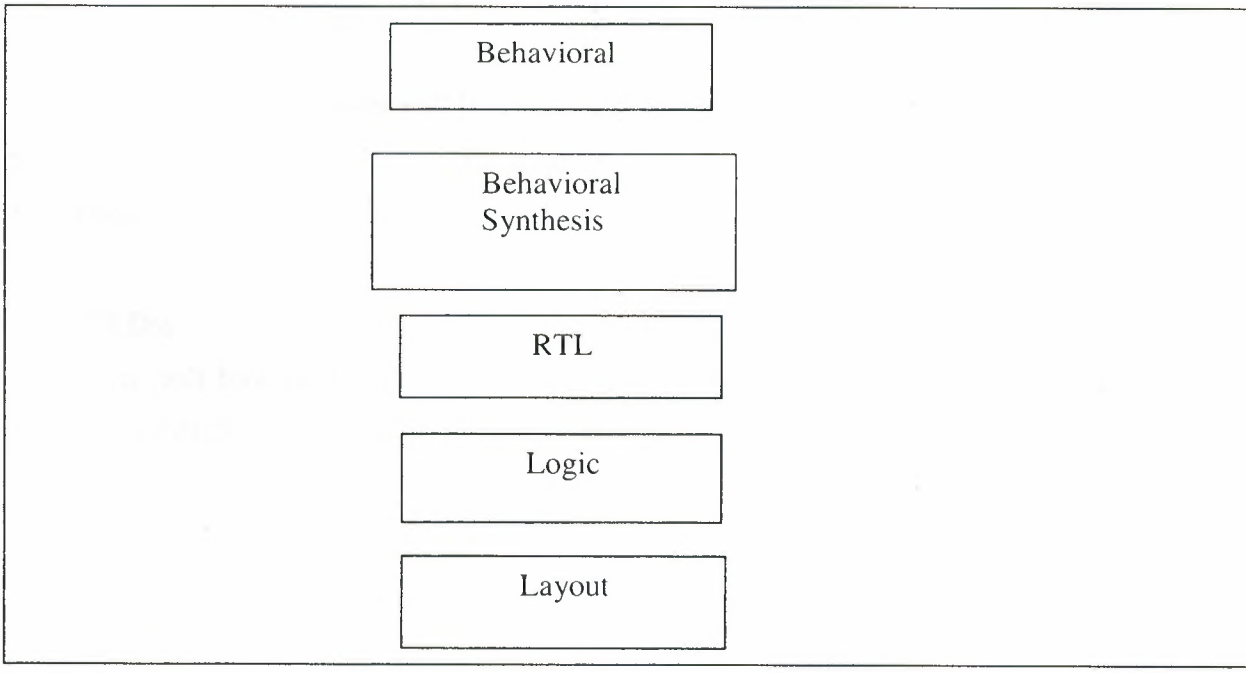

Figure 1- 6 Behavioral Synthesis

1.6. Overview of Synthesis for Xilinx Devices

1.6.1. What will be covered

PLD technologies:

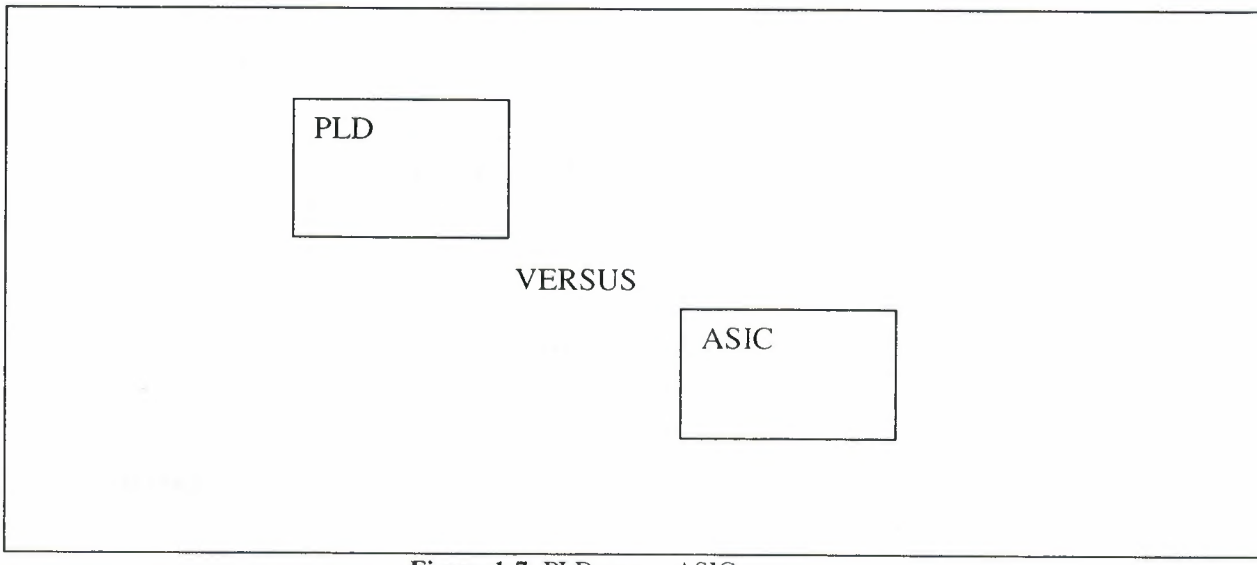

Figure 1-7 PLD versus ASIC

First PLD architectures will be discussed. How they differ from ASICs and how this effects design methodologies. Then we will architectures of two types of Xilinx PLD. These are FPGAs and CPLDs.

Xilinx PLDs:

Then will look at the architectures of the two types of Xilinx PLD. These are FPGAs and CPLDs.

Xilinx Architectures:

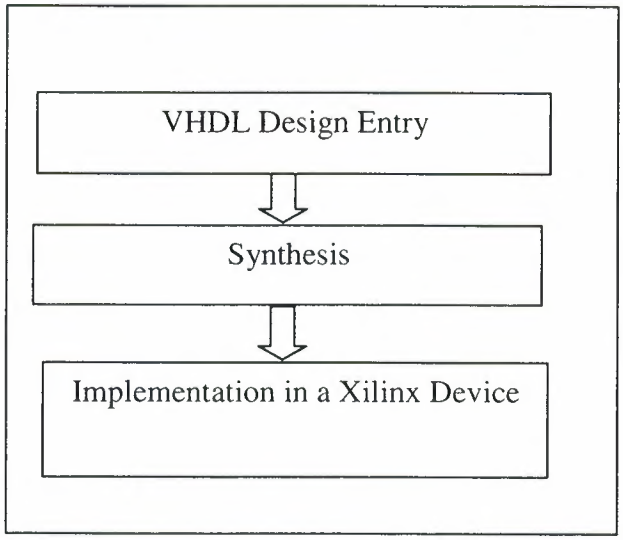

Figure 1-8 Xilinx Architecture

Then we will look at how these characteristics impact on the style in which you write your code.

1.6.2. Terminology

Here are the definitions Xilinx uses:

PLDs:

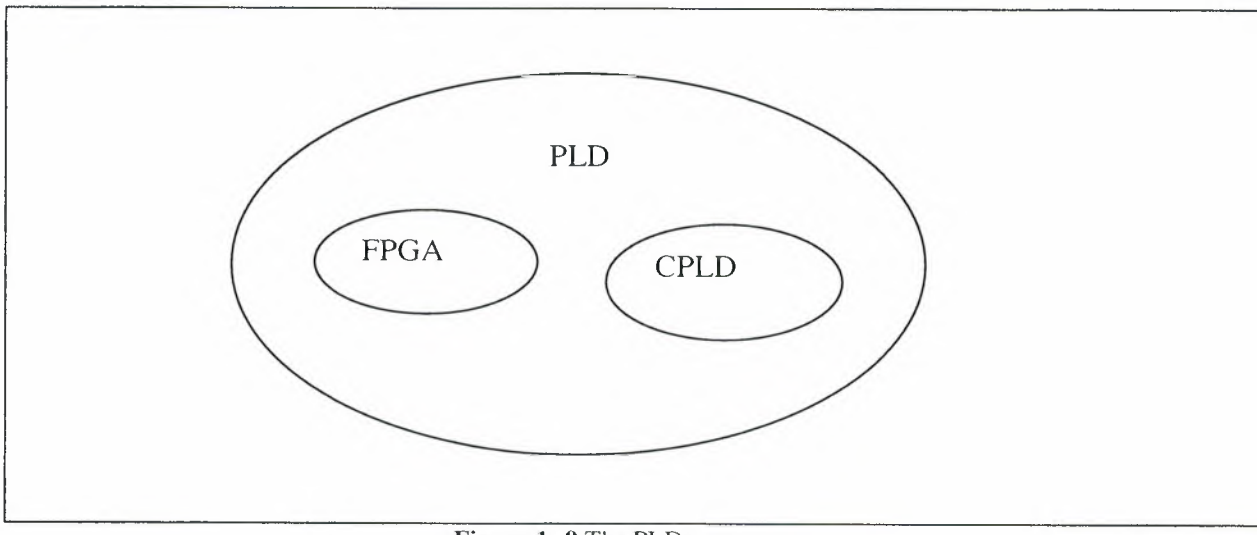

Figure 1- 9 The PLDs

The term PLD (Programmable Logic Device) covers all of the programmable devices that Xilinx offers. The two types of PLD that Xilinx offer are the FPGA ("Field Programmable Gate Array"), and the CPLD or "Complex PLD".

FPGAs, CLDs:

The Xilinx FPGA technologies are SRAM-based, such as the XC4000e. The Xilinx CPLD architecture are EPROM or FLASH technology based, en example of which is the XC9000 series.

Implementation:

The term "implementation is used to describe the process of turning the logic design into a physical design i.e. placing and routing the design and downloading into the target device.

1.6.3. PLD Synthesis Issues

Lets begin by looking into what we need to consider when using synthesis for the design of PLDs.

First synthesis users designed ASICs (Application Specific Integrated Circuits), where you can only program the device once.

Then it moved into PLD-based technologies. Where, you can re- program the device many times.

Synthesizing to PLD technologies:

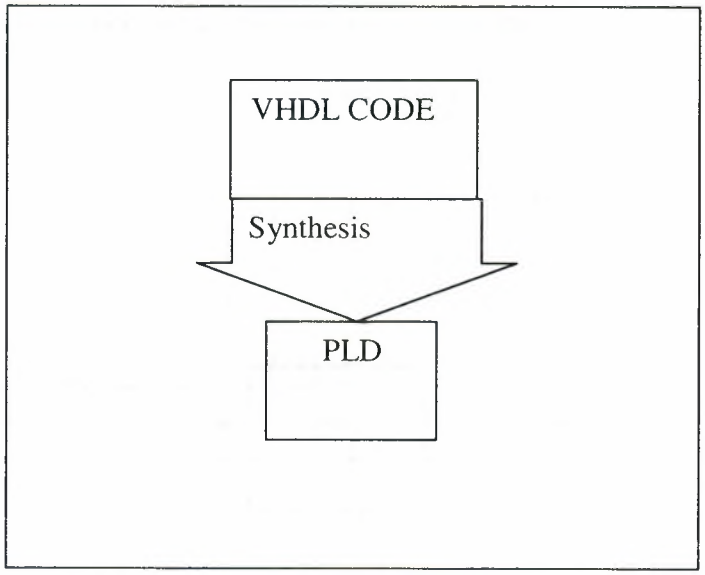

Figure 1- 10 Synthesis to PLD

From a synthesis perspective, the tool needs to contain specific algorithms to map the logic into the most efficient combination of the large building blocks.

1.6.4. Xilinx Device Architectures

We will give you an overview of the Xilinx device architectures.

1.6.5 FPGA Technologies

Xilinx FPGA technologies are based on static RAM building blocks which need to be downloaded with their logical function each time the system is powered up.

Configurable Logic Blocks (CLB):

Each building block in a given Xilinx FPGA device is known as Configurable Logic Block (CLB)

A CLB is a cell consisting of 8 or more inputs, 3 or more outputs, some combinational logic and two or more registers.

Connection by programmable switches:

The CPLDs are arranged into a fixed matrix and are connected by programmable switches. These form the required signal nets between CLBs.

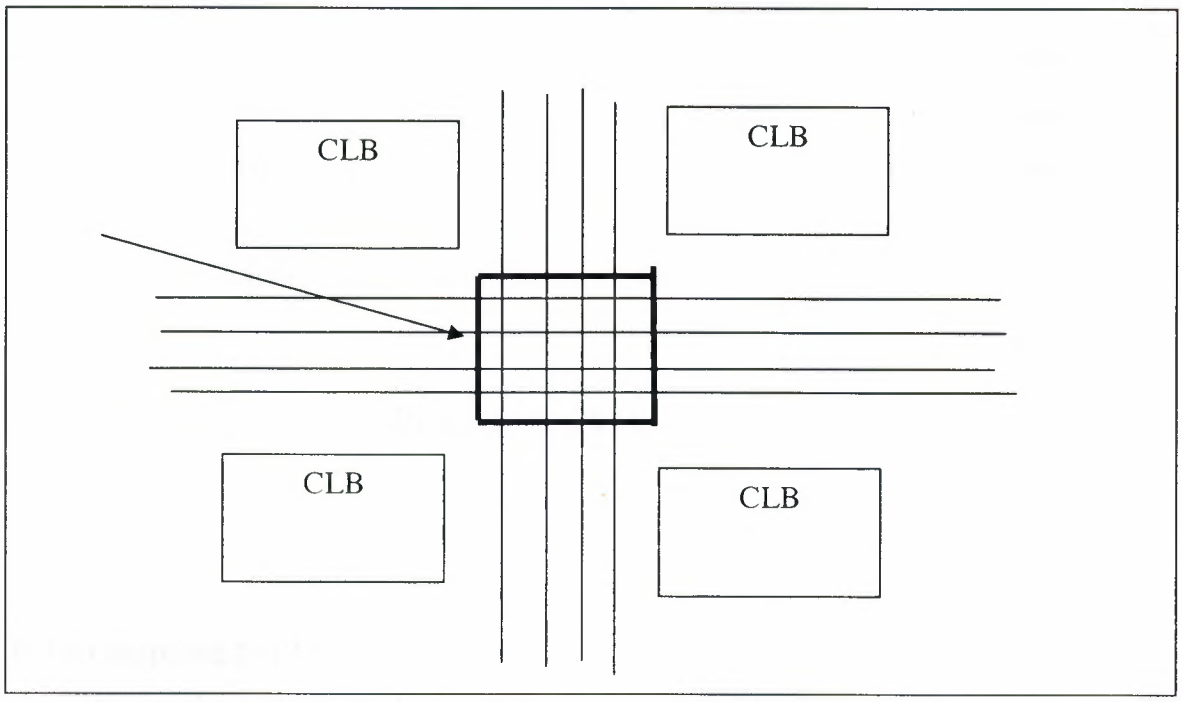

Figure 1- 11 Connection

IOBs provide IO (Input/Output) connections:

The matrix of logic cells is surrounded by IO cells called IOBs (Input Output Blocks). These IO cells have different structure from the CLBs and can be configured to provide different types of IO interface

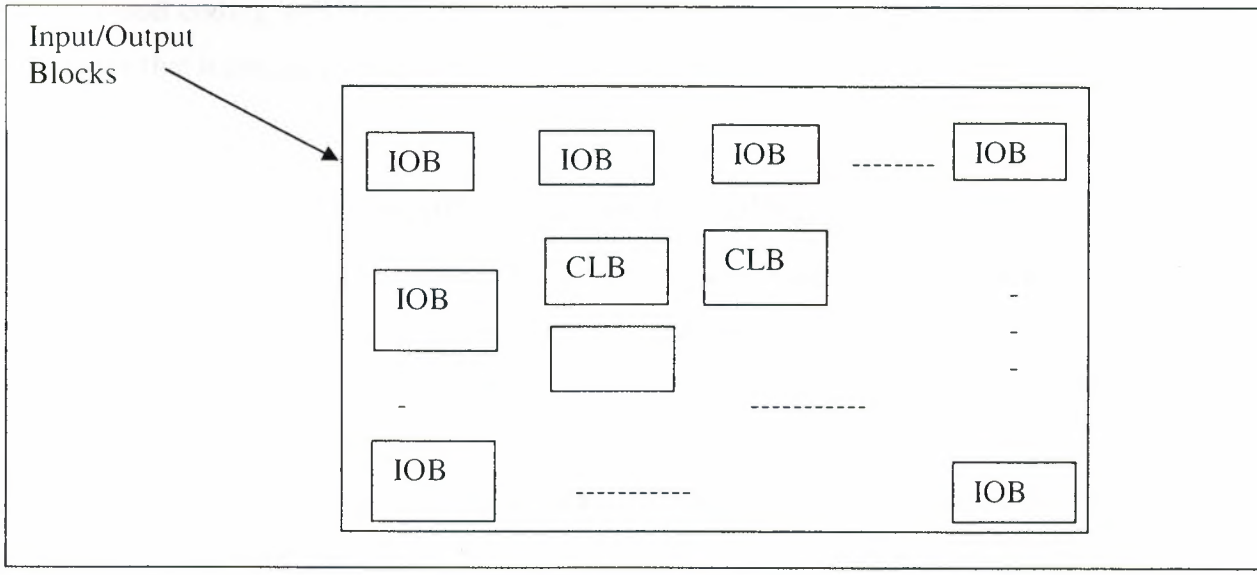

Figure 1- 12 I/O

1.6.6. CPLD Technologies

CPLD technologies are not based on static RAM building blocks and they don't need to be downloaded with their logical function each time the system is powered up. They have bigger physical size, and accommodate less function. CPLD technologies is mainly useful for smaller applications. CPLDs are based on a different type of building block than used in FPGAs. In a CPLD, each building block is referred to as a Function Block (FB). Each FB is comprised of macro cells, each capable of implementing a combinational or registered function.

These function blocks are connected via a switch matrix.

1.6.7. Where PLD Specific Issues Occur

So, having looked at the architecture of PLD technologies, we can now move on to look at the main areas of interest from the coding style and synthesis point of view.

Efficient mapping to PLD architecture

The synthesis tool must be able to map the VHDL Code into an efficient utilization of CLBs.

However, the style in which you write your code can help the synthesis tool to obtain better results.

Good Coding Style

Good coding style means that the synthesis tool can identify constructs within your code that it can easily map to technology features.

1.6.8. How the PLD Specific Issues are Handled

The key to using PLD resources efficiency is to write your VHDL so that it makes the best use of the architectures available within the target device.

Architecture independence

An important advantage of designing with VHDL is that the description can be independent of architecture, an can be re targeted to a new architecture if required.

However the code that is purely generic may not make the most efficient use of architecture specific features.

Architecture independence versus efficiency

A trade off exists here. You might want to keep the code architecture independent, leaving the synthesis tool to infer the best resources for the target device, or you might use architecture specific constructs to exploit pre-optimized functions at he expense of architecture independence.

Inference can be difficult

In fact not all functions can be inferred by a synthesis tool. For instance, some tools do not infer RAM. Hence users are sometimes required to make specific reference to these parts in their code. This is known as instantiation.

2. FIFO USING VIRTEX-II BLOCK RAM DESIGN WITH ISE

2.1. Design Description

My design is synchronous 512*36 (512 adresses and each address has 36 bit or data) FIFO.

FIFO is an acronym for First In, First Out. An abstraction is a ways of organizing and manipulation of data relative to time and prioritization.

In software FIFO is used in queue process.

In hardware FIFO is used commonly in electronic circuits for buffering and flow control.

2.1.1. Synchronous FIFO Using Common Clocks

Figure 1.1 is a block diagram of a synchronous FIFO. When both the Read and Write clocks originate from the same source, it simplifies the operation and arbitration of the FIFO, and the Empty and Full flags can be generated more easily. Binary counters are used for both the read (read_addr) and write (write_addr) address counters. Table 1-1 lists the Port Definitions for a synchronous FIFO design.

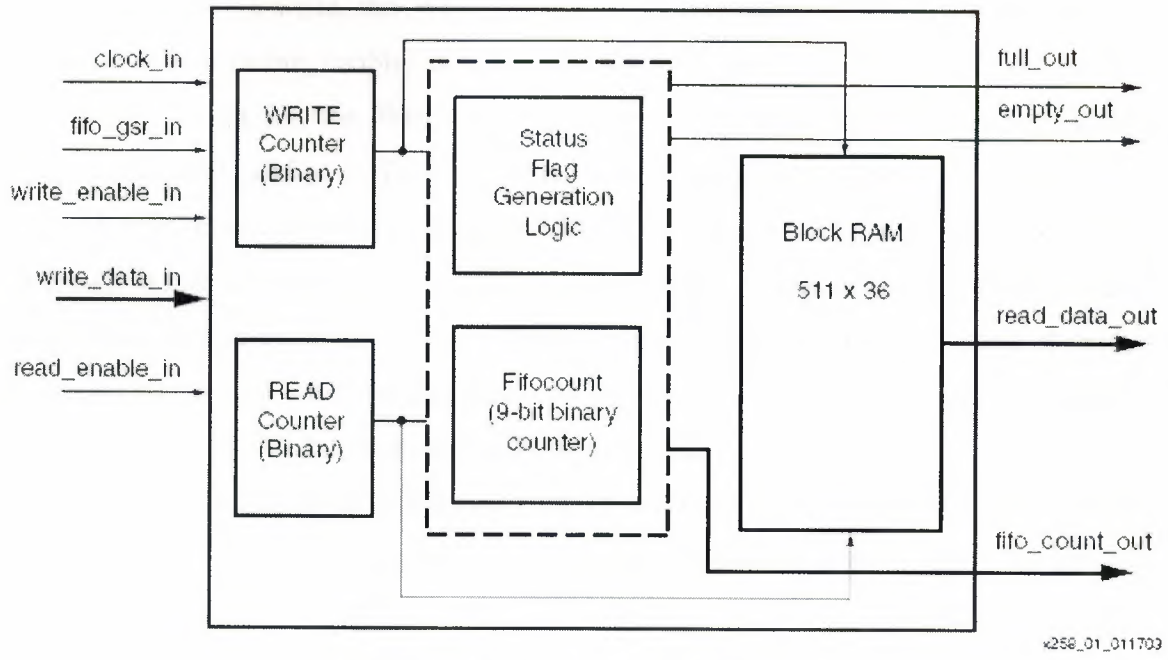

Figure 2.1 511*36 Synchronous FIFO

Table 1-1 Port Definitions

Signal Name	Port Direction	Port Width
clock_in	input	1
fifo_gsr_in	input	1
write_enable_in	input	1
write_data_in	input	36
read_enable_in	input	1
read_data_out	output	36
full_out	output	1
empty_out	output	1
fifocount_out	output	4

2.1.2. Synchronous FIFO Operation

To perform a read, Read Enable (read_enable) is driven High prior to a rising clock edge, and the Read Data (read_data) will be presented on the outputs during the next clock cycle. To do a Burst Read, simply leave Read Enable High for as many clock cycles as desired, but if Empty goes active after reading, then the last word has been read, and the next Read Data would be invalid.

To perform a write, the Write Data (write_data) must be present on the inputs, and Write Enable (write_enable) is driven High prior to a rising clock edge. As long as the Full flag is not set, the Write will be executed. To do a Burst Write, the Write Enable is left High, and new Write Data must be available every cycle.

A FIFO count (fifocount) is added for convenience, to determine when the FIFO is 1/2 full, ¾ full, etc.. It is a binary count of the number of words currently stored in the FIFO. It is incremented on Writes, decremented on Reads, and left alone if both operations are performed within the same clock cycle. In this application, only the upper four bits are sent to I/O, but that can easily be modified.

The Empty flag is set when either the fifocount is zero, or when the fifocount is one and only a Read is being performed. This early decoding allows Empty to be set immediately after the last Read. It is cleared after a Write operation (with no simultaneous Read). Similarly, the Full flag is set when the fifocount is 511, or when the fifocount is 510 and only a write is being performed. It is cleared after a Read operation (with no simultaneous Write). If both a Read and Write are done in the same clock cycle, there is no change to the status flags. During Global Reset (fifo_gsr), both these signals are driven High, to prevent any external logic from interfacing with the FIFO during this time.

I used the VHDL (Very High Speed Hardware Description Language) to design this FIFO.

2.2. Design Flow

My design flow is shown in Figure 2.

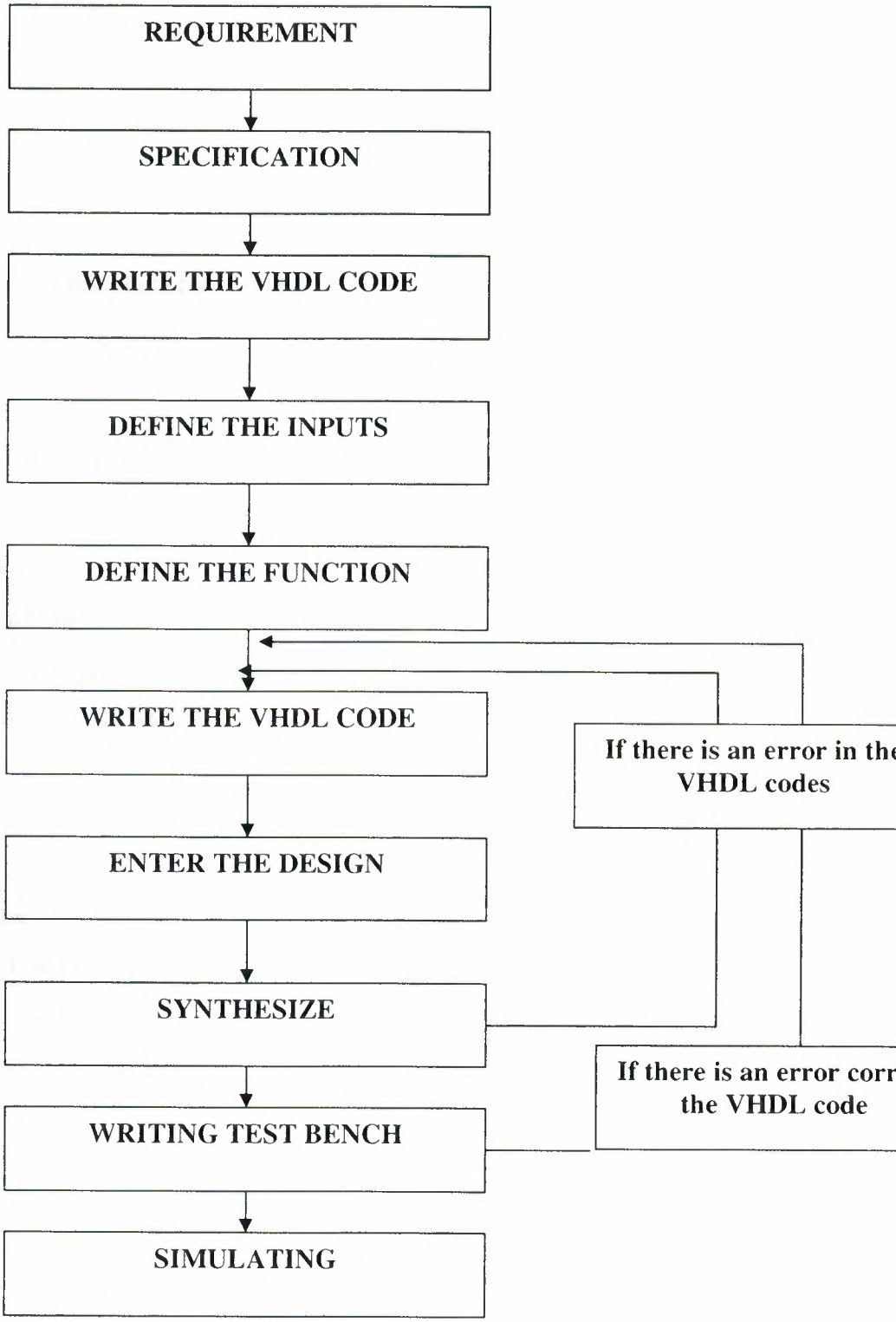

Figure 2.1 Design Process

2.2.1. Requirements

- FIFO
- Clock frequency is 160 MHz

2.2.2. Specification

- 512 locations
- 36 bits in each location
- The output and input will be synchronize with the clock

2.2.3. The Inputs and Outputs of the FIFO

inputs

- clock_in
- fifo_gsr_in
- write_enable_in
- write_data_in
- read_enable_in

outputs

- full_out
- empty_out
- read_data_out
- fifo_count_out

2.2.4. The Functions in VHDL Code

Write processes is in the VHDL code for the functions shown is in the blog diagram.Write processes for the following functions as shown in the blog diagram.

1. Processes for the Status Flag Generation Logic (proc1, proc2 and proc7 in the VHDL code)
2. Fifo count (9-bit binary counter) (proc3 and proc4 in my code)
3. Process for read counter (proc5)
4. Process for write counter (proc6)
5. Instantiatiated block RAM (RAMB16_S36_S36)

2.2.5. Write the VHDL Code

I used to entity section of the VHDL code to implement the inputs and outputs.I use the architecture section of the VHDL code to implement the processes.

```
----------------------------------------------------------------------------
--                                                                        --
-- Module     : fifoctlr_cc_v2.vhd        Last Update: 26/March/2008  --
--                                                                        --
-- Description : FIFO controller top level.                               --
--               Implements a 511x36 FIFO w/common read/write clocks.     --
--                                                                        --
-- The following VHDL code implements a 511x36 FIFO in a Virtex2          --
-- device.  The inputs are a Clock, a Read Enable, a Write Enable,        --
-- Write Data, and a FIFO_gsr signal as an initial reset.  The outputs    --
-- are Read Data, Full, Empty, and the FIFOcount outputs, which           --
-- indicate how full the FIFO is.                                         --
--                                                                        --
-- Designer    : Esra Tuğba Oğuzhanoglu                                   --
--                                                                        --
-- University  : YDU                                                      --
----------------------------------------------------------------------------

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

-- synopsys translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- synopsys translate_on

entity fifoctlr_cc_v2 is
  port (clock_in:       IN  std_logic;
```

```
        read_enable_in:  IN  std_logic;
        write_enable_in: IN  std_logic;
        write_data_in:   IN  std_logic_vector(35 downto 0);
        fifo_gsr_in:     IN  std_logic;
        read_data_out:   OUT std_logic_vector(35 downto 0);
        full_out:        OUT std_logic;
        empty_out:       OUT std_logic;
        fifocount_out:   OUT std_logic_vector(3 downto 0));
END fifoctlr_cc_v2;

architecture fifoctlr_cc_v2_hdl of fifoctlr_cc_v2 is
  signal clock:              std_logic;
  signal read_enable:        std_logic;
  signal write_enable:       std_logic;
  signal fifo_gsr:           std_logic;
  signal   read_data:                                  std_logic_vector(35   downto   0)   :=
"000000000000000000000000000000000000";
  signal write_data:         std_logic_vector(35 downto 0);
  signal full:              std_logic;
  signal empty:             std_logic;
  signal read_addr:         std_logic_vector(8 downto 0) := "000000000";
  signal write_addr:        std_logic_vector(8 downto 0) := "000000000";
  signal fcounter:          std_logic_vector(8 downto 0) := "000000000";
  signal read_allow:        std_logic;
  signal write_allow:       std_logic;
  signal fcnt_allow:        std_logic;
  signal fcntandout:        std_logic_vector(3 downto 0);
  signal ra_or_fcnt0:       std_logic;
  signal wa_or_fcnt0:       std_logic;
  signal emptyg:            std_logic;
  signal fullg:             std_logic;
  signal gnd_bus:           std_logic_vector(35 downto 0);
  signal gnd:               std_logic;
  signal pwr:               std_logic;
```

```
component BUFGP
  port (
    I: IN std_logic;
    O: OUT std_logic);
END component;

component RAMB16_S36_S36
  port (
    ADDRA: IN std_logic_vector(8 downto 0);
    ADDRB: IN std_logic_vector(8 downto 0);
    DIA:   IN std_logic_vector(31 downto 0);
    DIB:   IN std_logic_vector(31 downto 0);
    DIPA:  IN std_logic_vector(3 downto 0);
    DIPB:  IN std_logic_vector(3 downto 0);
    WEA:   IN std_logic;
    WEB:   IN std_logic;
    CLKA:  IN std_logic;
    CLKB:  IN std_logic;
    SSRA:  IN std_logic;
    SSRB:  IN std_logic;
    ENA:   IN std_logic;
    ENB:   IN std_logic;
    DOA:   OUT std_logic_vector(31 downto 0);
    DOB:   OUT std_logic_vector(31 downto 0);
    DOPA:  OUT std_logic_vector(3 downto 0);
    DOPB:  OUT std_logic_vector(3 downto 0));
END component;

BEGIN
  read_enable <= read_enable_in;
  write_enable <= write_enable_in;
  fifo_gsr <= fifo_gsr_in;
  write_data <= write_data_in;
```

```
  read_data_out <= read_data;
  full_out <= full;
  empty_out <= empty;
  gnd_bus <= "00000000000000000000000000000000";
  gnd <= '0';
  pwr <= '1';

------------------------------------------------------------------------
--                                                                    --
-- A global buffer is instantianted to avoid skew problems.           --
--                                                                    --
------------------------------------------------------------------------

gclk1: BUFGP port map (I => clock_in, O => clock);

------------------------------------------------------------------------
--                                                                    --
-- Block RAM instantiation for FIFO.  Module is 512x36, of which one  --
-- address location is sacrificed for the overall speed of the design.--
--                                                                    --
------------------------------------------------------------------------

bram1: RAMB16_S36_S36 port map (ADDRA => read_addr, ADDRB => write_addr,
        DIA => gnd_bus(35 downto 4), DIPA => gnd_bus(3 downto 0),
        DIB => write_data(35 downto 4), DIPB => write_data(3 downto 0),
        WEA => gnd, WEB => pwr, CLKA => clock, CLKB => clock,
        SSRA => gnd, SSRB => gnd, ENA => read_allow, ENB => write_allow,
        DOA => read_data(35 downto 4), DOPA => read_data(3 downto 0) );

------------------------------------------------------------
--                                                        --
-- Set allow flags, which control the clock enables for   --
-- read, write, and count operations.                     --
--                                                        --
```

```
------------------------------------------------------------

proc1: PROCESS (clock, fifo_gsr)
BEGIN
  IF (fifo_gsr = '1') THEN
    read_allow <= '0';
  ELSIF (clock'EVENT AND clock = '1') THEN
    read_allow <= read_enable AND NOT (fcntandout(0) AND fcntandout(1)
                AND NOT write_allow);
  END IF;
END PROCESS proc1;

proc2: PROCESS (clock, fifo_gsr)
BEGIN
  IF (fifo_gsr = '1') THEN
    write_allow <= '0';
  ELSIF (clock'EVENT AND clock = '1') THEN
    write_allow <= write_enable AND NOT (fcntandout(2) AND fcntandout(3)
                AND NOT read_allow);
  END IF;
END PROCESS proc2;

fcnt_allow <= write_allow XOR read_allow;

------------------------------------------------------------
--                                                        --
-- Empty flag is set on fifo_gsr (initial), or when on the --
-- next clock cycle, Write Enable is low, and either the   --
-- FIFOcount is equal to 0, or it is equal to 1 and Read   --
-- Enable is high (about to go Empty).                     --
--                                                        --
------------------------------------------------------------

ra_or_fcnt0 <= (read_allow OR NOT fcounter(0));
```

```
fcntandout(0) <= NOT (fcounter(4) OR fcounter(3) OR fcounter(2) OR fcounter(1) OR
fcounter(0));
fcntandout(1) <= NOT (fcounter(8) OR fcounter(7) OR fcounter(6) OR fcounter(5));
emptyg <= (fcntandout(0) AND fcntandout(1) AND ra_or_fcnt0 AND NOT
write_allow);

proc3: PROCESS (clock, fifo_gsr)
BEGIN
  IF (fifo_gsr = '1') THEN
    empty <= '1';
  ELSIF (clock'EVENT AND clock = '1') THEN
    empty <= emptyg;
  END IF;
END PROCESS proc3;

------------------------------------------------------------
--                                                        --
-- Full flag is set on fifo_gsr (but it is cleared on the  --
-- first valid clock edge after fifo_gsr is removed), or   --
-- when on the next clock cycle, Read Enable is low, and   --
-- either the FIFOcount is equal to 1FF (hex), or it is    --
-- equal to 1FE and the Write Enable is high (about to go  --
-- Full).                                                  --
--                                                        --
------------------------------------------------------------

wa_or_fcnt0 <= (write_allow OR fcounter(0));
fcntandout(2) <= (fcounter(4) AND fcounter(3) AND fcounter(2) AND fcounter(1));
fcntandout(3) <= (fcounter(8) AND fcounter(7) AND fcounter(6) AND fcounter(5));
fullg <= (fcntandout(2) AND fcntandout(3) AND wa_or_fcnt0 AND NOT read_allow);

proc4: PROCESS (clock, fifo_gsr)
BEGIN
  IF (fifo_gsr = '1') THEN
```

```
    full <= '1';
  ELSIF (clock'EVENT AND clock = '1') THEN
    full <= fullg;
  END IF;
END PROCESS proc4;

-------------------------------------------------------------
--                                                         --
-- Generation of Read and Write address pointers.  They now --
-- use binary counters, because it is simpler in simulation, --
-- and the previous LFSR implementation wasn't in the       --
-- critical path.                                          --
--                                                         --
-------------------------------------------------------------

proc5: PROCESS (clock, fifo_gsr)
BEGIN
  IF (fifo_gsr = '1') THEN
    read_addr <= "000000000";
  ELSIF (clock'EVENT AND clock = '1') THEN
    IF (read_allow = '1') THEN
      read_addr <= read_addr + '1';
    END IF;
  END IF;
END PROCESS proc5;

proc6: PROCESS (clock, fifo_gsr)
BEGIN
  IF (fifo_gsr = '1') THEN
    write_addr <= "000000000";
  ELSIF (clock'EVENT AND clock = '1') THEN
    IF (write_allow = '1') THEN
      write_addr <= write_addr + '1';
    END IF;
```

```
  END IF;
END PROCESS proc6;

------------------------------------------------------------
--                                                        --
-- Generation of FIFOcount outputs.  Used to determine how --
-- full FIFO is, based on a counter that keeps track of how --
-- many words are in the FIFO.  Also used to generate Full --
-- and Empty flags.  Only the upper four bits of the counter --
-- are sent outside the module.                           --
--                                                        --
------------------------------------------------------------

proc7: PROCESS (clock, fifo_gsr)
BEGIN
  IF (fifo_gsr = '1') THEN
    fcounter <= "000000000";
  ELSIF (clock'EVENT AND clock = '1') THEN
    IF (fcnt_allow = '1') THEN
      IF (read_allow = '0') THEN
        fcounter <= fcounter + '1';
      ELSE
        fcounter <= fcounter - '1';
      END IF;
    END IF;
  END IF;
END PROCESS proc7;

fifocount_out <= fcounter(8 downto 5);

END fifoctlr_cc_v2_hdl;
```

I use the xilinx – ISE integrated software and enviroment to create the FIFO Project an enter the VHDL code.

2.3. Design Steps

2.3.1. Create the FIFO Project

Create the FIFO project which will target the FPGA device on the Virtex-2.

To create the FIFO project:

1. Select **File > New Project...** The New Project Wizard appears.
2. Type **fifoctrl_cc_v2** in the Project Name field.
3. Enter to location and a fifo subdirectory is created automatically.
4. Verify that **HDL** is selected from the Top-Level Source Type list.

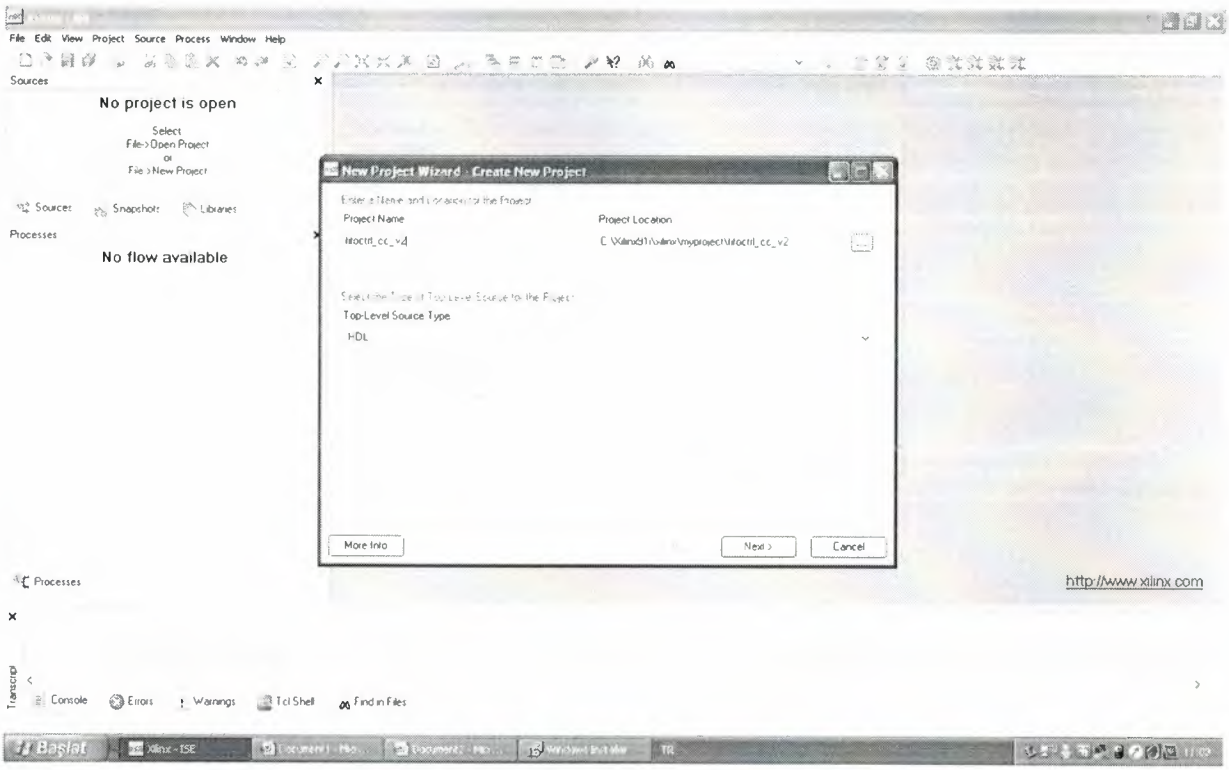

Figure 3.1 Project Name

5. Click **Next** to move to the device properties page.
6. Fill in the properties in the table as shown below:

- Product Category: **All**
- Family: **Virtex2**
- Device: **XC2V40**
- Package: **CS144**

- Speed Grade: **-6**
- Top-Level Source Type: **HDL**
- Synthesis Tool: **XST (VHDL/Verilog)**
- Simulator: **ISE Simulator (VHDL/Verilog)**
- Preferred Language: **VHDL**
- Verify that **Enable Enhanced Design Summary** is selected.

Leave the default values in the remaining fields.

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

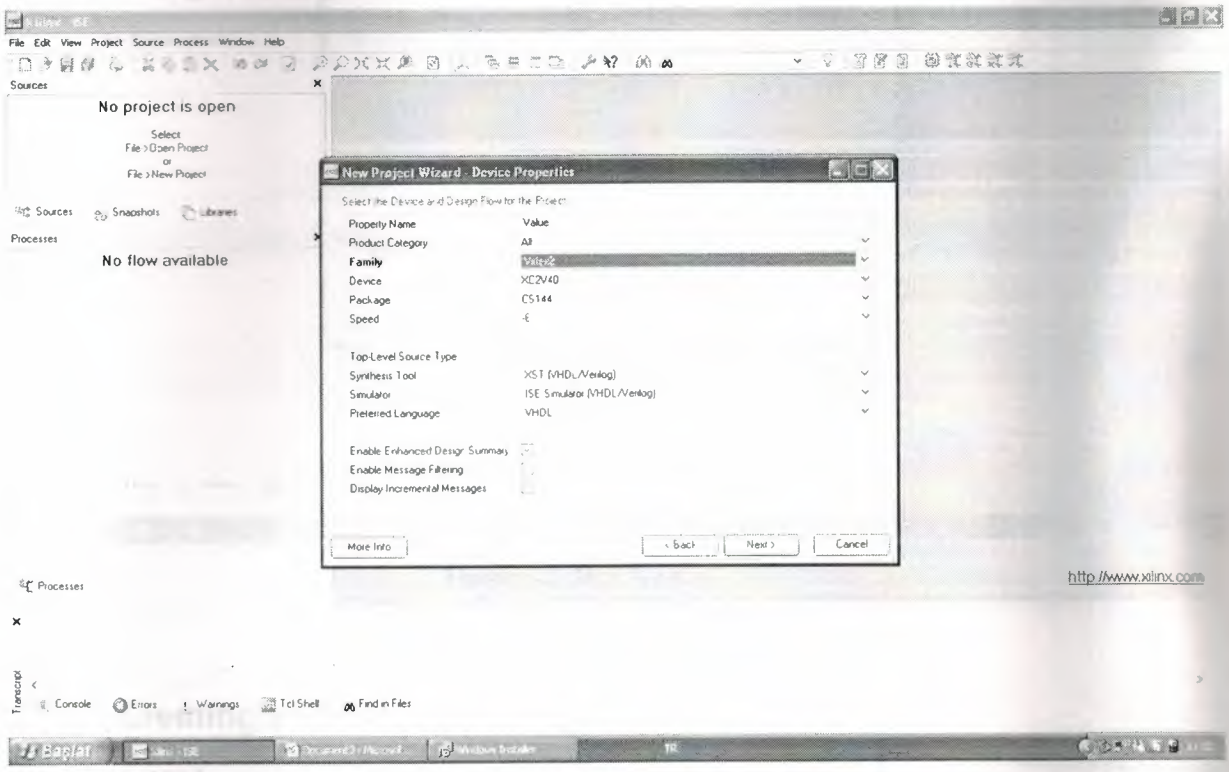

Figure 3.2 Project Device Properties

7. Click **Next** to proceed to the Create New Source window in the New Project Wizard. At

the end of the next section, my FIFO project will be complete.

2.3.2. Create the HDL Source of the FIFO

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

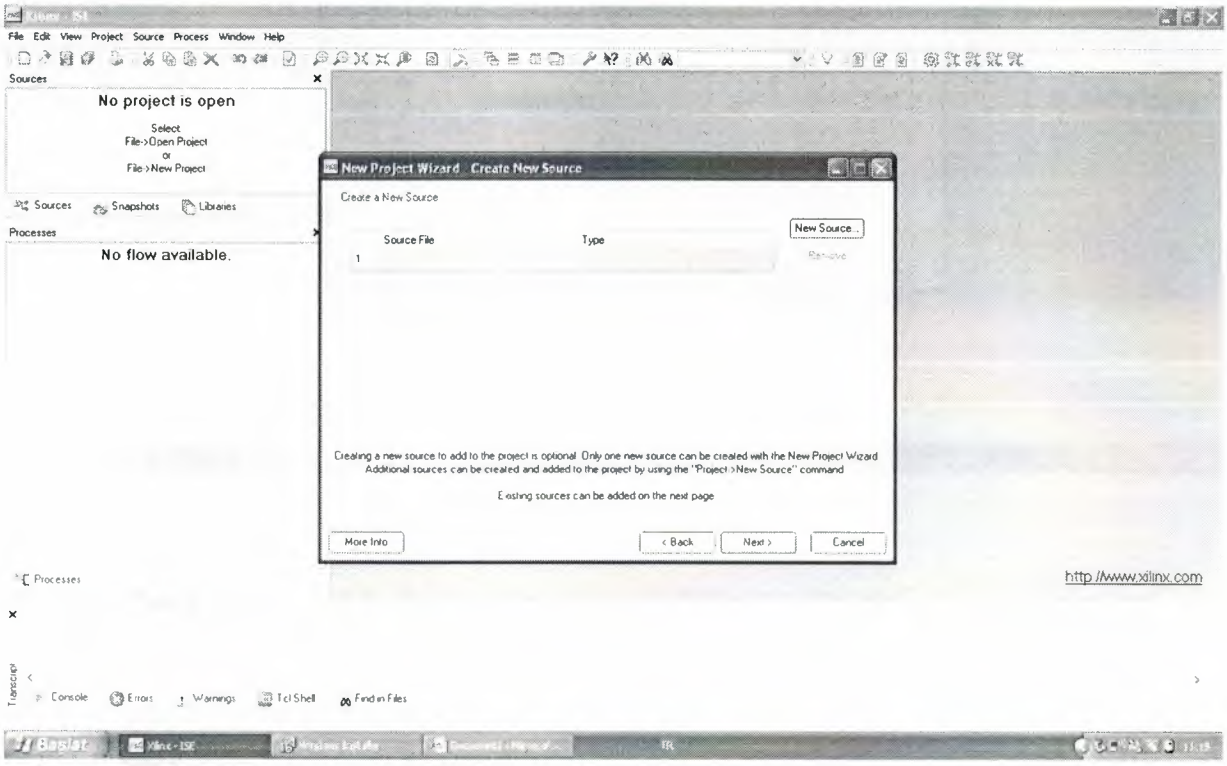

Figure 3.3 New Sources

2.3.3. Creating a VHDL Source

Create a VHDL source file for the project as follows:

1. Click the **New Source** button in the New Project Wizard.
2. Select **VHDL Module** as the source type.
3. Type in the file name **fifoctrl_cc_v2**.
4. Verify that the **Add to project** checkbox is selected.
5. Click **Next**.
6. Declare the ports for the fifo design by filling in the port information as shown below.

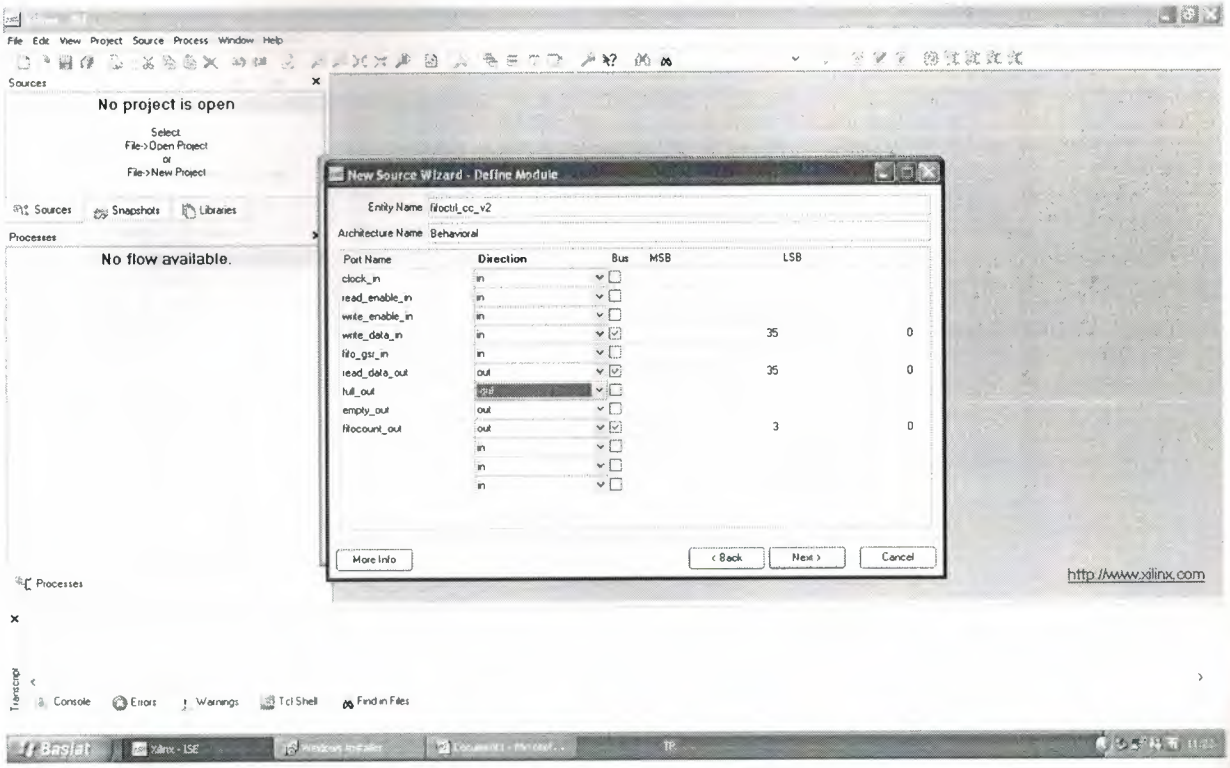

Figure 3.4 Define Module

7. Click **Next**, then **Finish** in the New Source Wizard - Summary dialog box to complete

the new source file template.

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

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

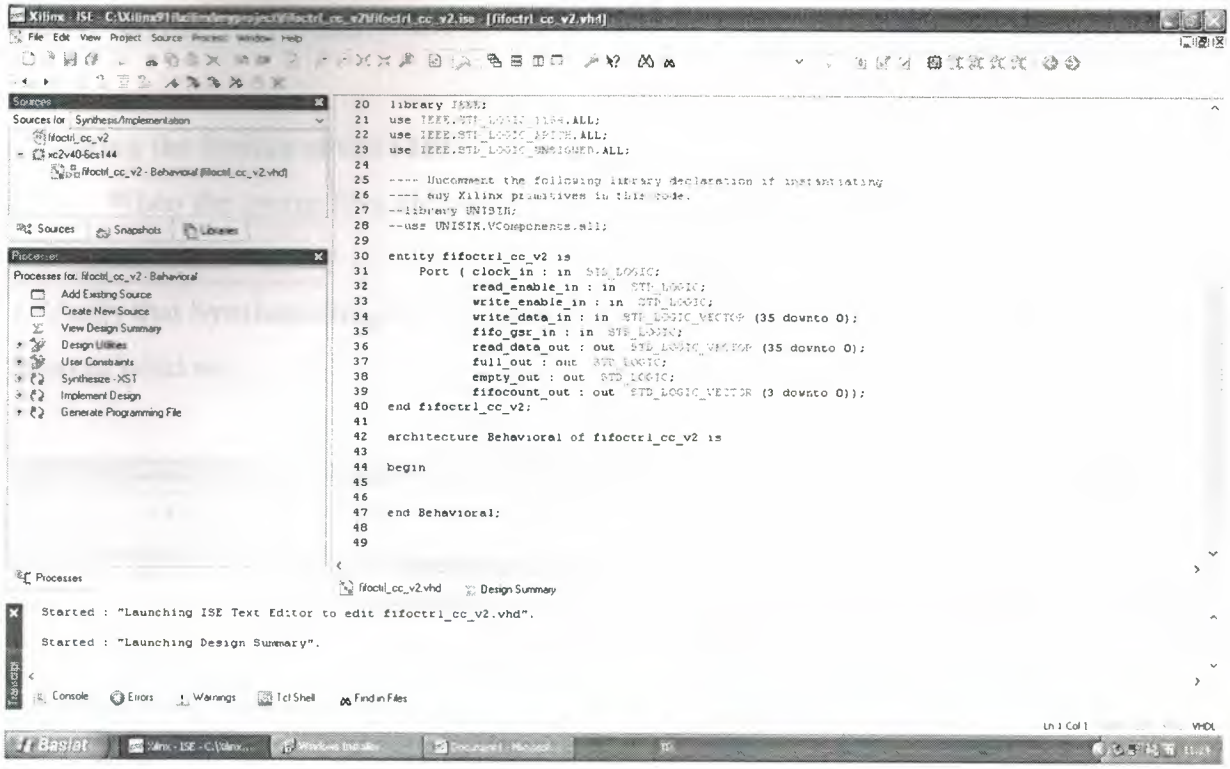

Figure 3.5 FIFO Project in ISE

Complete VHDL Source

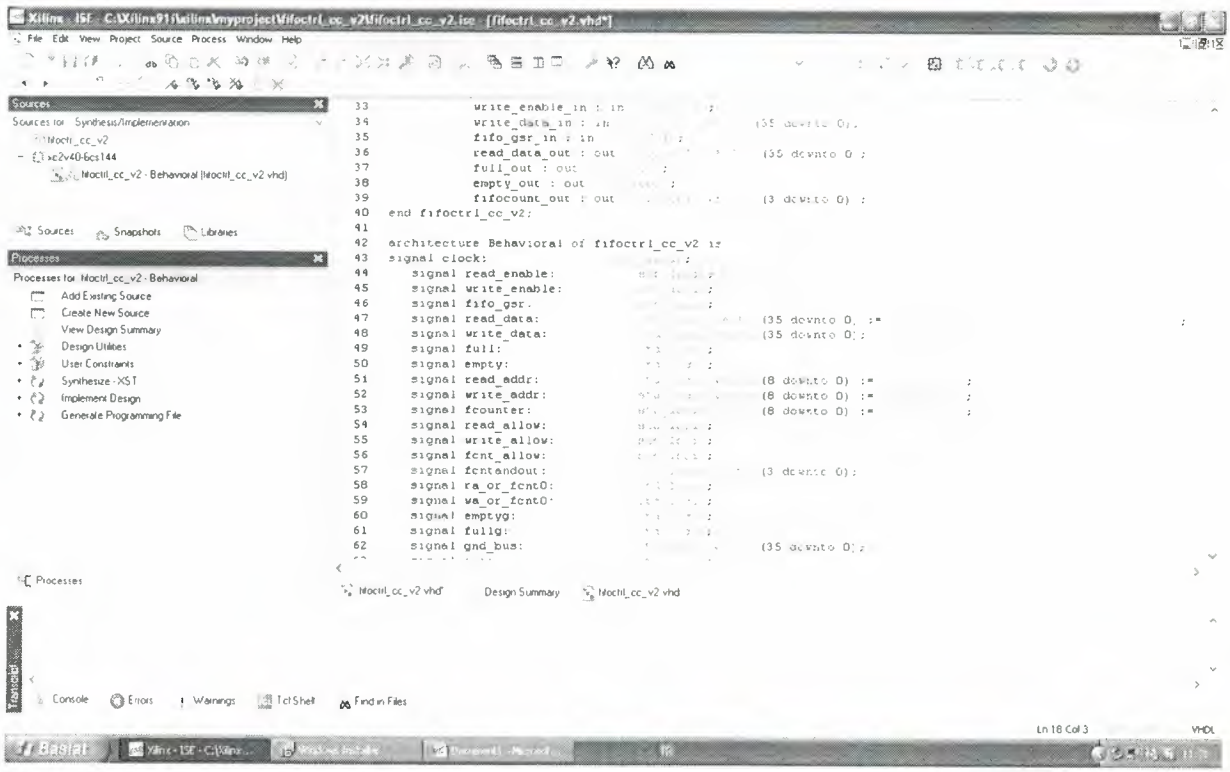

Figure 3.6 Completed VHDL Source

2.3.4. Syntax Checks

Checking the Syntax of the FIFO

When the source files are complete, check the syntax of the design to find errors and typos.

1. Verify that **Synthesis/Implementation** is selected from the drop-down list in the Sources window.

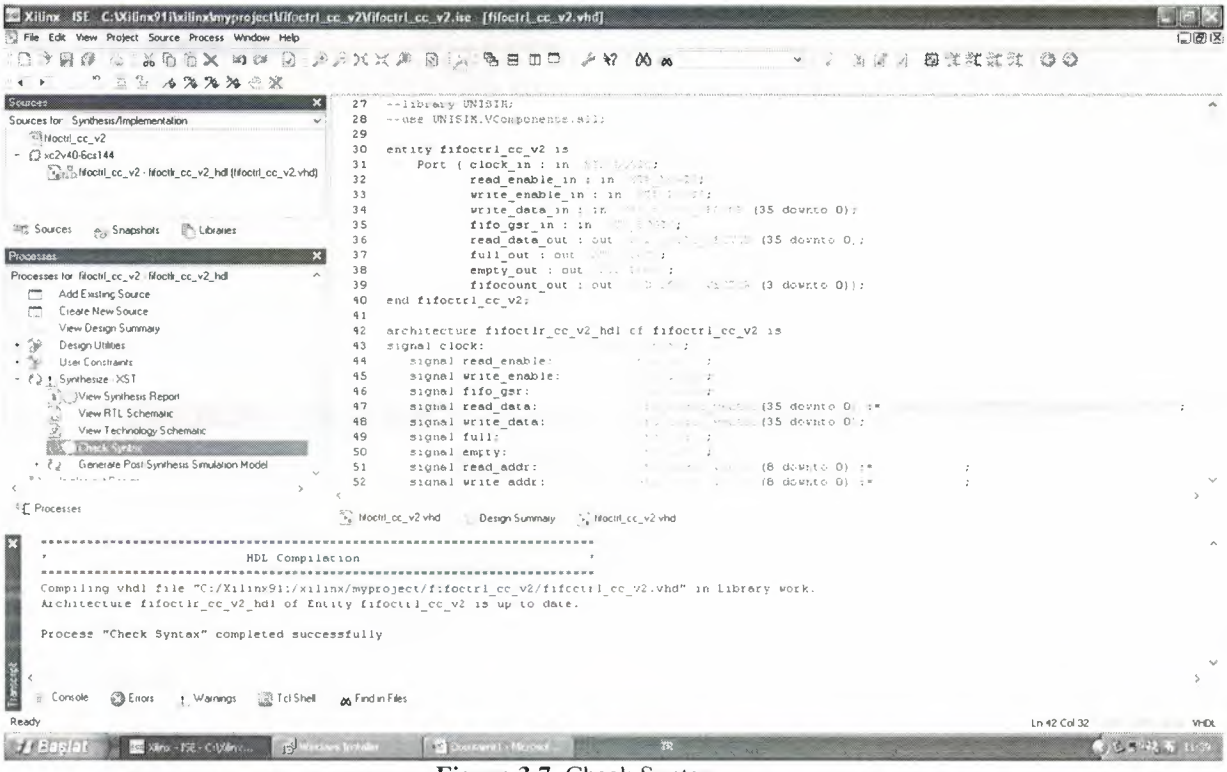

Figure 3.7 Check Syntax

2. Select the **fifoctrl_cc_v2** design source in the Sources window to display the related processes in the Processes window.

3. Click the "**+**" next to the Synthesize-XST process to expand the process group.

4. Double-click the **Check Syntax** process.

5. Syntax check completed successfully.

Note: I must correct any errors found in my source files. I can check for errors in the Console tab of the Transcript window. If I continue without valid syntax, I will not be able to simulate or synthesize my design.

2.3.5. Synthesize

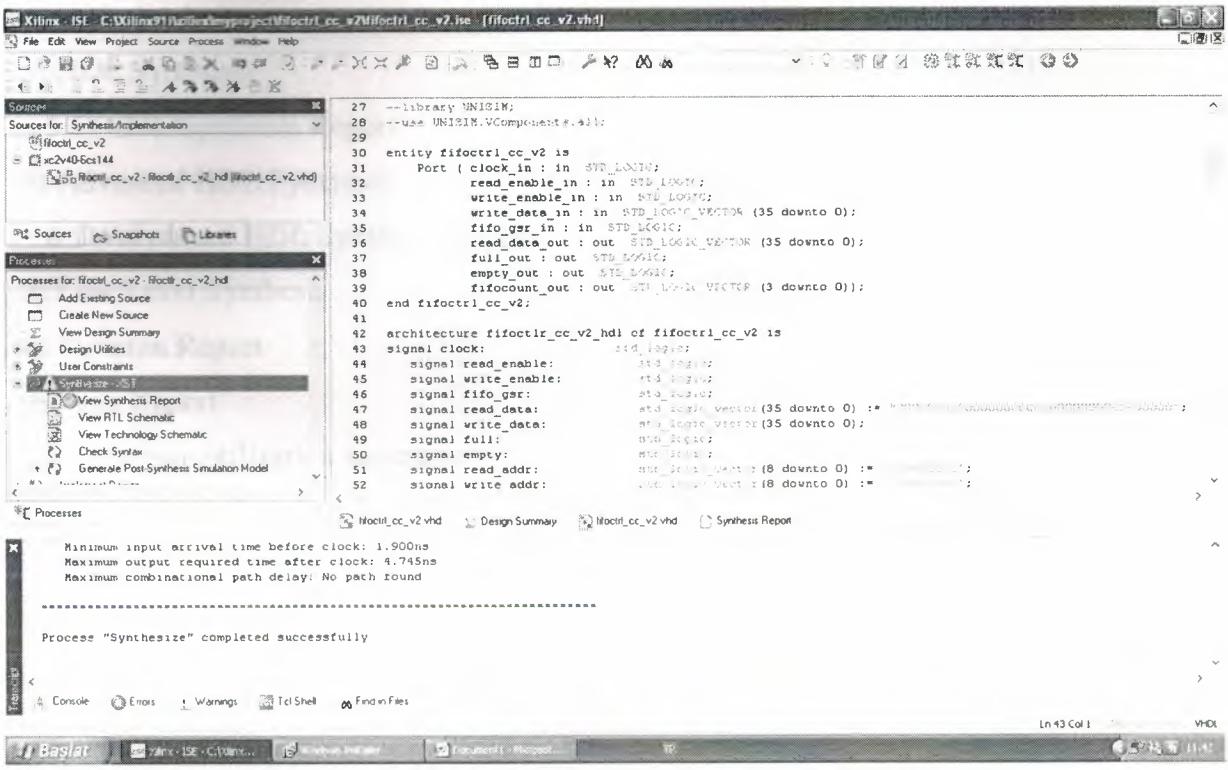

Figure 3.8 Synthesize

Synthesize completed successfully.

I use the xilinx 'synthesize tool' to synthesize the code. The synthesize result is shown below.

Release 9.1i - xst J.30

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

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

CPU : 0.00 / 1.11 s | Elapsed : 0.00 / 1.00 s

--> Parameter xsthdpdir set to ./xst

CPU : 0.00 / 1.16 s | Elapsed : 0.00 / 1.00 s

--> Reading design: fifoctlr_cc_v2.prj

TABLE OF CONTENTS

1) Synthesis Options Summary

2) HDL Compilation

3) Design Hierarchy Analysis

4) HDL Analysis

5) HDL Synthesis

5.1) HDL Synthesis Report

6) Advanced HDL Synthesis

6.1) Advanced HDL Synthesis Report

7) Low Level Synthesis

8) Partition Report

9) Final Report

9.1) Device utilization summary

9.2) Partition Resource Summary

9.3) TIMING REPORT

```
=========================================================================
============
*                  Synthesis Options Summary                  *
=========================================================================
============
---- Source Parameters
Input File Name                    : "fifoctlr_cc_v2.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "fifoctlr_cc_v2"
Output Format                      : NGC
Target Device                      : xc2v40-6-fg256

---- Source Options
Top Module Name                    : fifoctlr_cc_v2
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
```

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

---- Target Options
Add IO Buffers : YES
Global Maximum Fanout : 500
Add Generic Clock Buffer(BUFG) : 16
Register Duplication : YES
Slice Packing : YES
Optimize Instantiated Primitives : NO
Convert Tristates To Logic : Yes
Use Clock Enable : Yes
Use Synchronous Set : Yes
Use Synchronous Reset : Yes
Pack IO Registers into IOBs : auto
Equivalent register Removal : YES

---- General Options
Optimization Goal : Speed

Optimization Effort : 1

Library Search Order : fifoctlr_cc_v2.lso

Keep Hierarchy : NO

RTL Output : Yes

Global Optimization : AllClockNets

Read Cores : YES

Write Timing Constraints : NO

Cross Clock Analysis : NO

Hierarchy Separator : /

Bus Delimiter : <>

Case Specifier : maintain

Slice Utilization Ratio : 100

BRAM Utilization Ratio : 100

Verilog 2001 : YES

Auto BRAM Packing : NO

Slice Utilization Ratio Delta : 5

==========

==========

==========

==========

* HDL Compilation *

==========

==========

Compiling vhdl file "C:/Documents and Settings/esra/Desktop/ESRA/fifoctrl_cc_v2.vhd" in Library work.

Architecture fifoctlr_cc_v2_hdl of Entity fifoctlr_cc_v2 is up to date.

==========

==========

* Design Hierarchy Analysis *

```
=========================================================================
===========
Analyzing hierarchy for entity <fifoctlr_cc_v2> in library <work> (architecture
<fifoctlr_cc_v2_hdl>).


=========================================================================
===========
*                    HDL Analysis                    *
=========================================================================
===========
Analyzing Entity <fifoctlr_cc_v2> in library <work> (Architecture
<fifoctlr_cc_v2_hdl>).
WARNING:Xst:2211 - "C:/Documents and
Settings/esra/Desktop/ESRA/fifoctrl_cc_v2.vhd" line 111: Instantiating black box
module <BUFGP>.
WARNING:Xst:753 - "C:/Documents and
Settings/esra/Desktop/ESRA/fifoctrl_cc_v2.vhd" line 120: Unconnected output port
'DOB' of component 'RAMB16_S36_S36'.
WARNING:Xst:753 - "C:/Documents and
Settings/esra/Desktop/ESRA/fifoctrl_cc_v2.vhd" line 120: Unconnected output port
'DOPB' of component 'RAMB16_S36_S36'.
WARNING:Xst:2211 - "C:/Documents and
Settings/esra/Desktop/ESRA/fifoctrl_cc_v2.vhd" line 120: Instantiating black box
module <RAMB16_S36_S36>.
Entity <fifoctlr_cc_v2> analyzed. Unit <fifoctlr_cc_v2> generated.


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

Performing bidirectional port resolution...

Synthesizing Unit <fifoctlr_cc_v2>.

Related source file is "C:/Documents and Settings/esra/Desktop/ESRA/fifoctrl_cc_v2.vhd".

Found 1-bit register for signal <empty>.

Found 1-bit xor2 for signal <fcnt_allow>.

Found 9-bit updown counter for signal <fcounter>.

Found 1-bit register for signal <full>.

Found 9-bit up counter for signal <read_addr>.

Found 1-bit register for signal <read_allow>.

Found 9-bit up counter for signal <write_addr>.

Found 1-bit register for signal <write_allow>.

Summary:

inferred 3 Counter(s).

inferred 4 D-type flip-flop(s).

Unit <fifoctlr_cc_v2> synthesized.

==========

==========

HDL Synthesis Report

Macro Statistics

Counters : 3

9-bit up counter : 2

9-bit updown counter : 1

Registers : 4

1-bit register : 4

Xors : 1

1-bit xor2 : 1

==========

==========

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

Loading device for application Rf_Device from file '2v40.nph' in environment
C:\Xilinx91i.

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

Macro Statistics
# Counters                          : 3
 9-bit up counter                   : 2
 9-bit updown counter               : 1
# Registers                         : 4
 Flip-Flops                         : 4
# Xors                              : 1
 1-bit xor2                         : 1

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

=========================================================================
=========
*                  Low Level Synthesis                     *
=========================================================================
=========

Optimizing unit <fifoctlr_cc_v2> ...
```

Mapping all equations...

Building and optimizing final netlist ...

Found area constraint ratio of 100 (+ 5) on block fifoctlr_cc_v2, actual ratio is 8.

Final Macro Processing ...

==========

==========

Final Register Report

Macro Statistics

Registers : 31

Flip-Flops : 31

==========

==========

==========

==========

* Partition Report *

==========

==========

Partition Implementation Status

No Partitions were found in this design.

==========

==========

* Final Report *

```
=========================================================================
==========
Final Results
RTL Top Level Output File Name     : fifoctlr_cc_v2.ngr
Top Level Output File Name         : fifoctlr_cc_v2
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 82

Cell Usage :
# BELS                             : 89
#      GND                         : 1
#      INV                         : 2
#      LUT1                        : 16
#      LUT2                        : 10
#      LUT2_L                      : 2
#      LUT4                        : 6
#      LUT4_D                      : 1
#      LUT4_L                      : 1
#      MUXCY                       : 24
#      VCC                         : 1
#      XORCY                       : 25
# FlipFlops/Latches                : 31
#      FDC                         : 2
#      FDCE                        : 27
#      FDP                         : 2
# RAMS                             : 1
#      RAMB16_S36_S36              : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 81
```

```
#      IBUF                : 39
#      OBUF                : 42
=========================================================================
===========
```

Device utilization summary:

Selected Device : 2v40fg256-6

Number of Slices:	21	out of 256	8%
Number of Slice Flip Flops:	31	out of 512	6%
Number of 4 input LUTs:	38	out of 512	7%
Number of IOs:	82		
Number of bonded IOBs:	82	out of 88	93%
Number of BRAMs:	1	out of 4	25%
Number of GCLKs:	1	out of 16	6%

Partition Resource Summary:

No Partitions were found in this design.

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

TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT

GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

Clock Signal	Clock buffer(FF name)	Load
clock_in	BUFGP	32

Asynchronous Control Signals Information:

Control Signal	Buffer(FF name)	Load
fifo_gsr_in	IBUF	31

Timing Summary:

Speed Grade: -6

Minimum period: 3.389ns (Maximum Frequency: 295.072MHz)

Minimum input arrival time before clock: 1.900ns

Maximum output required time after clock: 4.745ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

Timing constraint: Default period analysis for Clock 'clock_in'

Clock period: 3.389ns (frequency: 295.072MHz)

Total number of paths / destination ports: 321 / 78

Delay: 3.389ns (Levels of Logic = 10)

Source: read_allow (FF)

Destination: fcounter_8 (FF)

Source Clock: clock_in rising

Destination Clock: clock_in rising

Data Path: read_allow to fcounter_8

Cell:in->out	Gate fanout	Net Delay	Delay	Logical Name (Net Name)
FDC:C->Q	23	0.449	0.947	read_allow (read_allow)
LUT2:I1->O	1	0.347	0.000	Mcount_fcounter_lut<0> (N6)
MUXCY:S->O	1	0.235	0.000	Mcount_fcounter_cy<0> (Mcount_fcounter_cy<0>)
MUXCY:CI->O	1	0.042	0.000	Mcount_fcounter_cy<1> (Mcount_fcounter_cy<1>)
MUXCY:CI->O	1	0.042	0.000	Mcount_fcounter_cy<2> (Mcount_fcounter_cy<2>)
MUXCY:CI->O	1	0.042	0.000	Mcount_fcounter_cy<3> (Mcount_fcounter_cy<3>)
MUXCY:CI->O	1	0.042	0.000	Mcount_fcounter_cy<4> (Mcount_fcounter_cy<4>)
MUXCY:CI->O	1	0.042	0.000	Mcount_fcounter_cy<5> (Mcount_fcounter_cy<5>)
MUXCY:CI->O	1	0.042	0.000	Mcount_fcounter_cy<6> (Mcount_fcounter_cy<6>)
MUXCY:CI->O	0	0.042	0.000	Mcount_fcounter_cy<7> (Mcount_fcounter_cy<7>)
XORCY:CI->O	1	0.824	0.000	Mcount_fcounter_xor<8> (Result<8>)
FDCE:D		0.293		fcounter_8

```
Total                    3.389ns (2.442ns logic, 0.947ns route)
                                 (72.1% logic, 27.9% route)

==========================================================================
==========
Timing constraint: Default OFFSET IN BEFORE for Clock 'clock_in'
  Total number of paths / destination ports: 38 / 38
--------------------------------------------------------------------------
  Offset:              1.900ns (Levels of Logic = 2)
  Source:              write_enable_in (PAD)
  Destination:         write_allow (FF)
  Destination Clock: clock_in rising

  Data Path: write_enable_in to write_allow
                               Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    IBUF:I->O             1    0.653   0.607  write_enable_in_IBUF (write_enable_in_IBUF)
    LUT4:I0->O            1    0.347   0.000  write_allow_and00001 (write_allow_and0000)
    FDC:D                      0.293          write_allow
    ----------------------------------------
    Total                    1.900ns (1.293ns logic, 0.607ns route)
                                     (68.0% logic, 32.0% route)

==========================================================================
==========
Timing constraint: Default OFFSET OUT AFTER for Clock 'clock_in'
  Total number of paths / destination ports: 42 / 42
--------------------------------------------------------------------------
Offset:              4.745ns (Levels of Logic = 1)
  Source:            fcounter_8 (FF)
  Destination:       fifocount_out<3> (PAD)
  Source Clock:      clock_in rising
```

```
Data Path: fcounter_8 to fifocount_out<3>
                          Gate     Net
  Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   FDCE:C->Q             4   0.449   0.552  fcounter_8 (fcounter_8)
   OBUF:I->O                 3.743        fifocount_out_3_OBUF (fifocount_out<3>)
  ----------------------------------------
  Total                    4.745ns (4.192ns logic, 0.552ns route)
                            (88.4% logic, 11.6% route)

==========================================================================
===========
CPU : 9.99 / 11.47 s | Elapsed : 10.00 / 11.00 s

-->

Total memory usage is 136208 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    4 (   0 filtered)
Number of infos    :    0 (   0 filtered)
```

Xilinx sythesize tool created the following design. Top level block diagram.

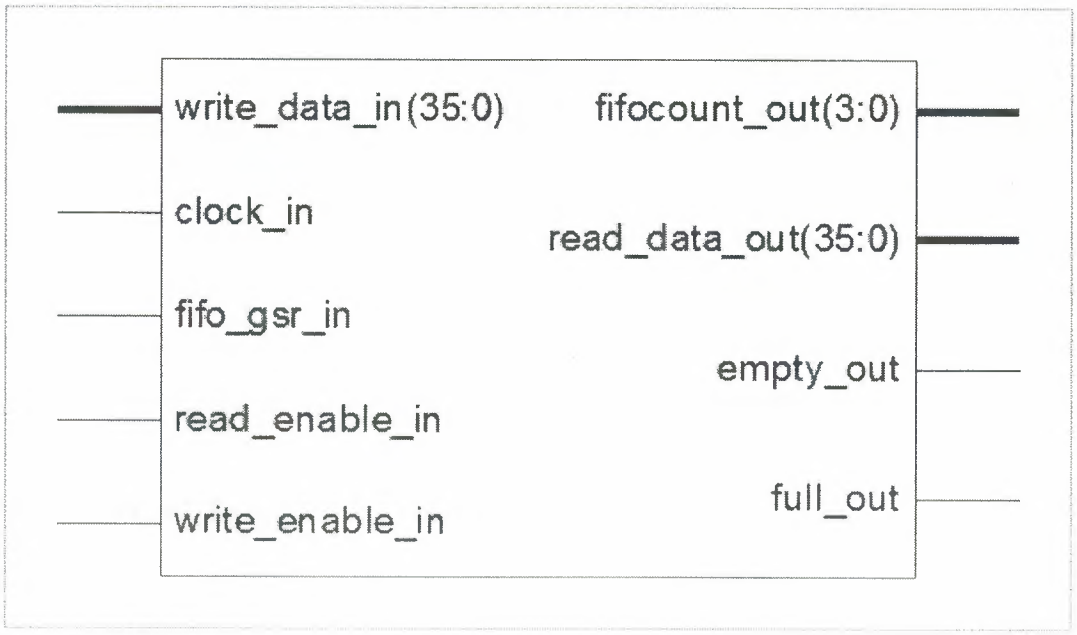

Table 3.9 Top Level Block Diagram

Detailed block diagram.

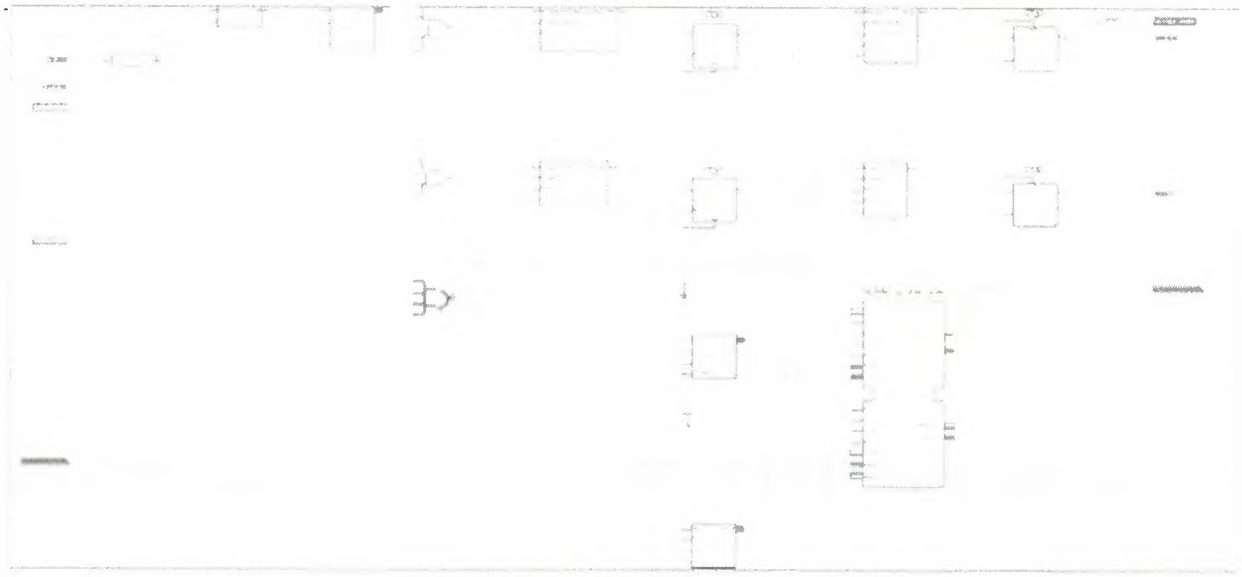

Table 3.10 Detailed Block Diagram

2.3.6. Design Simulation

2.3.6.1. Verifying Functionality Using Behavioral Simulation

Create a test bench waveform containing input stimulus you can use to verify the functionality of the fifo. The test bench waveform is a graphical view of a test bench.

Create the test bench waveform as follows:

1. Select the **fifoctrl_cc_v2** HDL file in the Sources window.

2. Create a new test bench source by selecting **Project → New Source**.

3. In the New Source Wizard, select **Test Bench WaveForm** as the source type, and type

Fifoctrl_cc_v2_tb in the File Name field.

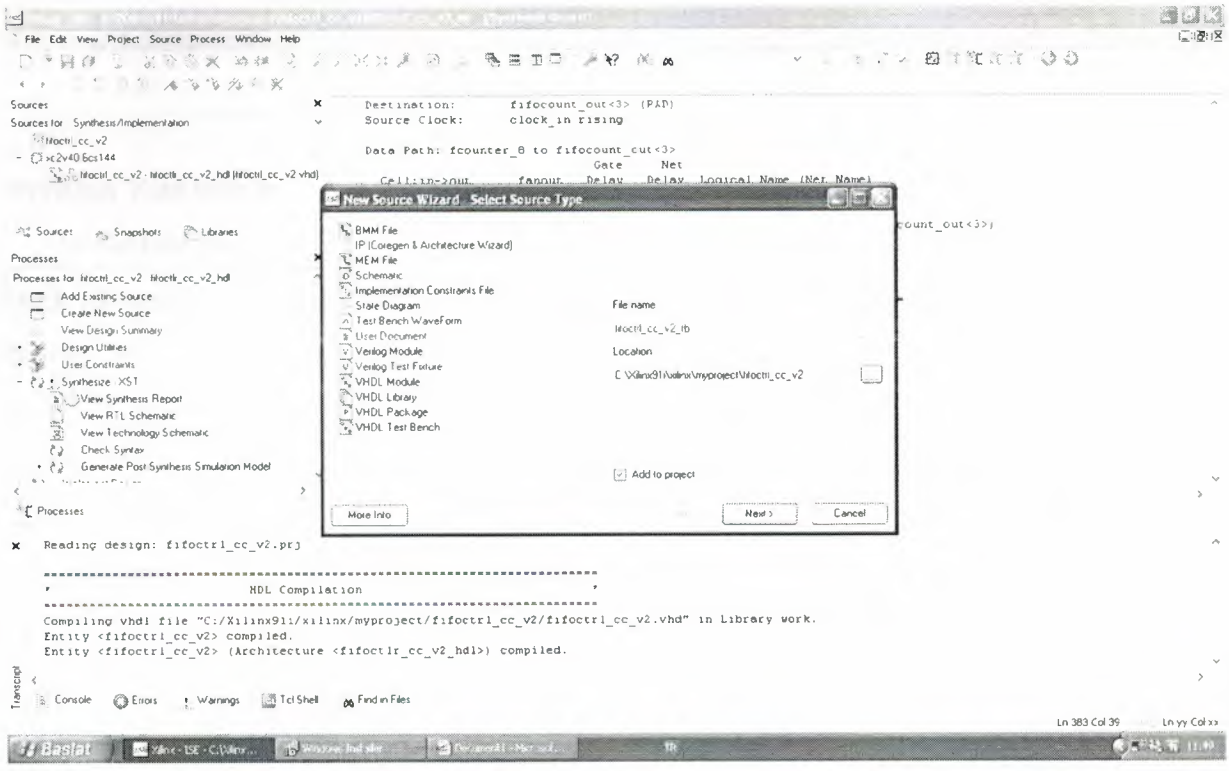

Figure 3.11 Create Test Bench

4. Click **Next**.

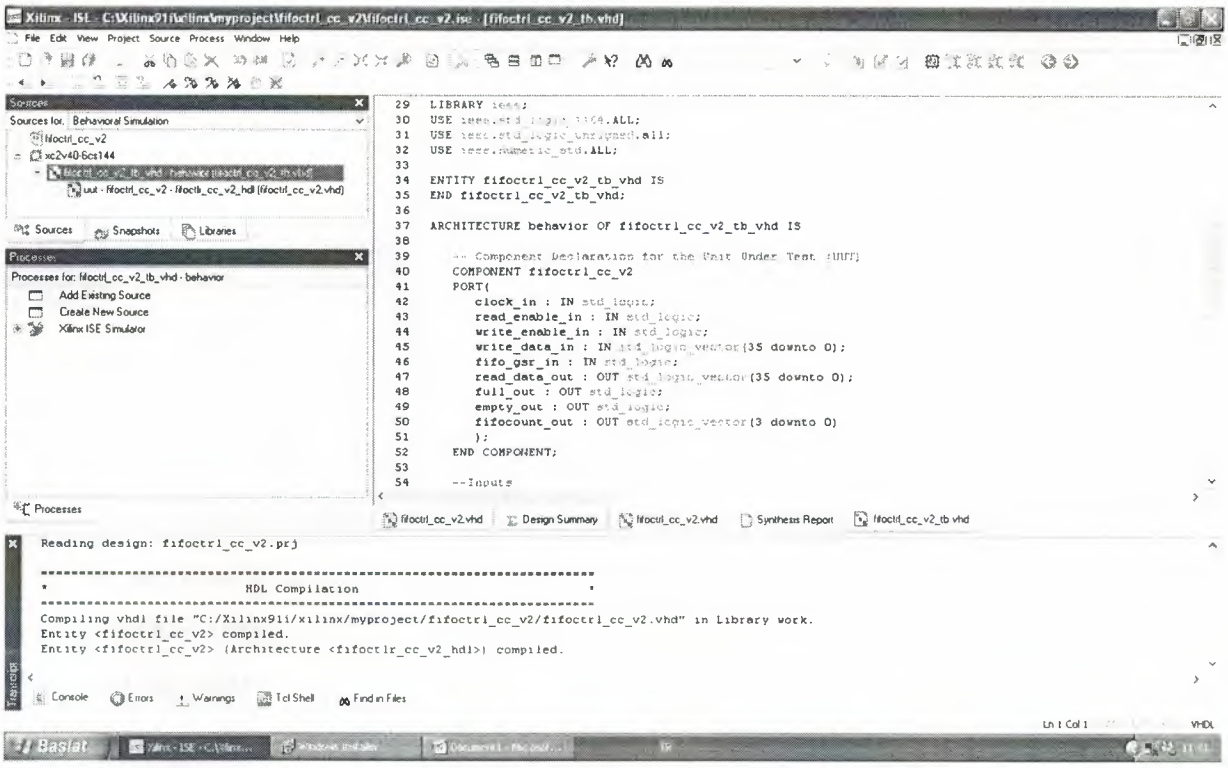

Figure 3.12 Created Test bench

Writing Test Bench

In the test bench generated 160 MHz. I wrote in the data in to the FIFO and read it back to verify data can be written and read correctly. The simulation result is shown below.

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   14:48:16 03/27/2008
-- Design Name:   fifoctlr_cc_v2
-- Module Name:   C:/Xilinx91i/xilinx/ESRA/fifoctrl_cc_v2/fifoctrl_cc_v2_tb.vhd
-- Project Name:  fifoctrl_cc_v2
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: fifoctlr_cc_v2
```

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

ENTITY fifoctrl_cc_v2_tb_vhd IS
END fifoctrl_cc_v2_tb_vhd;

ARCHITECTURE behavior OF fifoctrl_cc_v2_tb_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT fifoctlr_cc_v2
    PORT(
        clock_in : IN std_logic;
        read_enable_in : IN std_logic;
        write_enable_in : IN std_logic;
        write_data_in : IN std_logic_vector(35 downto 0);
        fifo_gsr_in : IN std_logic;
        read_data_out : OUT std_logic_vector(35 downto 0);
        full_out : OUT std_logic;
```

```
            empty_out : OUT std_logic;
            fifocount_out : OUT std_logic_vector(3 downto 0)
            );
        END COMPONENT;

        --Inputs
        SIGNAL clock_in :  std_logic := '0';
        SIGNAL read_enable_in :  std_logic := '0';
        SIGNAL write_enable_in :  std_logic := '0';
        SIGNAL fifo_gsr_in :  std_logic := '0';
        SIGNAL write_data_in :  std_logic_vector(35 downto 0) := (others=>'0');

        --Outputs
        SIGNAL read_data_out :  std_logic_vector(35 downto 0);
        SIGNAL full_out :  std_logic;
        SIGNAL empty_out :  std_logic;
        SIGNAL fifocount_out :  std_logic_vector(3 downto 0);

BEGIN

        -- Instantiate the Unit Under Test (UUT)
        uut: fifoctlr_cc_v2 PORT MAP(
            clock_in => clock_in,
            read_enable_in => read_enable_in,
            write_enable_in => write_enable_in,
            write_data_in => write_data_in,
            fifo_gsr_in => fifo_gsr_in,
            read_data_out => read_data_out,
            full_out => full_out,
            empty_out => empty_out,
            fifocount_out => fifocount_out
        );

clock_in <= not clock_in after 6 ns;
```

```
tb : PROCESS
BEGIN

        -- Wait 100 ns for global reset to finish
        wait for 100 ns;
        write_enable_in <= transport '1';
        write_data_in <= transport
std_logic_vector'("00000000000000000000000000000000"); --0
        fifo_gsr_in <= transport '0';

    WAIT FOR 12 ns; -- Time=240 ns
        write_data_in <= transport
std_logic_vector'("00000000000000000000000000000001"); --1

        -- ---------------------
        WAIT FOR 12 ns; -- Time=280 ns
        write_data_in <= transport
std_logic_vector'("00000000000000000000000000000010"); --2
        WAIT FOR 12 ns; -- Time=320 ns
        write_data_in <= transport
std_logic_vector'("00000000000000000000000000000011"); --3
        -- ---------------------
        WAIT FOR 12 ns; -- Time=360 ns
        write_data_in <= transport
std_logic_vector'("00000000000000000000000000000100"); --4
        -- ---------------------
        WAIT FOR 12 ns;
                -- ---------------------
        WAIT FOR 12 ns; -- Time=960 ns
        read_enable_in <= transport '1';
        write_enable_in <= transport '0';
        write_data_in <= transport
std_logic_vector'("00000000000000000000000000010011"); --13
```

```
        wait; -- will wait forever
    END PROCESS;

END;
```

5. The Associated Source page shows that you are associating the test bench waveform with the source file fifoctrl_cc_v2. Click **Next**.

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

Generate the clock and give inputs.

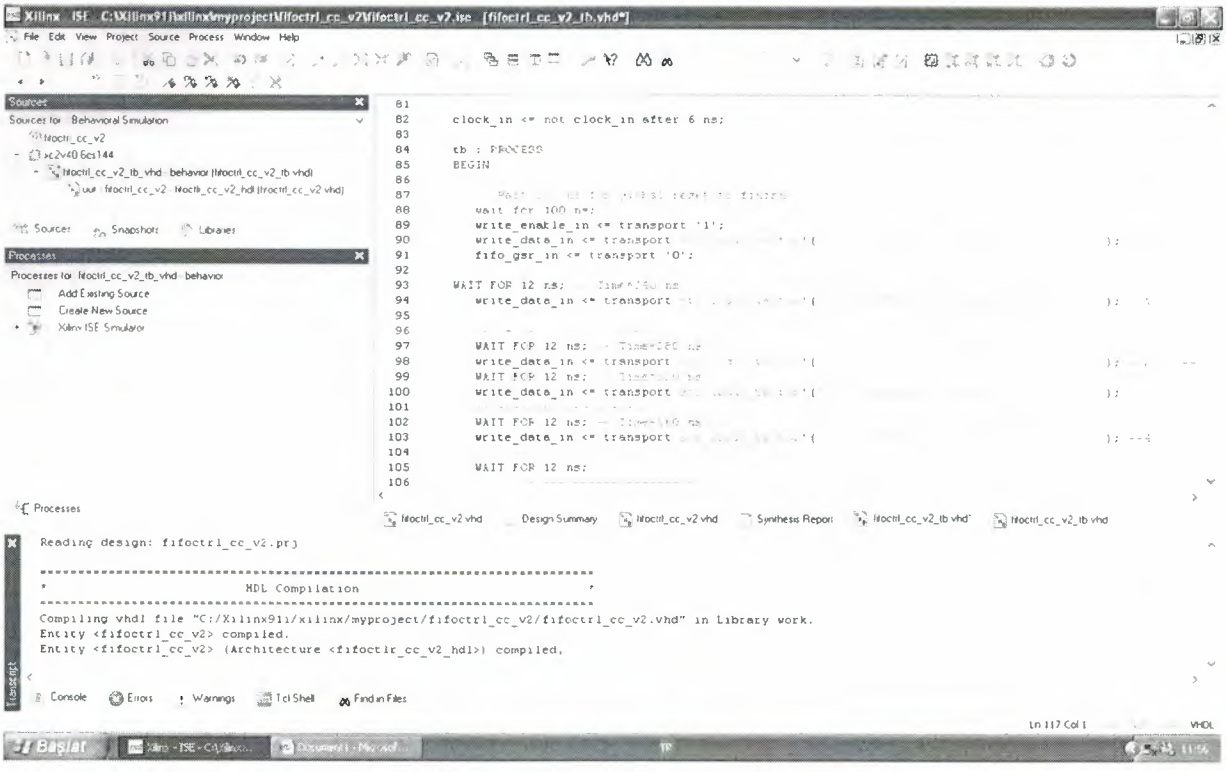

Figure 3.13 Generate The Clock

7. In the Sources window, select the **Behavioral Simulation** view to see that the test bench waveform file is automatically added to your project.

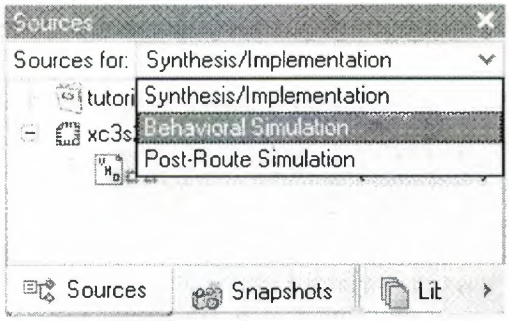

Figure 3.14 Behavior Simulation Selection

8. Close the test bench waveform.

2.3.7. Simulating Design Functionality

Verify that the fifo design functions as you expect by performing behavior simulation as follows:

1. Verify that Behavioral Simulation and fifoctrl_cc_v2_tb are selected in the Sources window.

2. In the Processes tab, click the "+" to expand the Xilinx ISE Simulator process and double-click the Simulate Behavioral Model process.

The ISE Simulator opens and runs the simulation to the end of the test bench.

3. To view my simulation results, select the Simulation tab and zoom in on the transitions.

The simulation waveform results will look like the following:

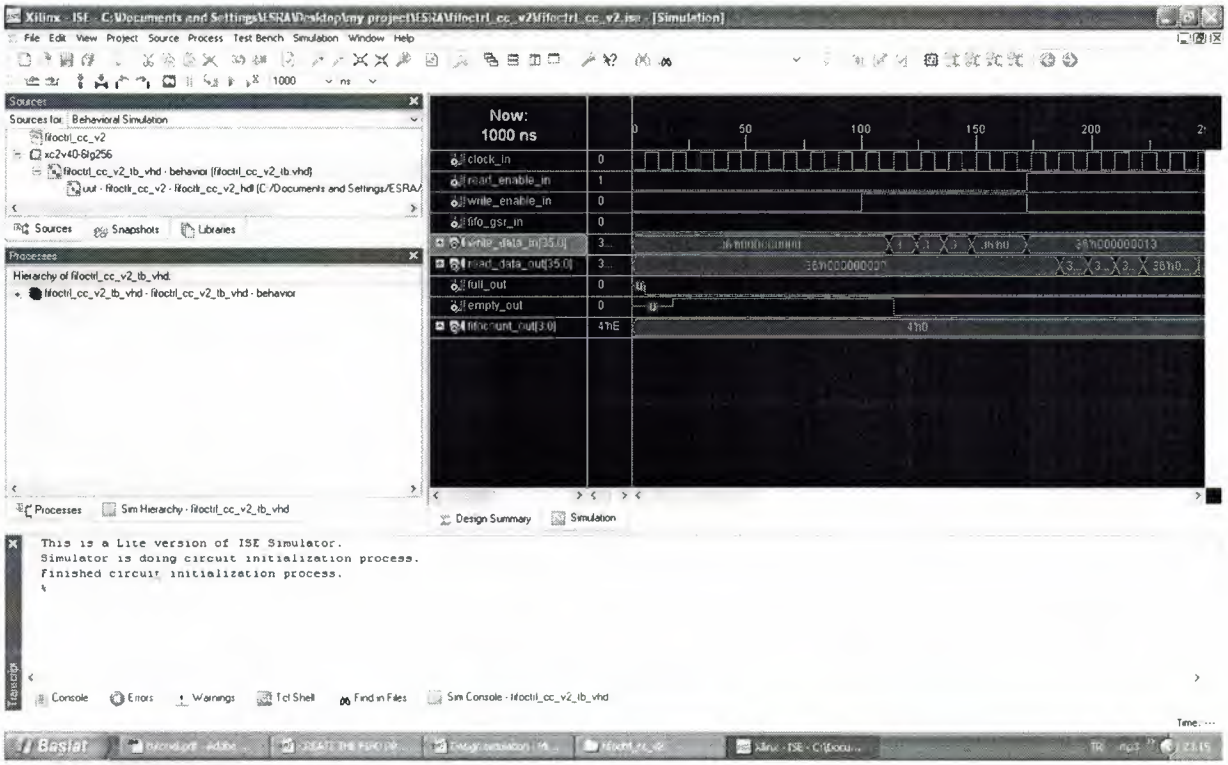

Figure 3.15 Simulation Results

4. Verify that the fifo is counting up and down as expected.

5. Close the simulation view. If you are prompted with the following message, "You have

an active simulation open. Are you sure you want to close it?", click Yes to continue.

I have now completed simulation of my design using the ISE Simulator.

2.3.8. Programming File Generation Report

Using the ISE tool the generate the programming file.

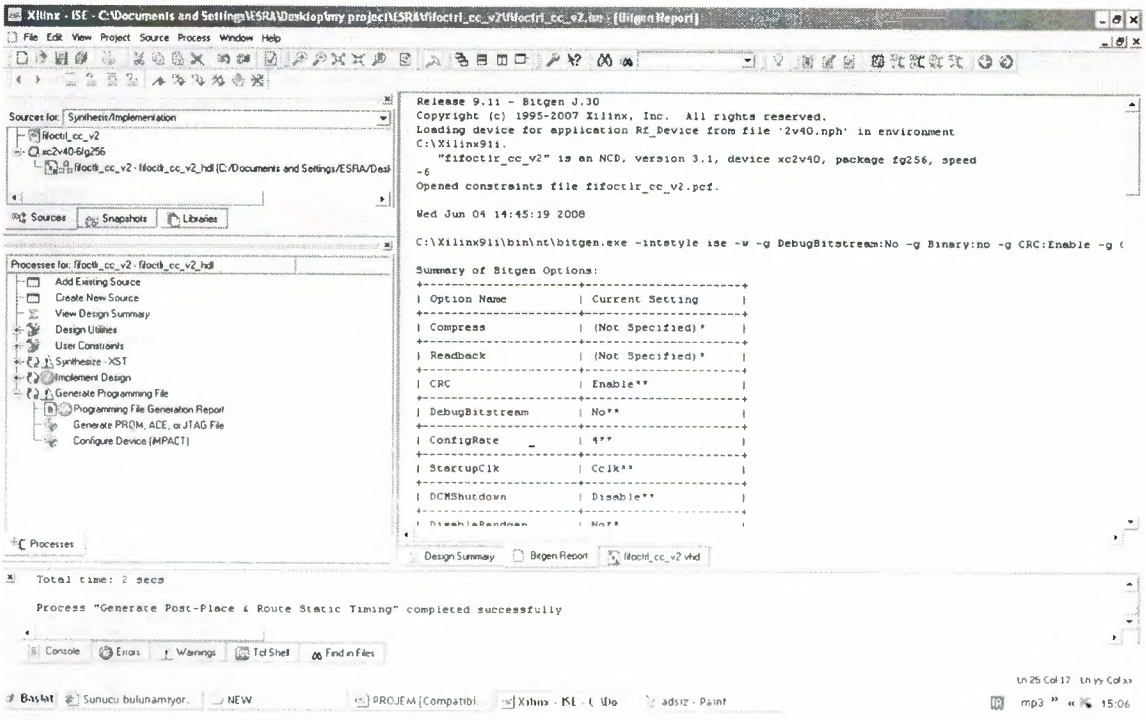

Figure 3.16 Generation Report

2.3.9. Programming the Device

Programming the Virtex-II FPGA.

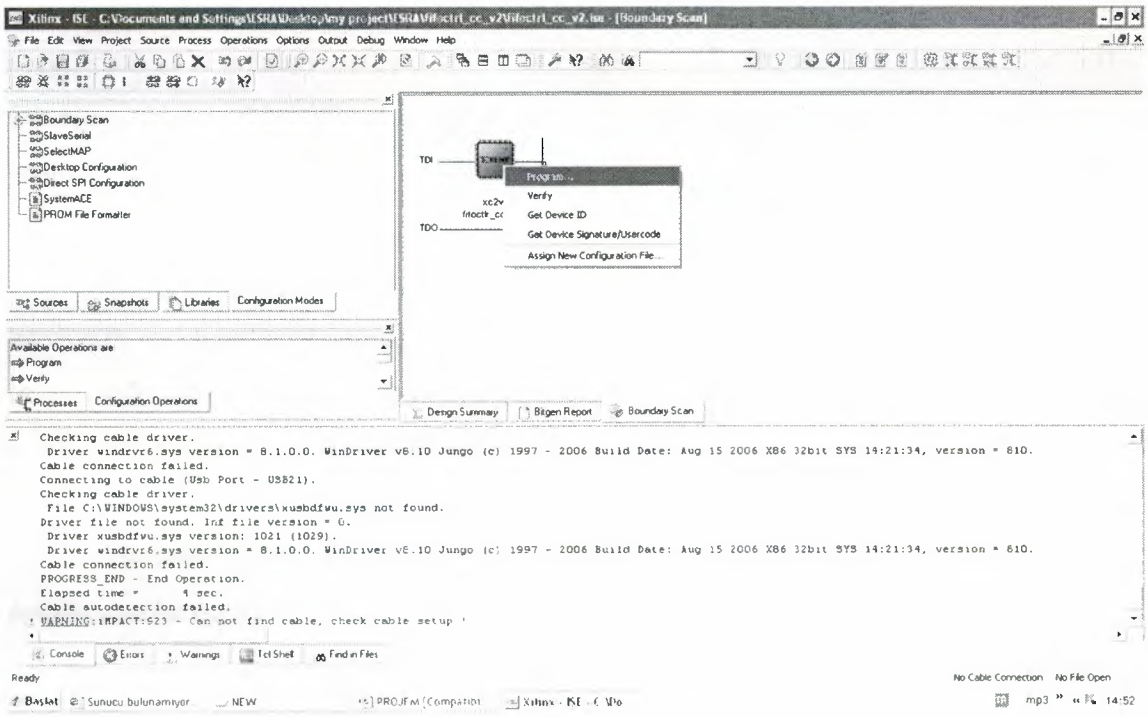

Figure 3.16 Programming Device

To program the device we just click the program.

CONCLUSION

In this project, the 512*36 FIFO is designed using VHDL language and the Xilinx ISE tools to program the Virtex-II FPGA. First, the requirements and specification is written define, the FIFO inputs and outputs are defined, the FIFO function is defined the VHDL code. Then the Xilinx ISE tools are used to complete the project.

The benefit that we gained by using a VHDL Xilinx is just to get rid of the hardware and bundle of cables and resistors, connections after all if only a single cable or a connection is missing or misplace the whole design can be easily destroyed and moreover its expensive and could be dangerous.

To sum up the programming in Xilinx has become so convinent that required results can be easily achieved with precise results.

REFERENCES

[1] M.Kadir Özakman, VHDL, Lecture Notes,"http://www.members.shaw.ca/kadirm"

[2] Xilinx Software Manuals "www.xilinx.com"

[3] Xilinx XAPP-258 , Application Notes

[4] Xilinx ISE Examples