

**NEAR EAST UNIVERSITY**



**Faculty of Engineering**

**Department of Computer Engineering**

**STOPWATCH DESIGN**

**Graduation Project  
COM-400**

**Student : Mehmet Şükrü İncili(20032310)**

**Supervisor : Mehmet Kadir Özakman**

**Nicosia-2008**

## ACKNOWLEDGEMENTS

*“First, I would like to thank my supervisor Mehmet Kadir Özakman for his Invaluable advice and belief in my work and myself over the course of this Graduation Project..*

*Second, I would like to Express my gratitude to Near East University for the scholarship that made the work possible.*

*Third, I thank my family for their constant encouragement and support during the preparation of this project.*

*Finally, I would also like to thank all my friends for their advice and support.”*

## ABSTRACT

I will design stopwatch at this project. Stopwatch is an electronic desgin. We are using it in different areas.

I will use XILINX ISE 9.1i software for to create the stopwatch design. I selected this program because is very useful for to do this electronic design. We can design many things that we are using xilinx ise software.

I will explain briefly why XILINX ISE software is useful and suitable for us.

Assume that you have a company you are working IT (information technology) sector. You are doing many specific solutions. In one day you need a 32 bit proccessor for example. You are calling xilinx company and then you are saying 'we want a 32 bit proccessor.' Then they are sending a spesific FPGA (field programmable logic gate) chip kit (virtex, spartan etc..) as special for your request. Then you are taking the kit and connecting the internet at where you are. They are loading a 32 bit proccessor software your kit using the internet. So you have a 32 bit proccessor.

It's very useful for the companies because if you want you can change your proccessor to ram or rom etc.. You can convert so many things. At present you have a kit you can create many things using this kit and XILINX software.

So I hope this technology will grow up and there will be so many vacany therefore I want to learn this technology and I used this software.

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b>	<b>i</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>TABLE OF CONTENTS</b>	<b>iii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>iv</b>
<b>INTRODUCTION</b>	<b>1</b>
<b>CHAPTER ONE : DESIGN DESCRIPTION</b>	<b>2</b>
1.1.INPUTS	2
1.2.OUTPUTS	3
1.3.FUNCTIONAL BLOCKS	3
<b>CHAPTER TWO : SOFTWARE AND HARDWARE REQUIREMENTS</b>	<b>5</b>
2.1.SOFTWARE REQUIREMENTS	5
2.2.HARDWARE REQUIREMENTS	5
<b>CHAPTER THREE : DESIGN STEPS AND DESIGN ENTRY</b>	<b>6</b>
3.1.CREATING THE STOPWATCH.VHD USING NEW PROJECT WIZARD	7
3.1.1.CODES IN <i>stopwatch.vhd</i>	12
3.1.2.CODES IN <i>statmach.vhd</i>	16
3.1.3.State Machine Diagram	19
3.2.CREATING A DCM MODULE	20
3.2.1.Using DCM Wizard	20
3.3.CREATING THE CORE GENERATOR MODULE	25
3.4.CREATING AN HDL-BASED MODULE	29
3.4.1.Using the New Source Wizard and HDL Editor	30
3.5.CREATING THE CNT60 VHDL MODULE	36
3.5.1.CODES IN <i>cnt60.vhd</i> MODULE	39
3.6.CREATING HEX2LED SOURCE USING NEW SOURCE WIZARD AND HDL EDITOR	41
3.6.1.CODES IN <i>HEX2LED.VHD</i> MODULE	43

<b>CHAPTER FOUR : ESTABLISHING THE TEST BENCH OF DESIGN</b>	<b>45</b>
4.1.CODES IN TEST BENCH	46
<b>CHAPTER FIVE : SYNTHESIS THE DESIGN</b>	<b>49</b>
<b>CHAPTER SIX : VIEW RTL SCHEMATIC</b>	<b>67</b>
<b>CHAPTER SEVEN : SIMULATION OF DESIGN</b>	<b>69</b>
<b>CHAPTER EIGHT : ABOUT DCM CLOCKING WIZARD</b>	<b>75</b>
8.1.DCM PORTS	75
<b>CHAPTER NINE : ABOUT BINARY COUNTER</b>	<b>84</b>
9.1.FEATURES	84
9.2.FUNCTIONAL DESCRIPTION	85
<b>CHAPTER TEN : ABOUT VIRTEX FPGA CHIPS</b>	<b>86</b>
10.1.BUILT FOR BANDWIDTH	86
10.2.LEGACY OF LEADERSHIP	88
10.3.PACKETS EVERYWHERE	89
10.4.TIME IS MONEY	91
<b>CHAPTER ELEVEN : ABOUT XILINX SOFTWARE AND COMPANY</b>	<b>93</b>
11.1.HISTORY OF XILINX	93
11.2.EFFECTIVE PARTNERSHIPS	98
11.3.BUSINESS OF XILINX	99
11.4.SUCCESS OF XILINX	100
11.5.VALUES OF XILINX	103
<b>CONCLUSION</b>	<b>105</b>
<b>REFERENCES</b>	<b>106</b>

## **LIST OF ABBREVIATIONS**

<b>ISE</b>	<b>Integrated Software Environment</b>
<b>FPGA</b>	<b>Field Programmable Gate Array</b>
<b>VHDL</b>	<b>Very high speed integrated circuit Hardware Description Language</b>
<b>DCM</b>	<b>Digital Clock Manager</b>
<b>RPM</b>	<b>Relationally Placed Macro</b>
<b>DUT</b>	<b>Design Under Test</b>
<b>UUT</b>	<b>Unit Under Test</b>
<b>RTL</b>	<b>Register Transfer Level</b>



# INTRODUCTION

My project is stopwatch I will design stopwatch using virtex chip and xilinx 9.1i software. Now I will explain briefly chapter's in my design and we will see step by step stopwatch design.

## ***Chapter one ;***

It's design description I defined inputs, outputs and functional blocks at this chapter.

## ***Chapter two ;***

I explained which software and hardware I used my project at this chapter.

## ***Chapter three ;***

It's design steps I defined design steps of my project at this chapter.

I am talking about the design entry and I am explaining which codes I used my project at this chapter.

## ***Chapter four ;***

I am talking about how I established the vhdl test bench of my stopwatch Design. I wrote the codes which I need to test my design at this chapter.

## ***Chapter five ;***

I am talking about synthesis of my design it is also doing check syntax at the same time and if I did wrong something I will see and I will fix them at this part.

## ***Chapter six ;***

I am creating RTL(register transfer level) schematic at this part.

## ***Chapter seven ;***

I am checking my design is it working correctly or not with simulation.

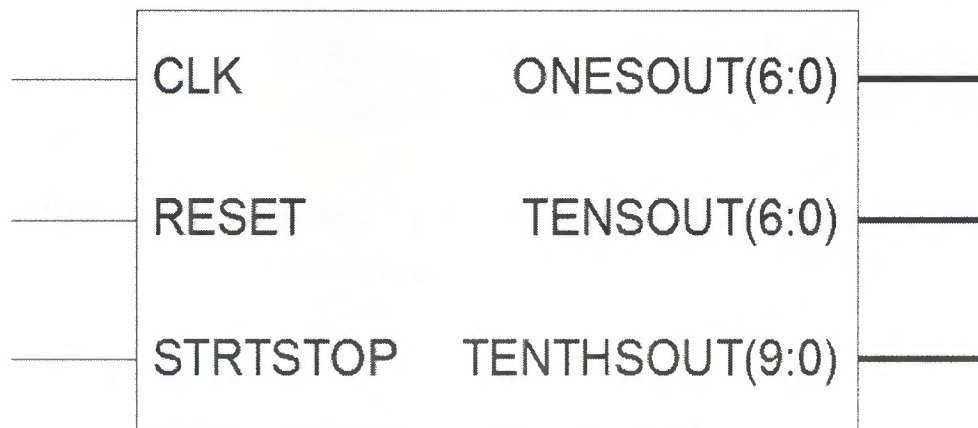
## ***Chapter eight,nine,ten and eleven ;***

This chapters includes information of about software and hardware which I used in my design.

## CHAPTER ONE

### DESIGN DESCRIPTION

My design is **STOPWATCH** has three inputs and three outputs.



*Figure 1-block diagram*

In the runner's stopwatch design, there are three external inputs and three external output buses. The system clock is an externally generated signal. The following list summarizes the input and output signals of the design.

#### 1.1.INPUTS

The following are input signals for the stopwatch design.

**CLK** → System clock for the Watch design.

**STRTSTOP** → Starts and stops the stopwatch. This is an active-low signal which acts like the start/stop button on a runner's stop-watch.

**RESET** → Resets the stopwatch to 00.0 after it has been stopped.



## 1.2.OUTPUTS

The following are outputs signals for the stopwatch design.

**TENSOUT[6:0]** → 7-bit bus which represents the Tens digit of the stopwatch value.

This bus is in 7-segment display format to be viewable on the 7-segment LED display.

**ONESOUT[6:0]** → Similar to TENSOUT bus above, but represents the Ones digit of the stopwatch value.

**TENTHSOUT[9:0]** → 10-bit bus which represents the Tenths digit of the stopwatch value. This bus is one-hot encoded.

## 1.3.FUNCTIONAL BLOCKS

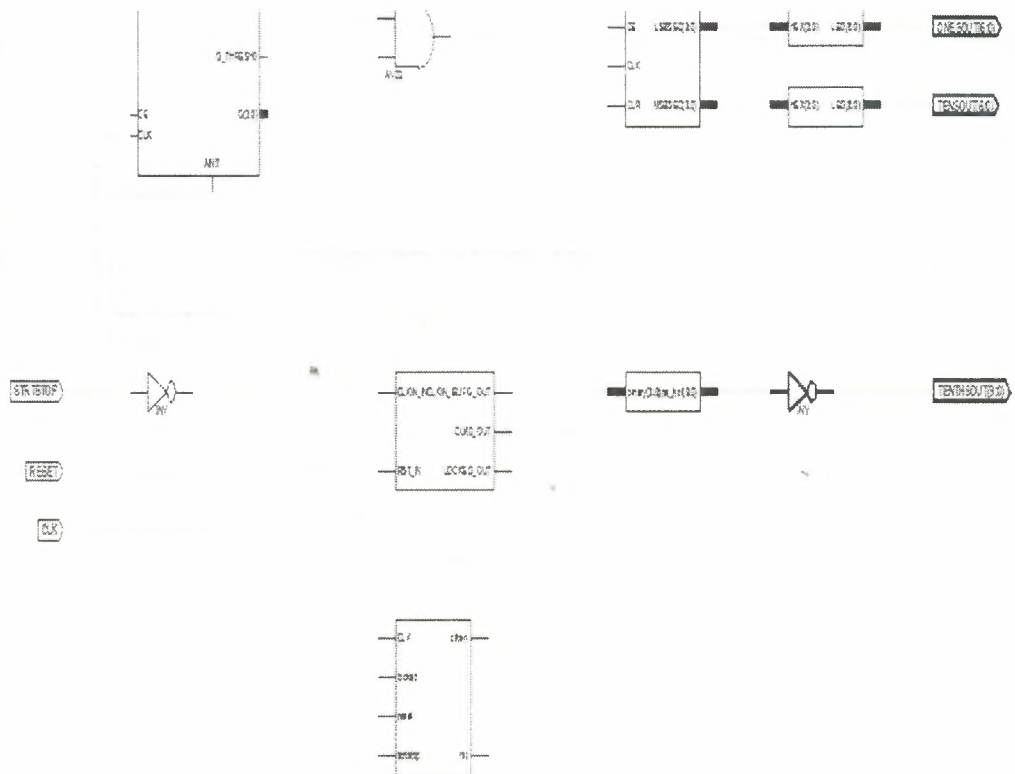


figure 2-functional blocks

The completed design consists of the following functional blocks.

### ***STOPWATCH***

→ Top level HDL file.

### ***STMACH\_V***

→ State Machine macro.

### ***CNT60***

→ VHDL module which counts from 0 to 59, decimal. This macro has two 4-bit outputs,

which represent the 'ones' and 'tens' digits of the decimal values, respectively.

### ***DCM1***

→ A single DCM clocking module created with Xilinx Architecture Wizard. DECODE HDL based macro. This macro converts a binary input to a one-hot output.

### ***TENTHS***

→ A Coregen 10-bit, one-hot encoded counter. This macro outputs the 'tenths' digit of the watch value as a 10-bit one-hot encoded value.

### ***HEX2LED***

→ HDL-based macro. This macro decodes the ones and tens digit values from hexadecimal to 7-segment display format.

## CHAPTER TWO

### SOFTWARE AND HARDWARE REQUIREMENTS

#### 2.1.SOFTWARE REQUIREMENTS

I will use VHDL this assignment.

**VHDL**---Very high speed integrated circuit Hardware Description Language

I will use XILINX ISE 9.1i

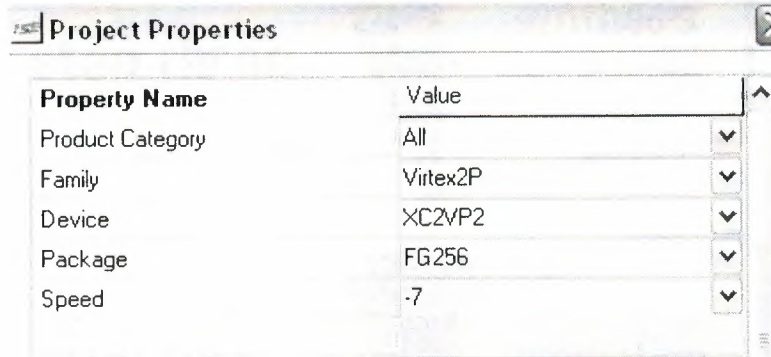
**ISE** --- Integrated Software Environment

#### 2.2.HARDWARE REQUIREMENTS

I will use FGPA chip.

**FGPA**---Field Programmable Gate Array

We are using this Project VIRTEX2P FPGA family.



*figure 3-properties of project*

# CHAPTER THREE


## DESIGN STEPS AND DESIGN ENTRY

**step one**

A black folder icon with a white label area.


**design entry**

**step two**

A black folder icon with a white label area.

**establish  
test bench**

**step three**

A black folder icon with a white label area.

**synthesis  
the design**

**step four**

A black folder icon with a white label area.

**simulation**

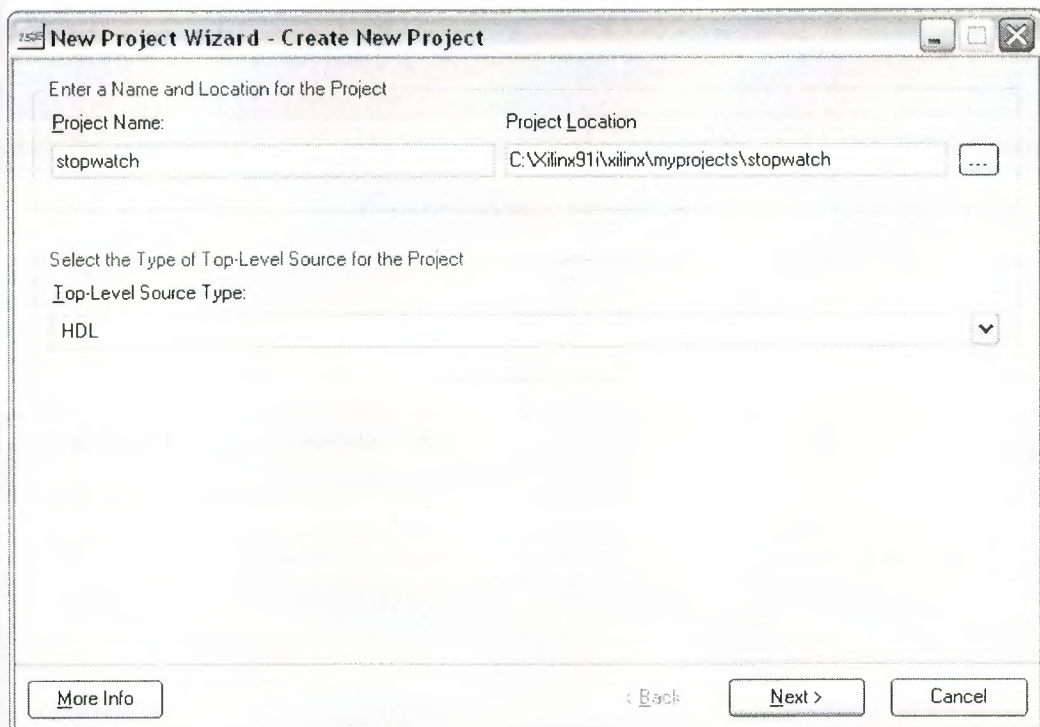
## DESIGN ENTRY

### 3.1.CREATING THE STOPWATCH.VHD USING NEW PROJECT WIZARD

Now we are open the xilinx 9.1 i and we select *file* and *new project* .

And we are writing as Project name '*stopwatch*'.

And click next.



*Figure 4-create new project*



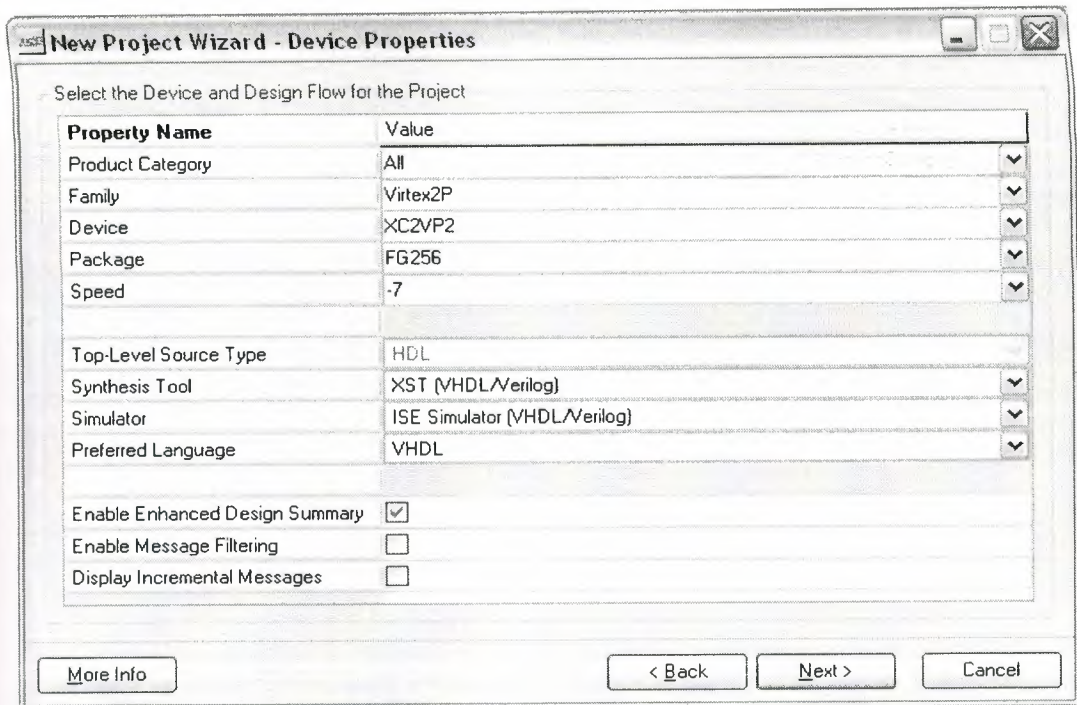


Figure 5-device properties

And we are setting these values as above.

And click next.

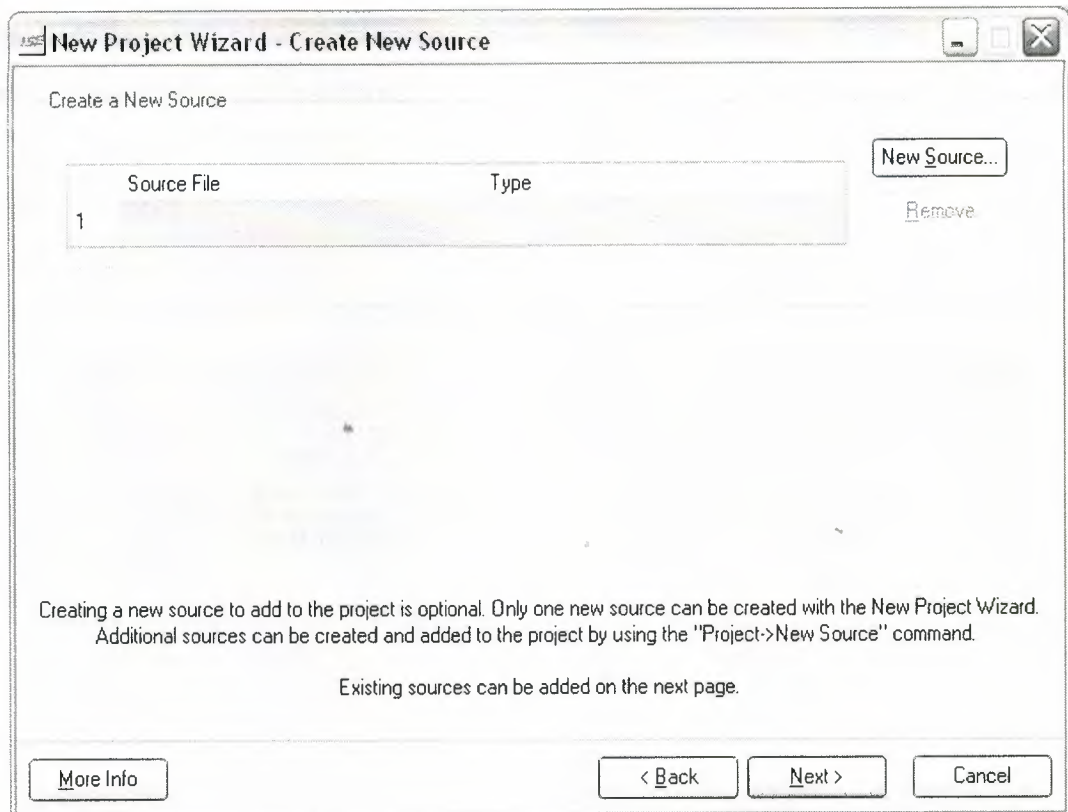


Figure 6 Then click next



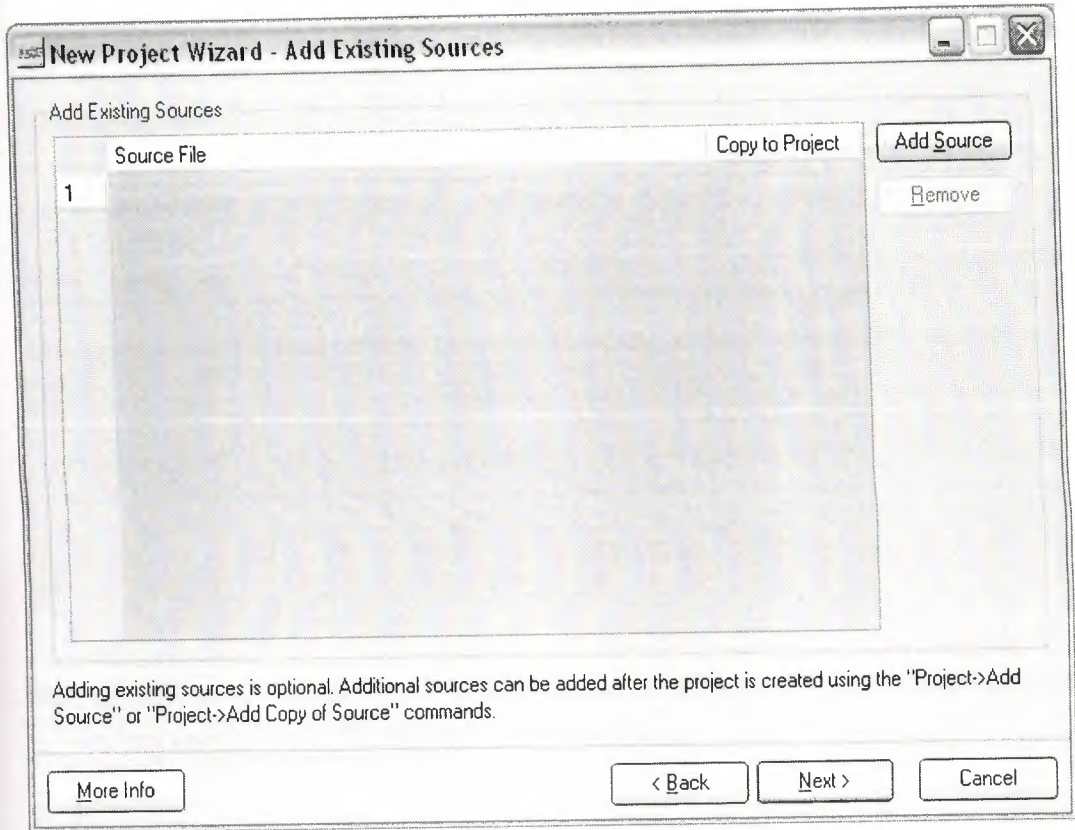


Figure 7-add existing source

Click **add source**

Then select these files this window

- Cnt60.vhd
- Statmach.vhd
- Stopwatch.vhd

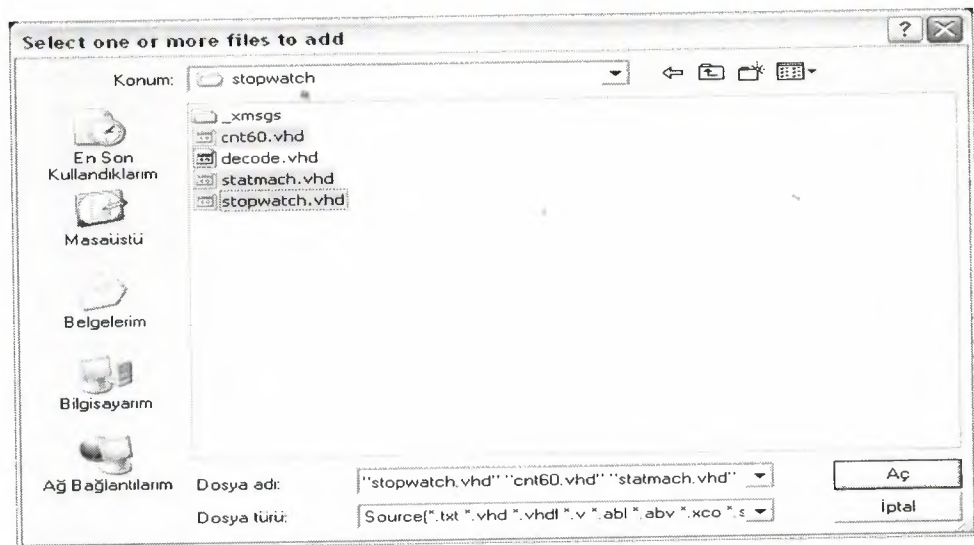


Figure 8 Click open

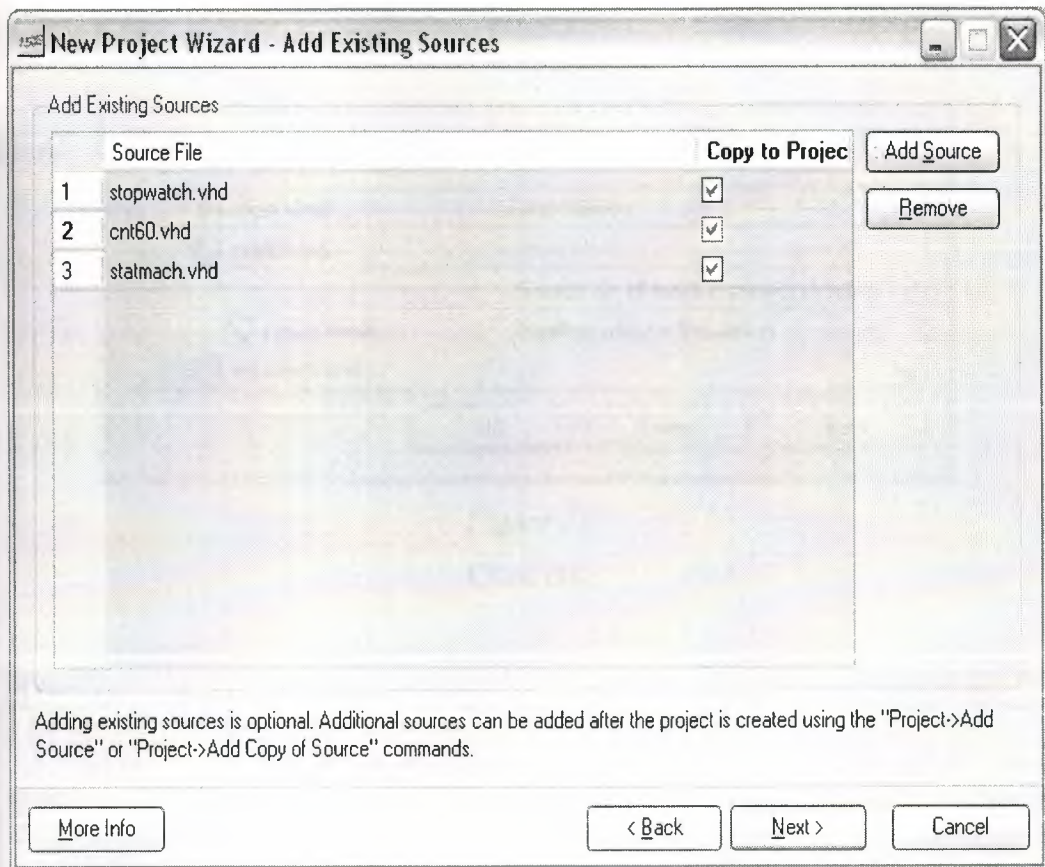


Figure 9

Select the boxes and click next.

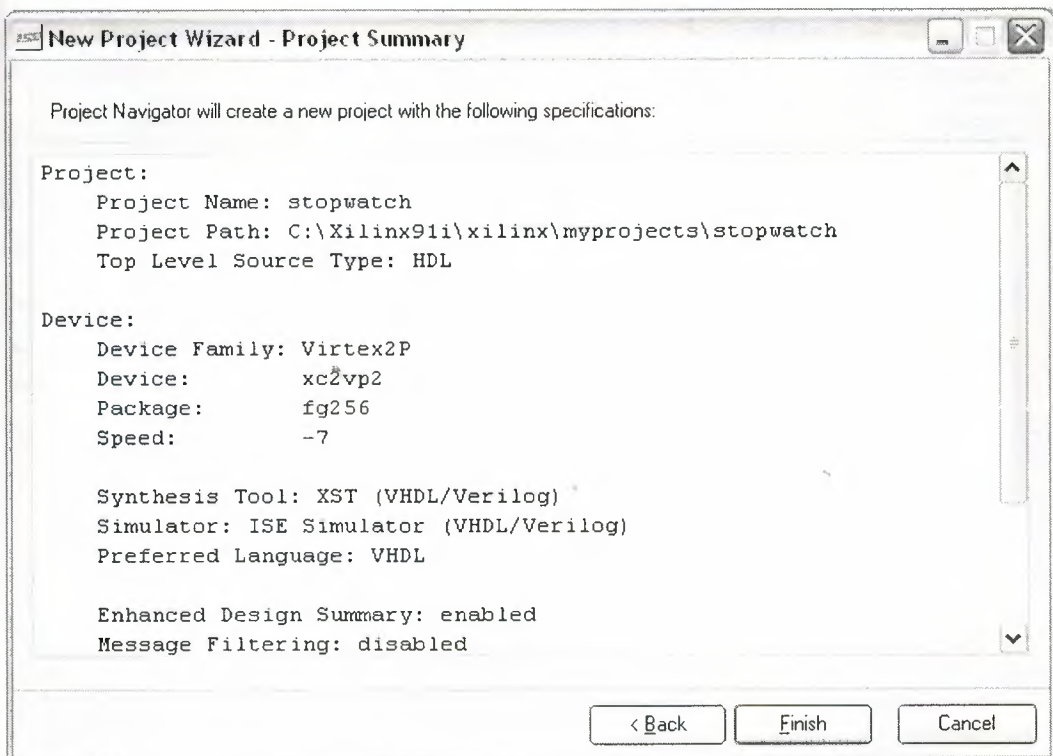


Figure 10 Click finish



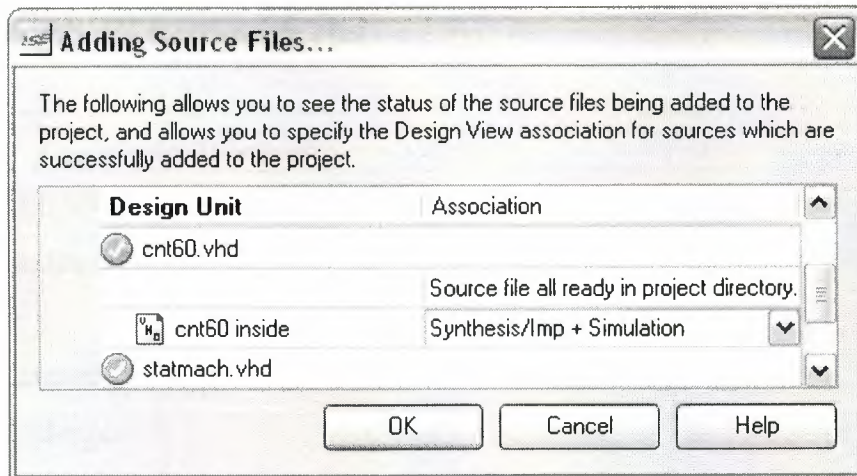


Figure 11

Click OK

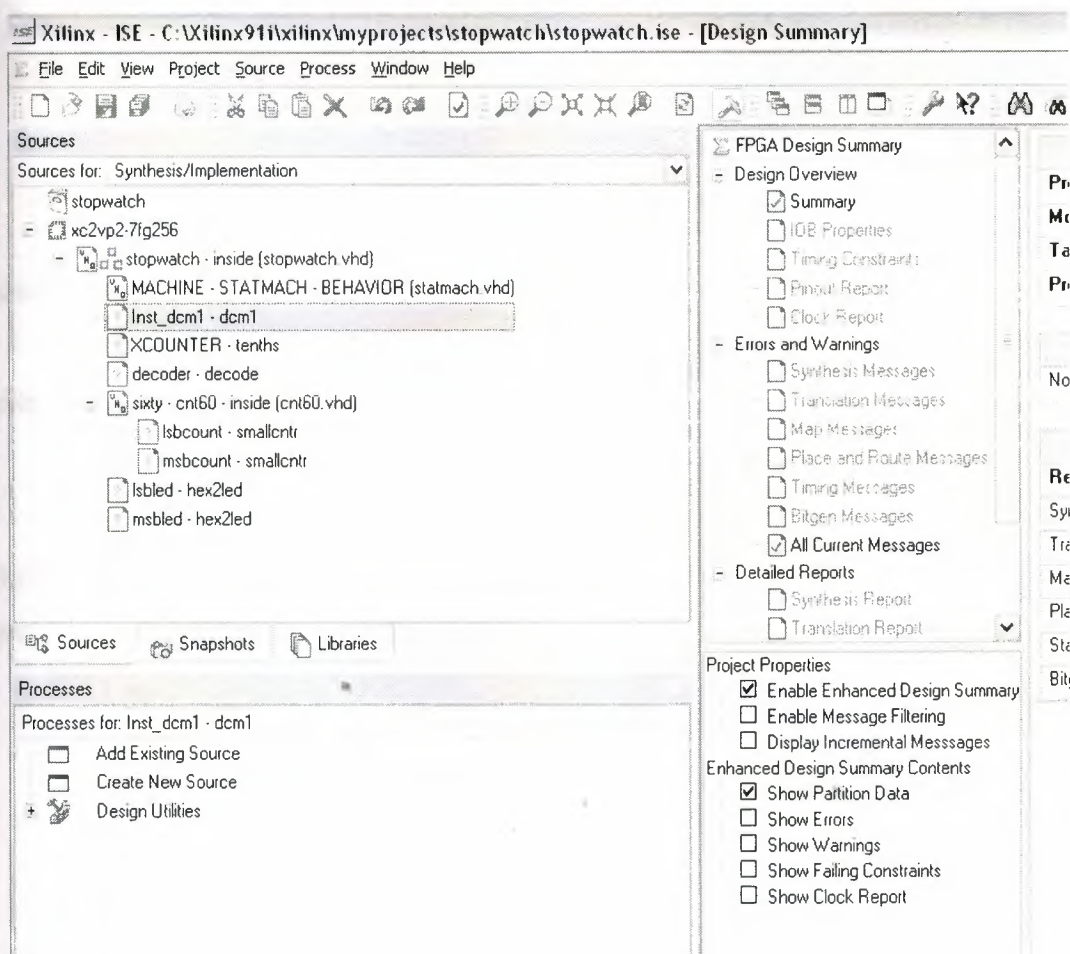


Figure 12-dcm1

### 3.1.1.CODES IN *stopwatch.vhd*

```
library IEEE;
use IEEE.std_logic_1164.all;
--synopsys translate_off
library UNISIM;
use unisim.vcomponents.all;
--synopsys translate_on

entity stopwatch is
    port ( CLK : in STD_LOGIC;
           RESET : in STD_LOGIC;
           STRTSTOP : in STD_LOGIC;
           TENTHSOUT : out STD_LOGIC_VECTOR(9 downto 0);
           ONESOUT : out STD_LOGIC_VECTOR(6 downto 0);
           TENSOUT : out STD_LOGIC_VECTOR(6 downto 0));
end stopwatch;

architecture inside of stopwatch is

    component statmach
        port ( CLK : in STD_LOGIC;
              RESET : in STD_LOGIC;
              STRTSTOP : in STD_LOGIC;
              locked : std_logic;
              CLKEN : out STD_LOGIC;
              RST : out STD_LOGIC);
    end component;

    component dcml
        PORT(
            RST_IN : IN std_logic;
```

```

        CLKIN_IN : IN std_logic;
        LOCKED_OUT : OUT std_logic;
        CLKD_OUT : OUT std_logic;
        CLKIN_IBUFG_OUT : OUT std_logic
    );
end component;

```

component tenths

```

    port (
        Q: OUT std_logic_VECTOR(3 downto 0);
        CLK: IN std_logic;
        Q_THRESHD: OUT std_logic;
        CE: IN std_logic;
        AINIT: IN std_logic);

```

end component;

-- FPGA Express Black Box declaration

```

attribute fpga_dont_touch: string;
attribute fpga_dont_touch of tenths: component is "true";

```

-- Synplicity black box declaration

```

attribute syn_black_box : boolean;
attribute syn_black_box of tenths: component is true;

```

component decode port (

```

        binary: in std_logic_vector(3 downto 0);
        one_hot: out std_logic_vector(9 downto 0));

```

end component;

component cnt60

```

    port ( CE : in STD_LOGIC;
           CLK : in STD_LOGIC;
           CLR : in STD_LOGIC;

```

```

        LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
        MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
end component;

```

```

component hex2led
    port ( HEX : in STD_LOGIC_VECTOR(3 downto 0);
          LED : out STD_LOGIC_VECTOR(6 downto 0));
end component;

```

```

signal strtstopinv : STD_LOGIC;
signal clkenable : STD_LOGIC;
signal rstint : STD_LOGIC;
signal xcountout : STD_LOGIC_VECTOR(9 downto 0);
signal xtermcnt : STD_LOGIC;
signal cnt60enable : STD_LOGIC;
signal lsbcnt : STD_LOGIC_VECTOR(3 downto 0);
signal msbcnt : STD_LOGIC_VECTOR(3 downto 0);
signal Q: std_logic_vector(3 downto 0);
signal clk_dcm : std_logic;
signal dcm_lock : std_logic;

```

```
begin
```

```

MACHINE:statmach port map(CLK=>clk_dcm,
    RESET=>RESET,
    STRTSTOP=>strtstopinv,
    locked => dcm_lock,
    CLKEN=>clkenable,
    RST=>rstint);

```

```

Inst_dcml: dcml PORT MAP(
    RST_IN => reset,
    CLKIN_IN => clk,

```



```

LOCKED_OUT => dcm_lock,
CLKO_OUT => clk_dcm,
CLKIN_IBUFG_OUT => open
);

```

XCOUNTER : tenths

```

port map (
    Q => Q,
    CLK => CLK_dcm,
    Q_THRESHO => xtermcnt,
    CE => clkenable,
    AINIT => rstint);

```

decoder: decode port map (

```

    binary => Q,
    one_hot => xcountout);

```

sixty: cnt60 port map(CE=>cnt60enable,

```

    CLK=>clk_dcm,
    CLR=>rstint,
    LSBSEC=>lsbcnt,
    MSBSEC=>msbcnt);

```

lsbled:hex2led port map(HEX=>lsbcnt,

```

    LED=>ONESOUT);

```

msbled:hex2led port map(HEX=>msbcnt,

```

    LED=>TENSOUT);

```

```

cnt60enable <= xtermcnt and clkenable;

```

```

TENTHSOUT <= not(xcountout);

```

```
strtstopinv <= not(STRTSTOP);
```

```
end inside;
```

### 3.1.2.CODES IN *statmach.vhd*

```
-- D:\XILINX\ISEEXAMPLES\WATCH_SC\STMACH_V.vhd  
-- VHDL code created by Visual Software Solution's StateCAD 5.02.x4  
-- Thu Jun 01 13:28:21 2000
```

```
-- This VHDL code (for use with Synopsys) was generated using:  
-- binary encoded state assignment with structured code format.  
-- Minimization is disabled, implied else is enabled,  
-- and outputs are manually optimized.
```

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;
```

```
ENTITY STATMACH IS
```

```
    PORT (CLK,reset,strtstop,locked: IN std_logic;
```

```
          clken,rst : OUT std_logic);
```

```
END;
```

```
ARCHITECTURE BEHAVIOR OF STATMACH IS
```

```
    SIGNAL sreg : std_logic_vector (2 DOWNTO 0);
```

```
    SIGNAL next_sreg : std_logic_vector (2 DOWNTO 0);
```

```
    CONSTANT CLEAR : std_logic_vector (2 DOWNTO 0) := "000";
```

```
    CONSTANT counting : std_logic_vector (2 DOWNTO 0) := "001";
```

```
    CONSTANT start : std_logic_vector (2 DOWNTO 0) := "010";
```

```
    CONSTANT stop : std_logic_vector (2 DOWNTO 0) := "011";
```

```
    CONSTANT stopped : std_logic_vector (2 DOWNTO 0) := "100";
```

```
    CONSTANT zero : std_logic_vector (2 DOWNTO 0) := "101";
```

BEGIN

PROCESS (CLK, reset, next\_sreg)

BEGIN

IF ( reset='1' ) THEN

sreg <= CLEAR;

ELSIF CLK='1' AND CLK'event THEN

sreg <= next\_sreg;

END IF;

END PROCESS;

PROCESS (sreg,strtstop,locked)

BEGIN

clken <= '0'; rst <= '0';

next\_sreg<=CLEAR;

CASE sreg IS

WHEN CLEAR =>

clken<='0';

rst<='1';

next\_sreg<=zero;

WHEN counting =>

clken<='1';

rst<='0';

IF ( strtstop='0' and locked = '1' ) THEN

next\_sreg<=counting;

else

next\_sreg<=stop;

END IF;

WHEN start =>

clken<='1';

rst<='0';

IF ( strtstop='0' and locked = '1' ) THEN

```

        next_sreg<=counting;
    else
        next_sreg<=start;
    END IF;
WHEN stop =>
    clken<='0';
    rst<='0';
    IF ( strtstop='0') THEN
        next_sreg<=stopped;
    else
        next_sreg<=stop;
    END IF;
WHEN stopped =>
    clken<='0';
    rst<='0';
    IF ( strtstop='1' and locked = '1') THEN
        next_sreg<=start;
    else
        next_sreg<=stopped;
    END IF;
WHEN zero =>
    clken<='0';
    rst<='0';
    IF ( strtstop='1' and locked = '1') THEN
        next_sreg<=start;
    else
        next_sreg<=zero;
    END IF;
WHEN OTHERS =>
    next_sreg<=CLEAR;

END CASE;
END PROCESS;
END BEHAVIOR;

```

### 3.1.3.State Machine Diagram

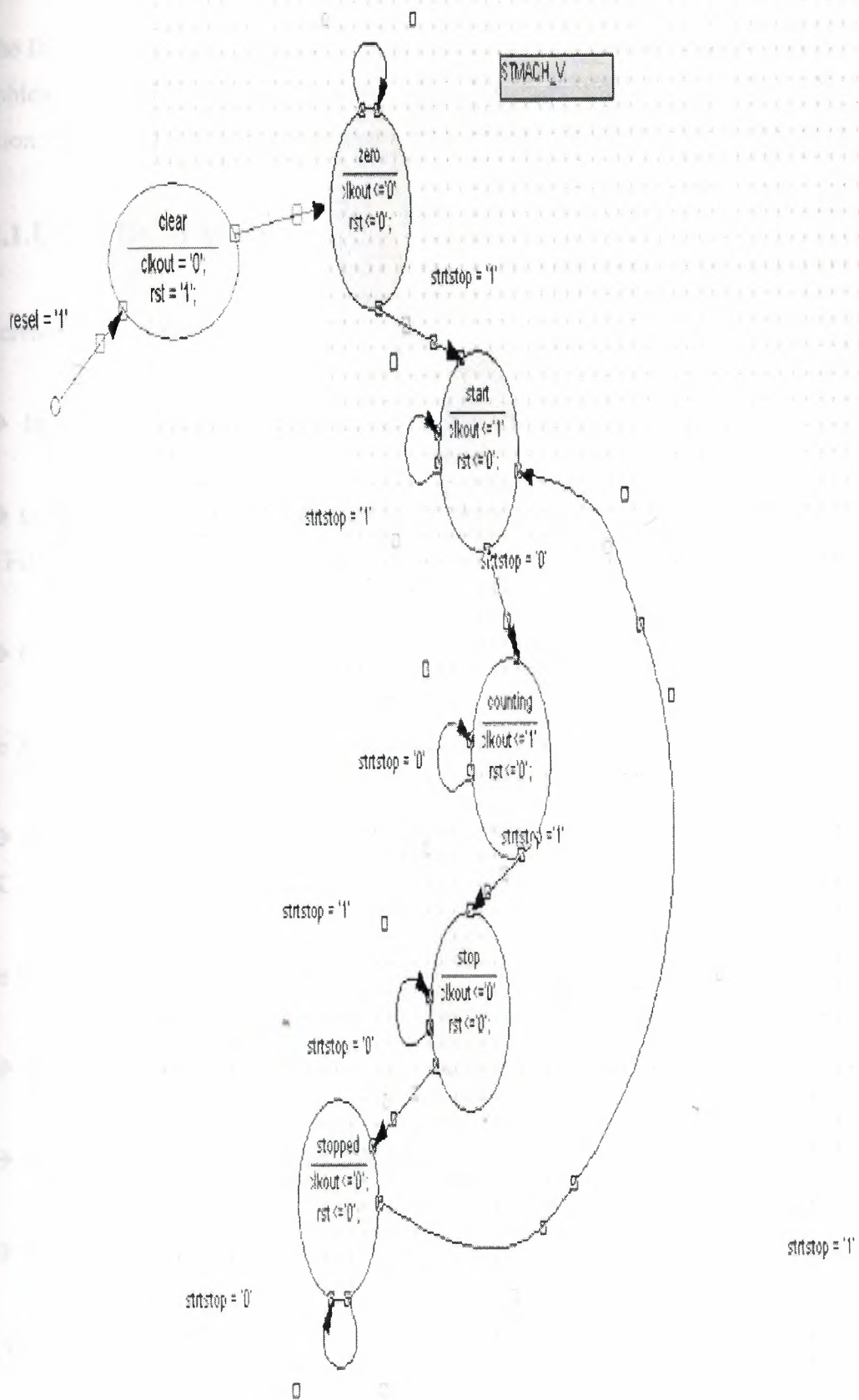


Figure 13-state machine diagram

## 3.2.CREATING A DCM MODULE

The DCM Wizard, one part of the Xilinx Architecture Wizard, enables a user to graphically select Digital Clock Manager (DCM) features that you wish to use. In this section, create a basic DCM module with CLK0 feedback and duty-cycle correction.

### 3.2.1.Using DCM Wizard

To create the DCM1 module:

1 → In Project Navigator, select **Project** → **New Source**.

2 → In the New Source dialog box, select **Architecture Wizard** and type 'DCM1' for the File Name.

3 → Click **Next**, then **Finish**.

The Xilinx Architecture Wizard is launched.

4 → In the Xilinx Architecture Wizard selection box, select **DCM Wizard** and click **OK**.

The DCM Wizard is launched.

5 → Deselect **RST** and **LOCKED**.

6 → Type **50** for the **Input Clock Frequency**.

7 → Verify the following settings:

◆ CLKIN Source: **External**

◆ Feedback Source: **Internal**



◆ Feedback Value: **1X**

◆ Phase Shift: **None**

◆ Duty Cycle Correction: **Yes**

8 → Select the **Advanced** button.

9 → Change Wait for DCM lock before DONE signal

10 → Select **OK** and **Next**.

An informational message displays the locked

11 → Select **OK** and **Finish**.

*DCM1.xaw* is added to the list of project source files

**Note:** The newly created *DCM1\_arwz.ucf* does not need constraints are passed into the relevant source file(s).

**Xilinx Clocking Wizard - General Setup**

The diagram shows a DCM (Digital Clock Manager) block with the following pins and connections:

- Inputs:** CLKIN, CLKFB, RST (checked), PSEN, PSINCDEC, PSCLK.
- Outputs:** CLK0 (checked), CLK90, CLK180, CLK270, CLKDV, CLK2X, CLK2X180, CLKFX, CLKFX180, STATUS, LOCKED (checked), PSDONE.

**Input Clock Frequency:** 50.000 MHz ☒ ns

**Phase Shift:** Type: NONE Value: 0

**CLKIN Source:** ☒ External ☐ Internal

**Feedback Source:** ☐ External ☒ Internal ☐ None

**Divide By Value:** 2

**Feedback Value:** ☒ 1X ☐ 2X

☒ Use Duty Cycle Correction

Buttons: More Info, Advanced, < Back, Next >, Cancel

Figure 14

Click advanced

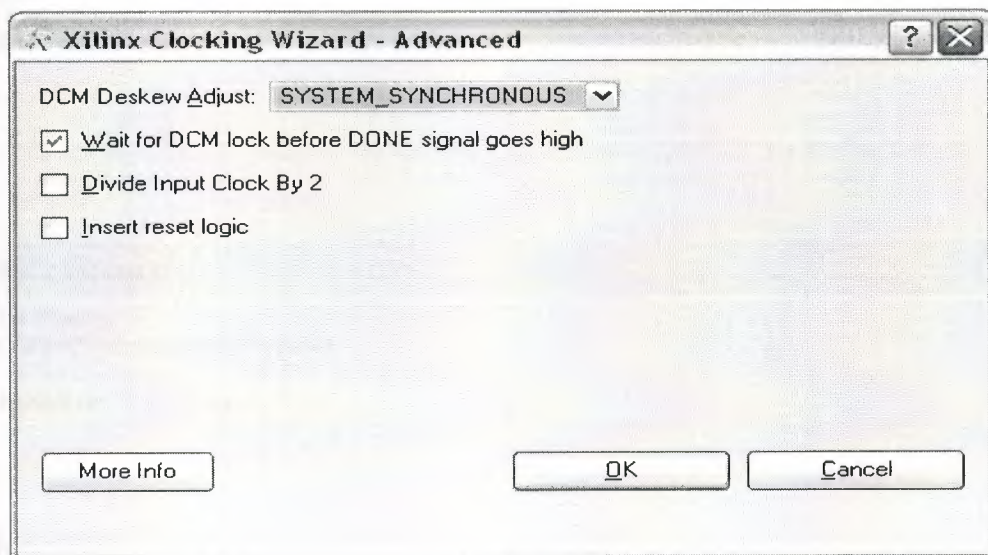


Figure 15 Select first one and click ok

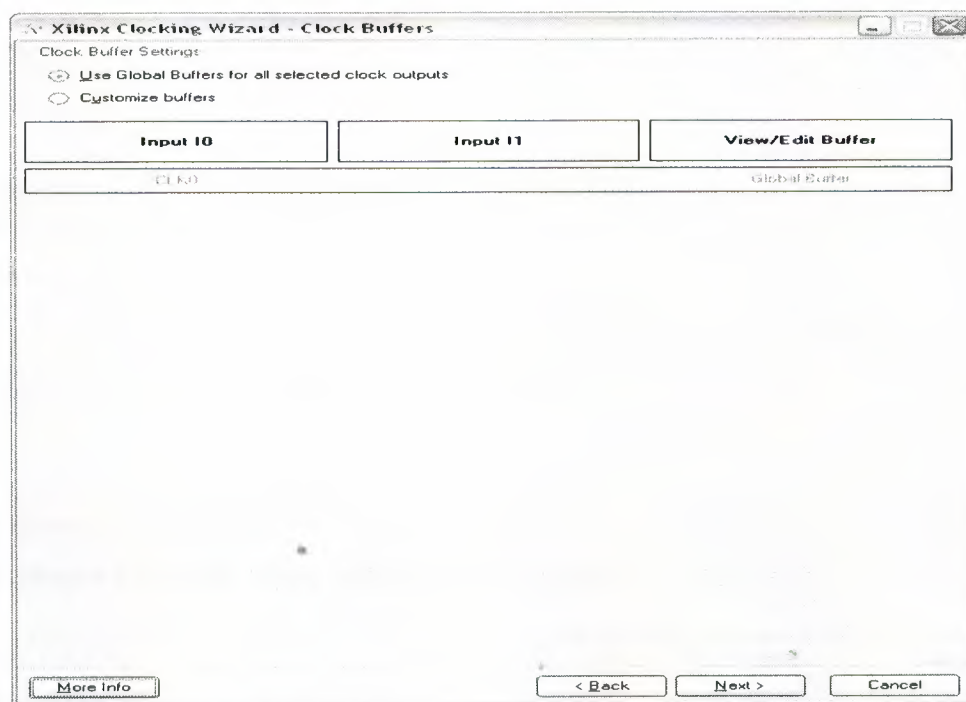


Figure 16

Click next

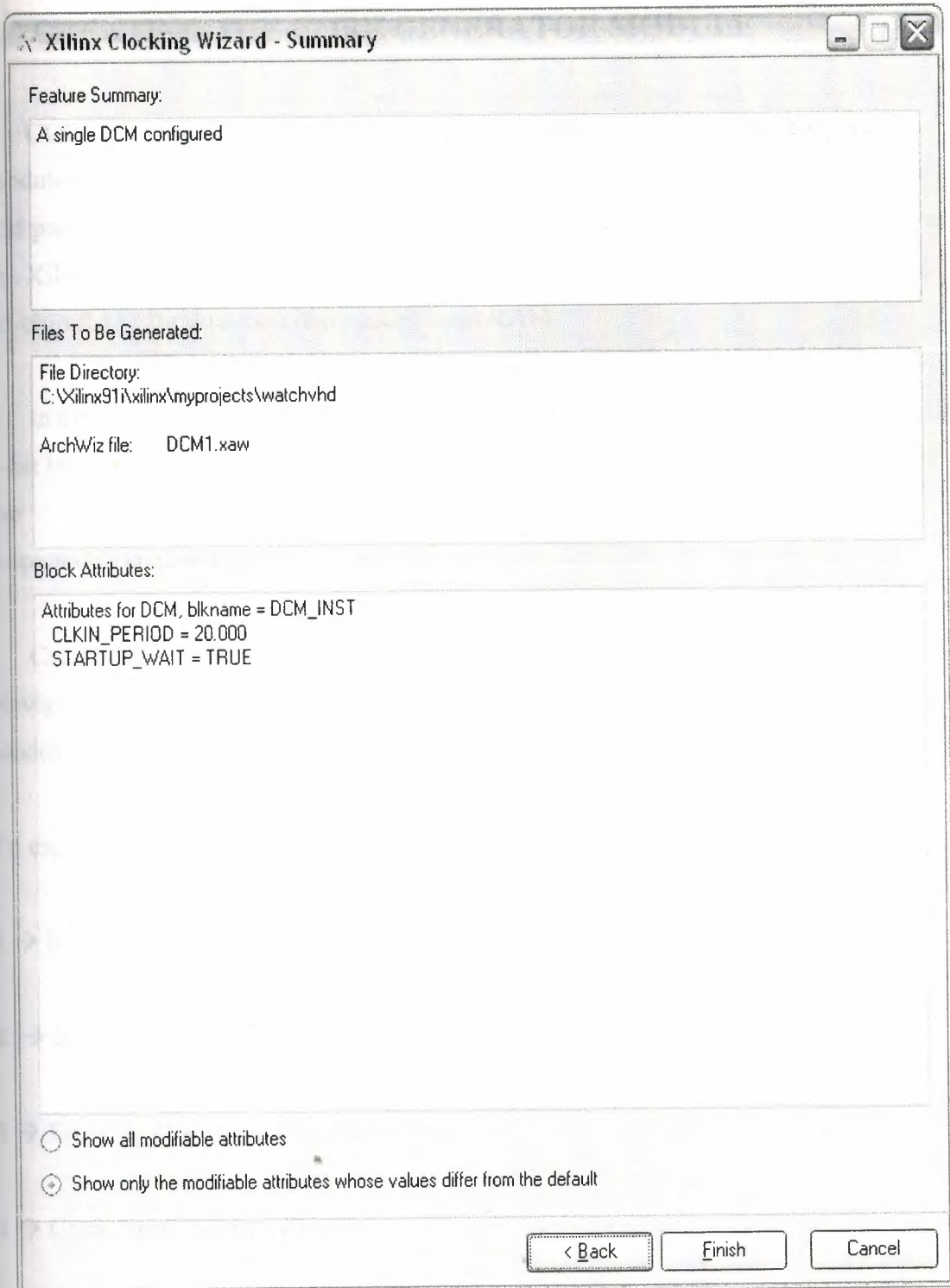


Figure 17 And click finish



### 3.3.CREATING THE CORE GENERATOR MODULE

CORE Generator is a graphical interactive design tool used to create high-level modules such as counters, shift registers, RAM and multiplexers. You can customize and preoptimize the modules to take advantage of the inherent architectural features of the Xilinx FPGA architectures, such as Fast Carry Logic for arithmetic functions and on-chip RAM for dual-port and synchronous RAM.

In this section, you will create a CORE Generator module called Tenthths. Tenthths is a 4-bit binary encoded counter. The 4-bit number is decoded to count the tenths digit of the stopwatch's time value.

Create the CORE Generator module using the New Source Wizard in Project Navigator. This invokes CORE Generator in which you can select and define the type of module you want.

To create the module:

- 1 → In Project Navigator, select **Project** → **New Source**.
2. → Select **Coregen IP** as the source type.
- 3 → Enter 'tenths' in the File Name field.
- 4 → Click **Next** and then **Finish**.

The Xilinx CORE Generator opens and displays a list of possible COREs available.

- 5 → Double-click on **Basic Elements - Counters**.
- 6 → Double-click on **Binary Counter** to open the Binary Counter dialog box.

This dialog box enables you to customize the counter to the design specifications.

7 → Fill in the Binary Counter dialog with the following settings:

◆ Component Name: **tenths**

Defines the name of the module.

◆ Output Width: **4**

Defines the width of the output bus.

◆ Operation: **Up**

Defines how the counter will operate. This field is dependent on the type of module you select.

◆ Count Style: **Count by Constant**

Allows counting by a constant or a user supplied variable.

◆ Count Restrictions: **Enable and Count To Value A (HEX)**

This dictates the maximum count value.



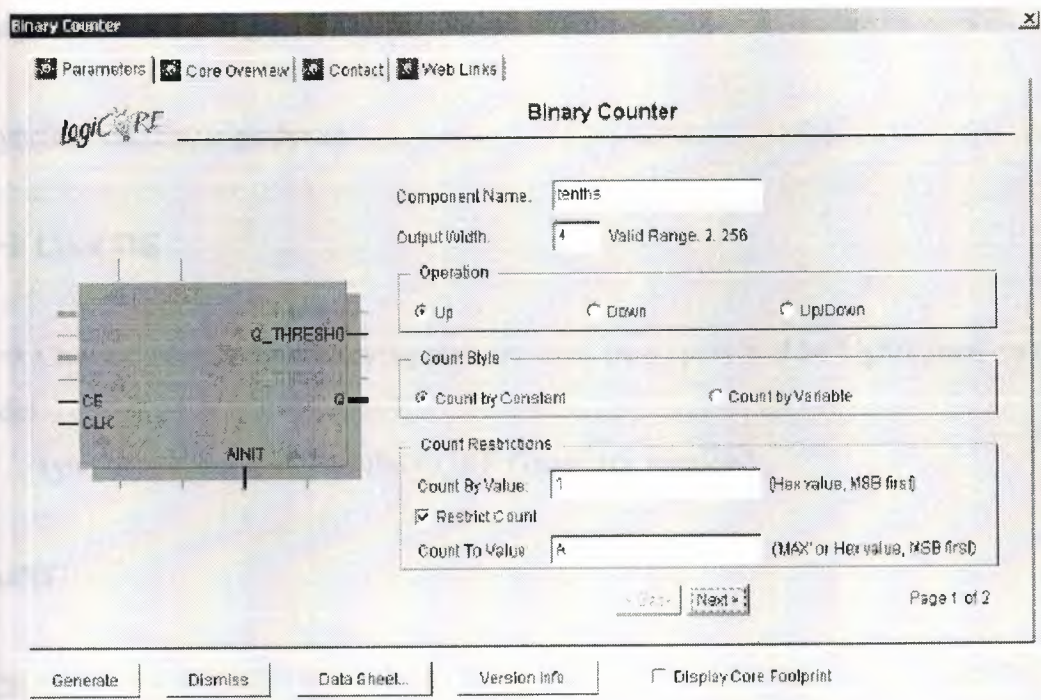


Figure 18-binary counter

8 → Select the **Next** button.

9 → Continue to fill in the Binary Counter dialog with the following settings:

♦ Threshold Options: **Threshold 0 set to A**

Signal goes high when the value specified has been reached.

♦ Threshold Options: **Registered**

10 → Click the **Register Options** button to open the Register Options dialog box.

11 → In the Register Options dialog box, enter the following settings:

♦ Clock Enable: **Selected**

♦ Asynchronous Settings: **Init with a value of 1**

♦ Synchronous Settings: **None**

12 → Click **OK**.

13 → Check that *only* the following pins are used (used pins will be highlighted on the model

symbol to the left side of the CORE Generator window):

♦ **AINIT**

♦ **CE**

♦ **Q**

♦ **Q\_THRESHOLD**

♦ **CLK**

14 → Click **Generate**.

*The module is created and automatically added to the project library.*

A number of other files are added to the project directory. These files are:

♦ *tenths.sym*

This is a schematic symbol file.

♦ *tenths.edn*

This file is the netlist that is used during the Translate phase of implementation.

◆ *tenths.vho* or *tenths.veo*

This is the instantiation template that is used to incorporate the CORE Generator module in your source HDL.

◆ *tenths.vhd* or *tenths.v*

These are simulation-only files.

◆ *tenths.xco*

This file stores the configuration information for the Tenths module and is used as a project source.

◆ *coregen.prj*

This file stores the Coregen configuration for the project.

15 → Click **Cancel** and close Core Generator.

### 3.4.CREATING AN HDL-BASED MODULE

Next, create a module from HDL code. With ISE, you can easily create modules from HDL code using the HDL Editor tool. The HDL code is then connected to your top-level HDL design through instantiation and is compiled with the rest of the design.

Now, you will author a new HDL module. This macro serves to convert the two 4-bit outputs of the CNT60 module into a 10-segment LED display format.

### 3.4.1.Using the New Source Wizard and HDL Editor

In order to create the module, first create a file using the New Source Wizard specifying the name and ports of the component. The resulting “skeleton” HDL file is then modified further in the HDL Editor.

To create the source file:

1 → Select **Project** → **New Source**.

A dialog box opens in which you specify the type of source you want to create.

2 → Select **VHDL Module**.

3 → In the File Name field, type '**decode**'.

4 → Click **Next**.

The *decode* component has a 4-bit input port named hex and a 10-bit output port named led.

To enter these ports:

5 → Click in the Port Name field and type **binary**.

6 → Click in the Direction field and set the direction to **in**.

7 → In the MSB field enter **3**, and in the LSB field enter **0**.

8 → Again click in the Port Name field and type **one\_hot**.

9 → Click in the Direction field and set the direction to **out**.

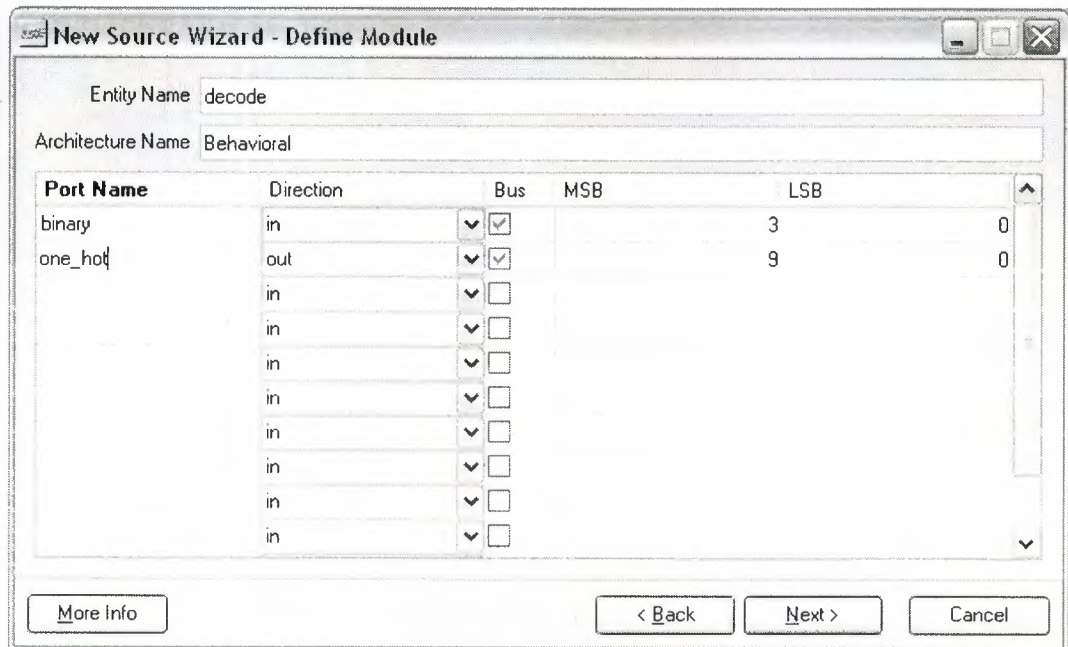
10 → In the MSB field enter **9**, and in the LSB field enter **0**.



11 → Click **Next** to complete the Wizard session.

A description of the module displays.

12 → Click **Finish** to open the empty HDL file in HDL Editor.

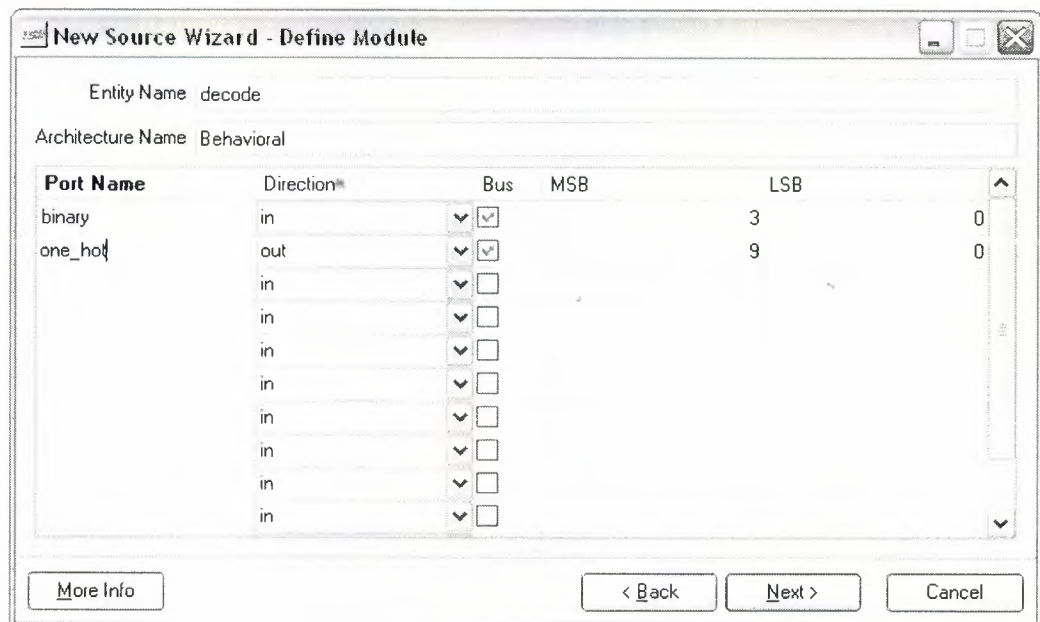


The dialog box titled "New Source Wizard - Define Module" shows the configuration for a new module. The "Entity Name" is "decode" and the "Architecture Name" is "Behavioral". A table lists the ports:

Port Name	Direction	Bus	MSB	LSB
binary	in	<input checked="" type="checkbox"/>		3
one_hot	out	<input checked="" type="checkbox"/>		9
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

Buttons at the bottom: More Info, < Back, Next >, Cancel.

Figure 19 Click next



The dialog box titled "New Source Wizard - Define Module" shows the configuration for a new module. The "Entity Name" is "decode" and the "Architecture Name" is "Behavioral". A table lists the ports:

Port Name	Direction	Bus	MSB	LSB
binary	in	<input checked="" type="checkbox"/>		3
one_hot	out	<input checked="" type="checkbox"/>		9
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

Buttons at the bottom: More Info, < Back, Next >, Cancel.

Figure 20 Click next





*Figure 21*  
Click Finish

Now we are selecting *decode.vhd* in source tab.

We will see this figure

```
18
19 -----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity decode is
31     Port ( binary : in  STD_LOGIC_VECTOR (3 downto 0);
32           one_hot : out STD_LOGIC_VECTOR (9 downto 0));
33 end decode;
34
35 architecture Behavioral of decode is
36
37 begin
38
39
40 end Behavioral;
41
```

Then we will write the following vhdl codes between **begin** and **architecture**.

with binary select

```
one_hot <=
    "0000000001" when "0001",      --1
    "0000000010" when "0010",      --2
    "0000000100" when "0011",      --3
    "0000001000" when "0100",      --4
    "0000010000" when "0101",      --5
    "0000100000" when "0110",      --6
    "0001000000" when "0111",      --7
    "0010000000" when "1000",      --8
    "0100000000" when "1001",      --9
    "1000000000" when "1010",      --10
    "0000000001" when others;      --1
```

I wrote this code to see the similar examples in **LANGUAGE TEMPLATES**.

To see the **LANGUAGE TEMPLATES**

- Click **edit**
- select **LANGUAGE TEMPLATES**

you will see the examples in **LANGUAGE TEMPLATES**

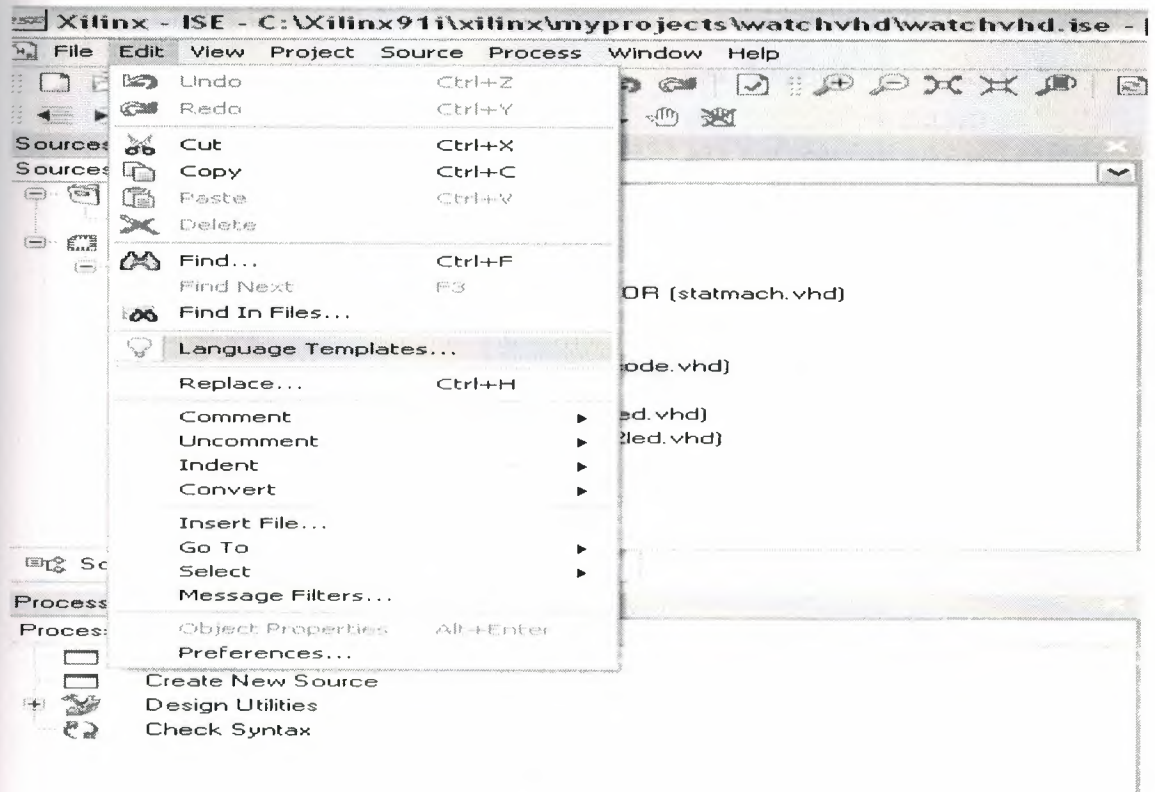


Figure 22

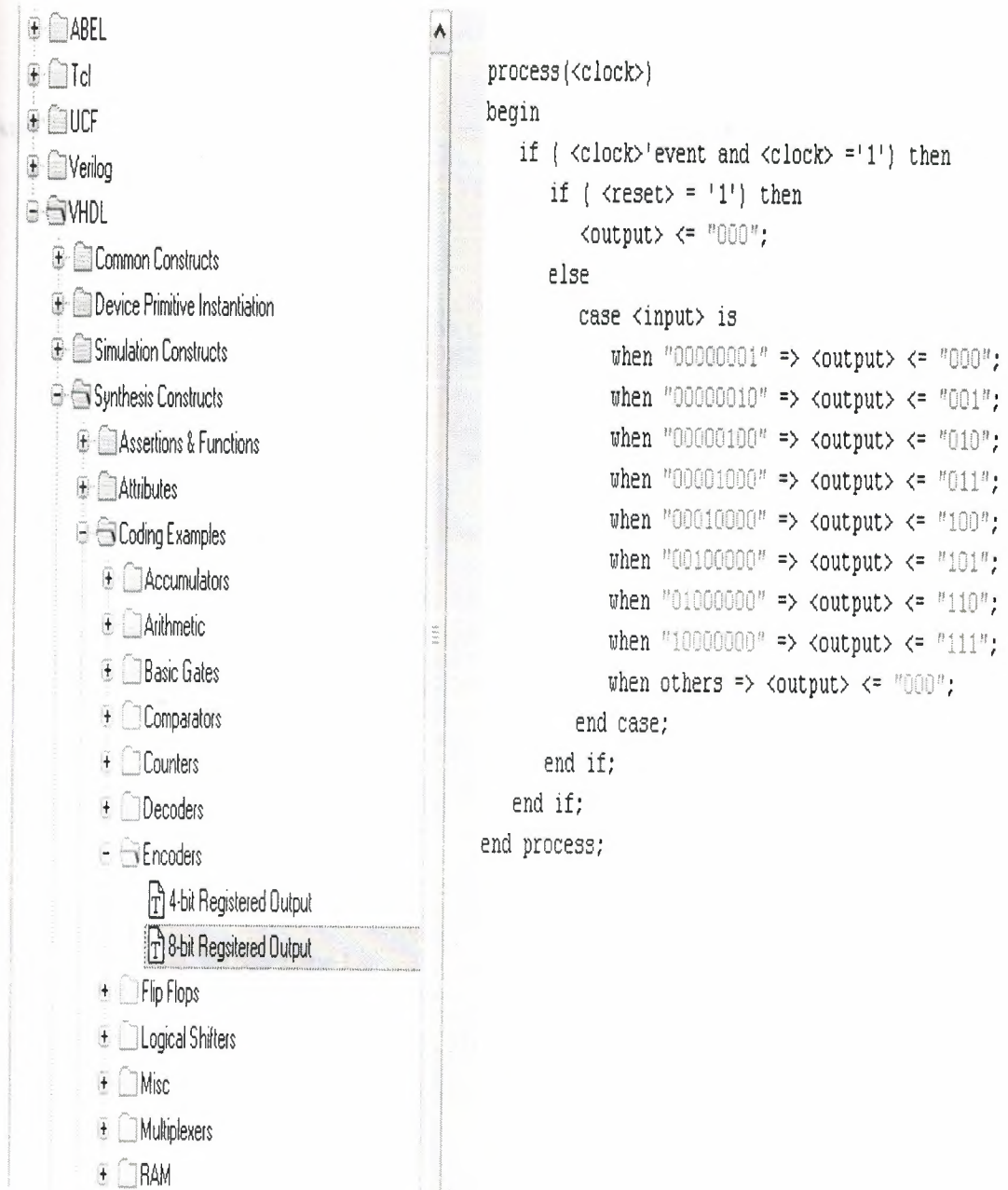


Figure 23  
**LANGUAGE TEMPLATES**



And after write this code then we will see this figure for the decode.vhd.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity decode is
7      Port ( binary : in std_logic_vector(3 downto 0);
8            one_hot : out std_logic_vector(9 downto 0));
9  end decode;
10
11  architecture behavioral of decode is
12
13  begin
14
15  with binary select
16      one_hot <= "0000000001" when "0001", --1
17                "0000000010" when "0010", --2
18                "0000000100" when "0011", --3
19                "0000001000" when "0100", --4
20                "0000010000" when "0101", --5
21                "0000100000" when "0110", --6
22                "0001000000" when "0111", --7
23                "0010000000" when "1000", --8
24                "0100000000" when "1001", --9
25                "1000000000" when "1010", --10
26                "0000000001" when others; --1
27
28  end behavioral;
29

```

Figure 24

decode.vhd

### 3.5.CREATING THE CNT60 VHDL MODULE

Click the project and new source

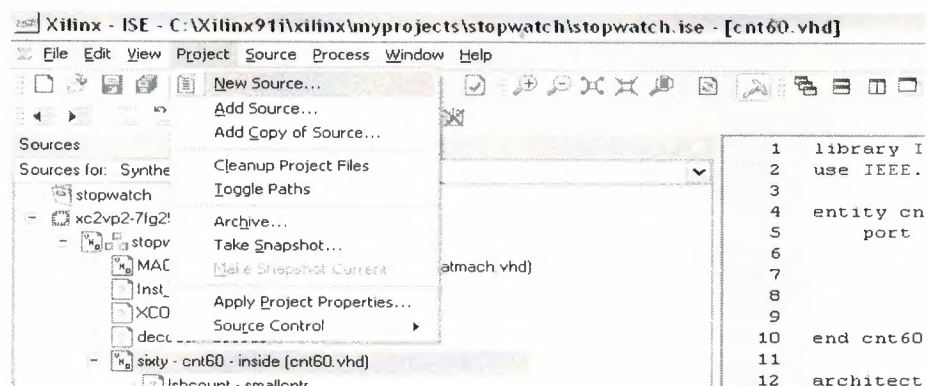


Figure 25



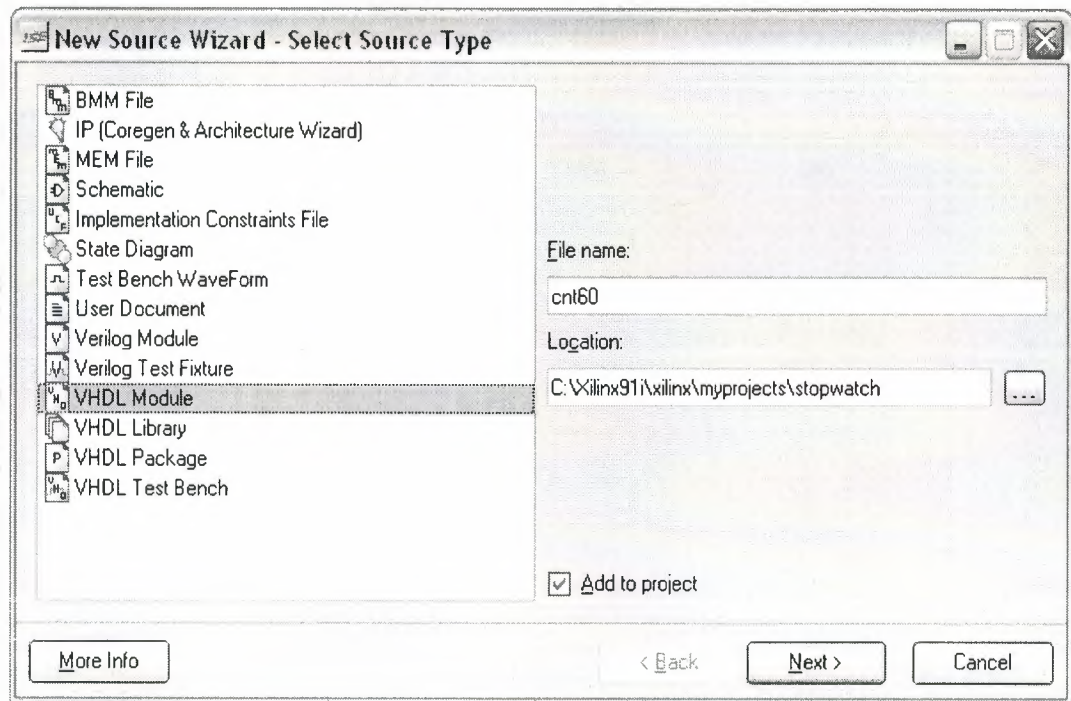


Figure 26

We are writing '**cnt60**' as file name and select the **vhdl module**.

And click **next**.

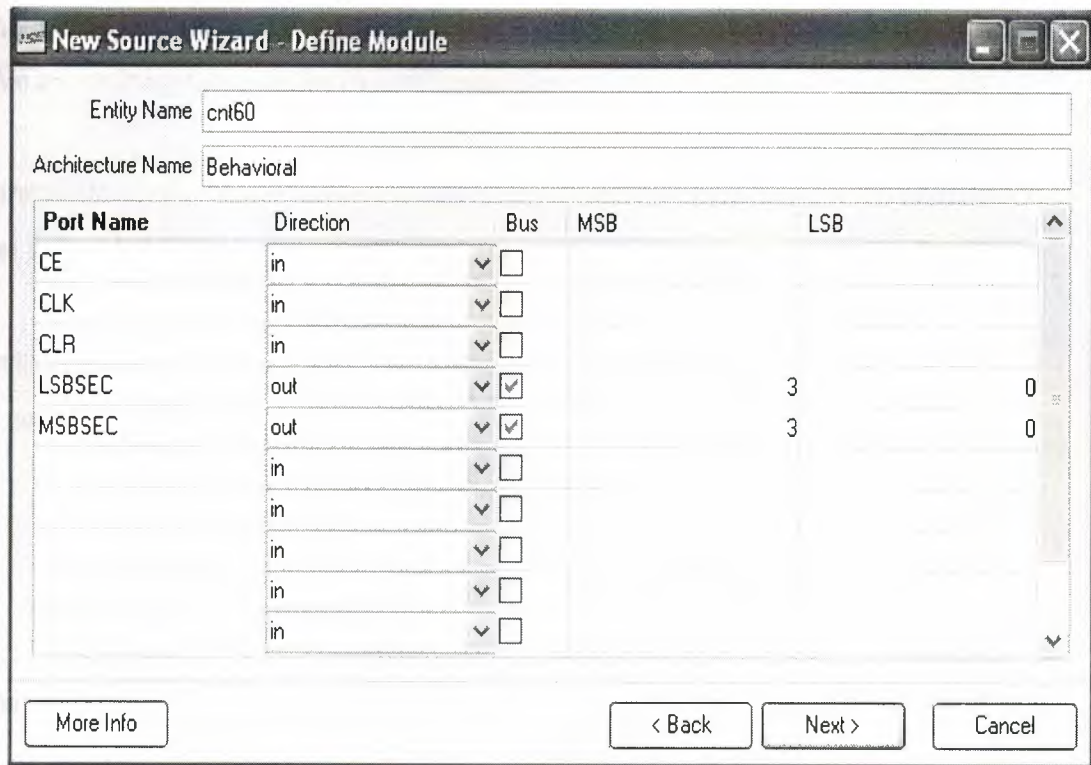
And write these **inputs** and **outputs**.

#### inputs

- CE
- CLK
- CLR

#### outputs

- LSBSEC
- MSBSEC



**New Source Wizard - Define Module**

Entity Name:

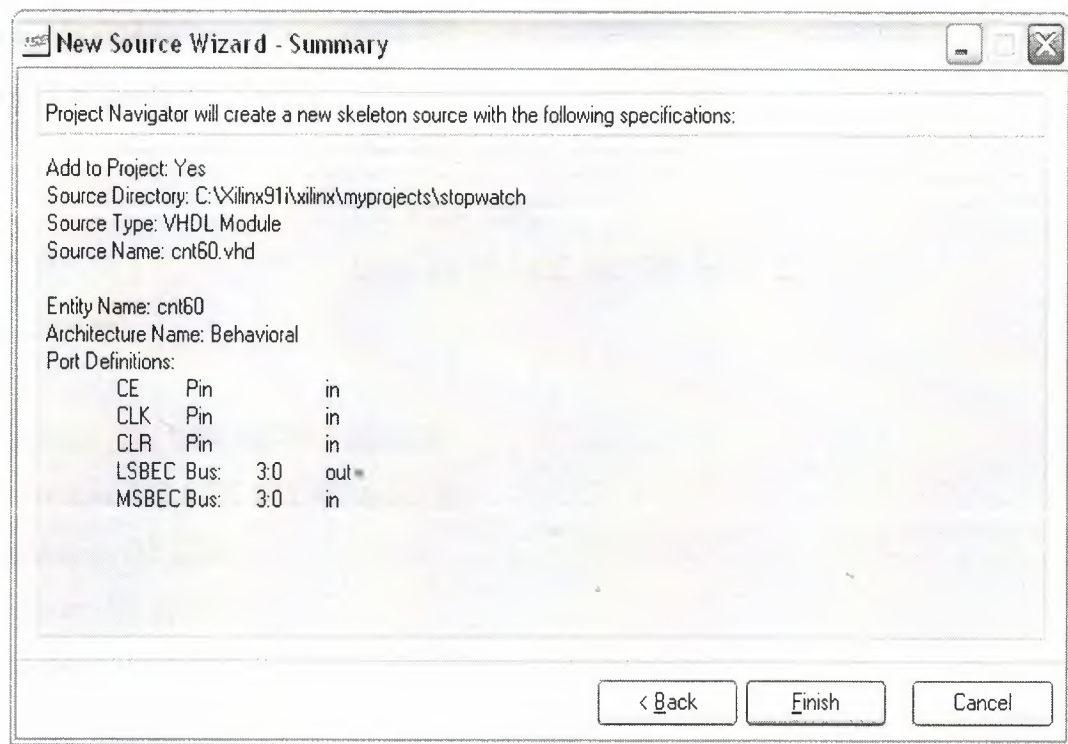
Architecture Name:

Port Name	Direction	Bus	MSB	LSB
CE	in	<input type="checkbox"/>		
CLK	in	<input type="checkbox"/>		
CLR	in	<input type="checkbox"/>		
LSBSEC	out	<input checked="" type="checkbox"/>		3 0
MSBSEC	out	<input checked="" type="checkbox"/>		3 0
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

More Info      < Back      Next >      Cancel

Figure 27

Then click next.



**New Source Wizard - Summary**

Project Navigator will create a new skeleton source with the following specifications:

Add to Project: Yes  
Source Directory: C:\Xilinx91\Xilinx\myprojects\stopwatch  
Source Type: VHDL Module  
Source Name: cnt60.vhd

Entity Name: cnt60  
Architecture Name: Behavioral

Port Definitions:

CE	Pin	in
CLK	Pin	in
CLR	Pin	in
LSBEC Bus:	3:0	out
MSBEC Bus:	3:0	in

< Back      Finish      Cancel

Figure 28

Click finish

### 3.5.1.CODES IN cnt60.vhd MODULE

We are writing these codes in cnt60 module

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

entity cnt60 is
    port ( CE : in STD_LOGIC;
          CLK : in STD_LOGIC;
          CLR : in STD_LOGIC;
          LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
          MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
end cnt60;

architecture inside of cnt60 is

    component smallcntr
        port ( CE : in STD_LOGIC;
              CLK : in STD_LOGIC;
              CLR : in STD_LOGIC;
              QOUT : out STD_LOGIC_VECTOR(3 downto 0));
    end component;

    signal lsbout : STD_LOGIC_VECTOR(3 downto 0);
    signal msbout : STD_LOGIC_VECTOR(3 downto 0);
    signal msbce : STD_LOGIC;
    signal lsbtc : STD_LOGIC;
    signal msbclr : STD_LOGIC;
    signal msbtc : STD_LOGIC;

begin

    lsbcoun: smallcntr port map(CE=>CE,CLK=>CLK,CLR=>CLR,QOUT=>lsbout);
```

```
msbcount: smallcntnr port map(CE=>msbce,CLK=>CLK,CLR=>msbclr,QDUT=>msbout);
```

```
process(lsbout)
```

```
begin
```

```
if(lsbout="1001") then
```

```
lsbtc<='1';
```

```
else
```

```
lsbtc<='0';
```

```
end if;
```

```
end process;
```

```
process(msbout)
```

```
begin
```

```
if(msbout="0110") then
```

```
msbtc<='1';
```

```
else
```

```
msbtc<='0';
```

```
end if;
```

```
end process;
```

```
msbce <= CE and lsbtc;
```

```
msbclr <= CLR or msbtc;
```

```
LSBSEC <= lsbout;
```

```
MSBSEC <= msbout;
```

```
end inside;
```

### 3.6.CREATING HEX2LED SOURCE USING NEW SOURCE WIZARD AND HDL EDITOR

In order to create the module, first create a file using the New Source Wizard specifying the name and ports of the component. The resulting “skeleton” HDL file is then modified further in the HDL Editor.

To create the source file:

**1 → Select Project → New Source.**

A dialog box opens in which you specify the type of source you want to create.

**2 → Select VHDL Module**

**3 → In the File Name field, type ‘hex2led’.**

**4 → Click Next.**

The *hex2led* component has a 4-bit input port named hex and a 7-bit output port named led.

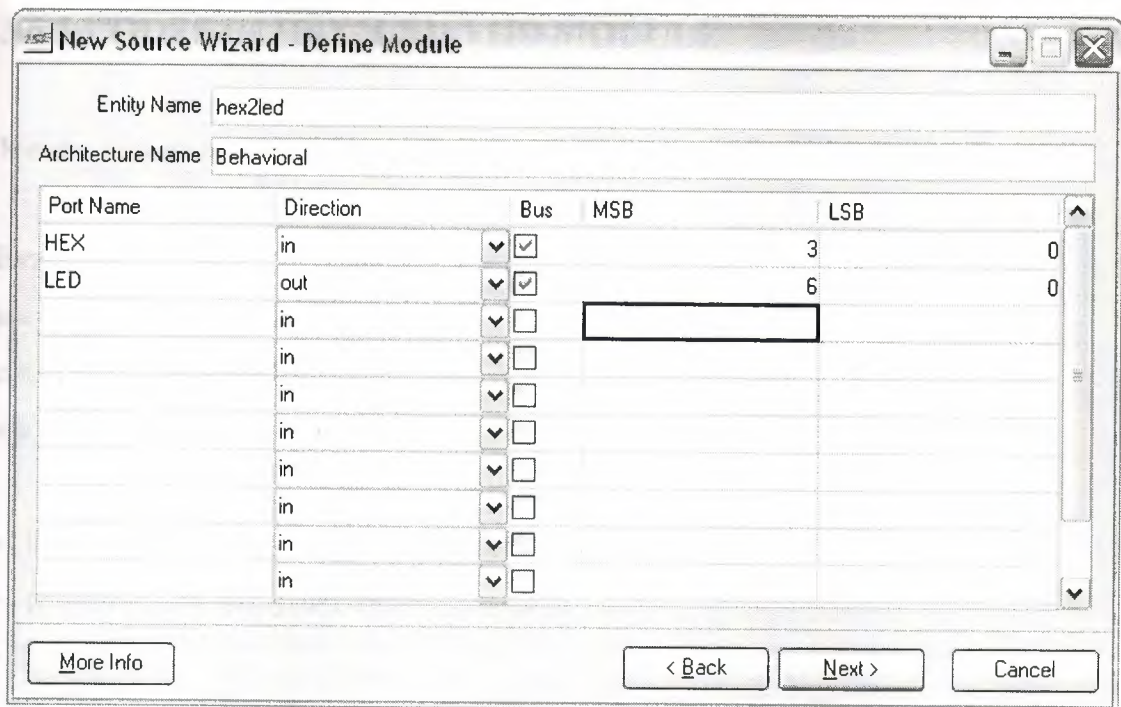
To enter these ports:

**5 → Click in the Port Name field and type HEX.**

**6 → Click in the Direction field and set the direction to in.**

**7 → In the MSB field enter 3, and in the LSB field enter 0.**





**New Source Wizard - Define Module**

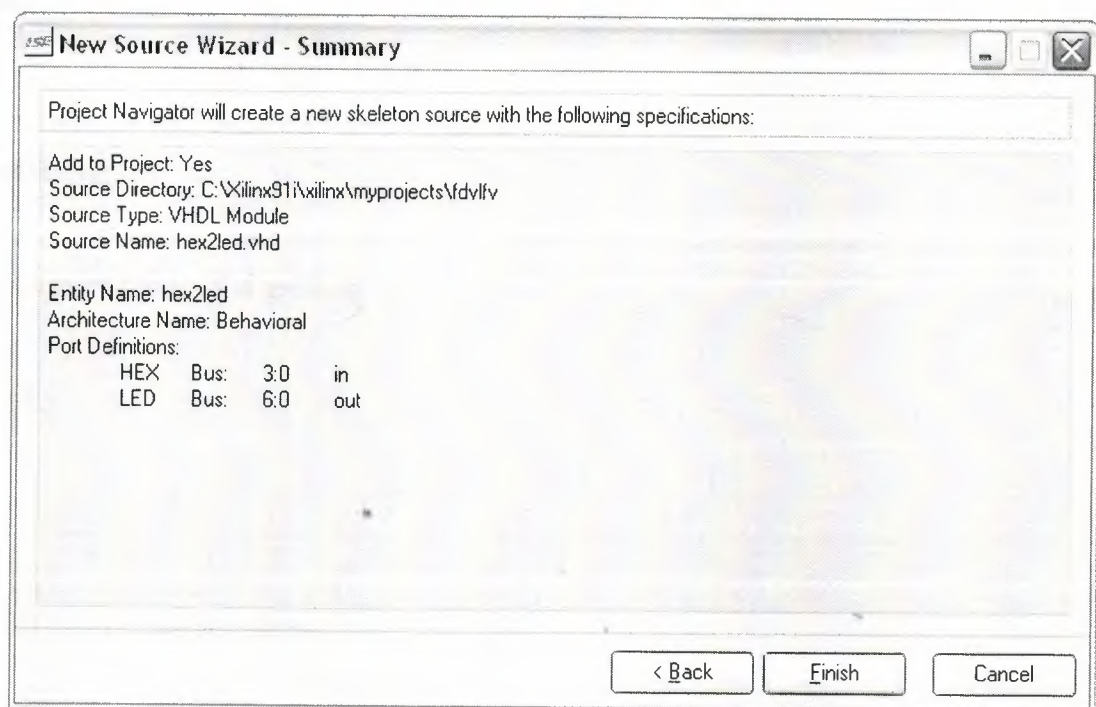
Entity Name:

Architecture Name:

Port Name	Direction	Bus	MSB	LSB
HEX	in	<input checked="" type="checkbox"/>		3
LED	out	<input checked="" type="checkbox"/>		6
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

Buttons:

Figure 29 Click next



**New Source Wizard - Summary**

Project Navigator will create a new skeleton source with the following specifications:

Add to Project: Yes  
Source Directory: C:\Xilinx91\Xilinx\myprojects\fdvlfv  
Source Type: VHDL Module  
Source Name: hex2led.vhd

Entity Name: hex2led  
Architecture Name: Behavioral

Port Definitions:

HEX	Bus:	3:0	in
LED	Bus:	6:0	out

Buttons:

Figure 30  
Click finish

### 3.6.1.CODES IN HEX2LED.VHD MODULE

*We are writing and completing these codes.*

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity hex2led is
    Port ( HEX : in std_logic_vector(3 downto 0);
          LED : out std_logic_vector(6 downto 0));
end hex2led;

architecture Behavioral of hex2led is

begin

    --HEX-to-seven-segment decoder
    -- HEX: in  STD_LOGIC_VECTOR (3 downto 0);
    -- LED: out STD_LOGIC_VECTOR (6 downto 0);
    --
    -- segment encoding
    -- 0
    -- ---
    -- 5 | 11
```

```
-- --- <- 6
-- 4| |2
-- ---
-- 3
```

with HEX SElect

```
LED<= "1111001" when "0001", --1
      "0100100" when "0010", --2
      "0110000" when "0011", --3
      "0011001" when "0100", --4
      "0010010" when "0101", --5
      "0000010" when "0110", --6
      "1111000" when "0111", --7
      "0000000" when "1000", --8
      "0010000" when "1001", --9
      "0001000" when "1010", --A
      "0000011" when "1011", --b
      "1000110" when "1100", --C
      "0100001" when "1101", --d
      "0000110" when "1110", --E
      "0001110" when "1111", --F
      "1000000" when others; --0
```

end Behavioral;

## CHAPTER FOUR

### ESTABLISHING THE TEST BENCH OF DESIGN

To establish the test bench of design we are selecting Project and new source then selecting the VHDL Test Bench module and typing stopwatch\_tb.vhd as file name.

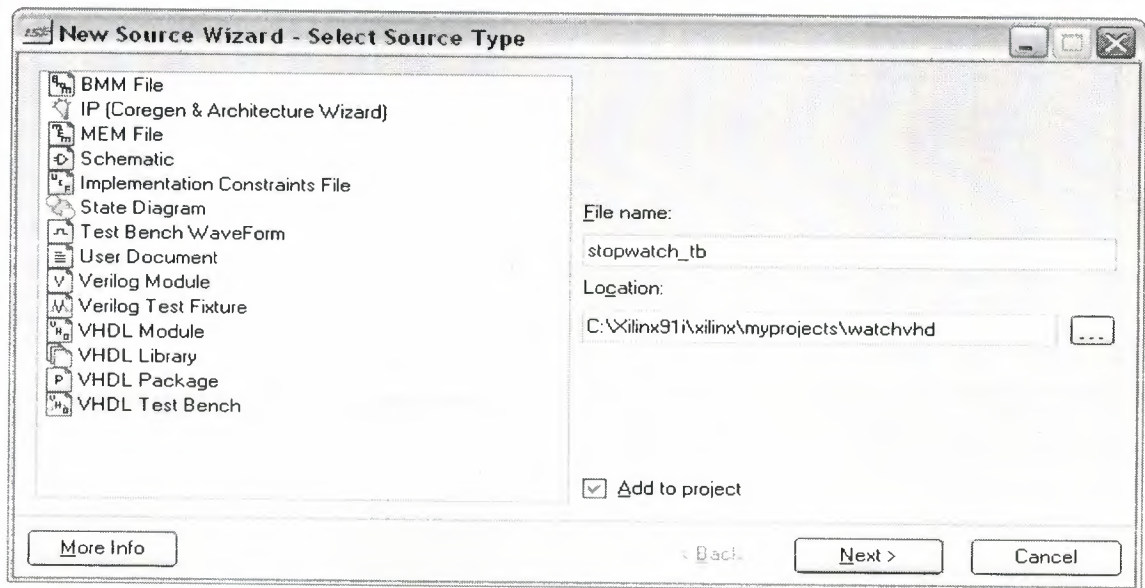


Figure 31 Click next

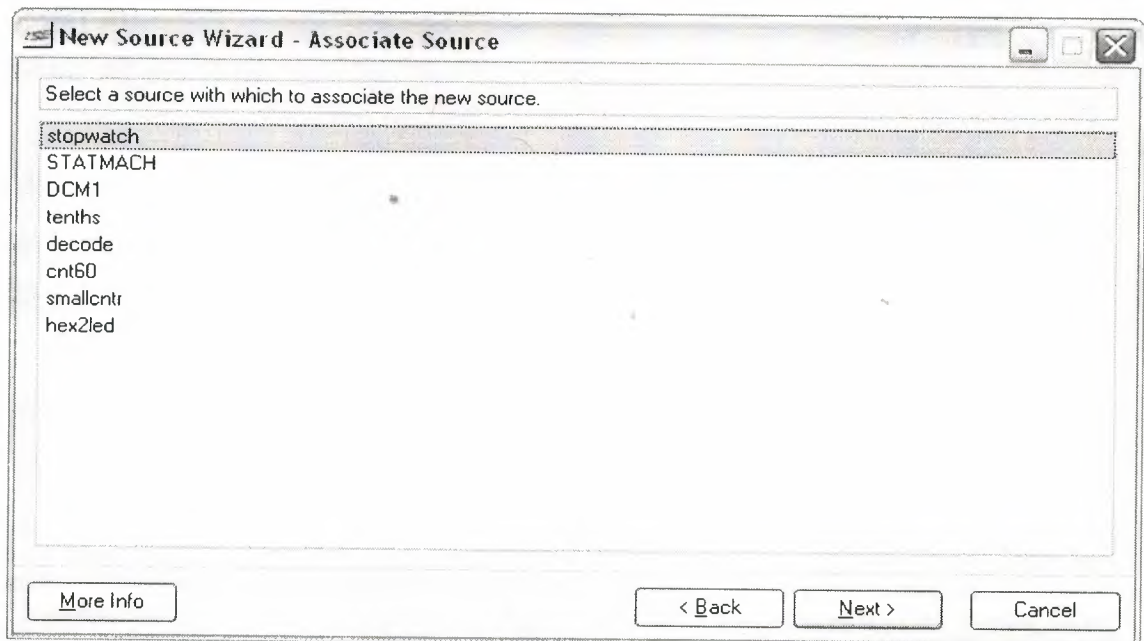




Figure 32 Select the stopwatch then click next.



Figure 33 Click finish

#### 4.1.CODES IN TEST BENCH

```
LIBRARY IEEE;
```

```
USE IEEE.std_logic_1164.all;
```

```
LIBRARY ieee;
```

```
USE IEEE.STD_LOGIC_TEXTIO.ALL;
```

```
USE STD.TEXTIO.ALL;
```

```
ENTITY tb IS
```

```
END tb;
```

```
ARCHITECTURE testbench_arch OF tb IS
```



COMPONENT stopwatch

PORT (

ONESOUT : out STD\_LOGIC\_VECTOR (6 DOWNTO 0);  
TENSOUT : out STD\_LOGIC\_VECTOR (6 DOWNTO 0);  
TENTHSOUT : out STD\_LOGIC\_VECTOR (9 DOWNTO 0);  
CLK : in STD\_LOGIC;  
RESET : in STD\_LOGIC;  
STRTSTOP : in STD\_LOGIC);

END COMPONENT;

SIGNAL ONESOUT : STD\_LOGIC\_VECTOR (6 DOWNTO 0);  
SIGNAL TENSOUT : STD\_LOGIC\_VECTOR (6 DOWNTO 0);  
SIGNAL TENTHSOUT : STD\_LOGIC\_VECTOR (9 DOWNTO 0);  
SIGNAL CLK : STD\_LOGIC;  
SIGNAL RESET : STD\_LOGIC;  
SIGNAL STRTSTOP : STD\_LOGIC;

constant ClockPeriod : Time := 10 ns;

BEGIN

UUT : stopwatch

PORT MAP (

CLK => CLK,  
RESET => RESET,  
STRTSTOP => STRTSTOP,  
TENTHSOUT => TENTHSOUT,  
ONESOUT => ONESOUT,  
TENSOUT => TENSOUT

);

generateclock: process

begin

```

        clk <= '1';
    loop
        wait for (ClockPeriod / 2);
        CLK <= not CLK;
    end loop;
end process;

stimulus: process
begin
    --Initialize Inputs
    reset <= '1';
    strtstop <= '1';
    --Wait until the Global Set/Reset deasserts
    wait for 100 ns;
    reset <= '0';
    --Wait long enough for the DCM to lock
    wait for 600 ns;
    strtstop <= '0';
    wait;
end process stimulus;

end testbench_arch;

```

We wrote these code after created the test bench

```

stimulus: process
begin
    --Initialize Inputs
    reset <= '1';
    strtstop <= '1';
    --Wait until the Global Set/Reset deasserts
    wait for 100 ns;
    reset <= '0';
    --Wait long enough for the DCM to lock
    wait for 600 ns;
    strtstop <= '0';
    wait;
end process stimulus;

end testbench_arch;

```

## CHAPTER FIVE

### SYNTHESIS THE DESIGN

We are clicking the *stopwatch\_tb.vhd* module in source tab.

And then we are clicking the *synthesize-XST* and *view synthesis report*

Release 9.1i - xst J.30

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

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

CPU : 0.00 / 0.67 s | Elapsed : 0.00 / 1.00 s

--> Parameter xsthdprdir set to ./xst

CPU : 0.00 / 0.67 s | Elapsed : 0.00 / 1.00 s

--> Reading design: stopwatch.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                      : "stopwatch.prj"  
Input Format                          : mixed  
Ignore Synthesis Constraint File : NO

---- Target Parameters

Output File Name                    : "stopwatch"  
Output Format                        : NGC  
Target Device                        : xc2vp2-7-fg256

---- Source Options

Top Module Name                    : stopwatch  
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 : stopwatch.lso  
Keep Hierarchy : NO  
RTL Output : Yes  
Global Optimization : AllClockNets  
Read Cores : YES  
Write Timing Constraints : NO  
Cross Clock Analysis : NO  
Hierarchy Separator : /  
Bus Delimiter : <>  
Case Specifier : maintain  
Slice Utilization Ratio : 100  
BRAM Utilization Ratio : 100  
Verilog 2001 : YES  
Auto BRAM Packing : NO



Slice Utilization Ratio Delta : 5



=====

=====

\* HDL Compilation \*

=====

=====

Compiling vhdl file "C:/Xilinx91i/xilinx/myprojects/watchvhd/smallcntr.vhd" in Library work.

Entity <smallcntr> compiled.

Entity <smallcntr> (Architecture <inside>) compiled.

Compiling vhdl file "C:/Xilinx91i/xilinx/myprojects/watchvhd/statmach.vhd" in Library work.

Entity <STATMACH> compiled.

Entity <STATMACH> (Architecture <BEHAVIOR>) compiled.

Compiling vhdl file "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" in Library work.

Entity <DCM1> compiled.

Entity <DCM1> (Architecture <BEHAVIORAL>) compiled.

Compiling vhdl file "C:/Xilinx91i/xilinx/myprojects/watchvhd/decode.vhd" in Library work.

Entity <decode> compiled.

Entity <decode> (Architecture <behavioral>) compiled.

Compiling vhdl file "C:/Xilinx91i/xilinx/myprojects/watchvhd/cnt60.vhd" in Library work.

Entity <cnt60> compiled.

Entity <cnt60> (Architecture <inside>) compiled.

Compiling vhdl file "C:/Xilinx91i/xilinx/myprojects/watchvhd/hex2led.vhd" in Library work.

Entity <hex2led> compiled.

Entity <hex2led> (Architecture <Behavioral>) compiled.

Compiling vhdl file "C:/Xilinx91i/xilinx/myprojects/watchvhd/stopwatch.vhd" in

Library work.

Entity <stopwatch> compiled.

Entity <stopwatch> (Architecture <inside>) compiled.

=====

=====

\* Design Hierarchy Analysis \*

=====

=====

Analyzing hierarchy for entity <stopwatch> in library <work> (architecture <inside>).

Analyzing hierarchy for entity <statmach> in library <work> (architecture  
<BEHAVIOR>).

Analyzing hierarchy for entity <dcm1> in library <work> (architecture  
<BEHAVIORAL>).

Analyzing hierarchy for entity <decode> in library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <cnt60> in library <work> (architecture <inside>).

Analyzing hierarchy for entity <hex2led> in library <work> (architecture  
<Behavioral>).

Analyzing hierarchy for entity <smallcntr> in library <work> (architecture <inside>).

=====

=====

\* HDL Analysis \*

=====

=====

Analyzing Entity <stopwatch> in library <work> (Architecture <inside>).

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/stopwatch.vhd" line 95: Unconnected output port 'CLKIN\_IBUFG\_OUT' of component 'dcm1'.

WARNING:Xst:2211 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/stopwatch.vhd" line 104: Instantiating black box module <tenths>.

WARNING:Xst:37 - Unknown property "fpga\_dont\_touch".

Entity <stopwatch> analyzed. Unit <stopwatch> generated.

Analyzing Entity <statmach> in library <work> (Architecture <BEHAVIOR>).

Entity <statmach> analyzed. Unit <statmach> generated.

Analyzing Entity <dcm1> in library <work> (Architecture <BEHAVIORAL>).

WARNING:Xst:2211 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 92: Instantiating black box module <IBUFG>.

WARNING:Xst:2211 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 96: Instantiating black box module <BUFG>.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'CLK90' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'CLK180' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'CLK270' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'CLKDV' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'CLK2X' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'CLK2X180' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'CLKFX' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'CLKFX180' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100: Unconnected output port 'STATUS' of component 'DCM'.

WARNING:Xst:753 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line 100:  
Unconnected output port 'PSDONE' of component 'DCM'.  
WARNING:Xst:2211 - "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd" line  
100: Instantiating black box module <DCM>.

Set user-defined property "CLK\_FEEDBACK = 1X" for instance <DCM\_INST> in  
unit <dcm1>.

Set user-defined property "CLKDV\_DIVIDE = 2.0000000000000000" for instance  
<DCM\_INST> in unit <dcm1>.

Set user-defined property "CLKFX\_DIVIDE = 1" for instance <DCM\_INST> in unit  
<dcm1>.

Set user-defined property "CLKFX\_MULTIPLY = 4" for instance <DCM\_INST> in  
unit <dcm1>.

Set user-defined property "CLKIN\_DIVIDE\_BY\_2 = FALSE" for instance  
<DCM\_INST> in unit <dcm1>.

Set user-defined property "CLKIN\_PERIOD = 20.0000000000000000" for instance  
<DCM\_INST> in unit <dcm1>.

Set user-defined property "CLKOUT\_PHASE\_SHIFT = NONE" for instance  
<DCM\_INST> in unit <dcm1>.

Set user-defined property "DESKEW\_ADJUST = SYSTEM\_SYNCHRONOUS" for  
instance <DCM\_INST> in unit <dcm1>.

Set user-defined property "DFS\_FREQUENCY\_MODE = LOW" for instance  
<DCM\_INST> in unit <dcm1>.

Set user-defined property "DLL\_FREQUENCY\_MODE = LOW" for instance  
<DCM\_INST> in unit <dcm1>.

Set user-defined property "DSS\_MODE = NONE" for instance <DCM\_INST> in  
unit <dcm1>.

Set user-defined property "DUTY\_CYCLE\_CORRECTION = TRUE" for instance  
<DCM\_INST> in unit <dcm1>.

Set user-defined property "FACTORY\_JF = C080" for instance <DCM\_INST> in  
unit <dcm1>.

Set user-defined property "PHASE\_SHIFT = 0" for instance <DCM\_INST> in unit  
<dcm1>.

Set user-defined property "STARTUP\_WAIT = TRUE" for instance <DCM\_INST>  
in unit <dcm1>.



Entity <dcml> analyzed. Unit <dcml> generated.

Analyzing Entity <decode> in library <work> (Architecture <behavioral>).

Entity <decode> analyzed. Unit <decode> generated.

Analyzing Entity <cnt60> in library <work> (Architecture <inside>).

Entity <cnt60> analyzed. Unit <cnt60> generated.

Analyzing Entity <smallcntr> in library <work> (Architecture <inside>).

Entity <smallcntr> analyzed. Unit <smallcntr> generated.

Analyzing Entity <hex2led> in library <work> (Architecture <Behavioral>).

Entity <hex2led> analyzed. Unit <hex2led> generated.

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

Performing bidirectional port resolution...

Synthesizing Unit <statmach>.

Related source file is "C:/Xilinx91i/xilinx/myprojects/watchvhd/statmach.vhd".

Found finite state machine <FSM\_0> for signal <sreg>.

```
-----
| States      | 6          |
| Transitions | 15         |
| Inputs      | 2          |
| Outputs     | 2          |
| Clock       | CLK (rising_edge) |
| Reset       | reset (positive) |
| Reset type  | asynchronous |
```



Reset State	000	
Encoding	automatic	
Implementation	LUT	

---

Summary:

inferred 1 Finite State Machine(s).

Unit <statmach> synthesized.

Synthesizing Unit <decode>.

Related source file is "C:/Xilinx91i/xilinx/myprojects/watchvhd/decode.vhd".

Found 16x10-bit ROM for signal <one\_hot>.

Summary:

inferred 1 ROM(s).

Unit <decode> synthesized.

Synthesizing Unit <hex2led>.

Related source file is "C:/Xilinx91i/xilinx/myprojects/watchvhd/hex2led.vhd".

Found 16x7-bit ROM for signal <LED>.

Summary:

inferred 1 ROM(s).

Unit <hex2led> synthesized.

Synthesizing Unit <smallcntr>.

Related source file is "C:/Xilinx91i/xilinx/myprojects/watchvhd/smallcntr.vhd".

Found 4-bit up counter for signal <qoutsig>.

Summary:

inferred 1 Counter(s).

Unit <smallcntr> synthesized.

Synthesizing Unit <dcm1>.

Related source file is "C:/Xilinx91i/xilinx/myprojects/watchvhd/DCM1.vhd".  
Unit <dcm1> synthesized.

Synthesizing Unit <cnt60>.

Related source file is "C:/Xilinx91i/xilinx/myprojects/watchvhd/cnt60.vhd".  
Unit <cnt60> synthesized.

Synthesizing Unit <stopwatch>.

Related source file is "C:/Xilinx91i/xilinx/myprojects/watchvhd/stopwatch.vhd".  
Unit <stopwatch> synthesized.

=====

=====

## HDL Synthesis Report

### Macro Statistics

# ROMs	: 3
16x10-bit ROM	: 1
16x7-bit ROM	: 2
# Counters	: 2
4-bit up counter	: 2

=====

=====

\* Advanced HDL Synthesis \*

=====

=====

Analyzing FSM <FSM\_0> for best encoding.

Optimizing FSM <MACHINE/sreg> on signal <sreg[1:3]> with sequential encoding.

-----  
State | Encoding  
-----

000 | 000

001 | 010

010 | 100

011 | 011

100 | 101

101 | 001  
-----

Loading device for application Rf\_Device from file '2vp2.nph' in environment  
C:\Xilinx91i.

Executing edif2ngd -noa "tenths.edn" "tenths.ngo"

Release 9.1i - edif2ngd J.30

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

INFO:NgdBuild - Release 9.1i edif2ngd J.30

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

Writing module to "tenths.ngo"...

Loading core <tenths> for timing and area information for instance <XCOUNTER>.

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

Macro Statistics

# FSMs	: 1
# ROMs	: 3
16x10-bit ROM	: 1
16x7-bit ROM	: 2
# Counters	: 2
4-bit up counter	: 2
# Registers	: 3

Flip-Flops

: 3

=====  
=====

=====  
=====

\* Low Level Synthesis \*

=====  
=====

Optimizing unit <stopwatch> ...

Mapping all equations...

Building and optimizing final netlist ...

Found area constraint ratio of 100 (+ 5) on block stopwatch, actual ratio is 2.

Final Macro Processing ...

=====  
=====

Final Register Report

Macro Statistics

# Registers : 11

Flip-Flops : 11

=====  
=====

=====  
=====

\* Partition Report \*

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

No Partitions were found in this design.  
-----

=====  
=====  
\* Final Report \*

=====  
Final Results

RTL Top Level Output File Name : stopwatch.ngr

Top Level Output File Name : stopwatch

Output Format : NGC

Optimization Goal : Speed

Keep Hierarchy : NO

Design Statistics

# IOs : 27

Cell Usage :

# BELS : 68

# GND : 2

# INV : 2

# LUT3 : 6

# LUT4 : 47

# MUXCY : 3

# MUXF5 : 4

# XORCY : 4



# FlipFlops/Latches	: 17
# FDC	: 3
# FDCE	: 12
# FDE	: 1
# FDPE	: 1
# Clock Buffers	: 1
# BUFG	: 1
# IO Buffers	: 27
# IBUF	: 2
# IBUFG	: 1
# OBUF	: 24
# DCMs	: 1
# DCM	: 1

=====

=====

#### Device utilization summary:

-----

Selected Device : 2vp2fg256-7

Number of Slices:	31 out of 1408	2%
Number of Slice Flip Flops:	17 out of 2816	0%
Number of 4 input LUTs:	55 out of 2816	1%
Number of IOs:	27	
Number of bonded IOBs:	27 out of 140	19%
Number of GCLKs:	1 out of 16	6%
Number of DCMs:	1 out of 4	25%

-----

#### 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
-----+-----+-----+		
CLK	Inst_dcm1/DCM_INST:CLK0	17
-----+-----+-----+		

### Asynchronous Control Signals Information:

-----

-----+-----+-----+		
Control Signal	Buffer(FF name)	Load
-----+-----+-----+		
rstint(MACHINE/sreg_Out01:O)	NONE(sixty/lsbcount/qoutsig_1)	9
RESET	IBUF	3
sixty/msbclr(sixty/msbclr:O)	NONE(sixty/msbcount/qoutsig_1)	4
-----+-----+-----+		

### Timing Summary:

-----

Speed Grade: -7

Minimum period: 3.509ns (Maximum Frequency: 285.006MHz)

Minimum input arrival time before clock: 2.228ns

Maximum output required time after clock: 4.226ns

Maximum combinational path delay: No path found

Timing Detail:

-----

All values displayed in nanoseconds (ns)

=====

Timing constraint: Default period analysis for Clock 'CLK'

Clock period: 3.509ns (frequency: 285.006MHz)

Total number of paths / destination ports: 182 / 31

-----

Delay: 3.509ns (Levels of Logic = 4)

Source: MACHINE/sreg\_FFd3 (FF)

Destination: XCOUNTER/BU28 (FF)

Source Clock: CLK rising

Destination Clock: CLK rising

Data Path: MACHINE/sreg\_FFd3 to XCOUNTER/BU28

	Gate	Net			
Cell:in->out	fanout	Delay	Delay	Logical Name (Net Name)	
-----					
FDC:C->Q	7	0.370	0.601	MACHINE/sreg_FFd3	
(MACHINE/sreg_FFd3)					
LUT3:I0->O	4	0.275	0.511	MACHINE/sreg_Out11 (clkenable)	
begin scope: 'XCOUNTER'					
LUT4:I1->O	5	0.275	0.526	BU3 (N7)	
LUT4:I1->O	4	0.275	0.413	BU20 (N119)	
FDPE:CE		0.263		BU28	
-----					

Total 3.509ns (1.458ns logic, 2.051ns route)  
(41.6% logic, 58.4% route)

Timing constraint: Default OFFSET IN BEFORE for Clock 'CLK'

Total number of paths / destination ports: 5 / 3

Offset: 2.228ns (Levels of Logic = 3)

Source: STRTSTOP (PAD)

Destination: MACHINE/sreg\_FFd3 (FF)

Destination Clock: CLK rising

Data Path: STRTSTOP to MACHINE/sreg\_FFd3

	Gate	Net			
Cell:in->out	fanout	Delay	Delay	Logical Name	(Net Name)
IBUF:I->O	5	0.878	0.564	STRTSTOP_IBUF	(STRTSTOP_IBUF)
LUT3:I0->O	1	0.275	0.000	MACHINE/sreg_FFd3-In_G	(N89)
MUXF5:I1->O	1	0.303	0.000	MACHINE/sreg_FFd3-In	
(MACHINE/sreg_FFd3-In)					
FDC:D		0.208		MACHINE/sreg_FFd3	

Total 2.228ns (1.664ns logic, 0.564ns route)  
(74.7% logic, 25.3% route)

Timing constraint: Default OFFSET OUT AFTER for Clock 'CLK'

Total number of paths / destination ports: 96 / 24

Offset: 4.226ns (Levels of Logic = 3)

Source: XCOUNTER/BU36 (FF)

Destination: TENTHSOUT<9> (PAD)

Source Clock: CLK rising

Data Path: XCOUNTER/BU36 to TENTHSOUT<9>

	Gate	Net			
Cell:in->out	fanout	Delay	Delay	Logical Name	(Net Name)
-----					
FDCE:C->Q	11	0.370	0.657	BU36 (Q<1>)	
end scope: 'XCOUNTER'					
LUT4:I0->O	1	0.275	0.332	TENTHSOUT<9>1 (TENTHSOUT_9_OBUF)	
OBUF:I->O	2.592			TENTHSOUT_9_OBUF (TENTHSOUT<9>)	
-----					
Total		4.226ns (3.237ns logic, 0.989ns route)			
		(76.6% logic, 23.4% route)			

=====

=====

CPU : 10.74 / 11.56 s | Elapsed : 11.00 / 12.00 s

-->

Total memory usage is 169048 kilobytes

Number of errors : 0 ( 0 filtered)

Number of warnings : 16 ( 0 filtered)

Number of infos : 0 ( 0 filtered)



## CHAPTER SIX

### VIEW RTL SCHEMATIC

To view RTL schematic we are selecting as follows

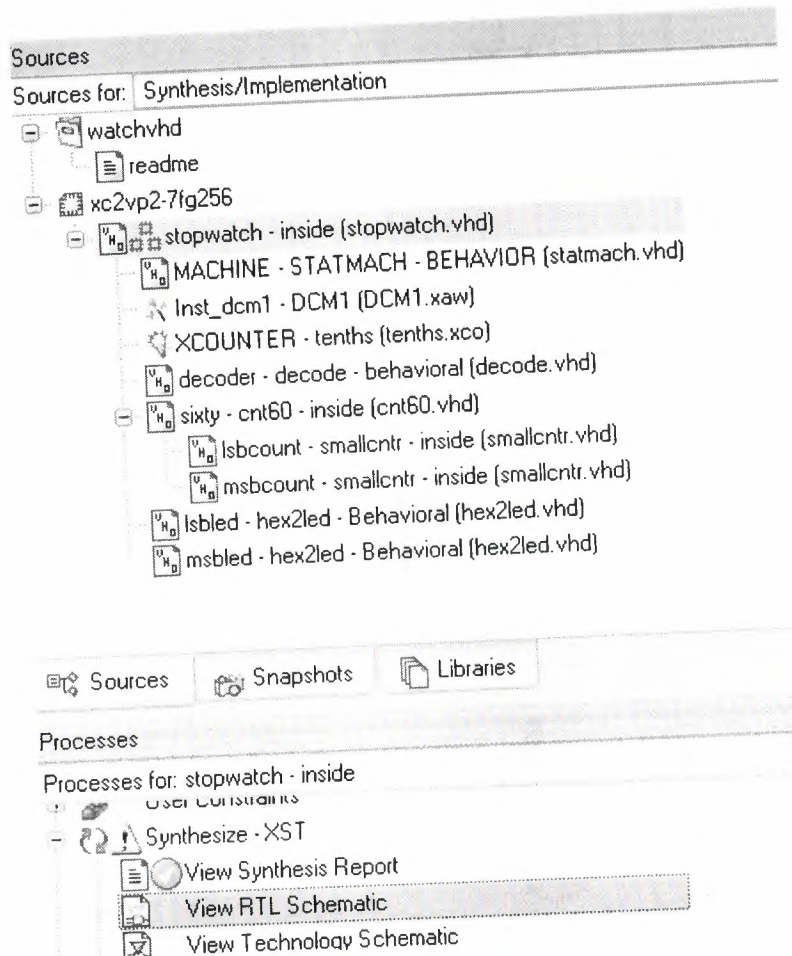


Figure 34

View rtl schematic



## CHAPTER SEVEN

### SIMULATION OF DESIGN

1. Select *behavioral simulation* on source tab.
2. And select the stopwatch\_tb.vhd source tab.
3. Select xilinx ise Simulator on process tab.
4. And click simulate behavioral model.

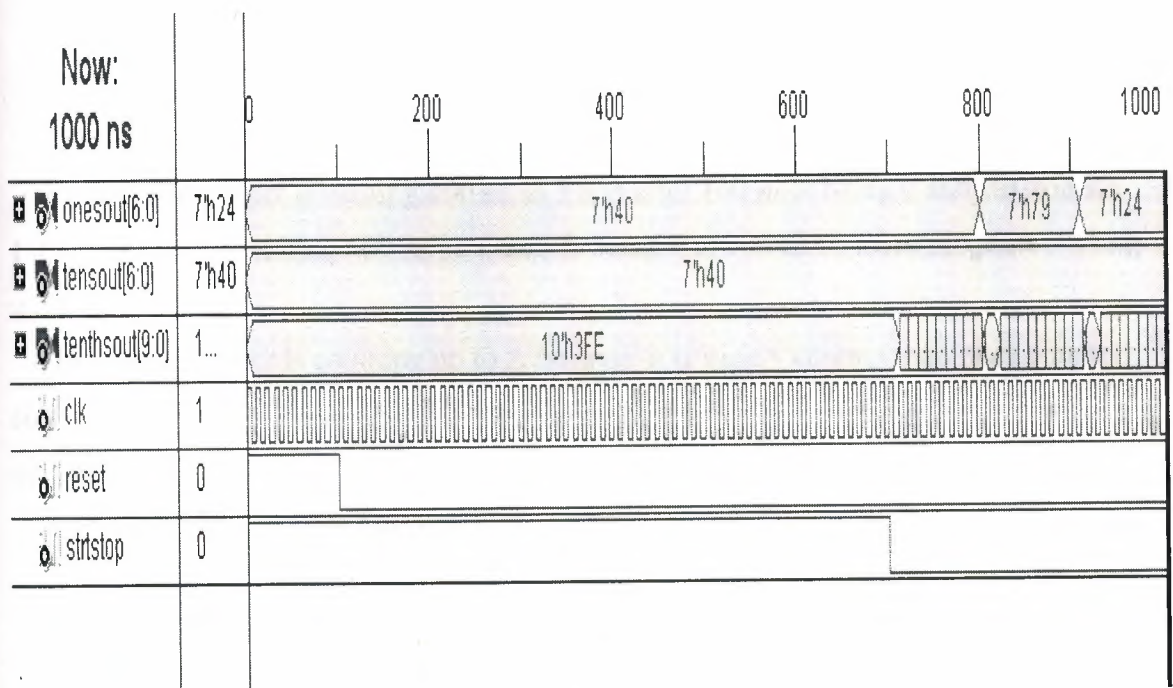


Figure 36 It's continue

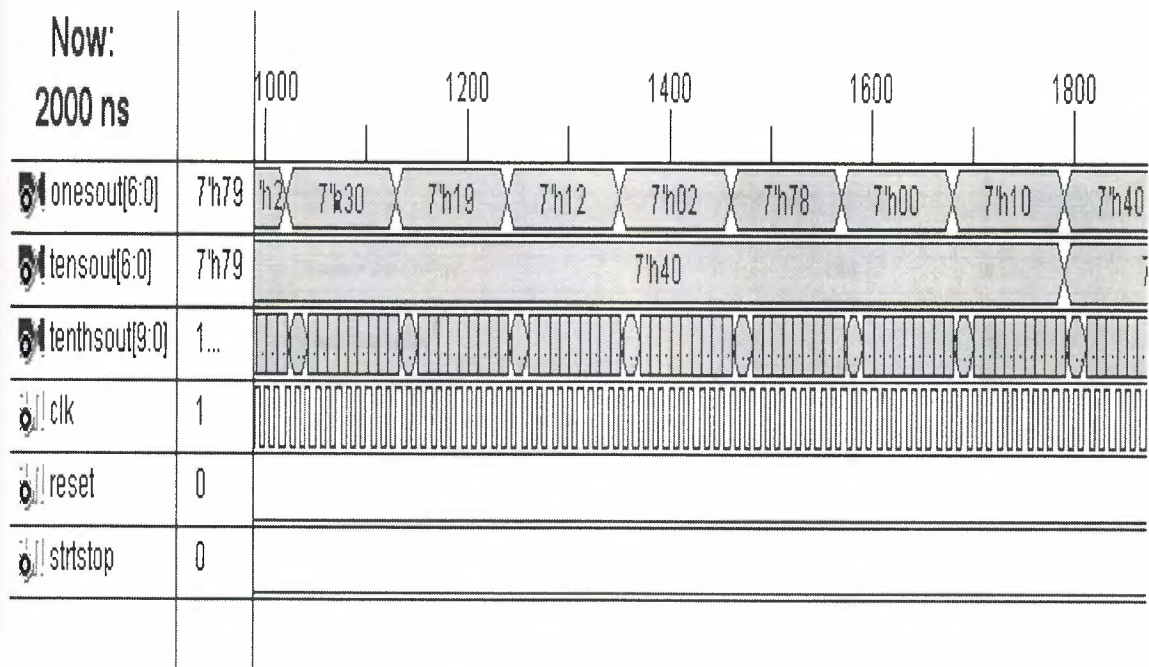


Figure 37-simulation

We are seeing reset is waiting 100 ns as 1 and after 100 ns is being 0 and strtstop stil 1. Then strtstop is waiting 600 ns as 1 then is being 0 at 700 ns so main program starting working.

The tenths counter is counting up to A because it is binary counter then ones counter starts count and it is counting up to nine then tens counter is starting then it is counting up to six.

Program is working that I hope.

After the simulation i did implement the design as follow.

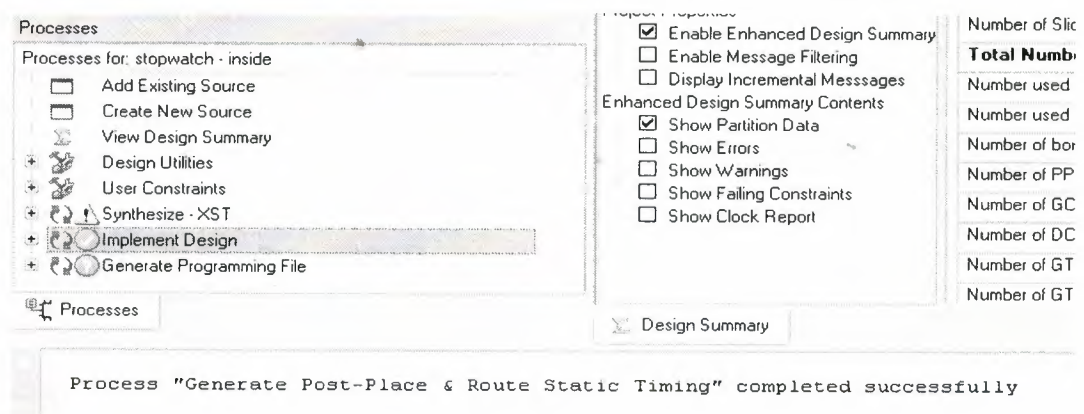


Figure38-implement the design



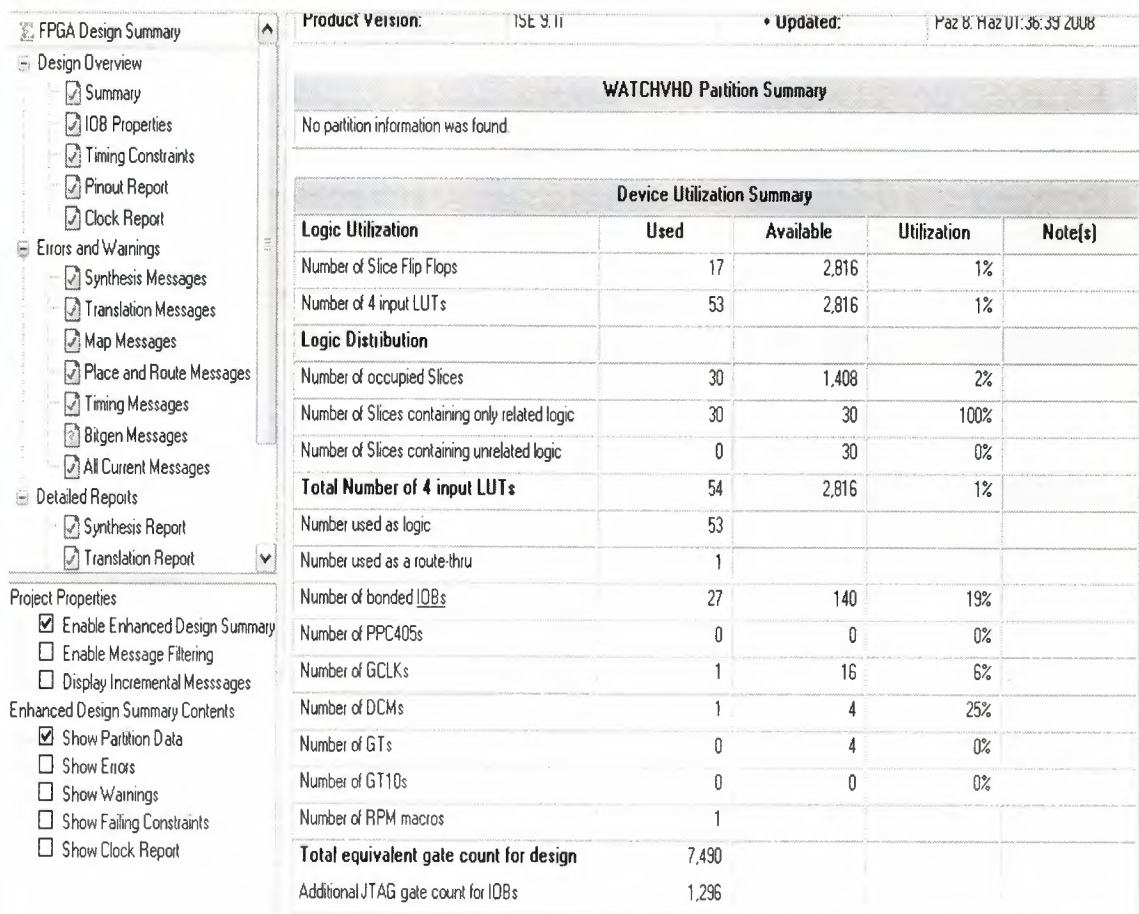


Figure39-device utilization summary

After implement the design I did generate the design as follow.

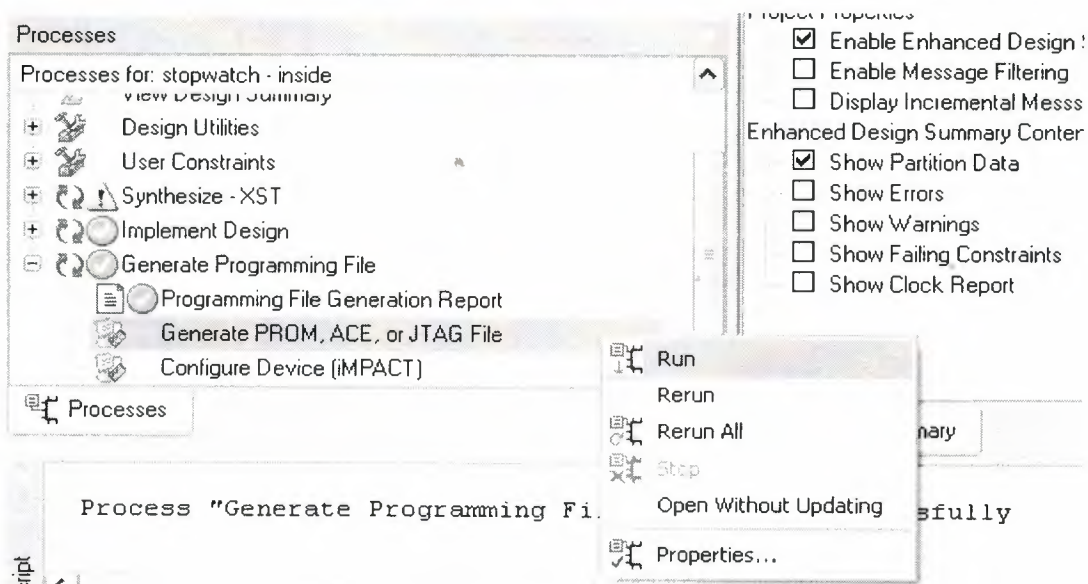


Figure40-implement the design



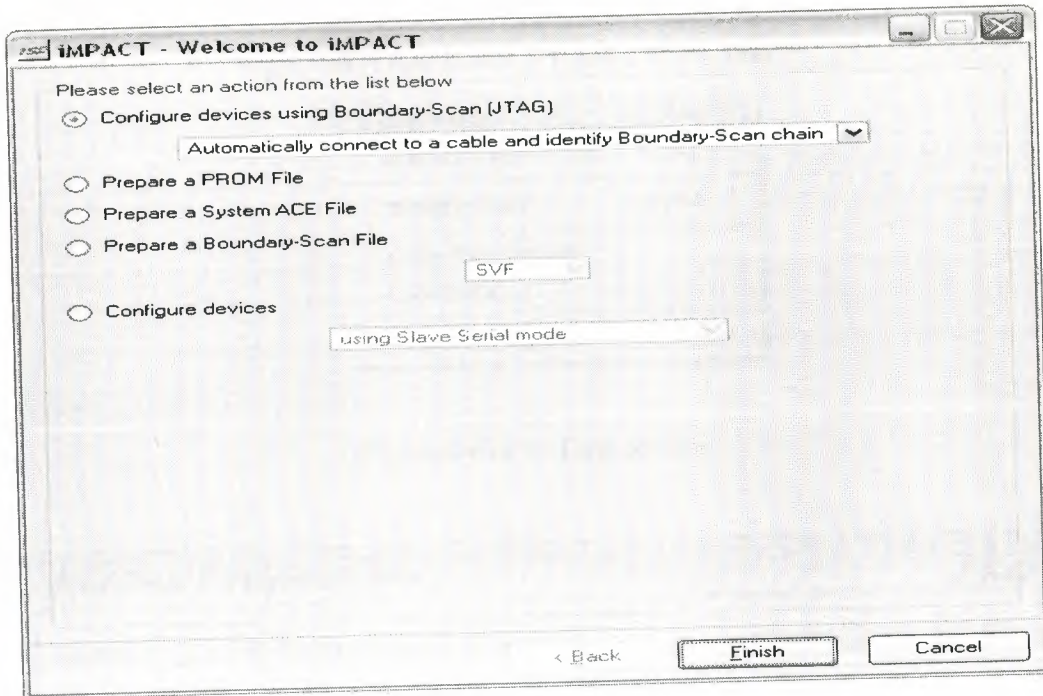


Figure41-select as figure and click finish

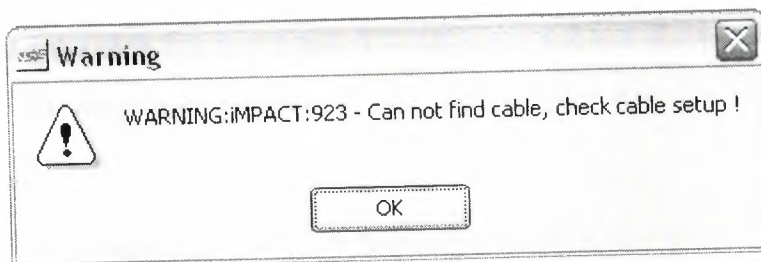


Figure42-warning

We are seeing this warning because of we have no hardware and cable I clicked ok.

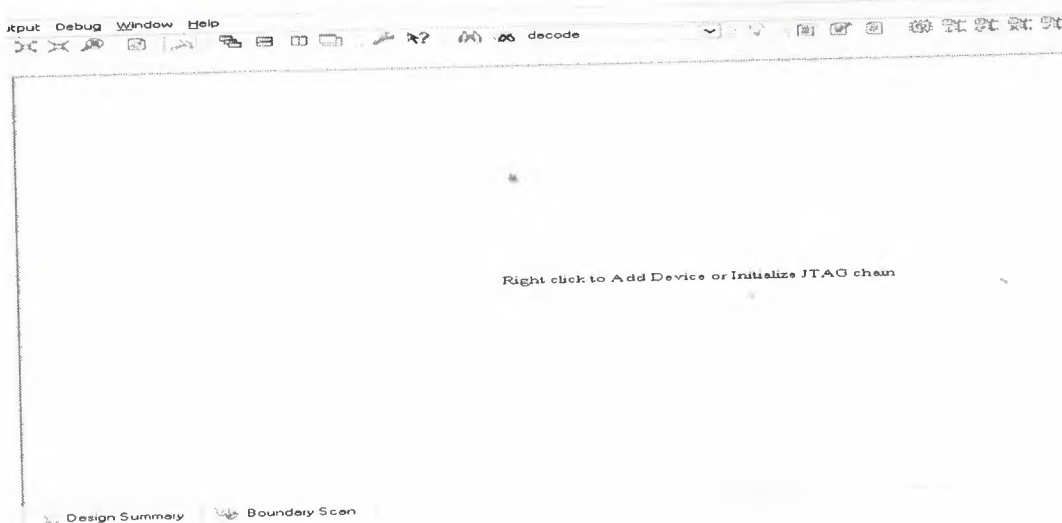


Figure43-right click and add device

Right click to Add Device or Initialize JTAG chain

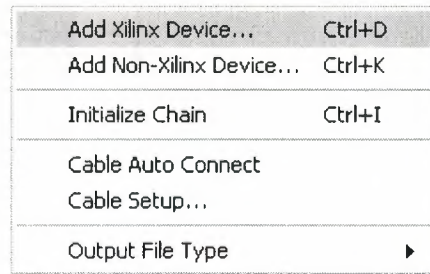


Figure44-add xilinx device

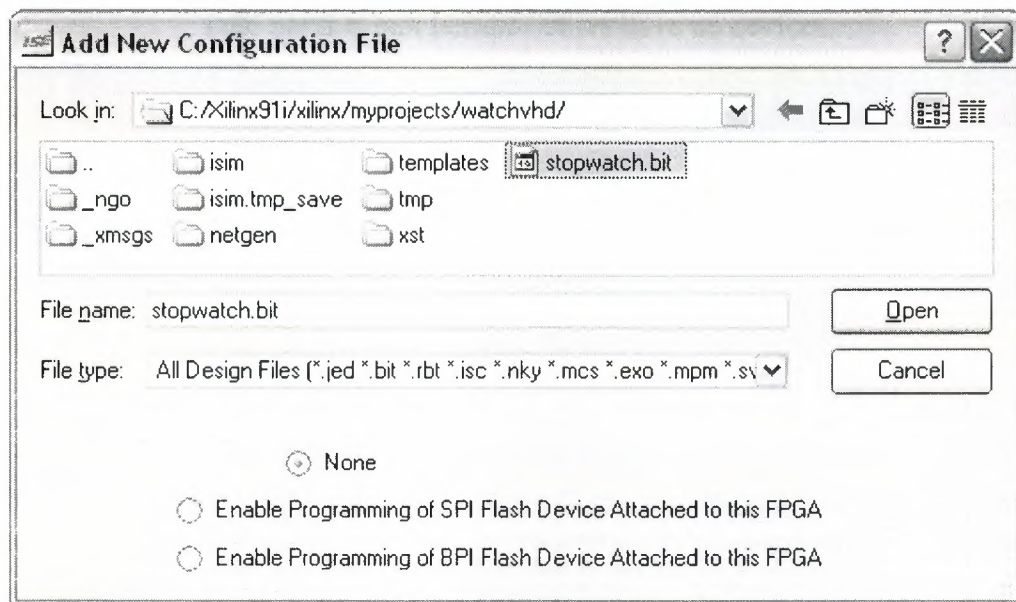


Figure45-select stopwatch.bit then click open

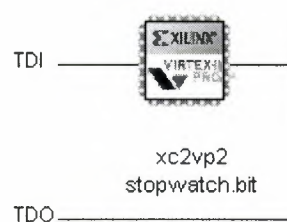
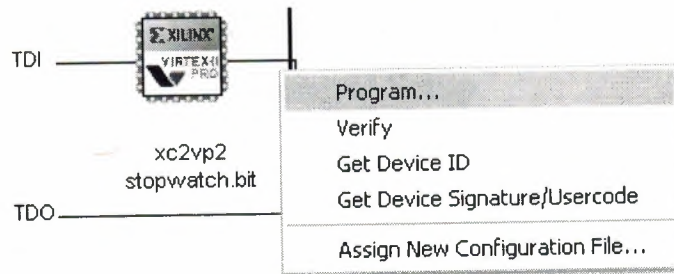


Figure46-stopwatch.bit



*Figure47-program*

I did not this part because of we have no devices.

## CHAPTER EIGHT

### ABOUT DCM CLOCKING WIZARD

#### General Setup

Use the General Setup dialog box to configure the Digital Clock Manager (DCM).

The primitive the Architecture Wizard will use to implement the DCM function depends on the FPGA device into which your design will be programmed:

- For Virtex-II, Virtex-II Pro and Spartan-3/3L devices, a DCM primitive will be used.
- For Spartan-3E/3A devices, a DCM\_SP primitive will be used.

#### 8.1.DCM PORTS

Each selectable pin has a corresponding checkbox to enable or disable the pin.

##### CLKIN

This pin is the clock input to the DCM. CLKIN is always enabled. CLKIN provides the source clock to the DCM. The frequency is specified in the **Input Clock Frequency** box. The source is specified in the **CLKIN Source** box.

##### CLKFB

This pin is the clock feedback input to the DCM. The selection in the **Feedback Source** box determines if CLKFB is enabled. A DCM requires a reference or feedback signal to provide delay-compensated output.

##### RST

This pin is the reset input to the DCM. RST is enabled when the checkbox is selected. If not enabled, RST will be tied to GND, and the DCM can only be reset upon configuration. For details about what occurs when RST is enabled, refer to



the "Using Digital Clock Managers (DCM)" section in the "Design Considerations" chapter of the relevant FPGA User Guide (links provided above).

## **PSEN**

This pin is the phase shift enable input to the DCM. PSEN is enabled when **Phase Shift Type** is set to **Variable**. When not enabled, PSEN is tied to GND.

To initiate variable phase shift operation, the PSEN input must be activated for one period of PSCLK. The phase change becomes effective after up to 100 CLKIN pulse cycles plus three PSCLK cycles, and is indicated by a High pulse on PSDONE. During the phase transition there are no sporadic changes or glitches on any output.

## **PSINCDEC**

This pin is the phase shift increment/decrement input to the DCM. PSINCDEC is enabled when **Phase Shift Type** is set to **Variable**. When not enabled, PSINCDEC is tied to GND.

The PSINCDEC signal is synchronous to PSCLK and is used to increment or decrement the phase shift factor. To increment or decrement the phase shift, the PSINCDEC signal must be High for increment, or Low for decrement.

For Virtex-II, Virtex-II Pro and Spartan-3/3L devices, each increment or decrement will move the clock phase by 1/256 of clock period. For Spartan-3E and Spartan-3A devices, each increment or decrement will correspond to approximately 25 ps.

## **PSCLK**

This pin is the phase shift clock input to the DCM. PSCLK is enabled when **Phase Shift Type** is set to **Variable**. When not enabled, PSCLK is tied to GND.

The PSCLK input can be sourced by the CLKIN signal to the DCM, or it can be a lower or higher frequency signal provided from any clock source (external or



internal). The frequency range of PSCLK is defined by PSCLK\_FREQ\_LF / PSCLK\_FREQ\_HF. For more information, see the product [Data Sheets](#).

### **CLK0 / CLK90 / CLK180 / CLK270**

These output DCM pins provide coarse phase shifting. Each of these outputs is enabled when its checkbox is selected.

The CLK0, CLK90, CLK180, and CLK270 outputs are each phase shifted by 1/4 of the input clock period relative to each other.

**Note** CLK90 and CLK270 are not available in high-frequency mode.

### **CLKDV**

This pin is the clock divide output of the DCM. CLKDV is enabled when the checkbox is selected.

The CLKDV output provides divided output clocks and the options are available in the **Divide By Value** list.

### **CLK2X / CLK2X180**

The CLK2X and CLK2X180 output pins double the clock frequency. Each of these outputs is enabled when its checkbox is selected. CLK2X180 is the opposite phase of CLK2X.

**Note** CLK2X and CLK2X180 are not available in high-frequency mode.

### **CLKFX / CLKFX180**

The CLKFX and CLKFX180 output pins provide fully digital, dedicated frequency synthesizer output to the DCM. Each of these outputs is enabled when its checkbox is selected. CLKFX180 is the opposite phase of CLKFX.

The output frequency is a function of the input clock frequency described by M and D, where M is the multiplier (numerator), and D is the divisor (denominator). M and D can be specified in the Clock Frequency Synthesizer dialog box. The Clock Frequency Synthesizer dialog box appears when CLKFX or CLKFX180 is enabled.

## STATUS

The STATUS pin is an 8-bit output bus from the DCM. STATUS is enabled when the checkbox is selected.

Only bits 0 to 2 are defined. Bits 3 to 7 are connected to a floating signal; this signal is optimized during synthesis.

STATUS[0] indicates the overflow of the phase shift numerator and that the absolute delay range of the phase shift delay line is exceeded. STATUS[1] indicates the loss of the input clock, CLKIN, to the DCM. STATUS[2] indicates that CLKFX has stopped.

## LOCKED

The LOCKED pin is an output to the DCM that activates after the DCM has achieved lock. LOCKED is enabled when the checkbox is selected.

To achieve lock, the DCM may need to sample several thousand clock cycles. After the DCM achieves lock, the LOCKED signal goes high. To guarantee that the system clock is established prior to the device waking up, the DCM can be set to delay the completion of the device. The STARTUP\_WAIT attribute activates this feature. Until the LOCKED signal activates, the DCM output clocks are not valid and can exhibit glitches, spikes, or other spurious movement.

## PSDONE

The PSDONE pin is the phase shift done output of the DCM. PSDONE is enabled when **Phase Shift Type** is set to **Variable**.

The PSDONE signal is synchronous to PSCLK and it indicates, by pulsing high for one period of PSCLK, that the requested phase shift was achieved. This signal also indicates that a new change to the phase shift numerator can be made. This output signal is not valid if the phase shift feature is not being used or is in FIXED mode.

## **Input Clock Frequency**

The input clock frequency determines the value of the DLL\_FREQUENCY\_MODE and DFS\_FREQUENCY\_MODE attributes for the DCM. The Clocking Wizard accepts values that fall within the ranges specified in the product Data Sheets. The default selection is low frequency. When the DLL\_FREQUENCY\_MODE attribute is set to HIGH, the only outputs available from the DCM are CLK0, CLK180, CLKDV and LOCKED.

**Note** The Clocking Wizard checks that each output can operate in the valid range. To check the valid range for each individual output, see the product Data Sheets.

**Note** The DLL\_FREQUENCY\_MODE and DFS\_FREQUENCY\_MODE attributes do not apply to the Spartan-3E and Spartan-3A architectures. In these architectures, the DCM function uses a DCM\_SP primitive instead of a DCM primitive.

The input clock frequency affects the CLKIN\_PERIOD attribute. This attribute is always set by the Clocking Wizard, but is used by BitGen (the bitstream generation program) only when the CLKFX / CLKFX180 output is used. This attribute does not affect any timing constraints.

## **FREQUENCY IS DIVIDED BY 2**

This text alerts you that the **Divide Input Clock by 2** option is selected in the Advanced dialog box (accessed by clicking the **Advanced** button below).

## **CLKIN Source**

CLKIN provides the source clock to the DCM.

## External

The Clocking Wizard connects a dedicated input buffer (IBUFG or IBUFGDS) to the CLKIN input pin. The output of the IBUFG or IBUFGDS is also brought out as a port. This is done to give you the ability to connect the output of the IBUFG or IBUFGDS to other clock components. For example, when a DCM is used with RocketIO, the output immediately after the IBUFGDS is required as the input for the BREFCLK or BREFCLK2 pin of the RocketIO transceiver (when using the BREFCLK pins to input the reference clock).

The default selection for **CLKIN Source** is **External**.

### Single

An IBUFG is instantiated in the module generated by the Clocking Wizard. This is the default selection when **CLKIN Source** is set to **External**.

### Differential

An IBUFGDS is instantiated in the module generated by the Clocking Wizard.

For more detailed information about these buffers, see the *Libraries Guides*, available from the Software Manuals collection.

## Internal

The Clocking Wizard connects the CLKIN input pin directly to the CLKIN pin on the DCM. The only components that can drive the CLKIN on the DCM are dedicated clock I/O, regular I/O, or clock buffers. When this DCM instantiation is placed into a design, you must connect the CLKIN pin to one of those components. Select **Internal** if you do not want an IBUFG to be inserted for CLKIN.

## Divide By Value



The value selected is the clock division factor associated with the CLKDV output pin. This selection is available only when the **CLKDV** checkbox is selected.

### **Feedback Source**

Specifies the feedback source to the CLKFB input of the DCM.

**Note** When either Internal or External is selected, a BUFG is connected to the DCM clock output used as the feedback (CLK0). This information is displayed in the table found in the Clock Buffers dialog box; click **Next** to access this dialog box. For any other connection, select **Customize buffers** in the Clock Buffers dialog box. You can also generate a board level clock by using a DDR register in the Clock Forwarding/Board Deskew flow.

#### **External**

The Clocking Wizard instantiates a global clock input buffer and connects it to CLKFB. The type of buffer is determined by the selection of **Single** or **Differential**.

##### **Single**

An IBUFG is instantiated in the module generated by the Clocking Wizard. This is the default selection when **Feedback Source** is set to **External**.

##### **Differential**

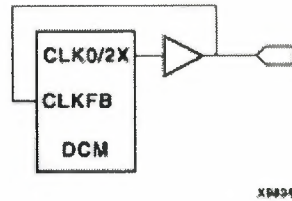
An IBUFGDS is instantiated in the module generated by the Clocking Wizard.

For more detailed information about these buffers, see the *Libraries Guides*, available from the [Software Manuals](#) collection.

#### **Internal**

The Clocking Wizard instantiates a global buffer (BUFG) and connects it as shown in the figure. The default selection for **Feedback Source** is **Internal**.





**None**

This is valid only when CLKFX or CLKFX180 output pin is used.

### Feedback Value

Specifies the feedback value for the DCM.

**1X**

The CLK0 output provides the feedback clock. This is the default selection.

**2X**

The CLK2X output provides the feedback clock. This output is not available in high-frequency mode.

**Note** 2X is not available for Virtex-II Pro devices.

### Use Duty Cycle Correction

This setting enables the 50/50 duty cycle for the 1X clock outputs CLK0, CLK90, CLK180 and CLK270.

### Phase Shift

This controls the fine phase shifting capabilities of the DCM.

**Note** For Spartan-3 devices, phase shift is only supported in Low Frequency mode. If the **Input Clock Frequency** for a Spartan-3 device is set for High Frequency mode the **Phase Shift** selections are disabled, the phase shift **Type** is set to **NONE**, and the phase shift **Value** is set to **0**. This applies to Spartan-3 devices; it does not apply to Spartan-3E and Spartan-3A devices.

## Type

Determines whether fine phase shifting will be used and, if so, determines the type of phase shift employed.

### NONE

This selection disables the phase shift feature.

### FIXED

This selection enables the fine fixed phase shifting.

### VARIABLE

This selection enables the variable fine phase shifting. When selected, PSEN, PSINCDEC, PSCLK, and PSDONE are enabled.

## Value

This setting determines the amount of phase introduced by the DCM. The phase shift value is the numerator in the following equations:

$$\text{Phase Shift (ns)} = (\text{Phase Shift Value}/256) * \text{PERIOD}_{\text{clk}_{\text{in}}}$$

$$\text{Phase Shift (Degrees)} = (\text{Phase Shift Value}/256) * 360$$

The full range of the phase shift value is always -255 to + 255, but its practical range varies with CLKIN frequency, as constrained by the FINE\_SHIFT\_RANGE attribute. For further details, see the product [Data Sheets](#)

## **CHAPTER NINE**

### **ABOUT BINARY COUNTER**

#### **9.1.FEATURES**

- Drop-in module for Virtex™, Virtex-E, Virtex-II, Virtex-II Pro™, Spartan™-II, Spartan-IIE and Spartan-3 FPGAs
- Generates Up, Down and Up/Down Counters
- Supports counts ranging from 2 to 256 bits wide
- Optional load capability
- Optional user programmable threshold outputs
- Optional clock enable and asynchronous and synchronous controls
- Counter increment value can be user defined or supplied externally
- User-programmable count limit
- Incorporates Xilinx Smart-IP™ technology for utmost parameterization and optimum implementation
- Uses relationally placed macro (RPM) mapping and placement technology, for maximum and predictable performance
- For use with v5.1i and later of the Xilinx CORE Generator™ System

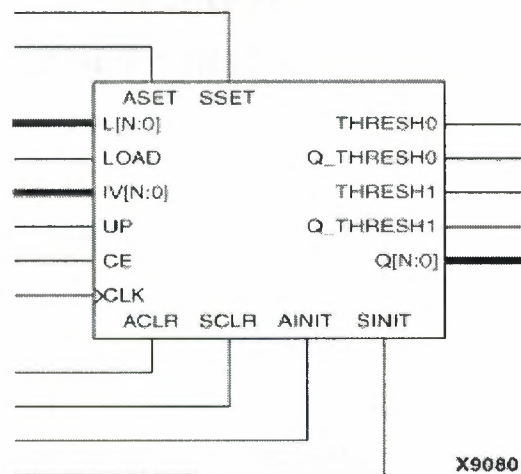


Figure 1: Core Schematic Symbol

## 9.2.FUNCTIONAL DESCRIPTION

The binary counter is used to create up counters, down counters and up/down counters with outputs of up to 256 bits wide. Support is provided for two threshold signals that can be programmed to become active when the counter reaches a user defined count. The upper limit of the count is user programmable, and the counter's increment value can

be user defined or provided via an external port. Options are provided for **Clock Enable**, **Asynchronous Set**, **Clear**, and **Init**, and **Synchronous Set**, **Clear** and **Init**. An optional

**Load** capability is also provided which can load the value on the Load port directly into the output register. The module can optionally be generated as a Relationally Placed Macro

(RPM) or as unplaced logic. When an RPM is generated the logic is placed in a column. When the counter reaches terminal count or the "count to value" the next count will be zero.

## CHAPTER TEN

### ABOUT VIRTEX FPGA CHIPS

#### *The Virtex-II Pro / Virtex-II Pro X*

#### *FPGA Family*

#### **The Next Logical Revolution**

The Virtex-II Pro™/Virtex-II Pro X Platform FPGA solution is the most technically sophisticated silicon and software product development in the history of the programmable logic industry. The goal was to revolutionize system architecture "from the ground up." To achieve that objective, the best circuit engineers and system architects from IBM, Mindspeed, and Xilinx co-developed the world's most advanced Platform FPGA silicon product. Leading teams from top embedded systems companies worked together with Xilinx software teams to develop the systems software and IP solutions that enabled this new system architecture paradigm. The result is the first Platform FPGA solution capable of implementing high performance system-on-a-chip designs previously the exclusive domain of custom ASICs, yet with the flexibility and low development cost of programmable logic. The Virtex-II Pro/Virtex-II Pro X family marks the first paradigm change from programmable logic to programmable systems, with profound implications for leading-edge system architectures in networking applications, deeply embedded systems, and digital signal processing systems. It allows custom user-defined system architectures to be synthesized, next-generation connectivity standards to be seamlessly bridged, and complex hardware and software systems to be co-developed rapidly with in-system debug at system speeds. Together, these capabilities usher in the next programmable logic revolution.

#### **10.1.BUILT FOR BANDWIDTH**

The Virtex-II Pro/Virtex-II Pro X family consists of eleven members. The nine Virtex-II Pro devices contain four to twenty RocketIO™ multi-gigabit transceivers (MGTs), while the two Virtex-II Pro X devices contain eight or twenty RocketIO X MGTs. Each Xilinx RocketIO/RocketIO X transceiver block contains a complete set of user-configurable supporting circuitry that addresses real-life, system-level challenges. These include standard 8B/10B encode/decode (plus 64B/66B encode/decode in the RocketIO X), programmable signal integrity adjustments for varying PCB trace lengths and materials, support for synchronization of multiple channels, and programmable



## CHAPTER TEN

### ABOUT VIRTEX FPGA CHIPS

#### *The Virtex-II Pro / Virtex-II Pro X*

#### *FPGA Family*

#### **The Next Logical Revolution**

The Virtex-II Pro™/Virtex-II Pro X Platform FPGA solution is the most technically sophisticated silicon and software product development in the history of the programmable logic industry. The goal was to revolutionize system architecture "from the ground up." To achieve that objective, the best circuit engineers and system architects from IBM, Mindspeed, and Xilinx co-developed the world's most advanced Platform FPGA silicon product. Leading teams from top embedded systems companies worked together with Xilinx software teams to develop the systems software and IP solutions that enabled this new system architecture paradigm. The result is the first Platform FPGA solution capable of implementing high performance system-on-a-chip designs previously the exclusive domain of custom ASICs, yet with the flexibility and low development cost of programmable logic. The Virtex-II Pro/Virtex-II Pro X family marks the first paradigm change from programmable logic to programmable systems, with profound implications for leading-edge system architectures in networking applications, deeply embedded systems, and digital signal processing systems. It allows custom user-defined system architectures to be synthesized, next-generation connectivity standards to be seamlessly bridged, and complex hardware and software systems to be co-developed rapidly with insystem debug at system speeds. Together, these capabilities usher in the next programmable logic revolution.

#### **10.1.BUILT FOR BANDWIDTH**

The Virtex-II Pro/Virtex-II Pro X family consists of eleven members. The nine Virtex-II Pro devices contain four to twenty RocketIO™ multi-gigabit transceivers (MGTs), while the two Virtex-II Pro X devices contain eight or twenty RocketIO X MGTs. Each Xilinx RocketIO/RocketIO X transceiver block contains a complete set of user-configurable supporting circuitry that addresses real-life, system-level challenges. These include standard 8B/10B encode/decode (plus 64B/66B encode/decode in the RocketIO X), programmable signal integrity adjustments for varying PCB trace lengths and materials, support for synchronization of multiple channels, and programmable

support for channel control commands. In addition, the RocketIO and RocketIO X blocks are the first FPGAembedded transceivers to reach baud rates of 3.125 Gbps and 10.3125 Gbps, respectively. Four RocketIO transceivers, employing sixteen PCB traces, can be used to support a fullduplex 10 Gbps channel by way of the RocketIO channel-bonding feature—or, a *single* RocketIO X transceiver can implement the same speed with just four PCB traces. This is equivalent to 256 traces of typical LVTTTL buses, or 68 traces of a high-speed, sourcesynchronous parallel LVDS bus. It allows a PCB trace reduction of up to 64X over conventional parallel buses, resulting in significant reductions in PCB complexity and EMI system noise. The RocketIO/RocketIO X technology fulfills higher bandwidth system requirements than currently possible, with cost savings coming from faster time-to-market, reduced printed circuit board (PCB) complexity, and lower component count.

Each of the larger devices incorporates one or two small yet powerful IBM® PowerPC™ 405 processor cores, each capable of more than 300 MHz clock frequency and 420 Dhrystone MIPS. While the processor cores occupy a small area of the die, they provide tremendous system flexibility where they are used. The PowerPC 405 cores are fully embedded within the FPGA fabric, where all processor nodes are controlled by the FPGA routing resources.

This provides the utmost architectural capability, where complex applications may be efficiently divided between high-speed logic implementation and high-flexibility software implementations. For example, a packet processing application using only the FPGA logic today for high-speed packet routing may be augmented to include a slave high-performance processor for exception handling or in-system statistics monitoring. In contrast, using a separate processor externally requires hundreds of additional interface pins, which degrades system performance and significantly increases FPGA I/O requirements and overall board costs.

The Virtex-II Pro/Virtex-II Pro X products are based on the most advanced FPGA fabric available: the Virtex-II™ architecture with IP-Immersion™ technology, which was developed to offer significant improvements in engineering productivity, silicon efficiency, and system flexibility. Unique features common in the Virtex-II Series—consisting of the Virtex-II and Virtex-II Pro families—include powerful SystemIO™ system connectivity solutions, digitally controlled impedance (DCI) technology,



comprehensive clocking solutions, high-speed Active Interconnect™ routing architecture, and bitstream encryption. These features together constitute the most complete Platform FPGA solution available, optimized for high performance system-level applications. The upward compatibility of the Virtex Series of products ensures benefits in engineering productivity, performance, design longevity, and continuing cost reduction.

## 10.2.LEGACY OF LEADERSHIP

Each of the Virtex families of FPGAs has been the most successful programmable product family in its class, starting with the introduction of the original Virtex family in 1998. The Virtex and Virtex-E families were recognized by the industry as the highest technology products available when they were first introduced. The Virtex-II family, which again achieved technology leadership in density, performance, and features, ushered in the era of Platform FPGAs—programmable devices with the system-level capability and performance to implement systems functionality. The Virtex-II Pro family continues the tradition of technology leadership as the most sophisticated Platform FPGA yet, again breaking the technology barrier for the benefit of leading-edge system architects.

The Virtex-II Pro/Virtex-II Pro X family is the first FPGA family to incorporate both serial transceiver technology and a hard processor core within a general-purpose FPGA device. This is significant for new high-bandwidth embedded processing applications such as packet processing, where both high device I/O bandwidth and high performance processor cores are needed together.

The Virtex-II Pro/Virtex-II Pro X devices are the *industry's first FPGAs in a 0.13-micron process*. The nine-layer metal, all-copper, low-k process technology is among the most advanced in the semiconductor industry. The combination of advanced Active Interconnect™ architecture and advanced process technology makes the Virtex-II Pro/Virtex-II Pro X family the highest performance FPGA in the world.

The RocketIO/RocketIO X multi-gigabit MGTs are the highest performance, most complete embedded serial transceivers available. They are user-configurable for up to 3.125 Gbps or 10.3125 Gbps baud rate per channel, which is many times the performance of other embedded transceivers at 1.25 Gbps. Each RocketIO/RocketIO X transceiver provides a complete set of common functionality available in standard

SerDes transceivers. In contrast, "programmable ASSP" products with clock/data recovery (CDR) provide only the most basic transceiver capability.

The IBM PowerPC 405 processor core used in the Virtex-II Pro/Virtex-II Pro X family is the highest performance embedded core available in FPGAs. The PowerPC architecture is used in many markets including communications, industrial control, test and measurement systems, and other performance-oriented markets. It is currently the most popular processor architecture in embedded applications.

### **10.3.PACKETS EVERYWHERE**

The Virtex-II Pro/Virtex-II Pro X family provides a powerful new paradigm for network processing where low latency is required, such as storage area networks, wireless infrastructure, and voice-over-IP networks. The digital convergence phenomenon drives the need for packet routing based on type and priority. For example, live voice and video data packets require significantly lower latency than data file packets. New data networking applications must now handle higher bandwidth traffic as well as more complex types of prioritized packets. In many cases, Virtex-II Pro/Virtex-II Pro X devices can offer higher overall performance than other solutions, including specialized network processors (NPs).

Using the Virtex-II Pro architecture, the most common packets may be quickly read and routed using FPGA logic, without incurring the lengthy software runtime needed by NPs. The FPGA logic interrupts the PowerPC processor core only when processor instructions are needed for special packet types.

For example, packets may be stored into a 16 KB dual-port memory area accessible by both the FPGA logic and the PowerPC 405 on-chip memory (OCM) port, allowing rapid change of control and packet disposition. By using the FPGA logic to process the most common packet types while the processor core handles the more specialized ones as a slave to the logic, the Virtex-II Pro architecture can provide higher overall performance than NPs, as well as more sophisticated processing capabilities than FPGA logic alone.

#### **Bridge, Anyone?**

Powerful protocol bridges for tying together disparate data stream formats are well-suited for the Virtex-II Pro/Virtex-II Pro X solution. New interface standards and protocols include PCI Express™, Infiniband®, Gigabit Ethernet, XAUI/10 Gigabit



Ethernet, RapidIO™, and HyperTransport™. These must interface seamlessly to one another, as well as to other standards such as PCI™, Fibre Channel, POS Phy Level 4, Flexbus 4, and others.

This presents a significant challenge to system developers because of changing standards, scarcity of off-the-shelf interface components, and the inflexibility of available solutions. System designers have had to assemble their own blend of FPGAs, discrete physical transceivers, and discrete communications processors to solve their complex system challenges. Even newer "programmable ASSPs" (application-specific standard products) with built-in serial transceivers fall short, because they frequently require companion FPGAs to supplement their logic capacity. The Virtex-II Pro solution, using the powerful Xilinx SystemIO™ capability to fully integrate silicon, software, and IP capabilities, provides the most flexible pre-engineered protocol bridge solutions available for fast time-to-market and low development cost.

### **Simplifying Complexity**

The Virtex-II Pro/Virtex-II Pro X solution offers a powerful paradigm for complex embedded systems found in signal processing, industrial control, image processing, networking, communications, and aeronautic applications. For the first time, complex embedded systems traditionally involving sophisticated hardware and software may be developed concurrently, emulated in actual hardware at speed, debugged in-system, and re-architected for performance within weeks, rather than months or years. In addition, full systems can be remotely upgraded as easily as software-only upgrades are performed today, using Compact Flash, CDROM, Internet, wireless transmission, or other flexible means. Hardware design is simplified using powerful development software and a large soft IP library to assemble logic- and processor-based platforms. Software development may be started earlier using the actual device in preconfigured sample platforms, without waiting for the new system board to be developed. In many cases, higher density Virtex-II Pro components may be used for early system development, whereby extra resources (including additional PowerPC processor cores) may be used to easily emulate board-level components yet to be developed. This flexibility, obviously unavailable in custom ASICs or ASSPs, allows systems to be emulated at speed, rather than simulated using software simulators at 100 or 1000 times slower. In-system debugging is further enhanced by the Xilinx ChipScope Pro tool, which provides comprehensive logic analysis—from probing internal nodes to full bus analysis with bus protocol adherence checks using an external logic analyzer via the



IEEE 1149.1 (JTAG) test access port. Using ChipScope Pro can result in orders of magnitude of improvement in engineering productivity.

Complex systems can be optimally repartitioned between FPGA logic and processor cores, allowing a continuum of possible trade-offs between the speed of logic and the flexibility of software code. For example, a first implementation of an echo cancellation algorithm might be all-software in compiled C code running on a PowerPC core, in order to allow the system software development to start. As the system is further optimized, part of the DSP algorithm could be retargeted using Matlab Simulink into FPGA logic to achieve a significantly faster but functionally identical system for production release.

In another example, an encryption application might implement the Diffie-Hellman key Exchange algorithm, whereby exponentiation and message management could be optimally partitioned into FPGA logic and an embedded processor, respectively. In this way, the programmable systems paradigm offers tremendous flexibility to allow system designers and architects to optimize the trade-offs in development time, system performance, and system costs.

It is significant that the embedded systems enabled by Virtex-II Pro solutions are "all-soft," in that both logic and software code are controlled by a soft data file. Because of this, the low cost of design maintenance and degree of design reuse is greatly enhanced. Whole system upgrades, including both hardware and software, can now be accomplished with one unified soft file using System ACE™ configuration solutions, offering the same low cost and ease of use as software-only upgrades.

#### **10.4.TIME IS MONEY**

The Virtex-II Series, comprising both the Virtex-II and Virtex-II Pro/Virtex-II Pro X families, offers significantly faster time-to-market and lower development costs than ASICs. Compared to a full-custom ASIC, the Virtex-II Pro solution eliminates the need for exhaustive verification during development, and allows hardware-software debug at system speeds rather than at slow software simulation speeds. In addition, the Virtex-II Pro/Virtex-II Pro X features of signal integrity, pre-engineered clocking capabilities, and an abundance of soft IP cores, significantly reduce development time. The Virtex-II Series offers significantly lower development costs than ASICs, due to lower tool costs, lower third-party IP costs, and absence of NRE costs. The Virtex-II

Series also increases engineering productivity by accelerating hardware availability for software development and increasing software debug speed. In addition, the availability of powerful development tools enables straightforward retargeting of other embedded processors into the PowerPC platform. Compared to other processor architectures, the PowerPC 405 core in most cases allows higher performance and more powerful capabilities, and thus can be used to accelerate preproduction of performance-sensitive applications.

The flexibility inherent in the Virtex-II Series allows system architects to fine-tune their architectural partitioning after the initial prototype is developed. That is, each subsystem function can be freely implemented as hardware only, software only, or any combination within the hardware-software continuum, depending on the trade-off between performance and complexity. For example, a wireless infrastructure system might initially implement a rake filter function in hardware, and then change to a firmware implementation as more software control is necessary during later development. This repartitioning would be impossible in custom ASICs without significant time and cost penalties.

The Virtex-II Series offers significantly more flexibility than fixed chip sets and ASSPs, allowing end user product differentiation and future-proofing. For a design requirement that can generally be met either by ASSPs or by Virtex-II Platform FPGAs, the initial design investment for an FPGA implementation may be higher. However, the advantages for Platform FPGA implementations include customizing of functionality, ease of design reuse, ability to fix design bugs, differentiation of user end products, and ownership and control of the entire system. These are important advantages in highly competitive markets where ASSPs have standing errata lists and unpredictable future availability. In contrast, properly developed Platform FPGA designs are soft designs that may be readily maintained and reused as needed. Therefore, FPGA methodologies can provide system manufacturers with greater competitive advantage in the short term, and greater ownership and control over their products in the long term.

Many high-bandwidth systems today use large FPGAs together with discrete SerDes transceivers, discrete communications processors, or other discrete components. The Virtex-II Pro/Virtex-II Pro X family can eliminate the need for many of these external components, enhancing time-to-market and performance, even providing system cost

benefits in many cases. Multi-chip solutions using FPGAs typically require over a hundred I/O pins to interface to each discrete quad 3.125 Gbps SerDes transceiver or discrete microprocessor.

The result is increased PCB complexity to accommodate the hundreds of traces, reduced system performance due to on-chip/off-chip connections, and higher overall system costs. In some cases, the increased FPGA pin-count requirement may force a higher-density FPGA to be used, again increasing the overall cost. In these cases, the Virtex-II Pro/Virtex-II Pro X devices can integrate the discrete components to achieve faster system development, higher system performance, and lower costs.

## **CHAPTER ELEVEN**

### **ABOUT XILINX SOFTWARE AND COMPANY**

How, many ask, did Xilinx (pronounced "Zylinks") get its unusual name? In 1984, when Xilinx was just forming, the new company tried to register several "sensible" names, but they were all taken. This became an expensive proposition and the founders, (being very frugal), decided to create an unusual name that wasn't taken. Thus, two of the founders came up with "Xilinx."

Xilinx Fellow Bill Carter, who was at Xilinx from the start, explains. "The 'X's' at each end represent programmable logic blocks. The "linx" represents programmable links that connect the logic blocks together, a key innovation embodied in FPGAs."

While Xilinx doesn't follow all the branding and phonetically-correct rules for naming a company, a Xilinx by any other name would not be as sweet.

#### **11.1.HISTORY OF XILINX**

Timeline of Significant Events in Xilinx History

**1984**

**Ross Freeman, Bernie Vonderschmitt, and Jim Barnett found Xilinx.**

The company's business and management mission and philosophy are created, a new kind of company is born. The concept for a new type of product, the Field Programmable Gate Array, takes shape.



**1985**

**Xilinx introduces its first product, the XC2064™.**

It's the first-ever FPGA, a radical new form of programmable logic.

**1987**

**Sales office established in Weybridge, England.**

This is the first sales office outside North America, targeted to serve the European market.

**1988**

**The company opens its first overseas office in Tokyo.**

Xilinx K.K. is born. Initial focus is to serve Seiko, our first wafer supplier/partner.

**1989**

**Xilinx founder, Ross Freeman, passes away.**

His dream for the team and the technology lives on.

**1990**

**Xilinx goes public at \$10 per share, after reaching several quarters of profitability.**

Shares are \$0.83 when adjusted for splits. This is a major milestone along the path to realizing the company vision.

**1991**

**The XC4000™ family of FPGAs is introduced.**

This is the first broadly adopted FPGA and will become the primary Xilinx revenue driver for the 90's.

**1993**

**Xilinx Scotland is established in Edinburgh.**

This team's focus on IP solutions core and software development brings Xilinx a considerable competitive advantage.

**1995**

**Xilinx Ireland officially opens in Dublin.**

This is our first major site in Europe, establishing manufacturing and engineering capability outside the U.S.

## **Xilinx Colorado is established in Boulder.**

Boulder employees significantly increase software development capability in the company. This is the first major North American site outside of Silicon Valley. It has since moved to Longmont, Colorado.

## **1996**

### **Wim Roelandts joins as CEO and President.**

He brings 30 years of Hewlett-Packard experience to his new assignment.

## **1997**

### **CREATIVE Values dialogue is created throughout the company.**

All employees participate in a process to articulate the values of Xilinx. These are: customer focus, respect, excellence, accountability, teamwork, integrity, very open communications and enjoying our work. The first letter of each value forms the acronym: CREATIVE.

## **1998**

### **Virtex®™ FPGA family is introduced.**

This is a major step in FPGA architecture and opens up new markets for the company. The Virtex family becomes the primary revenue driver to date.

## **1999**

### **Xilinx Albuquerque opens with acquisition of CoolRunner™ team and technology.**

This product line offers new low power and lower cost CPLD products to customers.

## **2000**

### **Xilinx revenue exceeds \$1B.**

The company reaches the billion-dollar milestone only 10 years after going public. Employees take out a full-page ad saying "Thanks a Billion for Your Leadership" as a tribute to CEO Wim Roelandts at a pivotal point for the company.

## **2003**



### **Spartan®<sup>TM</sup>-3 family of products is introduced.**

This very low-cost product is the world's first 90nm FPGA. The Spartan-3 technology puts us considerably ahead of our competition and in the company of premier advanced semiconductor manufacturers.

### **2003**

Wim Roelandts becomes Chairman of the Board.

**Bernie Vonderschmitt retires; the last company founder leaves an impressive legacy.**

### **2004**

**Xilinx celebrates its 20th anniversary.**

The company observes its first 20 years of life by honoring employees, customers, shareholders, partners, and local communities.

### **2008**

**Moshe Gavrielov is named President and CEO.**

Wim Roelandts remains Chairman of the Board.

### **How Xilinx Began**

#### **New Technology**

Two brilliant engineers and a marketing guru working in Silicon Valley in 1984 had a dream. Bernie Vonderschmitt, Ross Freeman, and Jim Barnett dreamed of starting a different kind of company.



**Ross Freeman, Xilinx co-founder, invented the "field programmable gate array" (FPGA), a new form of programmable logic.**



**Bernie Vonderschmitt, Xilinx co-founder, pioneered the revolutionary concept of a fabless semiconductor.**

They wanted to create a company that would develop and launch state-of-the-art technology in an entirely new field. And they wanted to lead it in such a way that the people who worked there loved their jobs, enjoyed working together, and were fascinated with their work.

The technology that propelled Xilinx into being was considered an off-the-wall concept in 1984. Invented by Xilinx co-founder Ross Freeman, the new semiconductor, now known as the field programmable gate array, was a completely new form of programmable logic.

These chips could be personalized by customers to perform a variety of functions by programming them with the help of software. "The concept," says Xilinx Fellow Bill Carter, who was the eighth employee to be hired in the new company in 1984, "required lots of transistors and, at that time, transistors were considered extremely precious. People thought that Ross's idea was pretty far out."

Ross postulated that transistors, because of Moore's Law (the doubling of transistor density every 18 months) would be getting less expensive and, therefore, less precious every year. In the years to come, a multi-billion dollar market for field programmable gate arrays (FPGAs) emerged, creating the foundation for the successful enterprise that Xilinx is today. Sadly, Ross Freeman passed away in 1989. The technology he invented is thriving and continues to delight more and more customers in an ever-widening breadth of industries.

## **11.2.EFFECTIVE PARTNERSHIPS**

Bernie Vonderschmitt, an engineer and an MBA graduate, came up with a powerful business model for the young company. When he was General Manager of the Solid State Division of RCA, he became convinced, working at the time with three in-house foundries making semiconductors, that semiconductor factories (or fabs) were expensive and burdensome. "If I ever start a semiconductor company, it will be fabless," he vowed. "We'll find partners who can do our manufacturing for us."

And that is exactly what Xilinx did in 1984. Since then, the idea has become so compelling and popular that today there are approximately 700 fabless semiconductor companies around the world.

However, the three founders wanted to not only revolutionize technology but the way companies are managed as well. Ross Freeman put it best. He hoped to start a company that had solid, ethical values, invited employee loyalty, made a good and useful product, helped make employees feel like owners, and encouraged people to enjoy their work.

The co-founders called this set of values and people objectives their "philosophy" and looked for employees who felt comfortable in this environment. And their theory - which has proven correct - was that if you created this kind of community atmosphere for clever and inventive people, they would stay, keeping their innovation and expertise in the company.

These original values regarding the treatment of employees and the way they interact with each other provided the basis for how Xilinx operates today. They help make Xilinx a great place to work.



And the technology that the three men introduced to the world is more popular than ever. It has become pervasive and mainstream, thanks to the technology and cost benefits that have come about because of Moore's Law.

The dream that Bernie, Ross, and Jim talked about in 1984 is a reality today, proving that dreams do come true.

### **11.3.BUSINESS OF XILINX**

There are three types of electronic devices: memory, processors, and logic. Memory devices store random information (contents of a spreadsheet or database); processors execute software instructions to perform a wide variety of tasks (running a data processing program or video game); and logic provides specific functions (communications between devices, and every other function a system must perform). There are two categories of logic devices: fixed or custom, and programmable or changeable. We are in the programmable logic business.

Xilinx leads the Programmable Logic Device (PLD) market - one of the fastest growing segments of the semiconductor industry. This market features a revolutionary technology called the field programmable gate array (FPGA) that our company pioneered in 1984.

Xilinx is the world's leading supplier of programmable logic solutions. We supply customers with "off-the-shelf" logic devices that customers can program to perform specific functions using the development tools we provide.

This programmability provides a revolutionary alternative to fixed or custom logic devices that typically require many months to design, test, and manufacture. Xilinx customers enjoy the benefit of faster time-to-market and increased product design flexibility as a result.

Our company's business is drawn from a variety of industry segments. In recent years, a large portion of our revenues came from the communications marketplace. However, we have become increasingly more diversified to include the consumer, industrial, and automotive sectors.

You can find Xilinx chips in a wide variety of digital electronic applications ranging from wireless base stations to HDTV to portable handsets.

Our extensive product line includes silicon solutions like the Virtex™ series FPGAs (high performance FPGAs for networking, communications, and video/imaging applications); Spartan™ FPGAs (ideal for high volume applications); and CoolRunner™ CPLD families (Complex Programmable Logic Devices that offer ultra low cost and low power). We also offer a powerful suite of high performance software design tools.

Xilinx has over 21,000 customers around the globe, including Alcatel, Cisco Systems, EMC, Ericsson, Fujitsu, Hewlett-Packard, IBM, Lucent Technologies, and Motorola. Most of our sales are handled by outside partners: both large distributors and independent sales representatives.

The company has a very flexible business model that has contributed to our success as an employer and a competitor. We are a "fabless" supplier and do not manufacture our logic devices. Instead, we have formed close strategic alliances with chip manufacturers like UMC and Toshiba. This strategy, along with outsourcing most of sales, allows us to focus more of our energies on R&D, marketing and technical support. The resulting flexibility gives us the ability to rearrange business priorities quickly as we respond to the cyclical nature of the semiconductor market. It also has made it easier for the company to avoid layoffs in periods of downturn.

What essentially defines Xilinx is vision. Our long-term business goal is to put a PLD in every piece of electronic equipment within the next 10 years. Our long-term management goal is to create a company that sets the standard for managing high technology companies. While these are ambitious undertakings, at Xilinx, visions have a way of turning into reality.

#### **11.4.SUCCESS OF XILINX**

While the rest of the industry continues the practices of layoffs and shaking-off excessive inventory, Xilinx is busy innovating, collaborating, and introducing new products to market. Unlike many of our counterparts, Xilinx views downturns as an



opportunity to focus on research and development, streamline operations, and deliver new products that change the FPGA landscape.

For the past few years, Xilinx has asserted a considerable market leadership position. We secured over 50% of the PLD market share: larger than all other public PLD companies combined. By creatively avoiding layoffs and empowering employees, we rose to become the fourth best company to work for in America (Forbes magazine).

Through the power of innovation and partnerships, Xilinx takes the FPGA-based value chain to a new level. By teaming with technology leaders in silicon fabrication, design automation, system level tools, IP, and design services, we deliver a complete value chain and strengthen our position as a strategic partner for our customers. Delivering this complete value chain enables the fastest innovation while reducing total development and system costs for our customers. It also reduces time to market and increases time in market for our customer's products.

In March 2002, through partnering with industry leaders IBM, WindRiver Systems, and Conexant, Xilinx delivered the Virtex™-II Pro programmable system solution. The solution was the first of its kind and is the most flexible tool ever invented for a designer. The Virtex-II Pro FPGA includes programmable logic fabric with high-speed embedded PowerPC processors and integrated 3.125 gigabit RocketIO™ serial transceivers supported by leading design tools. Recent additions to the family and lower price points have now made the Virtex-II Pro solution the de-facto standard for all programmable logic users.

The Virtex-II Pro solution responds to the issues facing design teams and their corporations. By delivering both high-performance processing and high bandwidth connectivity on a single device, many design challenges associated with integration, high-speed interfacing, high performance processing, and new design methodologies are effectively solved. The rapid rate of change in technology and standards demands a solution that is completely flexible and reduces inventory risks and NRE costs - the Virtex-II Pro solution delivers.

Xilinx is a company built on delivering maximum customer value and ongoing innovation throughout all of our product lines. Xilinx recently revamped all of its products from the new Spartan™-IIE cost-optimized FPGA solution to the

CoolRunner™-II RealDigital CPLD solution, the Virtex-II Pro platform for programmable systems, and the Virtex-II EasyPath solution for cost management. We also introduced the world's fastest and most productive software tool suite with our ISE 4.2i software release, numerous intellectual property cores, and the technical training necessary to decrease time-to-knowledge for the rapid assimilation of this new technology. We continue to focus on raising the bar by adding more value in every category of the value chain.

Through the years, Xilinx has evolved into a solutions company rather than remaining just a chip company. We can only be better tomorrow than we are today by working closely with our customers and anticipating their needs. Xilinx's job is to continue to expand our capabilities and our partnerships, so we can continue to be a strategic partner for our client companies.

Xilinx is an innovation engine and our employees are the keys to our innovation. Such innovation requires personnel policies that allow employees to make their own decisions and take risks. Our company values and corporate culture promote teamwork and very open communication. We know that keeping employees satisfied leads directly to innovation, customer satisfaction, and ultimately, increased profits. Our employees are inspired and know they make a real difference.

This unique work environment has resulted in breakthrough technology, marketing and community achievements. For example, Xilinx continues to support local schools through our Stock for Students program and made a \$1 million donation to the American Red Cross. Also, Xilinx was the first semiconductor company to simulcast training in North America and Europe through industry events like Programmable World 2002.

With a combination of innovative products, world-class partners, inspired employees and the recognition of the balance between business and community, our clients have taken Xilinx solutions, management, and employees to heart. This is a reminder that good people ultimately do come in first when they are inspired and empowered to be leaders

## 11.5.VALUES OF XILINX

Our values have helped set the character of our company. They are more than a set of lofty ideals put down on paper and left to yellow in conference rooms. Values are very much alive and well in Xilinx. We don't just talk about them, we try and live them every day.

Our values also provide the backdrop for the dialogue we have with colleagues. They help us make business decisions. They provide the framework for interacting with each other. What's especially impressive about our values is that they are accepted and acceptable around the world. The practices may be different in different places, but the values are relevant everywhere.

While a whirlwind of business change constantly surrounds us - and we accept change as the reality of today's high-tech industry - it's important to know that the values we depend upon are constant and unchanging. They are, in a very real sense, our permanent anchors.

The set of values we believe in started with our company's founders. The three men who followed their dream of starting a new enterprise were just as concerned about the work environment as they were about the new, innovative technology they were pioneering. Respect for the dignity of the individual was the cornerstone of the philosophy upon which Xilinx was founded.

In 1996, our company started a grassroots process to articulate our values. We looked very carefully at our business and made certain the values were connected with what would move us forward and foster our growth in the marketplace. Most importantly, we wanted to capture in words what we liked so much about working at Xilinx and the ideals our founders had set in motion.

The result was a description of the the eight Xilinx CREATIVE values.

Customer Focused

Respect



Excellence

Accountability

Teamwork

Integrity

Very Open Communication

Enjoying Our Work

It was the creativity of the founders and their innovative ideas that launched a new category of products for customers around the world. And it is the creativity of our products and patents that has propelled us to market leadership.

At Xilinx, we believe that making decisions based on our values translates directly to the bottom line, helps us be more successful in our business, and brings us closer to realizing our company vision of setting a new standard for managing a high-tech company. Customers are eager to do business with a company whose values are as excellent as their products.

Another way the values are kept alive is through a variety of appreciation programs. The most popular one is the Values Medallion award. Employees nominate someone who has exhibited a teamwork value, and, each quarter, several winners are randomly selected from the nominees. These individuals are awarded 10 shares of stock and receive public recognition from Wim.

How do we "enforce" the values? We don't. We leave it up to each individual to act in accordance with them. The values are there to direct our actions, to guide us professionally and personally, and to inspire us in the worst and the best of times.



## CONCLUSION

We are seeing the stopwatch design is working correctly.

At the beginning reset is 1 the program is waiting 100 ns and reset become 0. After that startstop waits 600 ns and becomes 0 then tenths counter which is binary counter starting the count and when it is reached the ten sixty counter starts the count.

Sixty counter has lsb and msb counter. The lsb starting count and is count up to nine then msb is increases one. Every time msb increases one when lsb is reached the nine.

## REFERENCES

- [http://www.members.shaw.ca/kadirm/VHDL\\_Course\\_notes.htm](http://www.members.shaw.ca/kadirm/VHDL_Course_notes.htm)
- [http://www.xilinx.com/products/silicon\\_solutions/fpgas/virtex/virtex\\_ii\\_pro\\_fpgas/capabilities/index.htm](http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fpgas/capabilities/index.htm)
- <http://www.xilinx.com/company/index.htm>
- Documents in xilinx9.1i