Digital Circuit Design Using Xilinx ISE Tools

Poras T. Balsara and Prashant Vallur

Table of Contents

Acknowledgements

We thank the ***Xilinx Corporation*** for providing the Xilinx Foundation Software and FPGA prototyping boards for educational purposes. We also thank Prof. Jan Van der Spiegel for allowing us to use parts of his Xilinx tutorials developed for his course in the Electrical Engineering Department at the University of Pennsylvania.

1. Introduction

Xilinx Foundation Series Tools is a suite of software tools used for the design of digital circuits implemented using *Xilinx Field Programmable Gate Array (FPGA)* or *Complex Programmable Logic Device (CPLD)*. The design procedure consists of (a) design entry, (b) compilation and implementation of the design, (c) functional simulation and (d) testing and verification. Digital designs can be entered in various ways using the above CAD tools: using a schematic entry tool, using a hardware description language (HDL), ABEL or VHDL or a combination of both. In this lab we will only use the design flow that involves the use of ABEL HDL.

The Foundation Series tools enable you to design combinational and sequential circuits starting with ABEL HDL design specifications. The steps of this design procedure are listed below:

1. Create ABEL design input file(s) using template driven editor.
2. Compile and implement the ABEL design file(s).
3. Create the test-vectors and simulate the design (functional simulation) without using a PLD (FPGA or CPLD).
4. Assign input/output pins to implement the design on a target device.
5. Download bitstream to an FPGA or CPLD device.
6. Test design on FPGA/CPLD device

An ABEL input file in the Xilinx Foundation software environment consists of the following segments:

- ***Header:*** module name, options and title.
- ***Declarations:*** pin, constant, sets, states, library.
- ***Logic Descriptions:*** equations, truth tables, state diagram.
- ***End:***

All your designs for this lab must be specified in the above ABEL input format. Note that the *state diagram* segment does not exist for combinational logic designs.

2. Programmable Logic Devices: *FPGA* and *CPLD*

In this lab digital designs will be implemented using two types PLDs: FPGA and CPLD. For the FPGA implementations we will mainly use the XC4010XL part which belongs to the XC4000 family of FPGAs. CPLD implementations will be done using the XC95108 which is part of the XC9000 CPLD family. These devices come in a variety of packages. We will be using devices that are packaged in 84 pin PLCC with the following part numbers: XC4010XL, PC84CMN9817 and XC95108, PC84ASJ9749. XC4010XL FPGA is a device with about 10K gates, whereas, XC95108 CPLD has about 2400 usable gates. Detailed information on these devices is available at the Xilinx website.

3. Creating a New Project in Xilinx Foundation Series Tools

Xilinx Foundation Series Tools can be started by clicking on the Project Navigator Icon on the Windows desktop. This should open up the Project Navigator window on your screen. This window shows (see Figure 1) the last accessed project.

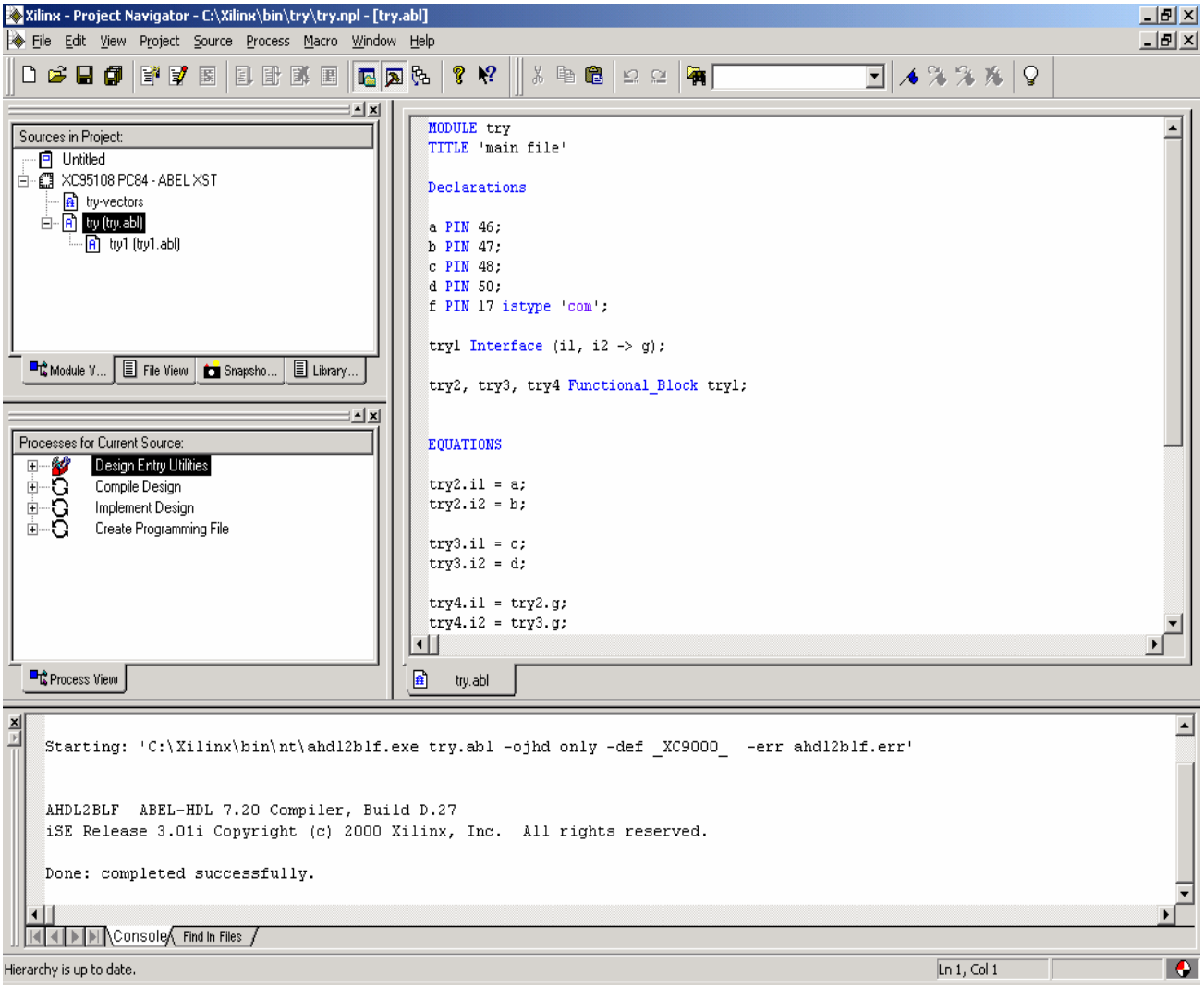

Figure 1: Xilinx Project Navigator window (*snapshot from Xilinx ISE software*)

3.1 Opening a project

Select **File->New Project** to create a new project. This will bring up a new project window (Figure 2) on the desktop. Fill up the necessary entries as follows:

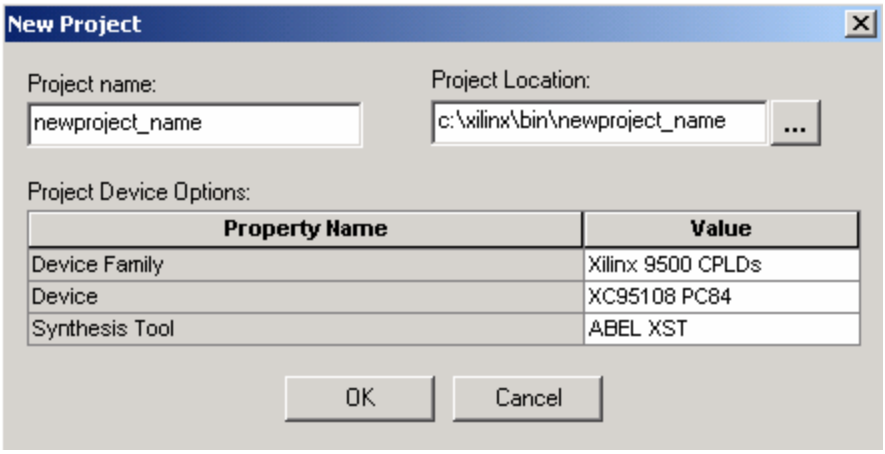

Figure 2: New Project Initiation window (*snapshot from Xilinx ISE software)*

- **Project Name**: Write the name of your new project
- **Project Location**: The directory where you want to store the new project
- For each of the properties given below, click on the '**value**' area and select from the list of values that appear.
    - **Device Family**: Family of the FPGA/CPLD used. In this laboratory we will mostly use the **Xilinx 9500 CPLDs.**
    - **Device**: The number of the actual device. For this lab you may enter **XC95108 PC84** (this can be found on the attached prototyping board)
    - **Synthesis Tool**: **ABEL XST**
    - Then click on **OK** to save the entries.

All project files such as schematics, netlists, ABEL files, modules, etc., will be stored in a subdirectory with the project name. A project can only have one top level HDL source file (or schematic). Modules can be added to the project to create a modular, hierarchical design (see Section 10).

In order to open an existing project in Xilinx Foundation Series Tools select **File->Open** to show the list of projects on the machine. Choose the project you want and click **OK**.

3.2 Creating an ABEL HDL input file for a combinational logic design

In this lab we will enter a design using a structural or RTL description using the ABEL HDL. You can create an ABEL HDL input file (**.abl** file) using the HDL Editor available in the Xilinx ISE Tools (or any text editor).

If adding an already existing source file (.abl file) to the project, in the project Navigator window, select **Project -> Add Copy Source** and browse through the disk for the source file.

If creating a new source file, in the Project Navigator window, select **Project -> New Source.** A window pops up as shown in Figure 3. (*Note*: *"**Add to project**" option is selected by default. If you do not select it then you will have to add the new source file to the project manually*.)

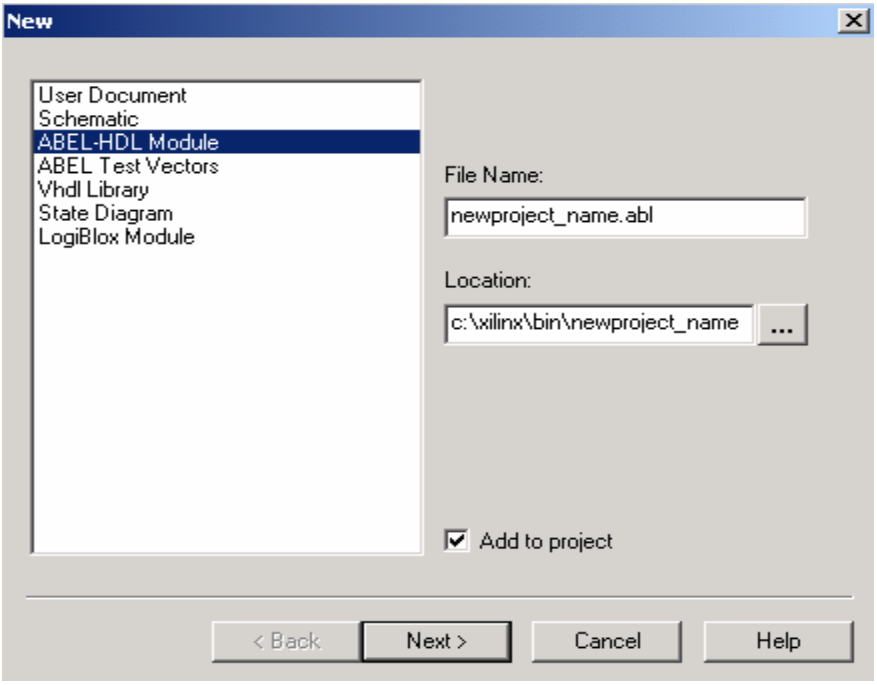

Figure 3: Creating ABEL-HDL source file *(snapshot from Xilinx ISE software)*

Select **ABEL-HDL Module** and in the "File Name:" area, enter the name of the ABEL source file you are going to create. *Make sure that the top-level ABEL file name matches with the name of the project*. Also make sure that the option **Add to project** is selected so that the source need not be added to the project again. Then click on **Next>** to accept the entries. This pops up the following window (Figure 4).

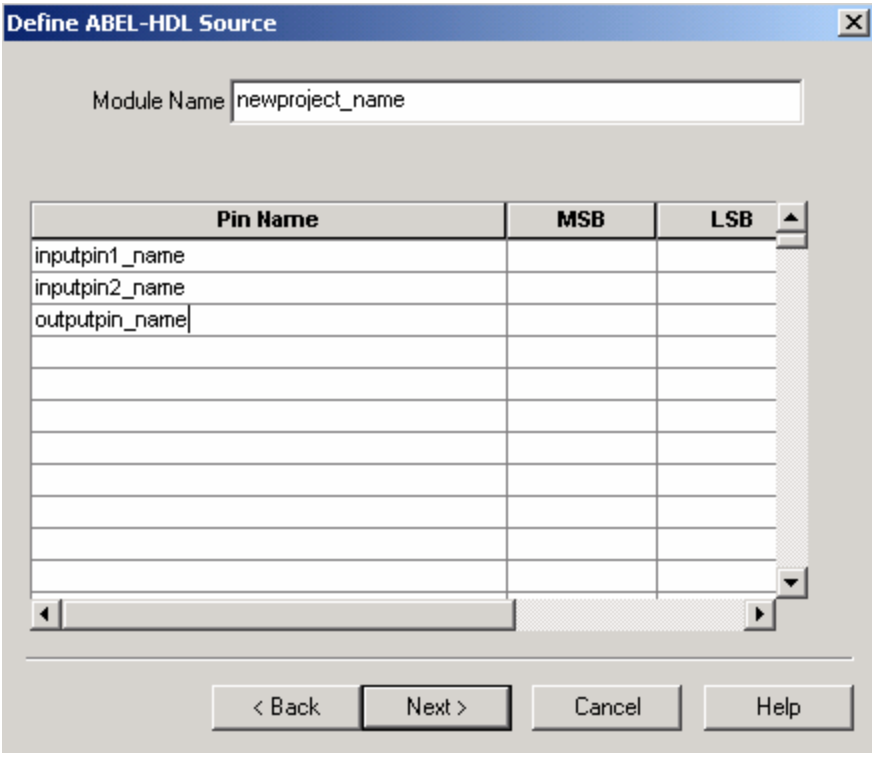

Figure 4: Define ABEL-HDL Source window (*snapshot from Xilinx ISE software*)

In the **Pin Name** column, enter the names of all input and output pins. A Vector/Bus can be defined by entering appropriate bit numbers in the **MSB/LSB** columns. Then click on **Next>** to get a window showing all the new source information (Figure 5). If any changes are to be made, just click on **<Back** to go back and make changes. If everything is acceptable, click on **Finish** to continue.

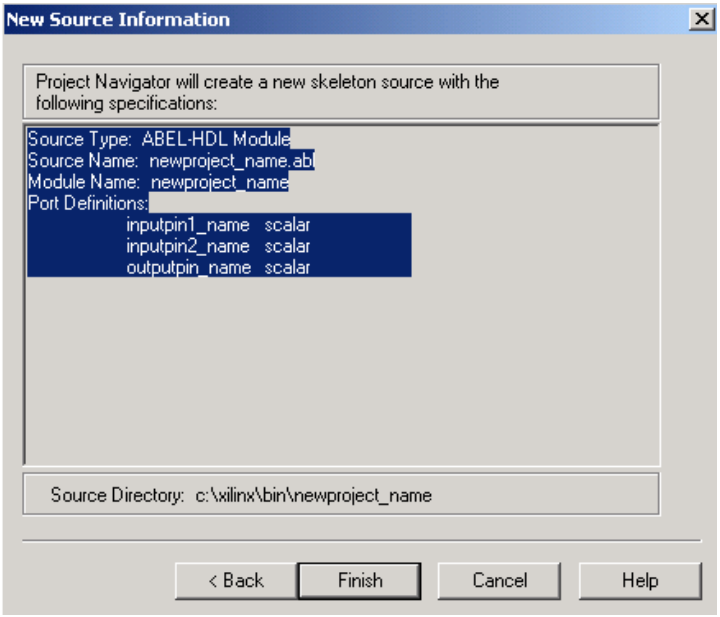

Figure 5: New Source Information window(*snapshot from Xilinx ISE software*)

Once you click on **Finish**, the source file will be displayed in the sources window in the **Project Navigator** (Figure 1).

If a source has to be removed, just right click on the source file in the **Sources in Project** window in the **Project Navigator** and select **Remove** in that. Then select **Project -> Delete Implementation Data** from the Project Navigator menu bar to remove any related files.

3.3 Editing the ABEL-HDL source file

The source file will now be displayed in the **Project Navigator** window (Figure 6). The source file window can be used as a text editor to make any necessary changes to the source file. All the input/output pins will be displayed. Make sure you enter the pintype for all output pins have to (combinational or sequential). For combinational, an output pin F can be declared as **F PIN istype 'com'** and if it's sequential, F **PIN istype 'reg'**. Enter the logic descriptions using: *Equations* or *Truth Tables* or *State Diagram* (for sequential circuits). Save your ABEL program periodically by selecting the **File->Save** from the menu. You can also edit ABEL programs in any text editor and add them to the project directory using "Add Copy Source".

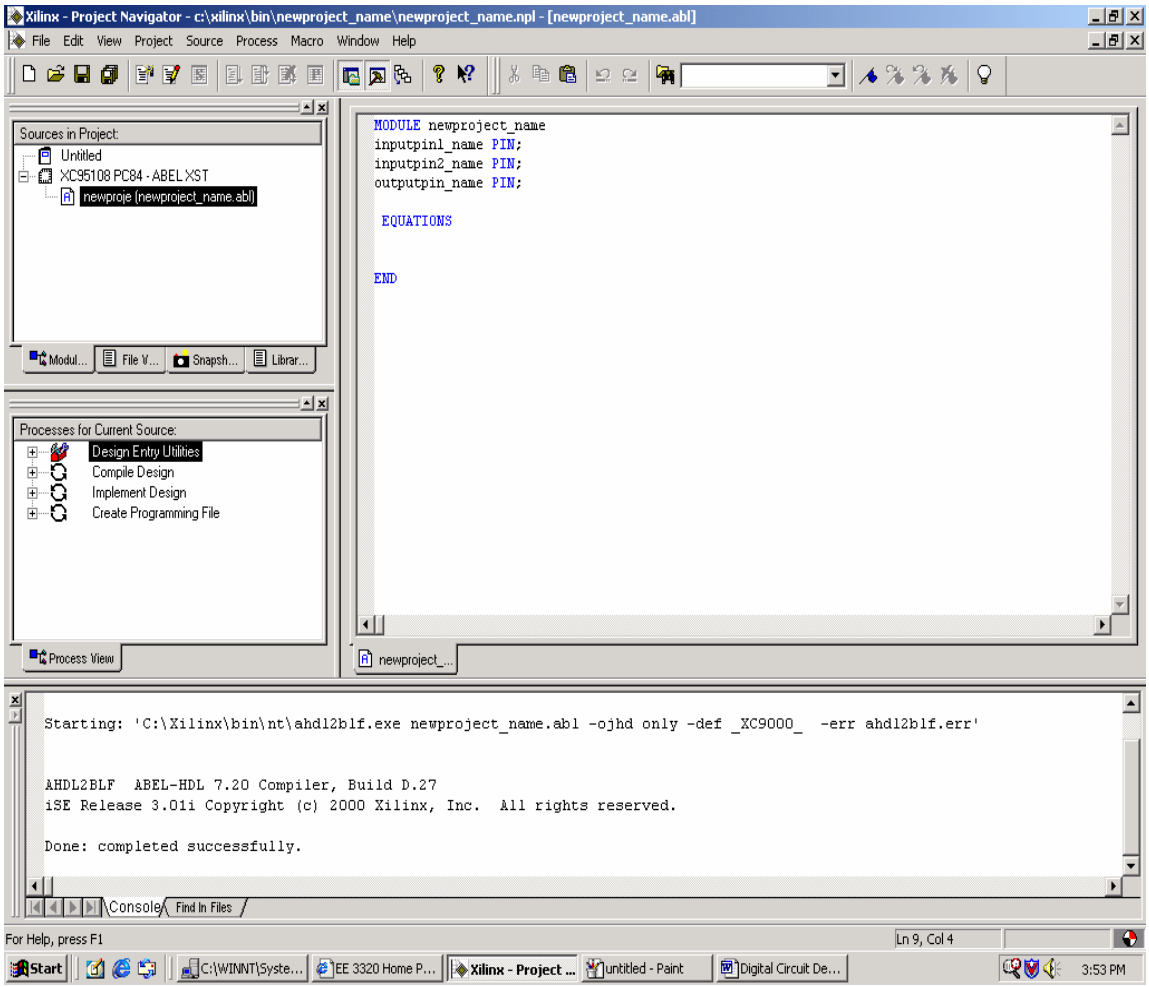

Figure 6: ABEL editor window in the Project Navigator (*from Xilinx ISE software*)

- **Adding *Logic Equations* to ABEL designs:**

  Logic equations can be easily added to the Logic Description section of an ABEL program. For example, an output ***Z*** can be described as,

  Z = (a & !b) # c;

  Remember that the names are case sensitive.

  Logical, arithmetic and relational operators available in ABEL are described in Table 1. Details of the relational operators are discussed in the section on sequential design.

Logical		Arithmetic		Relational	
&	AND	**-**	2's Complement	==	Equal
#	OR	**A-B**	Subtraction	**!=**	Not Equal
!	NOT	**A+B**	Addition	<	Less Than
$	XOR	<<	Shift Left	<=	Less Than / Equal
		>>	Shift Right	>	Greater Than
				<=	Greater Than / Equal

Table 1 Operation in ABEL HDL

- **Adding *Truth Tables* to ABEL designs:**

  Truth tables can also be included in a design by using the syntax shown given below. A table starts with a header with all the input and output names followed by rows of truth table entries. Combinational transitions are specified with a **->** and registered transitions (in sequential circuits, Section 9) are specified with a **:>**

```
TRUTH_TABLE
([input1, input2, ... inputn] -> [output1, output2, ... outputm])
            [0, 0, ... 0] -> [1, 0, ... 1];
            [0, 0, ... 1] -> [0, 1, ... 0]
            ...
            [1, 1, ... 1] -> [0, 1, ... 1];
```

Suppose we want to describe an AND gate. It can be done using the logic equation as shown in Figure 7 or using the Truth Table as shown in Figure 8. We can use of either of them or both of them to define various output functions in the design.

```
MODULE newproject_name
TITLE 'AND Gate'

Declarations

inputpin1_name PIN;
inputpin2_name PIN;
outputpin_name PIN istype 'com';

EQUATIONS

outputpin_name = inputpin1_name & inputpin2_name ;

END
```

newproject_...

Figure 7: AND gate description using Equations (*snapshot from Xilinx ISE software*)

```
MODULE newproject_name
TITLE 'AND gate'

Declarations

inputpin1_name PIN;
inputpin2_name PIN;
outputpin_name PIN istype 'com';

TRUTH_TABLE

( [inputpin1_name, inputpin2_name] -> outputpin_name );

        [0,0] -> 0;
        [0,1] -> 0;
        [1,0] -> 0;
        [1,1] -> 1;


END
```

newproject_...

Figure 8: AND gate description using Truth Table (*from Xilinx ISE software*)

4. Compilation and Implementation of the Design

The design has to be compiled and implemented before it can be checked for correctness by running functional simulation or downloaded onto the prototyping board. With the top-level ABEL file opened (can be done by double-clicking that file) in the HDL editor window in the right half of the Project Navigator, and the view of the project being in the **Module view** , the **compile design** and **implement design** options can be seen in the **process view**. **Design entry utilities** and **Create Programming File** options can also be seen in the process view. The former can be used to include user constraints, if any and the latter will be discussed later.

To compile the design, double-click on the option **Compile Design** in the **Processes** window. It will go through steps like **Check Syntax**, **Compile Logic, Interpret Feedbacks, Reformat Logic** and **Optimize Hierarchy**. If any of these steps could not be done or done with errors, it will place a **X** mark in front of that, otherwise a tick mark will be placed after each of them to indicate the successful completion. If everything is done successfully, a tick mark will be placed before the **Compile Design** option. If there are warnings, one can see mark in front of the option indicating that there are some warnings. One can look at the warnings or errors in the **Console** window present at the bottom of the Navigator window. ***Every time the design file is saved; all these marks disappear asking for a fresh compilation***.

To implement the design, double-click on the **Implement Design** option. It has many steps in it e.g., **Translation, Fitter, Timing Analysis** and **Launch Tools.** One can use the timing analyzer and post fit chipviewer from the Launch Tools section once the implementation is completed successfully. If the implementation is done successfully, a tick mark will be placed in front of the **Implement Design** option.

The above two steps can be done in a single step by just double-clicking on the **Implement Design** option straightaway. This will do the compilation first and then the implementation. In our example, when we compile the design, it shows warning and it indicates the warning in the **Console** as shown in the Figure 9.

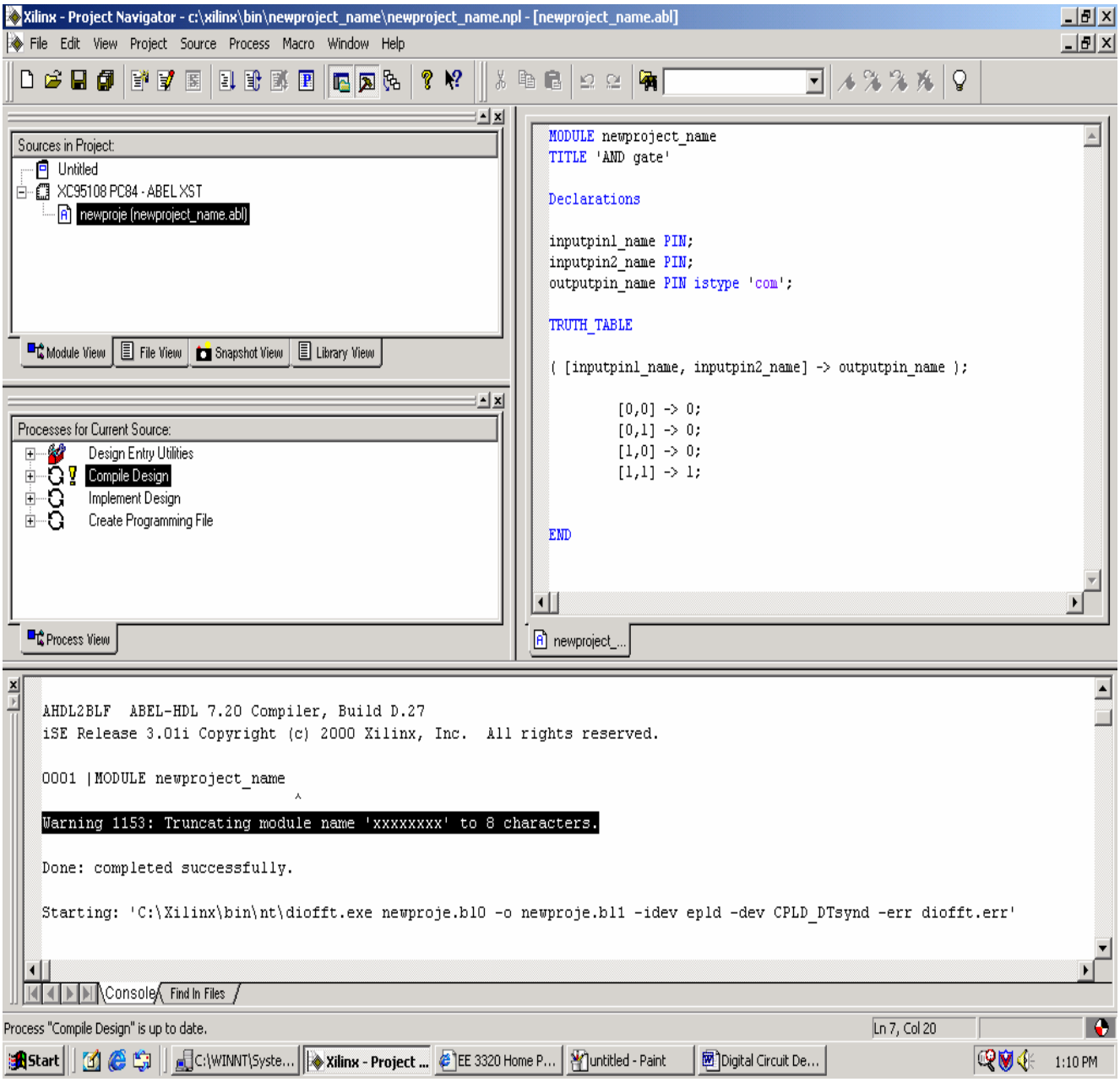

Figure 9 : Compiling the Design *(snapshot from Xilinx ISE software)*

After that, when the design is implemented, it ignores the warnings and completes the implementation successfully (shown in Figure 10).

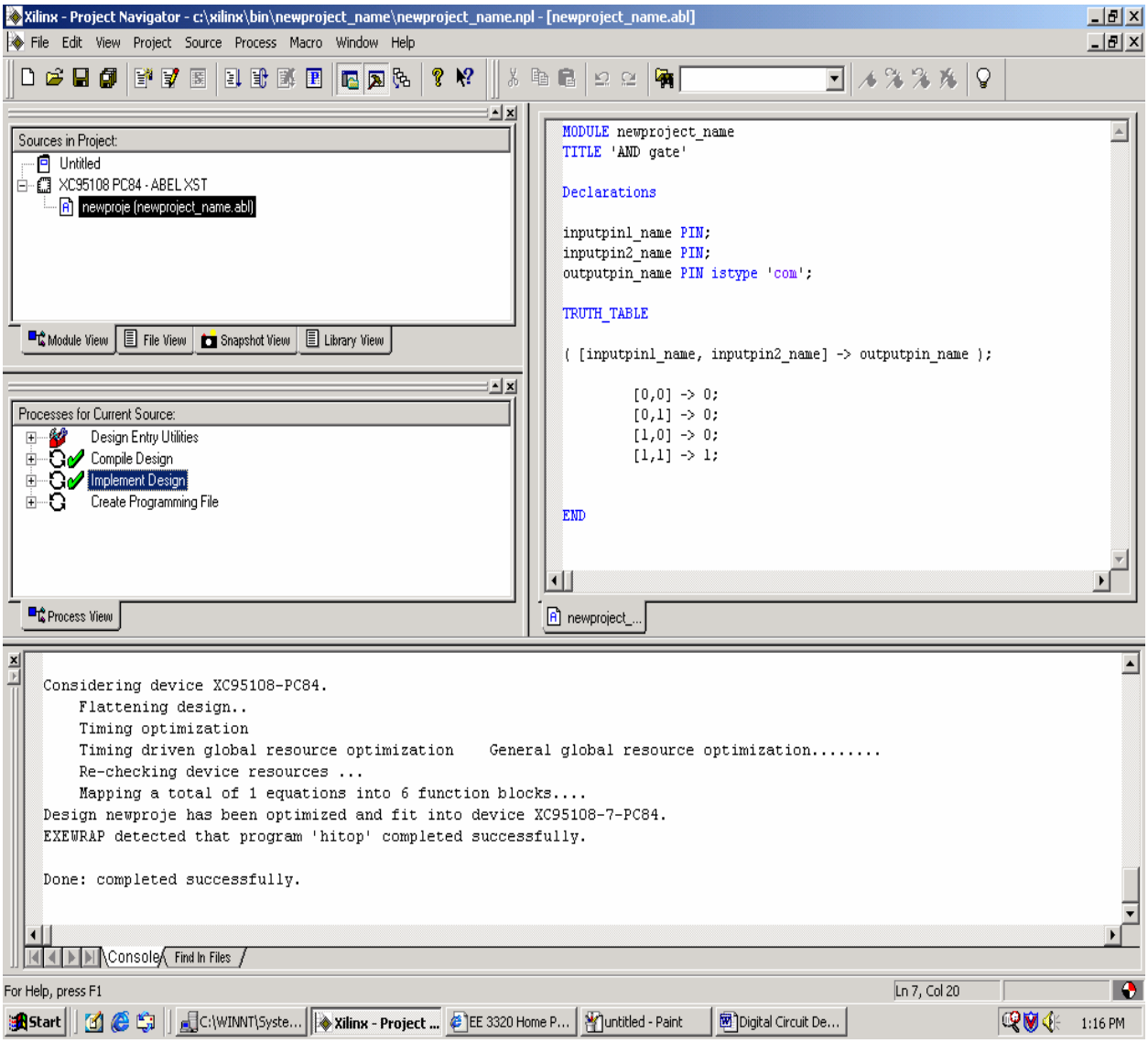

Figure 10 : Implementing the Design (*snapshot from Xilinx ISE software)*

5. Functional Simulation of Combinational Designs

5.1 Adding the test vectors

To check the functionality of a design, we have to apply test vectors and simulate the circuit. We can add test vectors in the top-level ABEL file as shown in the Figure 11.

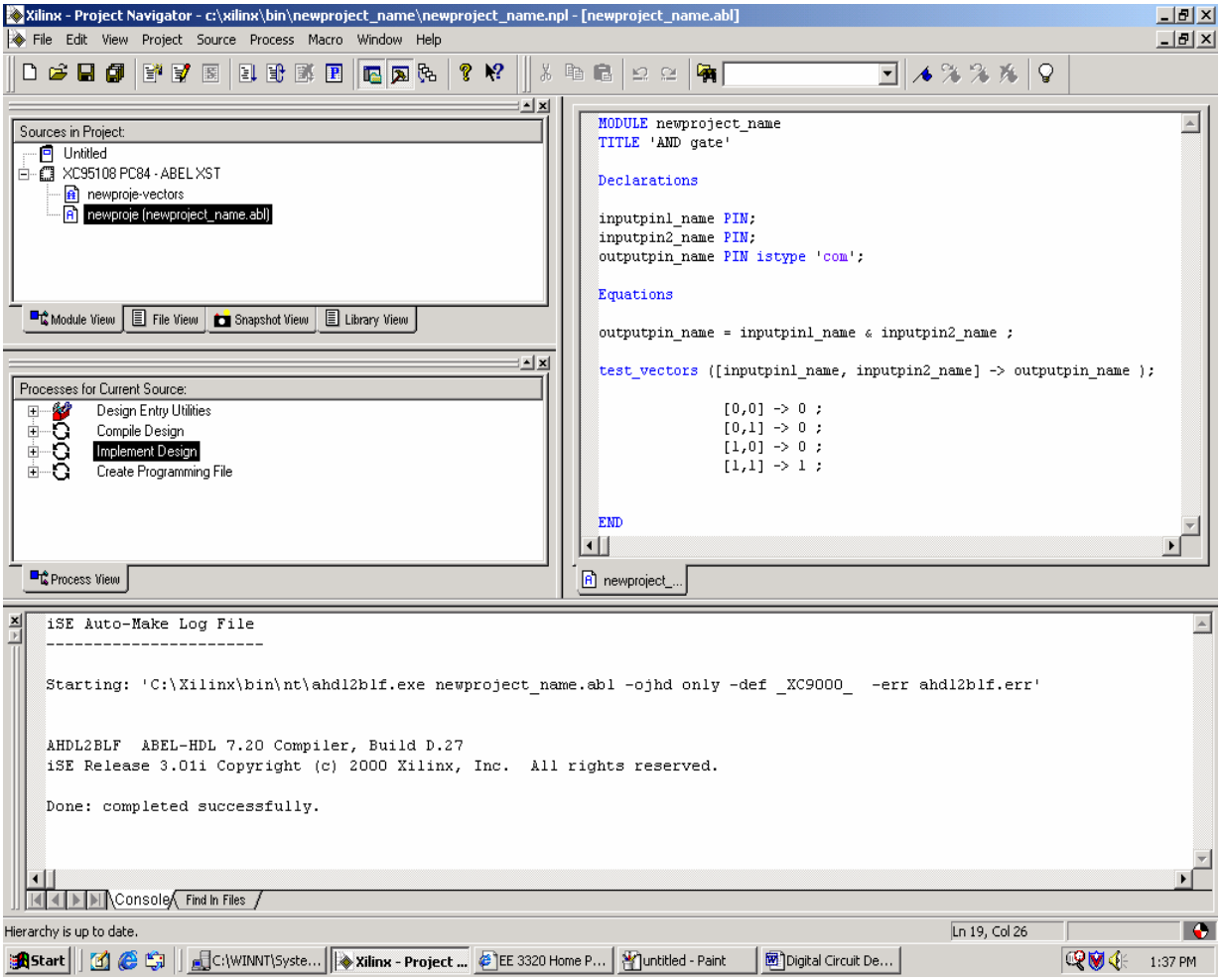

Figure 11 : Adding test vectors to the design (*snapshot from Xilinx ISE software)*

One can see the test vectors file (newprojec-vectors) added to the **Sources in Project** window on the left-hand side of the Project Navigator window. Next, compile and implement the design as before. Alternatively, you may add the test_vectors at the same time you create the rest of the ABEL file.

5.2 Simulating and Viewing the Output Waveforms

Click on the **File View** in the **Sources in Project** window. Then double-click on the test vectors file (newproje-vectors in our example). That file will be displayed on the right-hand side. Then in the **Processes View** window (left-bottom), double-click on the **Compile Test Vectors** option. Then double-click on the **Simulate Equations** option. When both are done successfully, a tick mark will be placed before each of them. Then click on the + mark in front of the **Simulate Equations** option to get the options: **Equation Simulation Report** and **Equation Simulation Waveform** as shown in Figure 12.

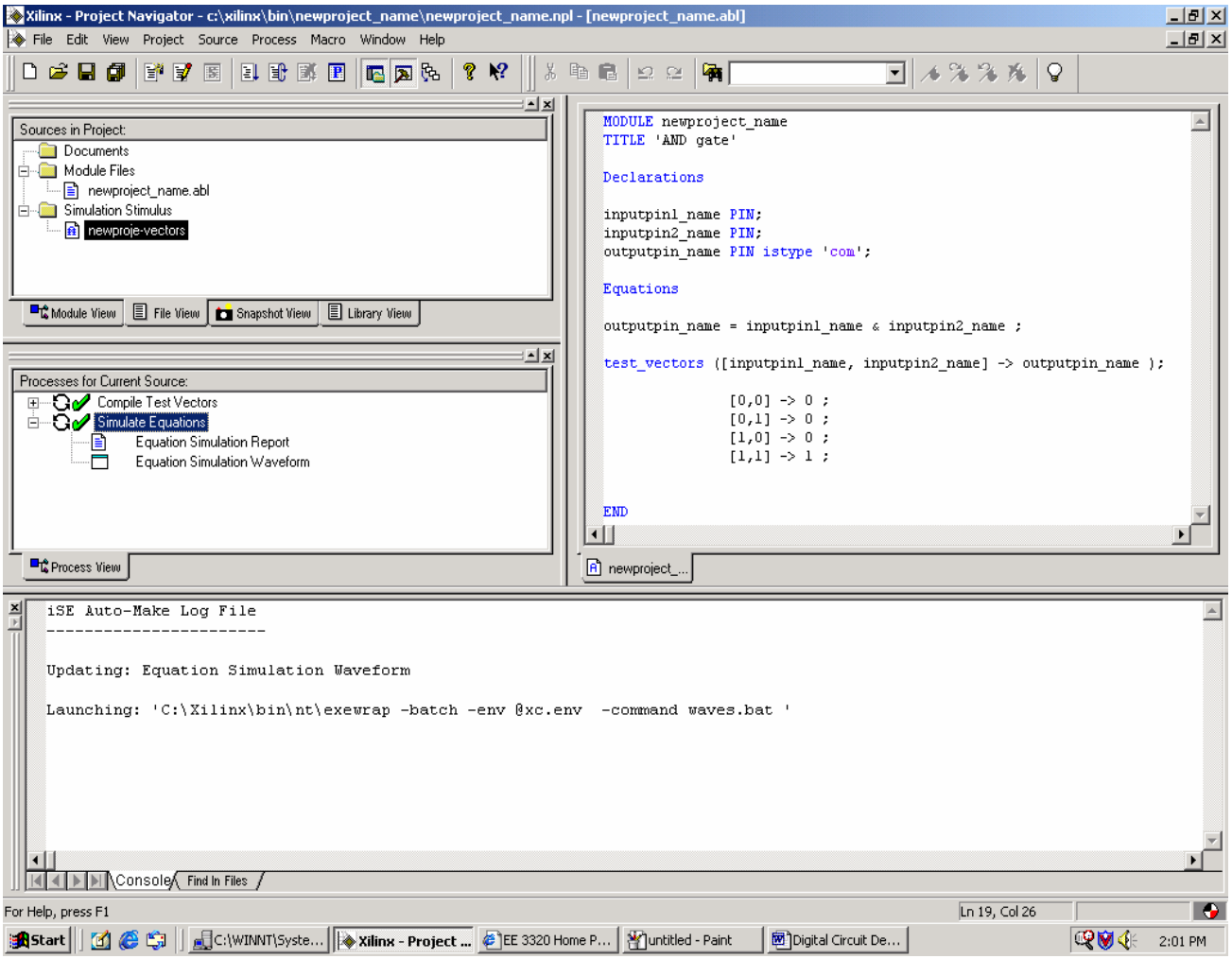

Figure 12 : Simulating the design (*snapshot from Xilinx ISE software)*

5.2.1 Equation Simulation Report

When this reporting option under the **Simulate Equations** option is double-clicked, a report window pops up and it shows which test vectors have passed and which test vectors have failed as shown in Figure 13.

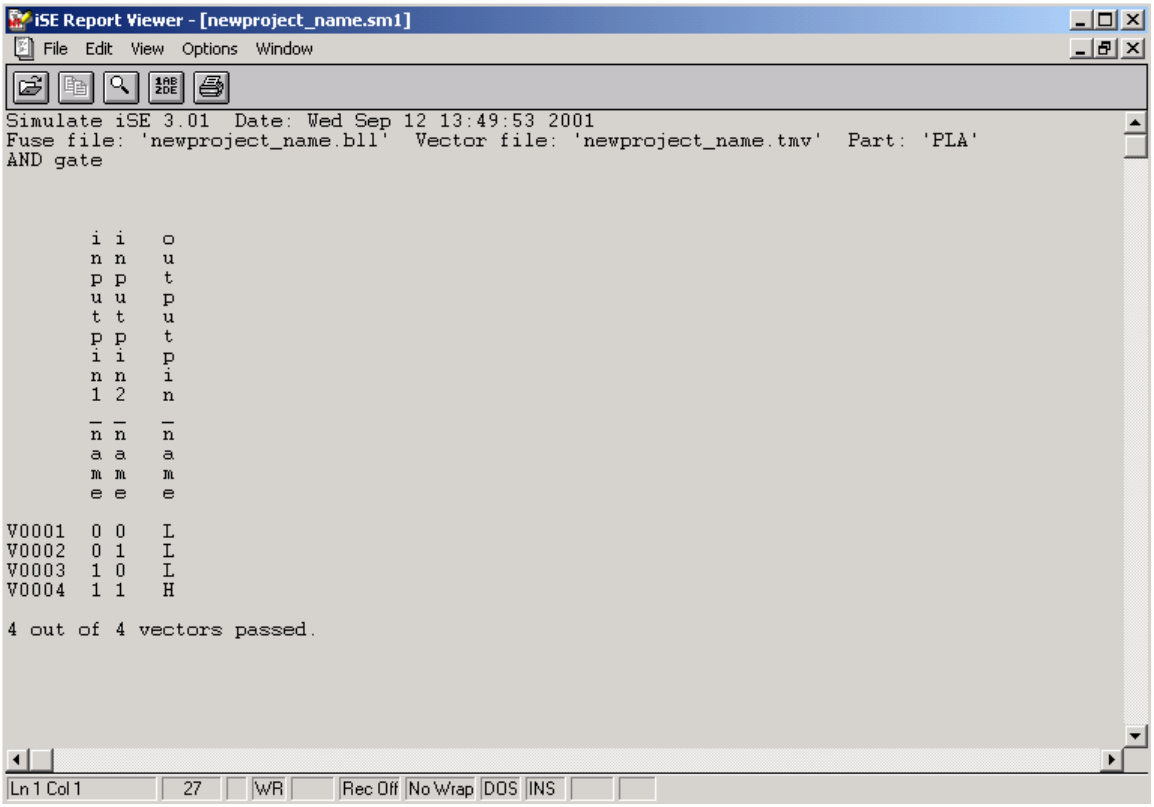

Figure 13 : Equation Simulation Report *(snapshot from Xilinx ISE software)*

As we have given correct values of output for all the test vectors, all of them have passed. This can be seen in the above figure. If there is an error (i.e. a mismatch between the actual and expected values), it will be indicated in this report window (Figure 13). In case of an error, first check if the test_vectors were typed in correctly and then debug the circuit.

5.2.2 Equation Simulation Waveform

When this option under **Simulate Equations** option is double-clicked, a window pops up with no waveform in it. Then select **Edit -> Show**. A form appears as shown in Figure 14. In that form, we can select all the signals to be displayed.

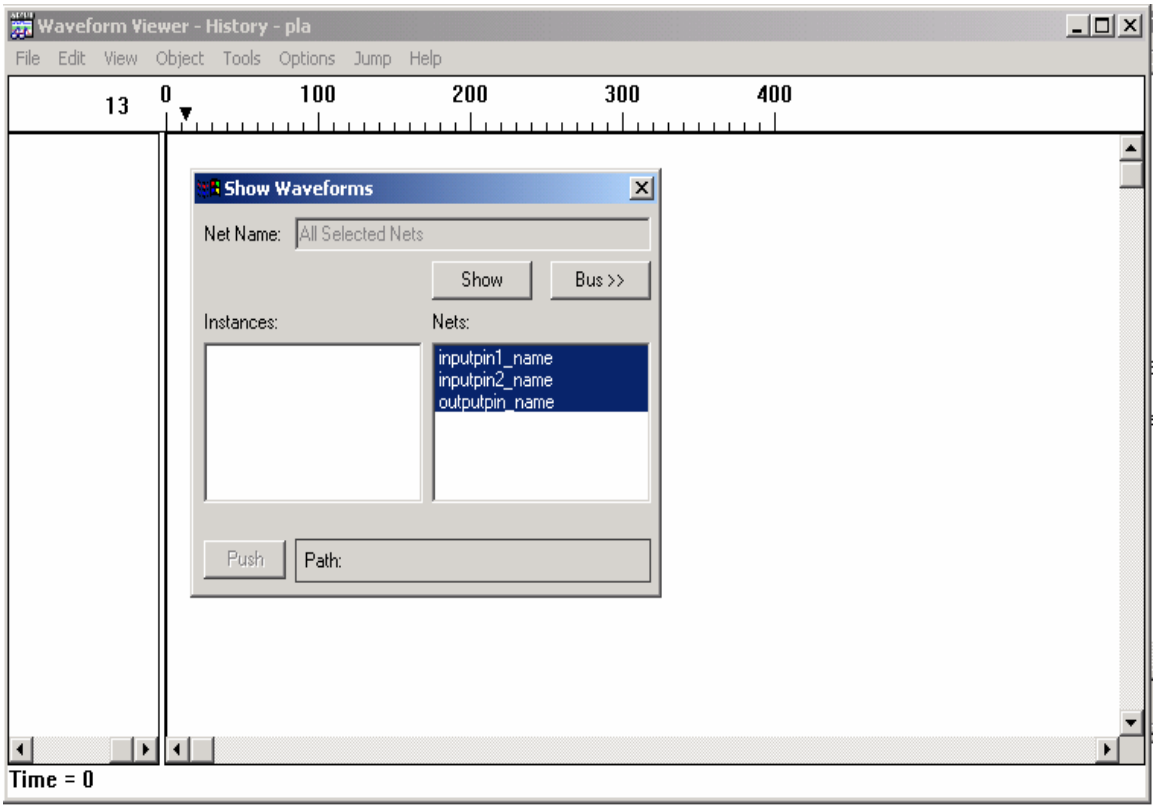

Figure 14 : Selecting the signals to be displayed (*snapshot from Xilinx ISE software)*

Press the **Ctrl** button on the keyboard and click on all the signals to be displayed, release the **Ctrl** button and click on **Show** in the form. Then a waveform appears on that **waveform viewer** window as shown in Figure 15. You can use the **Bus>>** button to collectively display a bunch of signals as a bus.

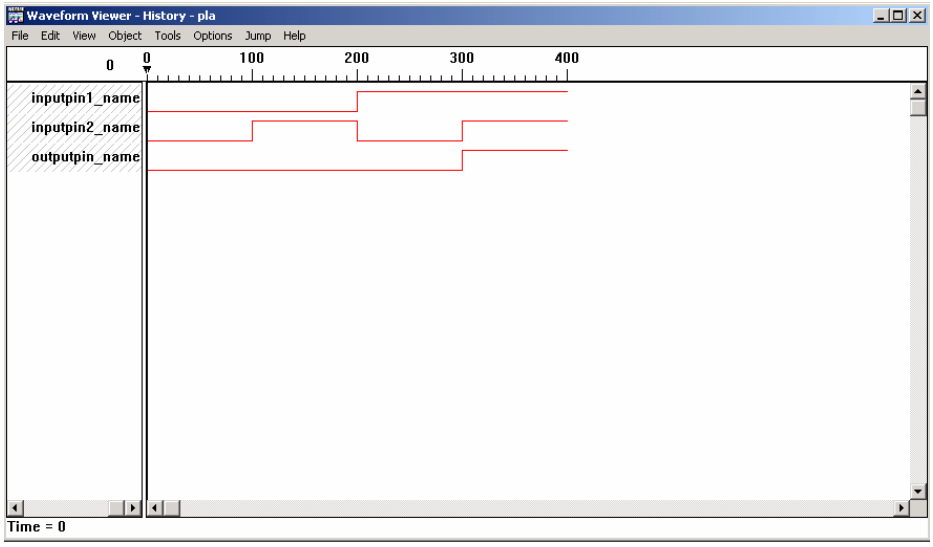

Figure 15 : Simulation results waveform *(snapshot from Xilinx ISE software)*

Various options like displaying some signals as a bus, placing some markers, setting some triggers, etc. are present in the waveform viewer. Explore these options for future use.

NOTE: The simulation waveform always gives the actual output for the test_vectors specified and not the expected values. The simulation report is the only place wherein mismatches between expected and actual values are indicated.

5.3 Saving the simulation results

To save the simulation results, select **File -> Save as** in the waveform viewer window and save the waveform. Then select **File -> Exit** to exit the waveform viewer and return to the Project Navigator.

6. Preparing bitstream for the XC95108 CPLD chip

A bitstream needs to be prepared for the design to be downloaded onto the prototyping board. This is done as follows:

- Assign pin numbers to the input and output pins in the ABEL – HDL design file. The pin numbers can be assigned by looking at section 8 of this tutorial. Then save the design file and implement the design again. ***Note that you can assign pin numbers only to top-level ABEL file.*** Let's assign some valid pin numbers for our example and implement it. Then the Project Navigator window looks as shown in Figure 16.

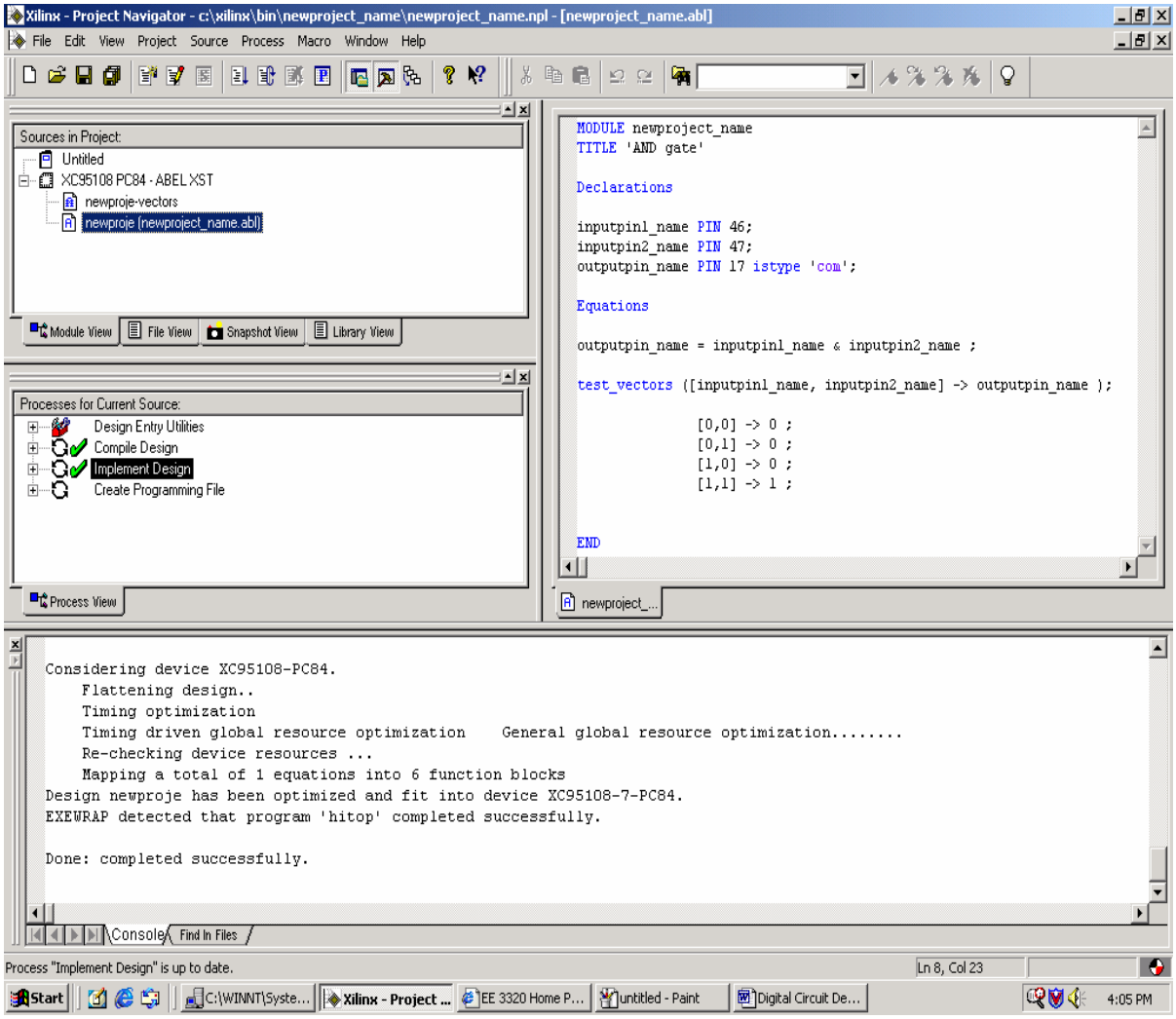

Figure 16 : Assigning I/O pin numbers (*snapshot from Xilinx ISE software*)

- Click on the **Module View** in the **Sources in Project** window. Then, in the **Process View** window (left-bottom), double-click on the **Create Programming File** option. Once it is done successfully, click on the + mark on its left to get **Launch JTAG Programmer** option. Double-click on it. That brings up the **Xilinx JTAG Programmer** window as shown in Figure 17.

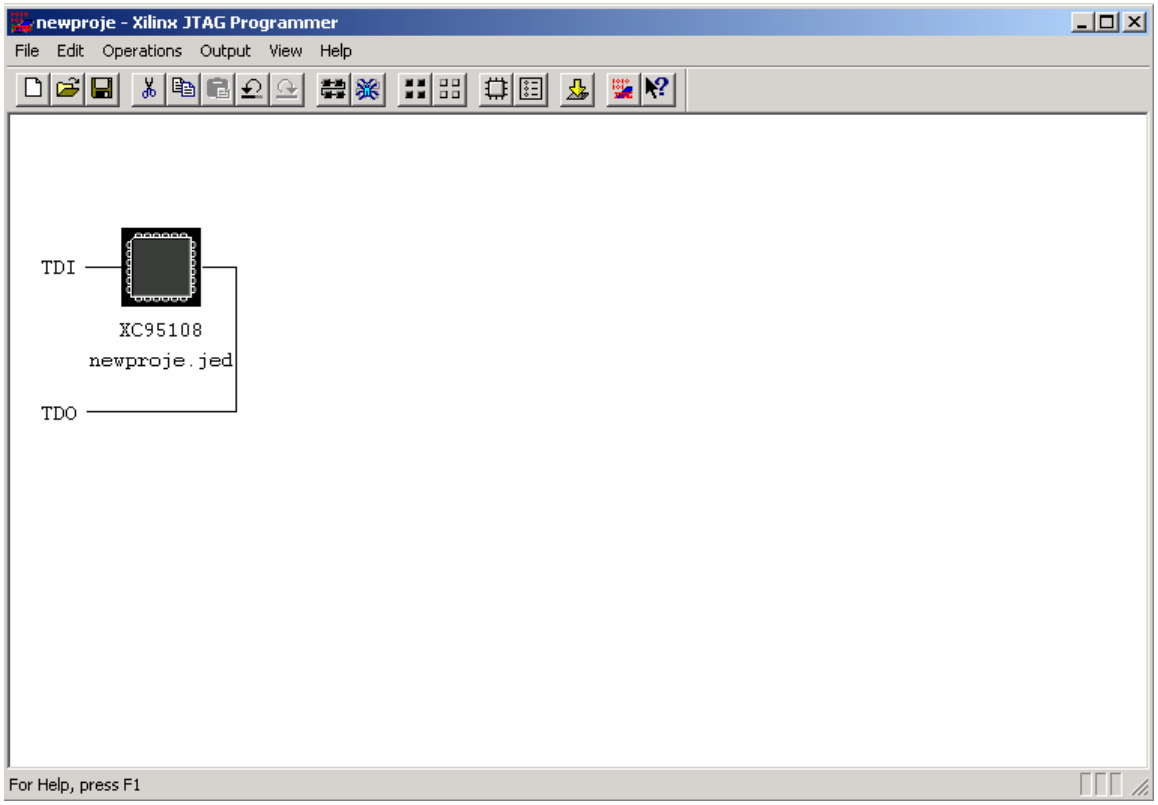

Figure 17 : JTAG Programmer window (*snapshot from Xilinx ISE software)*

- Select **Output->Create SVF File** menu item in the JTAG Programmer window. An SVF options window pops up, click on **OK** to accept the default settings. A pop-up window lets you specify the name and directory of the SVF file. (It is recommended that you place this file in the top level directory of your project so that it can be easily found later.) Click **Save** after choosing the file.
- Next, select **Operations->Program** from the menu. Click **OK** in the pop-up window that appears (don't change any of the other parameters). Then select File **-> Exit** to exit the JTAG programmer and return to the Project Navigator.

7. Downloading a Design to the Prototyping Board

After successfully creating the bitstream file (.svf file), configure the CPLD. This can be done in either of the following two ways:

7.1 Downloading through GXSTOOLS

- Select **Start -> Programs -> XSTOOLs -> GXSLOAD** from the Windows desktop or simply double-click on **GXSLOAD** icon on the Window desktop.

The following window pops up.

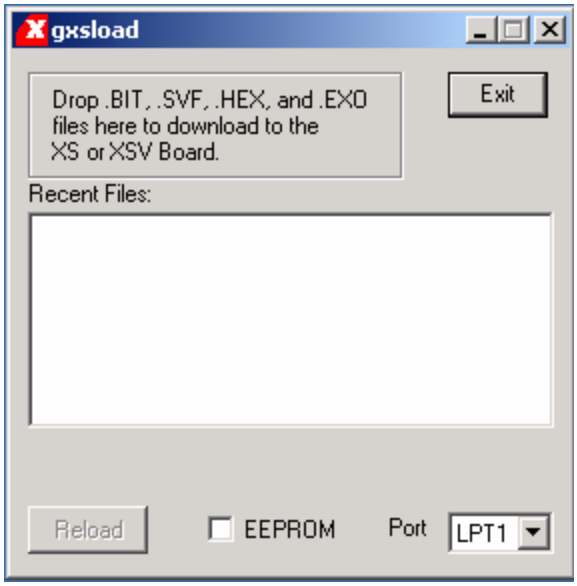

Figure 18 : GXSLOAD (*snapshot from GXSTOOLS software*)

- Select **Start -> Search-> For Files or Folders…**from the Windows desktop and search for the .svf file created in the above steps. Once that file is found, simply drag and drop that file onto the GXSLOAD window. Then the downloading starts and once it is done successfully, a message window appears indicating that the board is programmed.

7.2 Downloading through XSTOOLS

- Select **Start -> Run -> command** from the Windows desktop to open a **DOS** window. Then go to xstools directory in the C: drive. Then at the prompt, type

C:\XSTOOLS> xsload <.svf file file name with entire path>

For our example, it is xsload c:\xilinx\bin\newproject_name\newproje.svf

8. Testing a Digital Logic Circuit

Testing a downloaded design requires connecting the inputs of the design to switches or ports and the outputs of the design to LEDs or 7-segment displays. In case of sequential circuits, the clock input(s) must also be connected to clock sources. These inputs and outputs can be connected to appropriately on the Digital Lab workbench. Alternatively, the inputs can also be applied from the PC as described later in this section.

The next step in testing is to apply a sequence of test vectors to the inputs in the design and check the outputs of the design against what is expected. A test vector is a set of input values for the inputs of a design. The **XS40** and **XS95** prototyping boards have some built-in features that enable you to test your designs quickly and easily.

8.1 Observing outputs using the on-board LEDs

The **XS40** and **XS95** boards have one on-board 7-segment display (see Figure 19) that is connected to the corresponding on-board FPGA/CPLD chip. This display can be used to observe the outputs of your design without using any additional wires if the design conforms to the pin assignments for the on-board 7-segment display. If your design has more than 7 outputs, you will have to use other LEDs available on the Digital Lab workbench. The figure below shows the 7-segment display with the conventional labeling of individual segments.

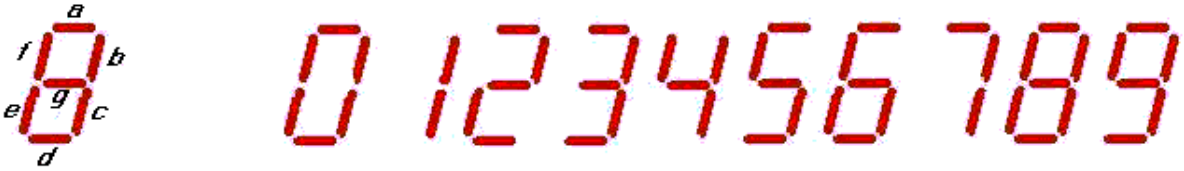

Figure 19: 7-segment display

For each prototyping board, Table 2 shows the FPGA/CPLD pin assignments for the LED segments of the on-board 7-segment display.

LED Segment	XC4010XL (XS40 Board)	XC95108 (XS95 Board)
S0 (seg. *d*)	pin# 25	pin# 21
S1 (seg. *c*)	pin# 26	pin# 23
S2 (seg. *e*)	pin# 24	pin# 19
S3 (seg. *g*)	pin# 20	pin# 17
S4 (seg. *b*)	pin# 23	pin# 18
S5 (seg. *f*)	pin# 18	pin# 14
S6 (seg. *a*)	pin# 19	pin# 15

Table 2. Pin assignments for 7-segment LEDs

8.2 Applying inputs using XSPORT/GXSPORT

Each board also has a few pins connected to the host PC through the parallel port. These pins can be used to apply input stimuli from the PC (DOS window) to test your design. The pins are also used for downloading design bitstreams. Input test vector can be applied to your design by using GXSPORT or XSPORT

- **GXSPORT**: Select **Start -> Programs -> XSTOOLs -> GXSPORT** from Windows desktop or simply double-click on the **GXSPORT** icon on the Windows desktop. The following window pops up.

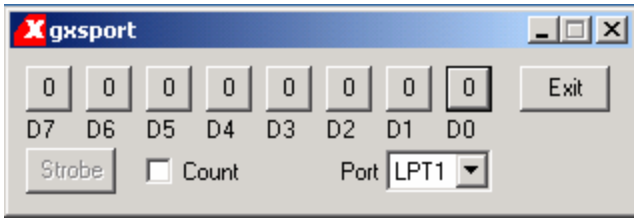

Figure 20 : GXSPORT (*snapshot from GXSTOOLS software*)

In the above GXSPORT window, we apply inputs by clicking on the bits from D7 to D0 and changing them as we need. D7….D0 correspond to B7….B0 of Table 3. When you click a 0, it changes to 1 and vice-versa. Change the inputs as needed and then click on the **Strobe** button to see the output on LED. When the **Count** option is selected, each click of the **Strobe** button increments the 8-bit value (D7…D0) by 1.

- **XSPORT** : Select **Start -> Run -> command** from the windows desktop open a DOS window. Change the directory to C:XSTOOLS. Then at the command prompt, type

```
C:\XSTOOLS> xsport  B7B6B5B4B3B2B1B0
```

For our example,as used only B1 and B0 pins by giving the pin numbers 47 and 46 (refer to Table 3) , we can set the inputs as

C:\XSTOOLS> xsport 00

C:\XSTOOLS> xsport 01 , etc..

Note: If you apply less than 8 bits, **XSPORT** will set the remaining the most significant bits to zero. If your design needs more than 8 input bits, you will have to apply them by using switches on the Digital Lab workbench. Table 3 lists the pins on each board that are connected to the parallel port (XSPORT).

XSPORT Argument	XC4010XL (XS40 Board)	XC95108 (XS95 Board)
B0	pin# 44	pin# 46
B1	pin# 45	pin# 47
B2	pin# 46	pin# 48
B3	pin# 47	pin# 50
B4	pin# 48	pin# 51
B5	pin# 49	pin# 52
B6	pin# 32	pin# 81
B7	pin# 34	pin# 80

Table 3. Pin assignments for XSPORT parallel port

So, for our example,

XSPORT Test Command	LED segment S3 (pin 17)
Xsport 00	0
Xsport 01	0
Xsport 10	0
Xsport 11	1

9. Design and Simulation of Sequential Circuits using ABEL HDL

The procedure to create ABEL HDL design files for sequential circuits is the same as that for combinational circuits. The main differences between combinational and sequential designs are: *(i)* definition of flip-flops (**registered** outputs or nodes in the Declaration section of a sequential design), *(ii)* state assignment, and *(iii)* specification of state machine (*Mealy* or *Moore*) in the Logic Description section. In an ABEL program, a sequential circuit can be specified by using a ***state transition table*** or a ***state machine program.***

9.1 Specification using a State Transition table

Simple sequential circuits can be specified as state transition tables because they do not have a large number of states and transitions. A state transition table entry is similar to a truth table entry for combination circuits. The syntax for a transition table entry is:

```
[Present State, Inputs] :> [Next State] -> [Outputs]
```

The example below shows the ABEL program for a sequence detector that detects the sequence **101** on its input. It produces a **1** on its output for one clock cycle when it detects the sequence. It will keep checking for the proper bit sequence and does not reset to the initial state after it has recognized the string. E.g., for input, **`INP="...110110101...",`** will generate the output, **`OUT="...000100101..."`**. The detector initializes to a reset state when input, **`RESET`** is activated.

```
MODULE seq_det
TITLE 'Detects sequence 101'

DECLARATIONS
C       PIN;                "clock input
RESET   PIN;                "reset input
INP     PIN;                "input port for sequence
OUT     PIN ISTYPE 'com';   "Output port
Q1..Q0  PIN ISTYPE 'reg';   "2 D flip-flops for state

EQUATIONS
[Q1..Q0].CLK = C;  “Connect clock input to D-flip-flops

TRUTH_TABLE
([Q1,Q0,RESET,INP] :> [Q1,Q0] -> OUT)
 [0, 0, 0, 0] :> [0, 0] -> 0;   "State 0, Input 0
 [0, 0, 0, 1] :> [0, 1] -> 0;   "State 0, Input 1
 [0, 0, 1, 0] :> [0, 0] -> 0;   "State 0, Reset
 [0, 0, 1, 1] :> [0, 0] -> 0;
 [0, 1, 0, 0] :> [1, 0] -> 0;   "State 1, Input 0
 [0, 1, 0, 1] :> [0, 1] -> 0;   "State 1, Input 1
```

```
 [0, 1, 1, 0] :> [0, 0] -> 0;   "State 1, Reset
 [0, 1, 1, 1] :> [0, 0] -> 0;
 [1, 0, 0, 0] :> [0, 0] -> 0;   "State 2, Input 0
 [1, 0, 0, 1] :> [0, 1] -> 1;   "State 2, Input 1
 [1, 0, 1, 0] :> [0, 0] -> 0;   "State 2, Reset
 [1, 0, 1, 1] :> [0, 0] -> 0;
 [1, 1, 0, 0] :> [0, 0] -> 0;   "Unreachable states
 [1, 1, 0, 1] :> [0, 0] -> 0;
 [1, 1, 1, 0] :> [0, 0] -> 0;
 [1, 1, 1, 1] :> [0, 0] -> 0;

TEST_VECTORS  ([C,Q1,Q0,RESET,INP] -> [Q1,Q0,OUT]0
                [.C., 0, 0, 0, 0] -> [0, 0, 0];
                [.C., 0, 0, 0, 1] -> [1, 0, 0];
                [.C., 1, 0, 0, 0] -> [0, 0, 0];
                [.C., 0, 0, 1, 0] -> [0, 0, 0];
                [.C., 0, 0, 1, 1] -> [0, 0, 0];

END seq_det
```

9.2 Specification using a State Machine program

For large, complex state machines it is easier to specify them as programs. This requires an enumeration of states in the Declaration section and the state machine program description in the Logic Description section. Like any programming language, the state machine program uses *relational operators* to test for equality, less than or greater than (as shown in the example below). The example below shows the ABEL program for the sequence detector described in section 9.1

```
MODULE seq_det1
TITLE 'Detects sequence 101'

DECLARATIONS
C       PIN;                "clock input
RESET   PIN;                "reset input
INP     PIN;                "input port for sequence
OUT     PIN ISTYPE 'com';   "output port
Q1..Q0  PIN ISTYPE 'reg';   "2 D flip-flops for state

"State Assignment Table
"---- Q1, Q0
ST0 = [0, 0];
ST1 = [0, 1];
ST2 = [1, 0];

EQUATIONS
[Q1..Q0].CLK = C;         "Connect clock input to D-FFs
[Q1..Q0].ACLR = RESET;    "Asynchronous clear FFs with Reset

STATE_DIAGRAM [Q1, Q0]
  STATE ST0:
    IF (INP == 1) THEN ST1
    ELSE ST0;
```

```
  STATE ST1:
    IF (INP == 0) THEN ST2
    ELSE ST1;
  STATE ST2:
    IF (INP == 1) THEN ST1 WITH OUT = 1
    ELSE ST0;

TEST_VECTORS  ([C,Q1,Q0,RESET,INP] -> [Q1,Q0,OUT]0
                [.C., 0, 0, 0, 0] -> [0, 0, 0];
                [.C., 0, 0, 0, 1] -> [1, 0, 0];
                [.C., 1, 0, 0, 0] -> [0, 0, 0];
                [.C., 0, 0, 1, 0] -> [0, 0, 0];
                [.C., 0, 0, 1, 1] -> [0, 0, 0];


END seq_det1
```

9.3 Simulation of sequential designs

Except for the additional clock signal, simulation of sequential designs can be done using test_vectors in the same way it was done for combinational. Look at the example shown in Figure 21. For each set of inputs, a clock pulse is applied and the resulting simulation waveform is as shown in Figure 22. In the test_vectors, the expected output value to be given is the one corresponding to the rising edge of the clock (Note: All the flip-flops defined by " istype 'reg'" in the ABEL file are rising-edge triggered)

```
MODULE seq
TITLE 'some sequential circuit'

ck PIN 46;
d PIN 47;
reset PIN 48;
q PIN 17 istype 'reg';

EQUATIONS

q.clk = ck;
q.aclr = reset;

q:=d;

test_vectors ([reset,d,ck] -> q)

             [0,0,.c.] -> 0;
             [0,1,.c.] -> 1;
             [1,0,.c.] -> 0;
             [1,1,.c.] -> 0;

END
```

Figure 21 : A simple D-Flipflop example *(snapshot from Xilinx ISE software)*

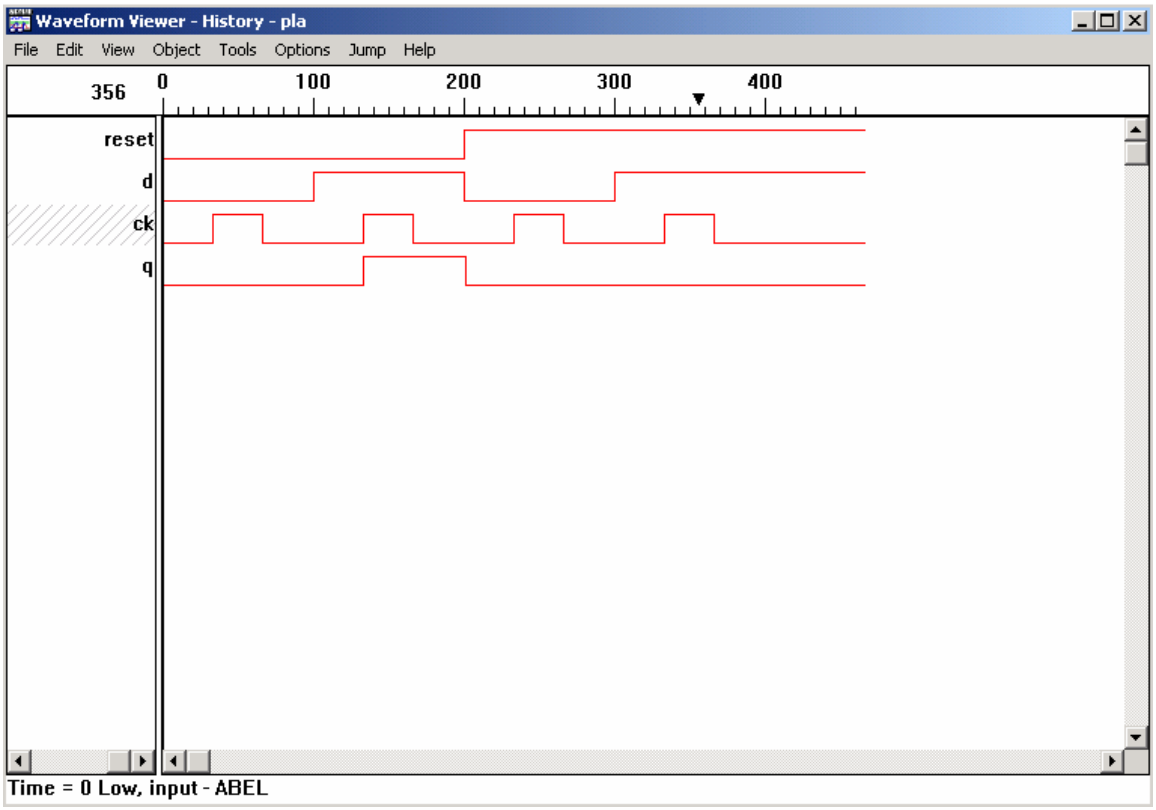

Figure 22: Simulation results of the above example (*snapshot from Xilinx ISEsoftware)*

10. Hierarchical Circuit Design Using Modules

It is always a good practice to keep a design modular and hierarchical. This is important for designs of moderate to high complexity. Often, you will use a circuit (module) over and over again. Instead of creating these modules every time you need them, it would be more efficient to make a cell or module out of them. You can then use this module every time to need it by instantiating the module in your circuit. ABEL HDL supports hierarchical design by creating *modules* of circuit components that can be used in another design. A module is specified like other designs with an additional **`INTERFACE`** specification in the Header section. In the example depicted in Figure 21, a 4-bit equivalence circuit is designed using 1-bit equivalence circuit modules.

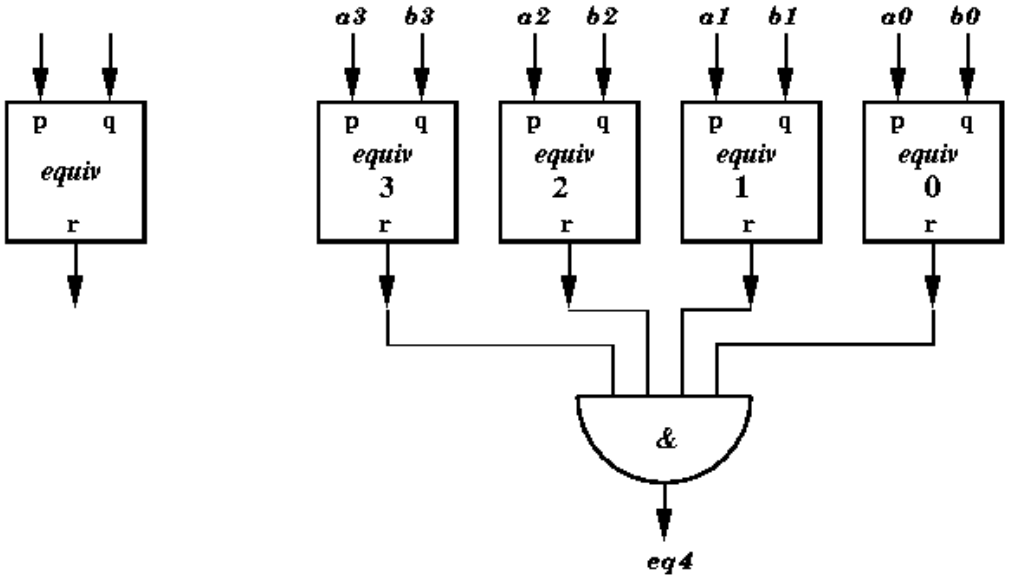

Figure 21: Hierarchical circuit design example: 4-bit equivalence circuit

- **Module Definition:** A module (functional block) definition is specified in a file, separate from the top-level design file using the module.

```
MODULE equiv
INTERFACE (p,q -> r);     "interface spec: p,q inputs: r output
TITLE '1-Bit Equivalence Circuit'
```

```
DECLARATIONS
p,q   PIN;
r     PIN ISTYPE 'com';

EQUATIONS
r = (p & q) # (!p & !q);

END equiv
```

- **Module Usage:** A design using a module includes a declaration of module interface and instantiation of each module in the Declaration section. There should not be any **INTERFACE** specification in the top-level design. (**NOTE:** *There is a limitation in Student Version of ABEL based Xilinx software because of which it cannot handle more than four functional_block statements at the top level.*)

```
MODULE equiv4bit
TITLE '4-bit Equivalence Circuit'

DECLARATIONS
a3..a0   PIN;                "4-bits of Input1
b3..b0   PIN;                "4-bits of Input2
eq4      PIN ISTYPE 'com';   "Output bit

"declare modules to be used
equiv INTERFACE (p,q -> r);

"instantiation of modules
equiv3, equiv2, equiv1, equiv0 FUNCTIONAL_BLOCK equiv;

EQUATIONS

"top level interconnections
equiv0.p = a0;    "Module 0
equiv0.q = b0;

equiv1.p = a1;    "Module 1
equiv1.q = b1;

equiv2.p = a2;    "Module 2
equiv2.q = b2;

equiv3.p = a3;    "Module 3
equiv3.q = b3;

eq4 = equiv3.r & equiv2.r & equiv1.r & equiv0.r;

END equiv4bit
```

NOTE : *A module is created in the same way as the top-level ABEL file. We can either use the design wizard provided by the Xilinx or create our own.*