



## MATrix-II

**VLSI Protoboard**  
**Version: 1**

**ni logic Pvt. Ltd.,**  
25/B5, Bandal Complex,  
Paud Road, Kothrud,  
Pune – 411 038.  
Maharashtra.

[www.ni2designs.com](http://www.ni2designs.com)

**Preface:**

This manual is for the users of MATrix-II with in-depth details of product including reference designs and examples.

**Copyrights:**

We acknowledge that the following organizations claim trademark rights in their respective products or services mentioned in this document, specifically:

Altera, Quartus-II, ACEX, ByteBlaster, etc

Xilinx, ISE, iMPACT, Spartan-II, etc.

**For further details and queries contact:**

ni logic Pvt. Ltd,  
25/B-5, Bandal Complex,  
Bhusari Colony, Paud Road, Kothrud,  
Pune – 411 038,  
Maharashtra, India

Tele-Fax: +91-20-25286948

Ph: +91-20-30904378

[info@ni2designs.com](mailto:info@ni2designs.com)

[www.ni2designs.com](http://www.ni2designs.com)

# MATrix-II VLSI Protoboard

## User Manual

### **Table of contents**

1. Introduction
  - a. Introduction to Programmable logic and PLDs
  - b. Product Introduction
2. Features and Specifications
3. Diagrams
  - a. I/O Connections for Xilinx FPGA
  - b. I/O Connections for Altera FPGA
  - c. I/O Connections for CPLDs
  - d. Baseboard legend diagram
4. Precautions
5. Design Flow
  - a. PLD design flow
  - b. Microcontroller design flow
  - c. Xilinx webpack design flow
  - d. Altera Quartus-II design flow
  - e. Keil design flow
6. Configuration/Downloading
  - a. Types of Modes
  - b. Jumper setting for mode selection
  - c. Using Variable frequency generator
  - d. Programming Atmel Microcontroller
7. Pin Assignments
8. Jumper settings
9. Using External Adaptors
10. Sample codes
11. Glossary

## Chapter 1: Introduction

### **1.a What is Programmable Logic?**

In the world of digital electronic systems, there are three basic kinds of devices: memory, microprocessors, and logic. Memory devices store random information such as the contents of a spreadsheet or database. Microprocessors execute software instructions to perform a wide variety of tasks such as running a word processing program or video game. Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform.

### **Fixed Logic versus Programmable Logic**

Logic devices can be classified into two broad categories - fixed and programmable. As the name suggests, the circuits in a fixed logic device are permanent, they perform one function or set of functions - once manufactured, they cannot be changed. On the other hand, programmable logic devices (PLDs) are standard, off-the-shelf parts that offer customers a wide range of logic capacity, features, speed, and voltage characteristics - and these devices can be changed at any time to perform any number of functions.

With fixed logic devices, the time required to go from design, to prototypes, to a final manufacturing run can take from several months to more than a year, depending on the complexity of the device. And, if the device does not work properly, or if the requirements change, a new design must be developed. The up-front work of designing and verifying fixed logic devices involves substantial "non-recurring engineering" costs, or NRE. These NRE costs can run from a few hundred thousand to several million dollars.

With programmable logic devices, designers use inexpensive software tools to quickly develop, simulate, and test their designs. Then, a design can be quickly programmed into a device, and immediately tested in a live circuit. There are no NRE costs and the final design is completed much faster than that of a custom, fixed logic device.

Another key benefit of using PLDs is that during the design phase customers can change the circuitry as often as they want until the design operates to their satisfaction. That's because PLDs are based on re-writable memory technology - to

change the design, the device is simply reprogrammed. Once the design is final, customers can go into immediate production by simply programming as many PLDs as they need with the final software design file.

## CPLDs and FPGAs

The two major types of programmable logic devices are field programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs). Of the two, FPGAs offer the highest amount of logic density, the most features, and the highest performance. The largest FPGA provides millions of "system gates" (the relative density of logic). These advanced devices also offer features such as built-in hardwired IP cores (such as the IBM Power PC, PCI cores, microcontrollers, peripherals, etc), substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to-device signaling technologies. FPGAs are used in a wide variety of applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing.

CPLDs, by contrast, offer much smaller amounts of logic - up to about 10,000 gates. But CPLDs offer very predictable timing characteristics and are therefore ideal for critical control applications. Low power CPLDs are also available and are very inexpensive, making them ideal for cost-sensitive, battery-operated, portable applications such as mobile phones and digital handheld assistants.

## The PLD Advantage

Fixed logic devices and PLDs both have their advantages. Fixed logic devices, for example, are often more appropriate for large volume applications because they can be mass-produced more economically. For certain applications where the very highest performance is required, fixed logic devices may also be the best choice.

However, programmable logic devices offer a number of important advantages over fixed logic devices, including:

- PLDs offer customers much more flexibility during the design cycle because design iterations are simply a matter of changing the programming file, and the results of design changes can be seen immediately in working parts.
- PLDs do not require long lead times for prototypes or production parts - the PLDs are already on a distributor's shelf and ready for shipment.
- PLDs do not require customers to pay for large NRE costs and purchase expensive mask sets - PLD suppliers incur those costs when they design

their programmable devices and are able to amortize those costs over the multi-year lifespan of a given line of PLDs.

- PLDs can be reprogrammed even after a piece of equipment is shipped to a customer. In fact, thanks to programmable logic devices, a number of equipment manufacturers now have the ability to add new features or upgrade products that already are in the field. To do this, they simply upload a new programming file to the PLD, via the Internet, creating new hardware logic in the system.

## Conclusion

The value of programmable logic has always been its ability to shorten development cycles for electronic equipment manufacturers and help them get their product to market faster. As PLD suppliers continue to integrate more functions inside their devices, reduce costs, and increase the availability of time-saving IP cores, programmable logic is certain to expand its popularity with digital designers.

## 1.b Product Information

The **MATrix-II ( Multiple Application Tricks)** is a low cost universal platform for testing and verifying designs based on the Xilinx and Altera PLDs. The purpose of **MATrix-II** is to teach the basic concepts of VLSI designing along with various electronics circuits. The **MATrix-II** has been revised and also extended to some basic electronic circuits for application development and their realization. Using this protoboard the user can verify his PLD designs with complete applications and also it gives a complete set of modules for project development for final year students.

**MATrix-II** supports multiple vendor devices from Xilinx and Altera, who are world leader in PLD manufacturing. The **MATrix-II** supports **Spartan-2** and **XC9500** series of devices from Xilinx; and **ACEX 1K** and **MAX7000s** series of devices from Altera.

The basic version of **MATrix-II** comes with 50,000 gate **XC2S50-PQ208** FPGA, **XC9572** or **MAX7128S** CPLD and **EP1k50TQ144** FPGA from Altera along with supporting circuitry to ease prototype efforts (optional PLD modules available).

**MATrix-II** comes along with various adaptors, which are optional to user. Every adaptor is pluggable with baseboard with the help of connectors.

1. Keypad adaptor
2. LCD adaptor
3. Dot matrix rolling display card
4. Relay card
5. 89C51 adaptor
6. ADC/DAC adaptor

With these adaptors user expands his choice and features to prototype and solve his design needs and requirements.

Above modules are optional to the baseboard and can be used by plugging into it.

## Chapter 2: Features & Specifications

### 2.a Features and Specifications

- Multi-vendor device support for Xilinx and Altera PLDs.
- Packages supported PLCC84, TQ144 and PQ208.
- Up to 140 user I/Os.
- Voltage support to +1.2V, +2.5V, +3.3V & +5V devices,
- All FPGA I/Os accessible through headers.
- Four Multiplexed 7-Segment displays (with segment map).
- Interface to RS232 with 9-pin D-type connector.
- User selectable configuration modes, using FLASH PROM / JTAG / Slave Serial.
- Byte-blaster cable interface for configuration of Altera FPGAs.
- On board 8-MHz Clock oscillator (user selectable).
- Variable frequency generator (from 100 Hz to 10 KHz range).
- Higher frequency board support.
- Configurable 24 switches as I/P or O/P.
- 16 digital LED indicated outputs.
- Power on Reset and configuration reset key.
- Support for different I/O Standards.
- 4x4 Keyboard matrix card.
- Interface to Atmel AT89s8252 microcontroller.
- Facility for I<sup>2</sup>C interface.
- 8-bit ADC/DAC add-on card.
- Four 5x7 Dot Matrix displays.
- Optically isolated relay card.
- 16x2 character LCD display with contrast control.
- Short circuit protection circuit.

## 2.b Individual Module Specification \*

- **Keyboard adaptor**
  - 4x4 membrane keypad
- **Liquid Crystal Display (LCD) adapter**
  - 16 x 2 characters LCD display with Contrast control
- **Relay Card**
  - 2 Optically isolated relays
  - NC, NO, COMM I/Os on power header
  - Relay ON indication
- **Dot Matrix rolling display panel**
  - Matrix of four 5x7 LED display
  - Total 140 LED matrix on board.
- **Pluggable Micro controller Card**
  - Atmel AT89S8252 ISP microcontroller
  - 8KB ISP Flash & 2KB of EEPROM.
  - Coupled with FPGA for embedded application development.
  - All 32 I/O lines accessible to FPGA.
  - On board reset circuit
  - Timer, interrupt and ISP ports.
- **ADC/DAC Card**
  - 8 channel ADC 0809 with sampling speed of 20KHz on single channel.
  - Single channel DAC0800 with 150ns settling time.
  - Facility for onboard gain and reference voltage adjustment.
- **Xilinx CPLD Module**
  - XC9572 PC84-15C containing 72 macrocells with 1,600 usable gates.
- **Xilinx FPGA Module**
  - 30,000 gate density XC2S30 PQ208-5 **FPGA from Xilinx**.
- **Altera CPLD Module**
  - EPM7128SLC84-15C device containing 128 macrocells with 2,500 usable gates.
- **Altera FPGA Module**
  - 50,000 gate density EP1K50 TQ144-3C **FPGA from Altera**.

**\*Note: Above module are optional with the product.**

## Power supply

Required voltages are generated onboard through regulators; other supply voltages are applied from external power supply.

Here is the list of voltages on board used.

+12V

+5V

-5V

+3.3V

+2.5V

+1.2V

**\* Note:** The above specifications and features of product are subject to change with new versions of product.

## Connectors

Header Name	Ident
Relay Header	JP5
Digital I/Os and Rolling display	JP3
8051 Header	JP2
LCD Header	JP4
ADC/DAC & Keypad	JP1
PLD Header	JH1, JH2, JH3, JH4
Power supply	JP6

## Jumpers

Jumpers are provided on baseboard for selection of

1. Configuration mode pins
2. Bypassing the PROM
3. Selecting configurable Input or Output
4. Selecting the O/P LEDs.

- J8-J11 Mode selection headers
- J6, J7 PROM bypass
- S0-S7 SW1
- S8-S15 SW2
- S16-S23 SW3

## Downloading cable

### **For Xilinx PLDs**

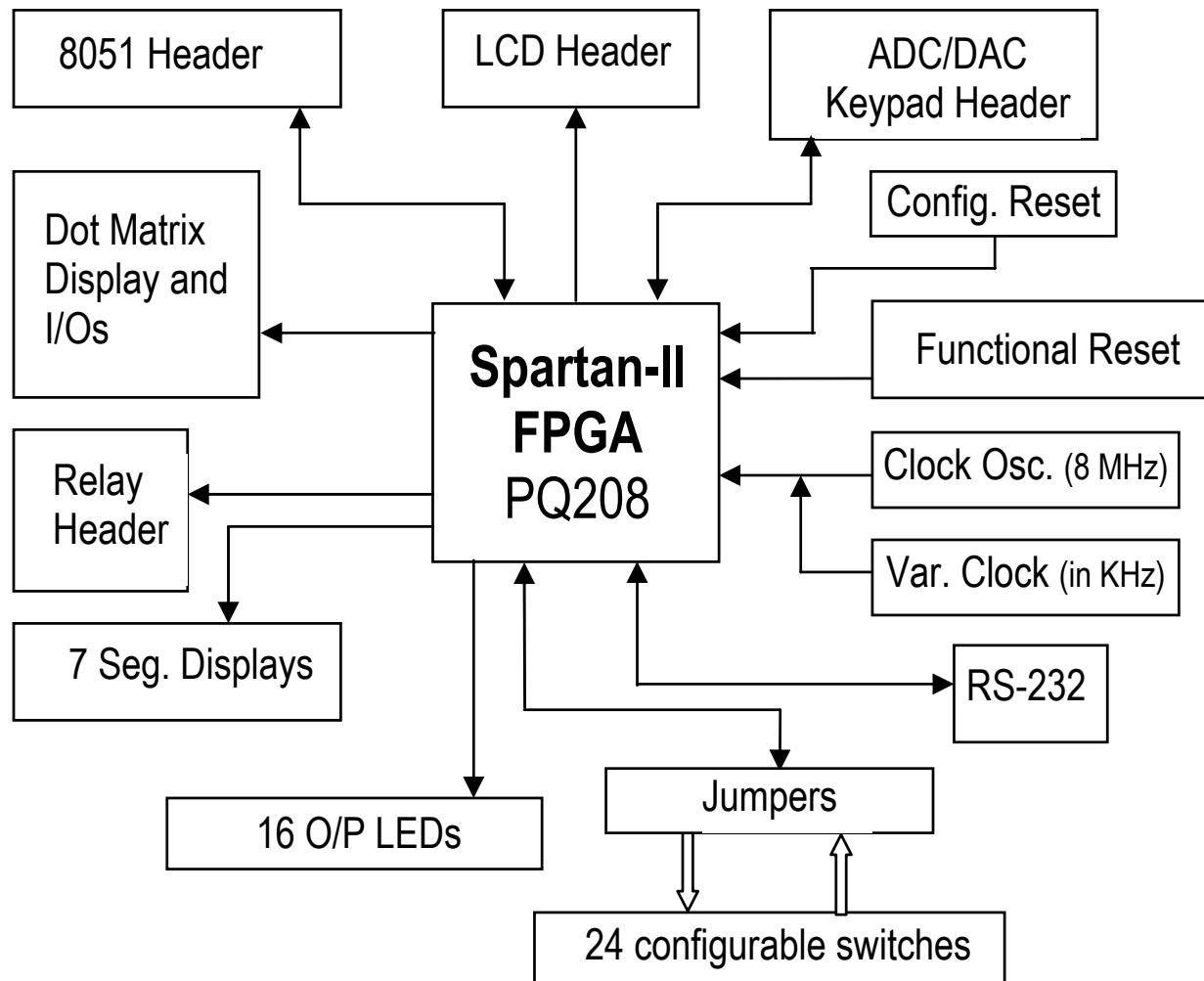
For configuration of Xilinx FPGA and CPLD from PC, a 9 pins D Type connector is provided on baseboard. The **MATrix** can be connected to PC's parallel port with cable provided having 25 pins D Type connector on other end.

### **For Altera PLDs**

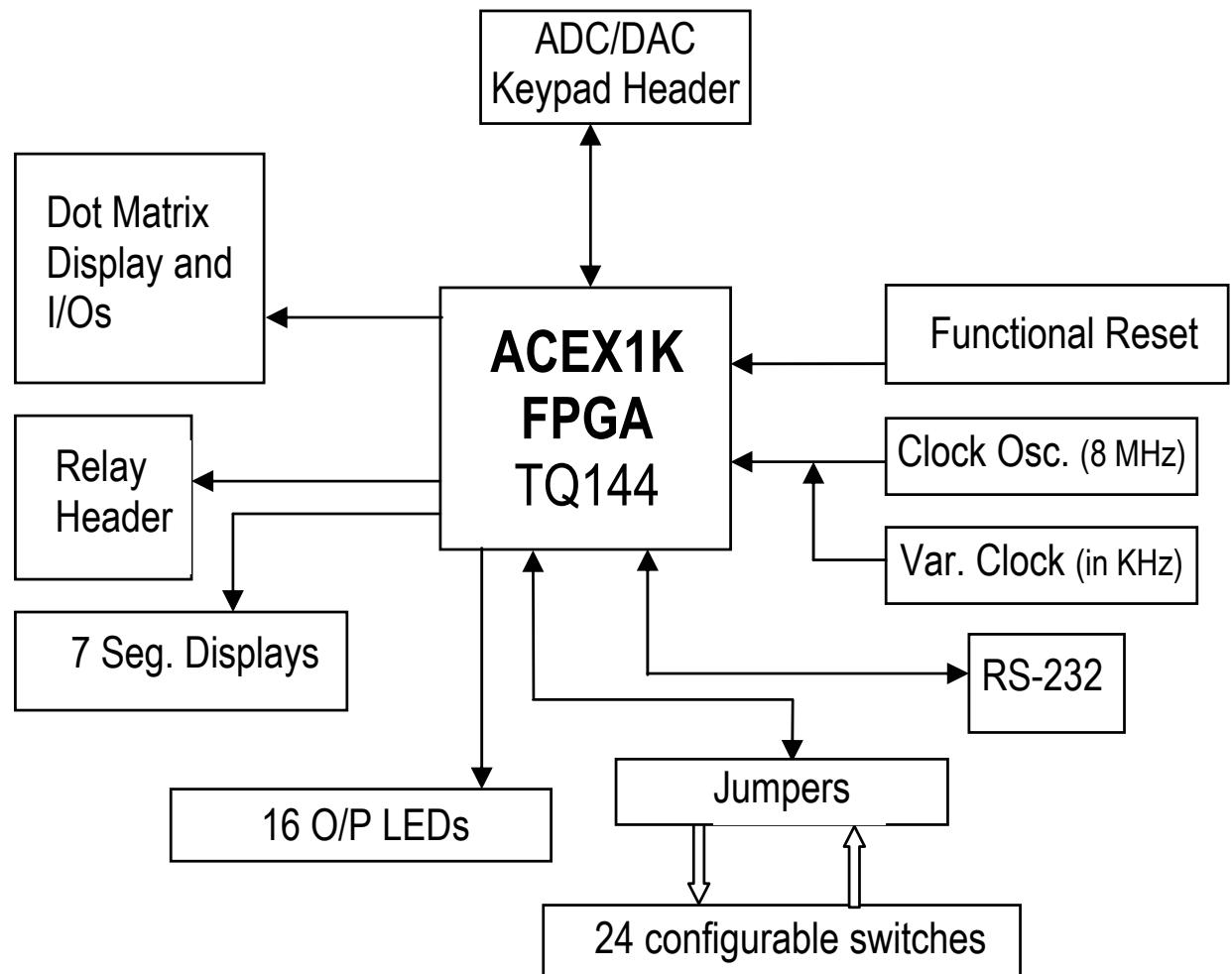
Altera PLD adaptors have onboard JTAG header. User has to connect the programming JTAG cable provided on this header and other end to PCs parallel port to configure the PLDs.

**\*Note:** Kindly remove the cables by its headers only. Removing the cables by handling its wire may cause damage to its joints.

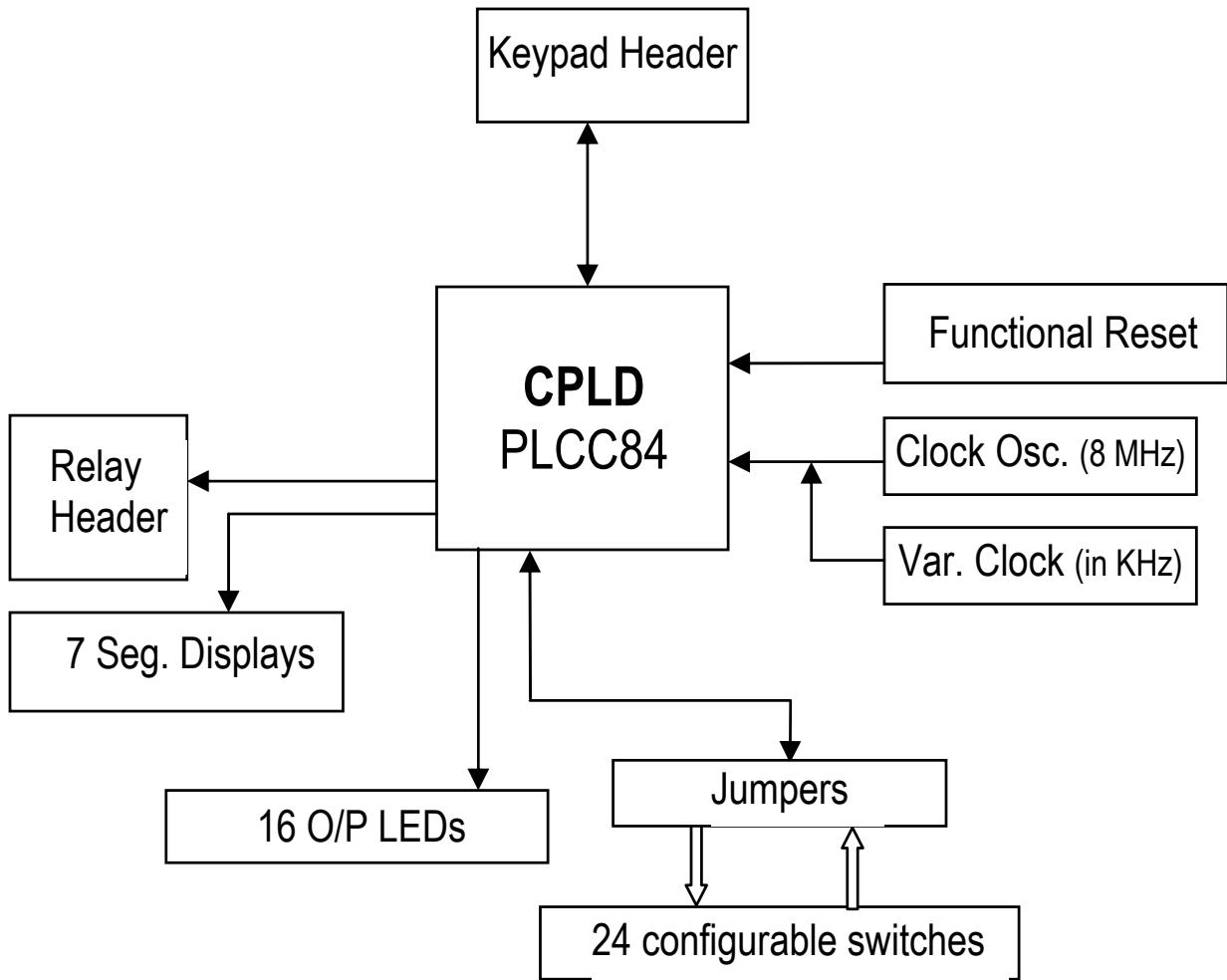
## Chapter 3: Diagrams



3.a System Connection Diagram (Xilinx FPGA)

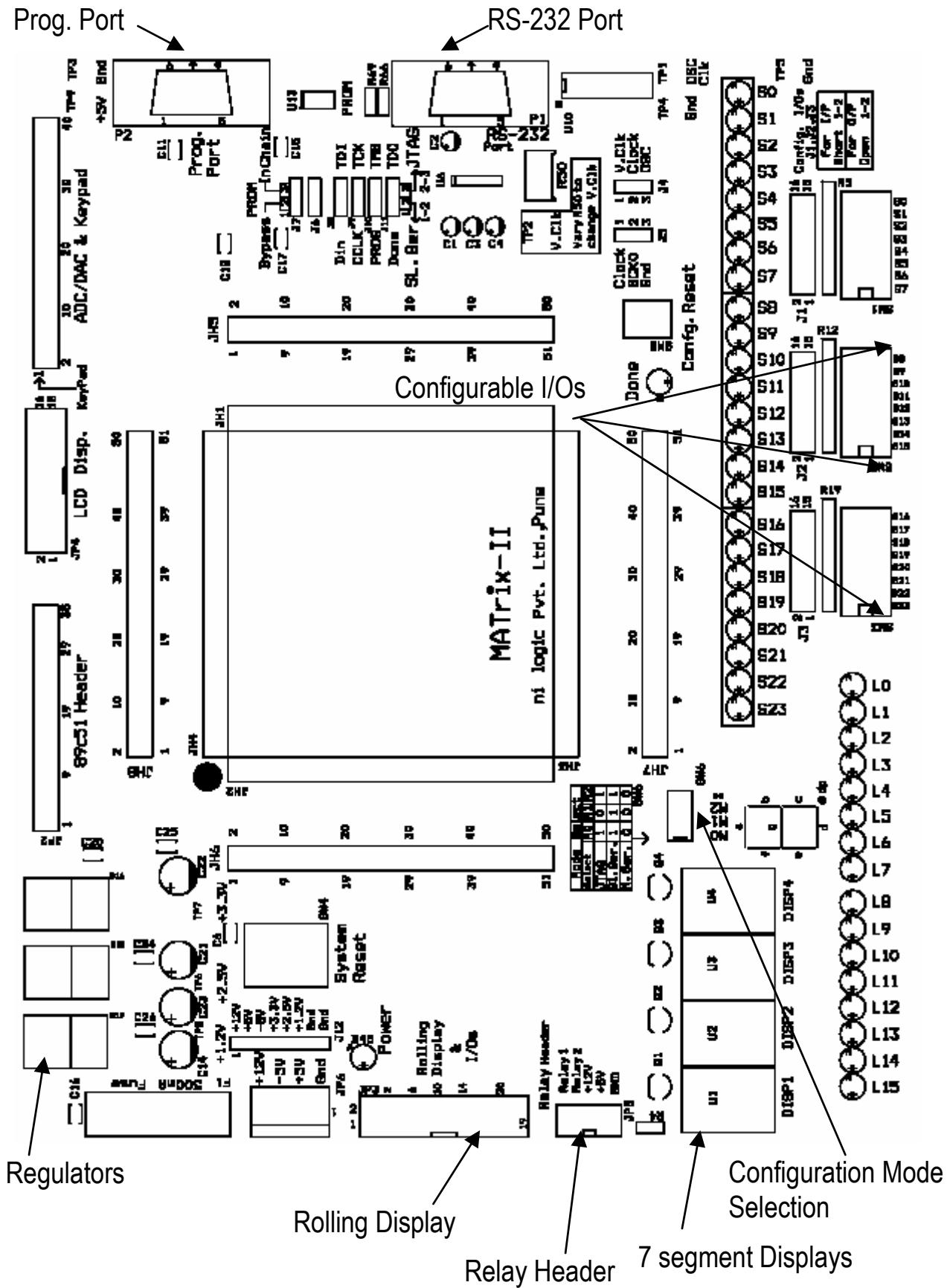


### 3.b System Connection Diagram (Altera FPGA)



### 3.c System Connection Diagram (CPLDs)

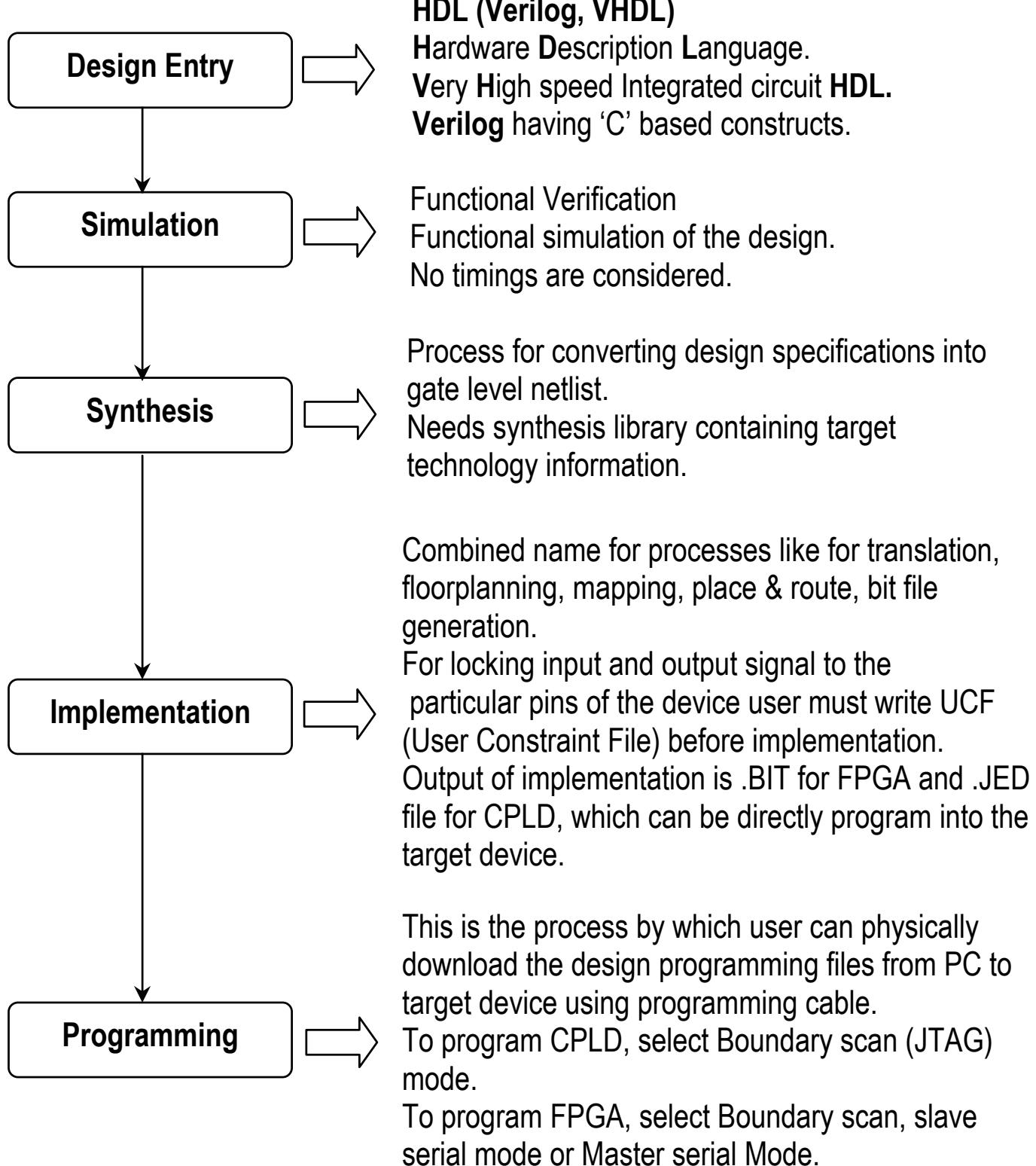
### 3.d MATrix board component legend diagram



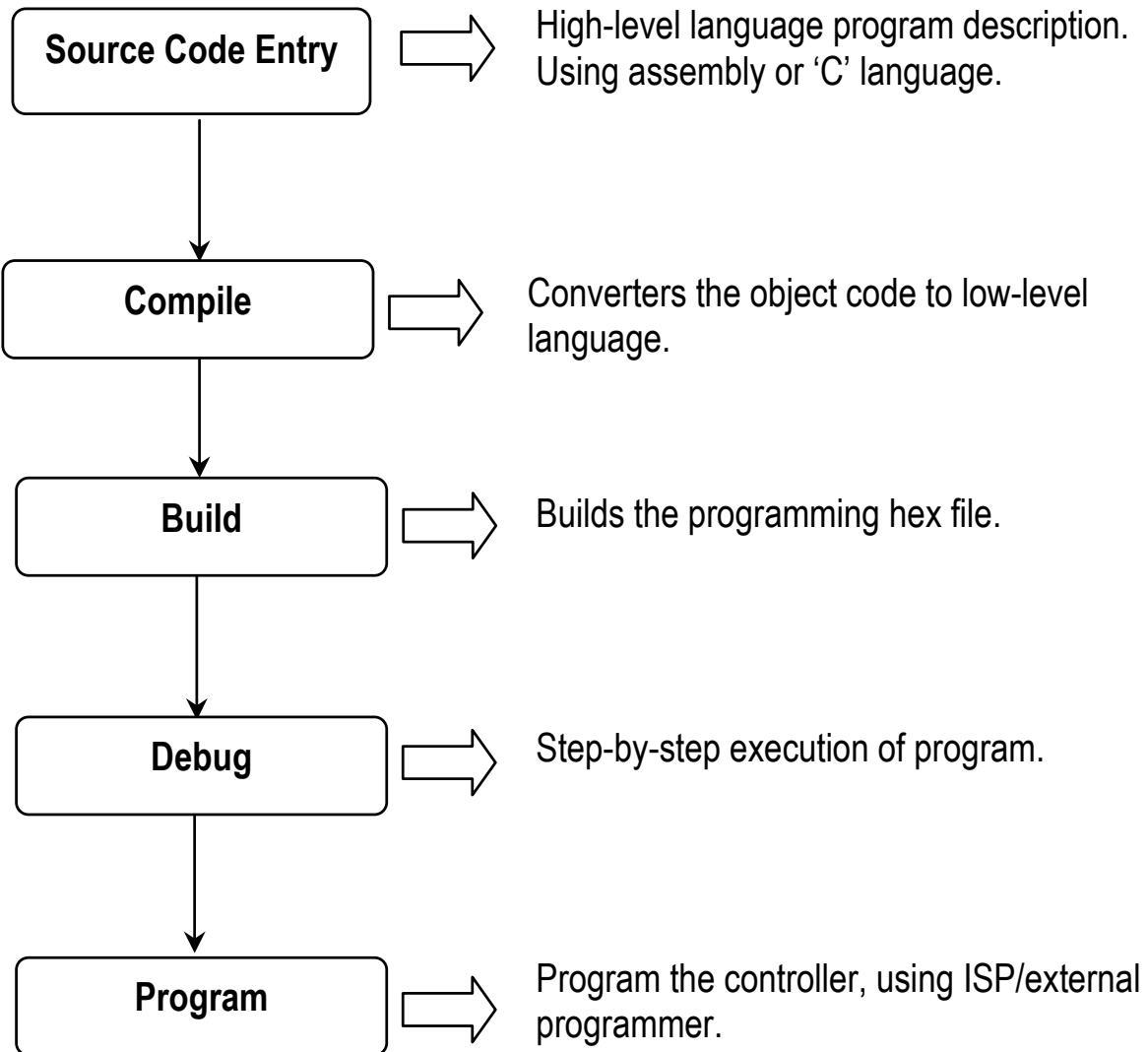
## Chapter 4: Precautions

- Verify the power on LED status after applying power to the trainer.
- Connect the 9 pin D connector of the cable to the trainer only after confirming the above
- During downloading make sure that the jumper selections are proper.
- Select the proper configuration mode during programming, else programming can fail.
- Take care for adaptor position before plugging on the board; this may cause damage to PLD device on power ON if plugged incorrectly.
- Insert the PLD adaptor by looking at the **circle** marks given on the baseboard and adaptor card.
- Do not touch the FPGA, as your body static charge may damage FPGA.
- Before implementation, it is necessary to lock the pins in user constraint file (UCF) as per the design and I/Os used.
- For downloading the bit stream, the downloading circuit requires a stable supply; hence it is recommended to use power supply given along with the trainer board.
- PLD devices are sensitive to surge currents and voltages.
- The devices supported on MATrix works 5V logic family, user can read device datasheets before applying external signals to device.
- Kindly remove the cables from holding its headers only. Removing the cables from holding its wires may cause damage to cable joints.

## Chapter 5: Design Flow



## Microcontroller Design Flow



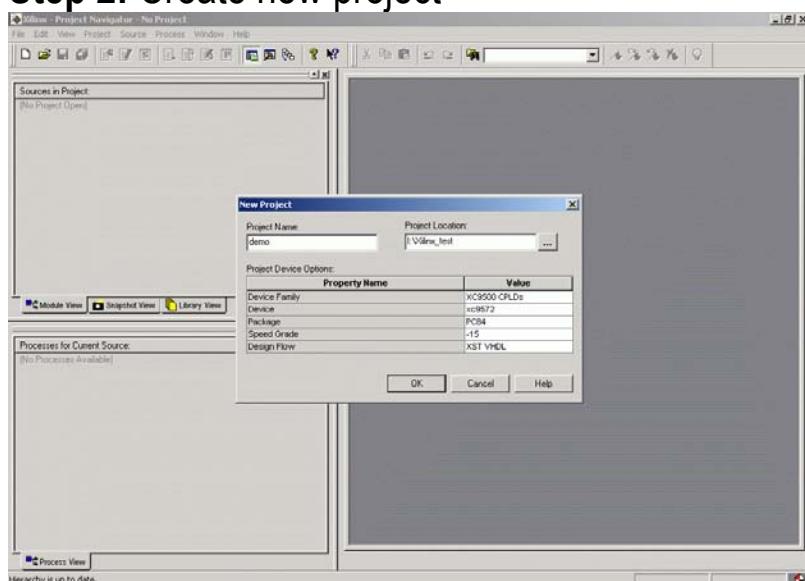
## Using EDA Tools

In this chapter we will see how a project can be created in Xilinx and Altera EDA tools, and how we can proceed to use MATrix to perform our experiments. We take the example of half adder and implement on both vendor devices.

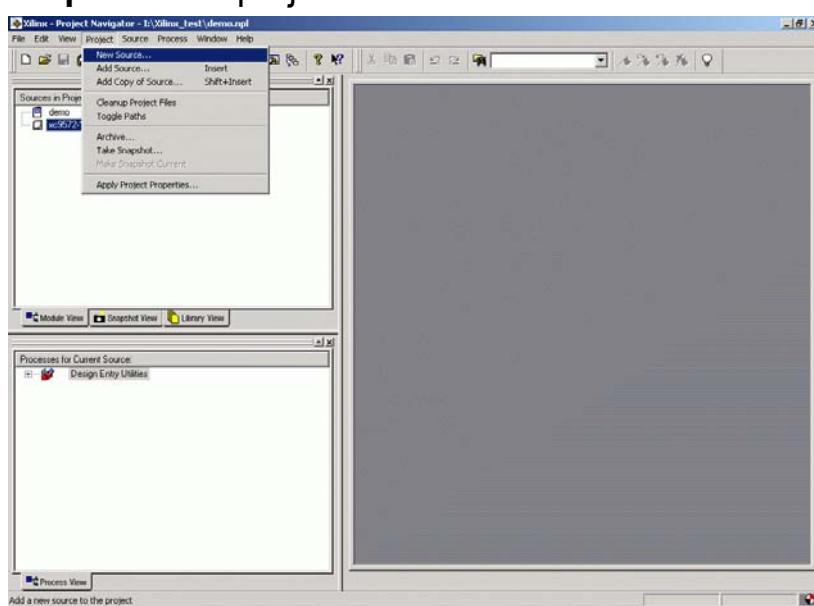
### **Design flow for Xilinx ISE series softwares**

**Step 1:** Open ISE webpack software.

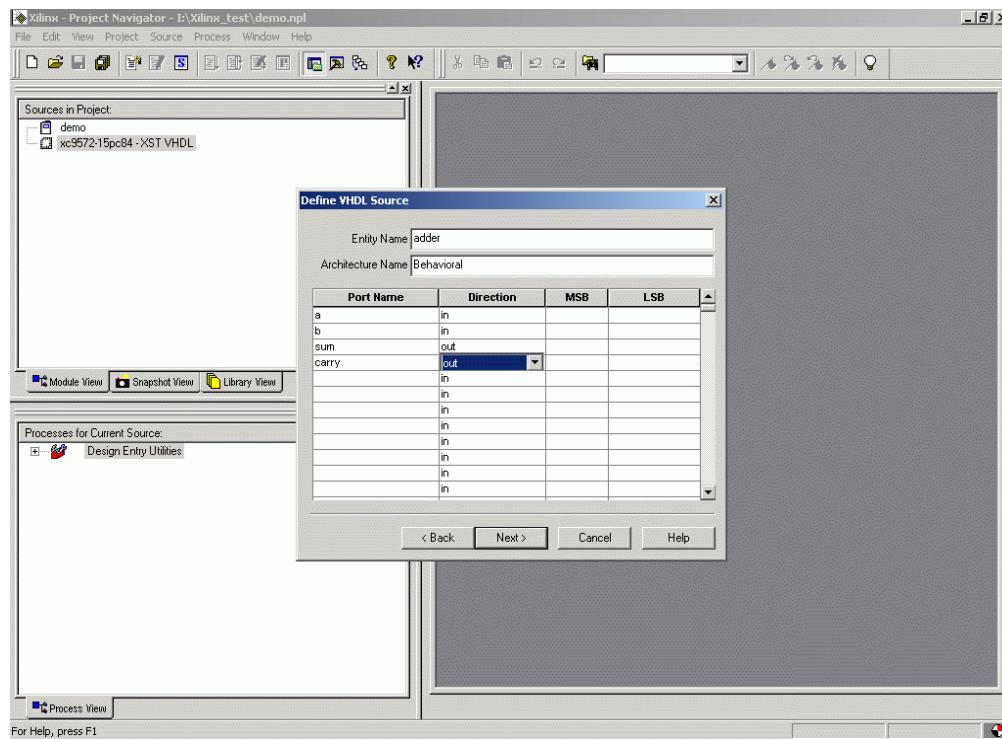
**Step 2:** Create new project



**Step 3:** Go to project menu and select new source



## Step 4: Select VHDL source file, name it adder, click next, and enter entity I/Os as A, B, Sum & Carry.



## Step 5: Write VHDL code for half adder.

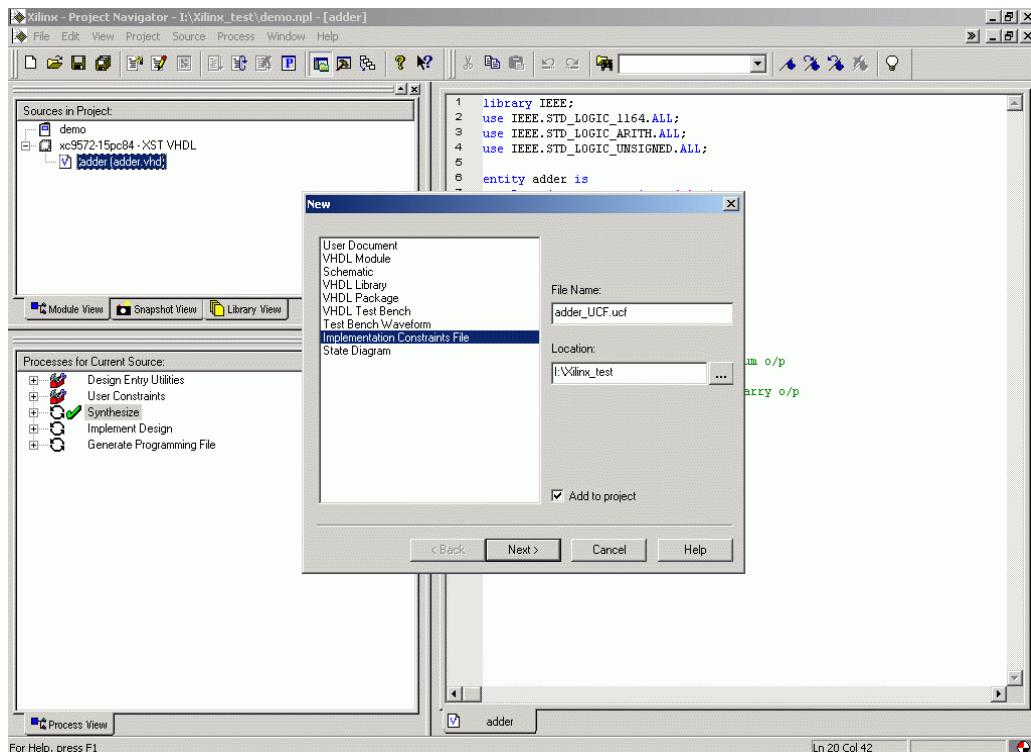
```

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 adder is
7     Port ( a      : in std_logic;
8            b      : in std_logic;
9            sum   : out std_logic;
10           carry : out std_logic);
11 end adder;
12
13 architecture Behavioral of adder is
14
15 begin
16
17 sum    <= a xor b;    -- XOR Gate Sum o/p
18 carry  <= a and b;  -- AND Gate carry o/p
19
20
21 end Behavioral;
22

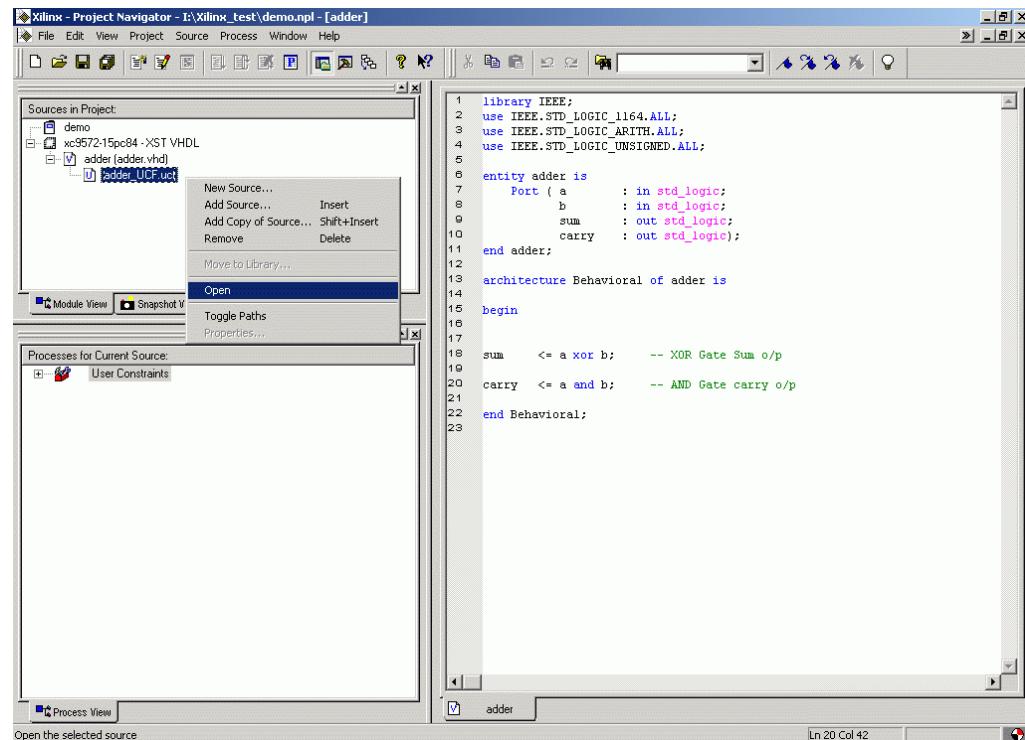
```

The screenshot shows the Xilinx Project Navigator interface with the "adder.vhd" file selected in the project tree. The main window displays the VHDL code for a half adder. The code includes library declarations for IEEE std\_logic packages and their arithmetic and unsigned variants. It defines an entity "adder" with four ports: "a" and "b" as inputs, and "sum" and "carry" as outputs. The architecture "Behavioral" contains a single process block with two assignments: one for the sum output using an XOR operation, and one for the carry output using an AND operation. The status bar at the bottom indicates "Ln 20 Col 42".

**Step 6:** Create new source file for implementation constraint file. Name it **adder\_UCF**, and associate with the corresponding design file.

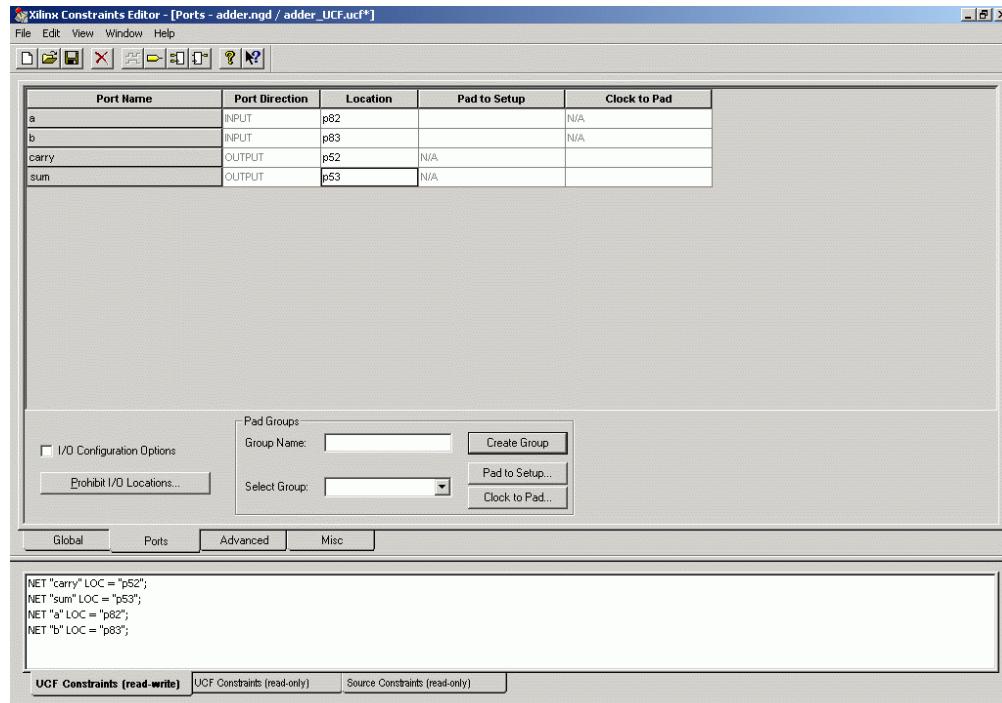


**Step 7:** To assign the pin location of the design, open the UCF file, to run the constraint editor where we have to lock the I/Os of design to a particular pin number.

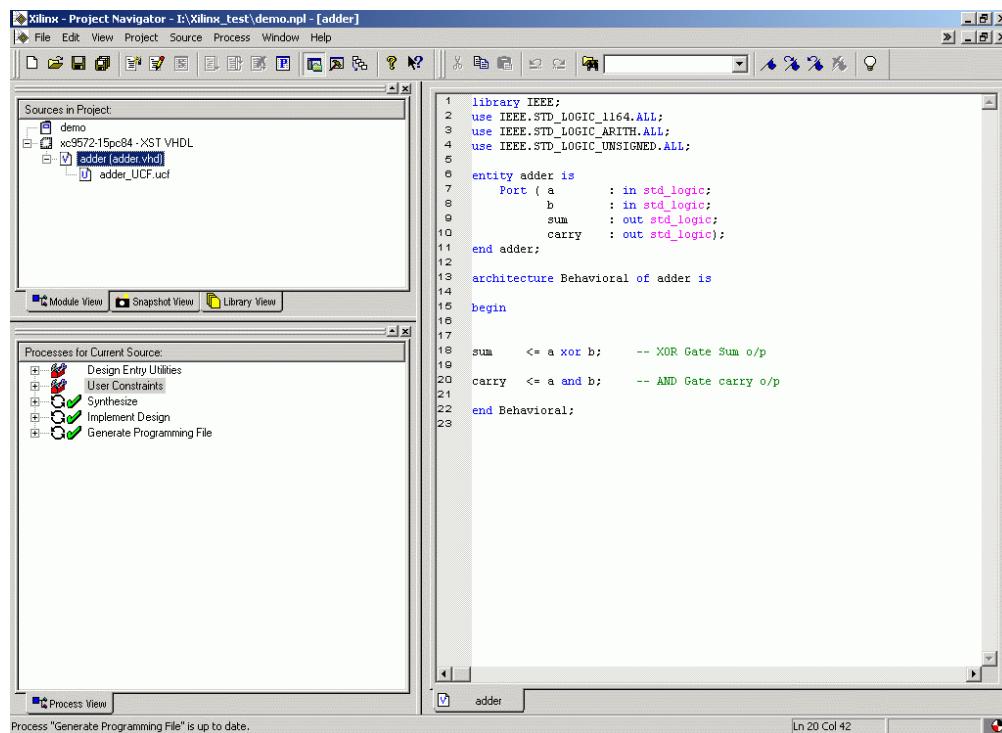


**Step 8:** Once the constraint editor is open, goto **ports** tab, and assign the pins by referring the Pin assignment chapter.

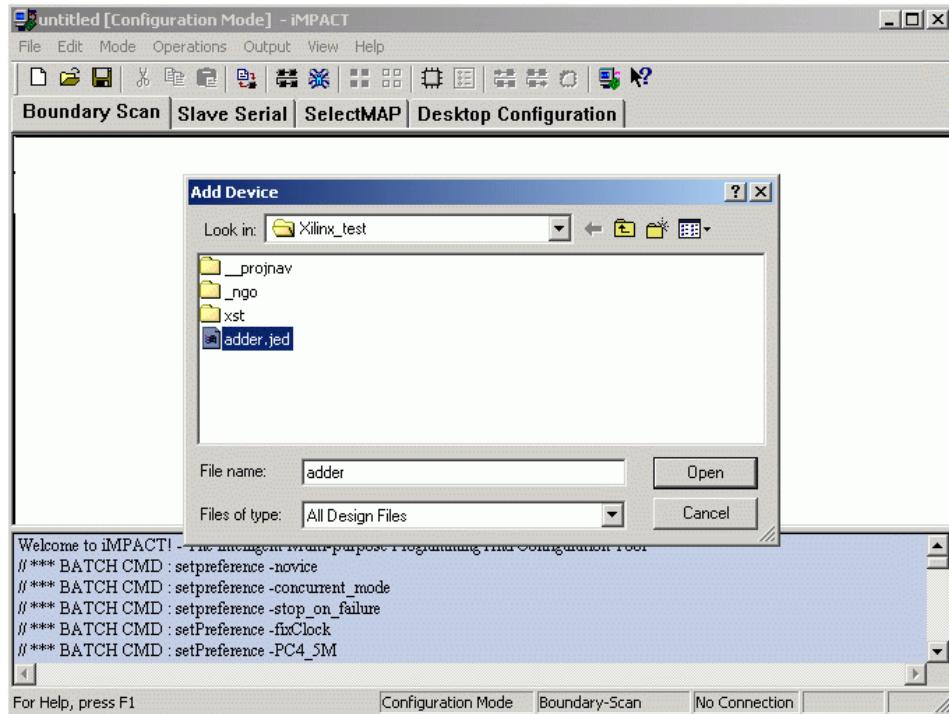
Eg: net A loc =p82;



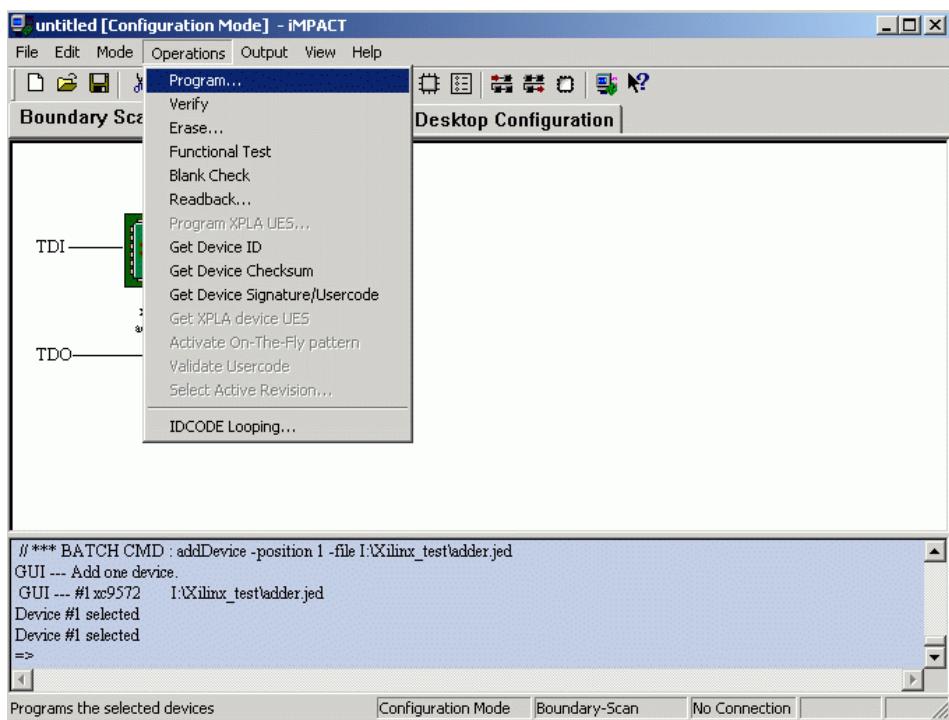
**Step 9:** Save the UCF file and come back to project navigator. Now selecting the **adder** design file, run the **synthesis** process, there after **Implementation** and finally run generate **programming file** option.



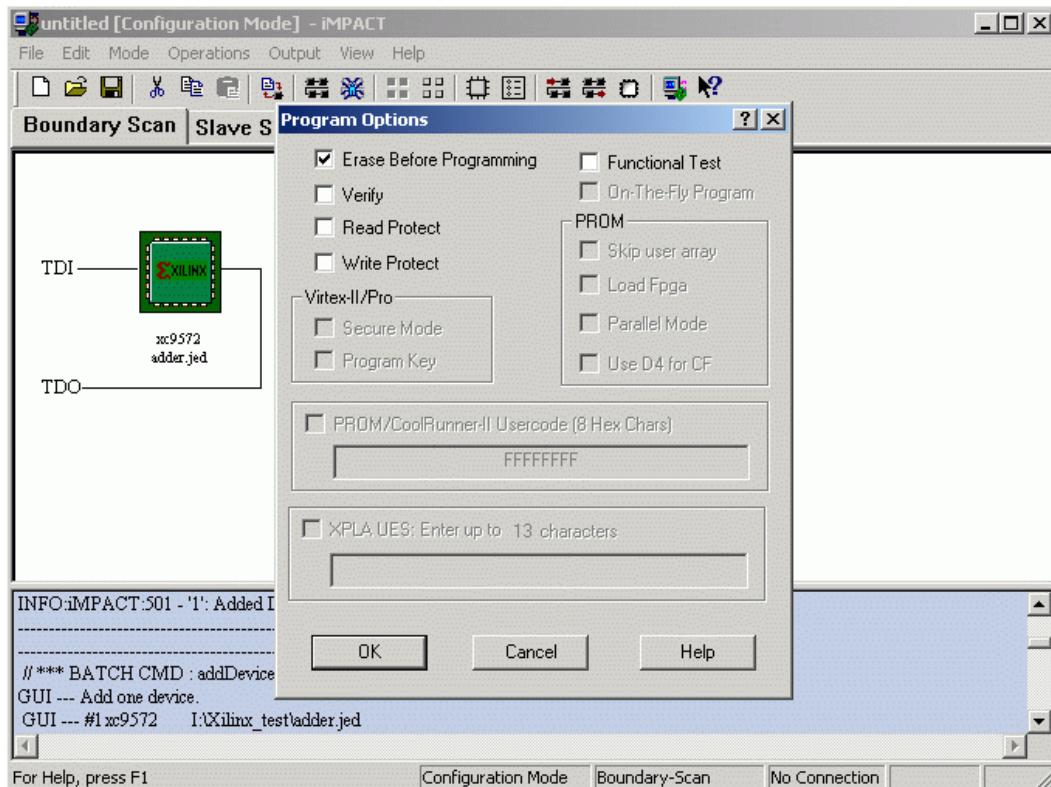
**Step 10:** After the all the three processes are successfully over, run the configure device (impact) option to open the impact programmer. After opening impact add Xilinx device design file, for CPLD it is \*.jed file, and for FPGA it is \*.bit file.



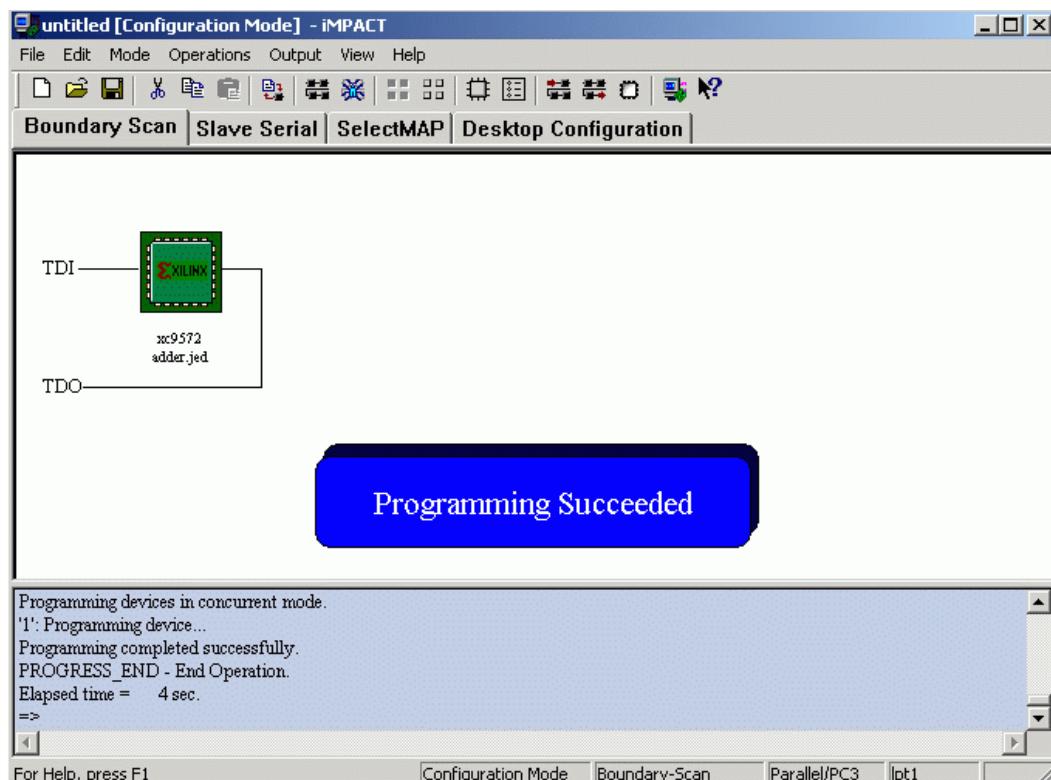
**Step 11:** Go to **operations** option and select program option.



## Step 12: Keep the erase option enabled and click OK.



## Step 13: After erasing the CPLD, the programming would start and will configure the particular device.



Now check the functionality on the board and verify it by applying different inputs.

## Design Flow for Quartus-II series of software of Altera.

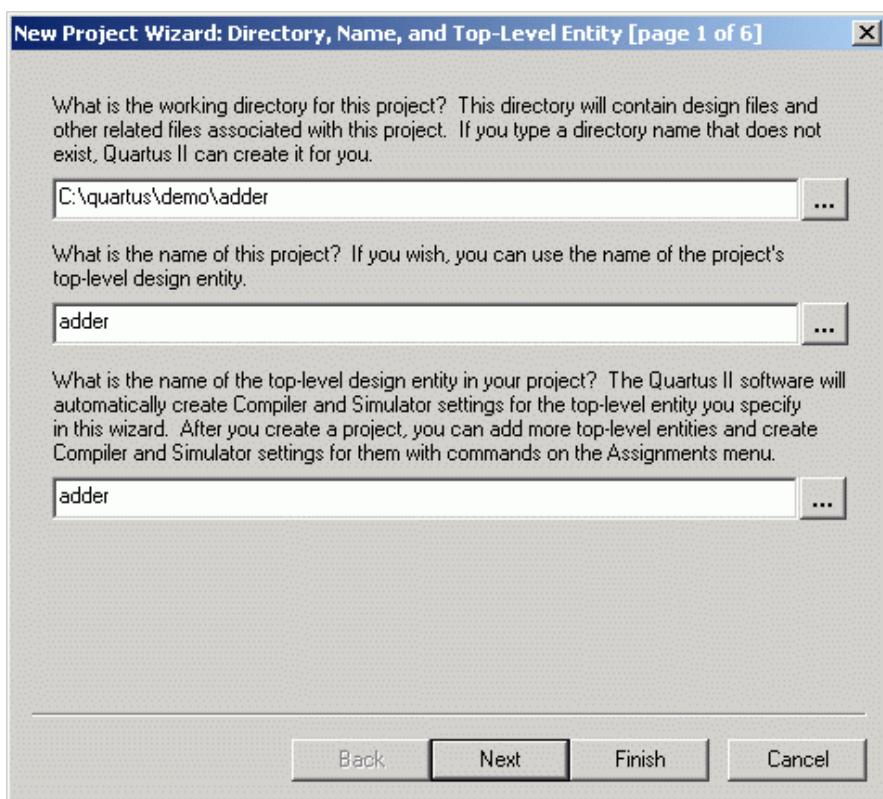
Install Quartus-II (version 3.0 & above) software on your machine, the supported platforms are windows NT/XP/2000.

We take the same **half adder** example for implementing on the Altera MAX7000s CPLD.

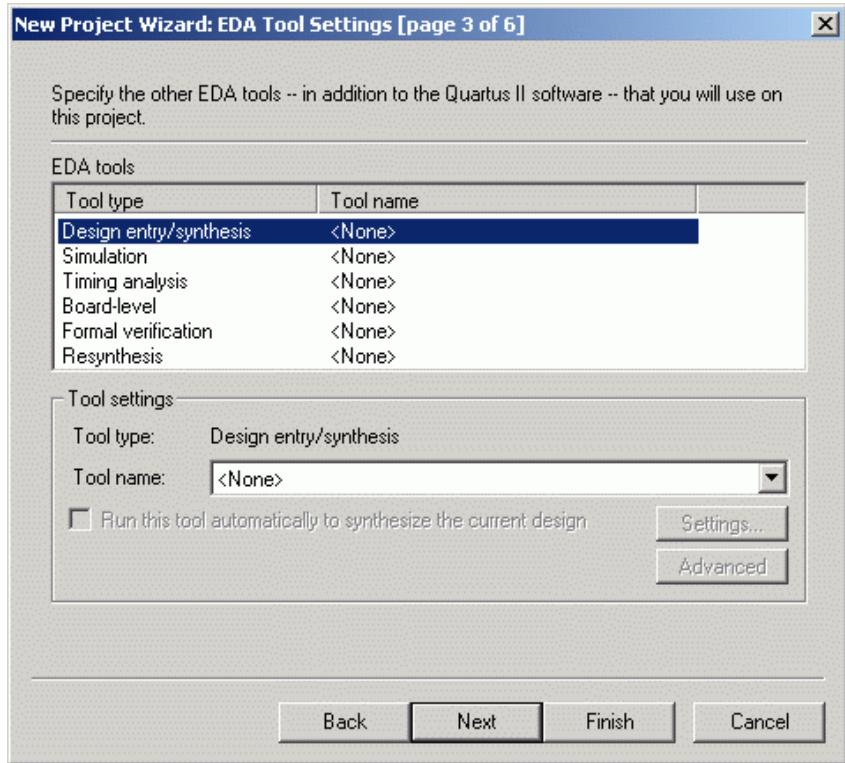
**Step 1:** Start Quartus-II (version 3.0 & above) software.

**Step 2:** For new project creation, go to **File** option and select new project wizard. In the opened window, specify project location and design and entity name. For eg. Entity name **adder**, and top design name also **adder**.

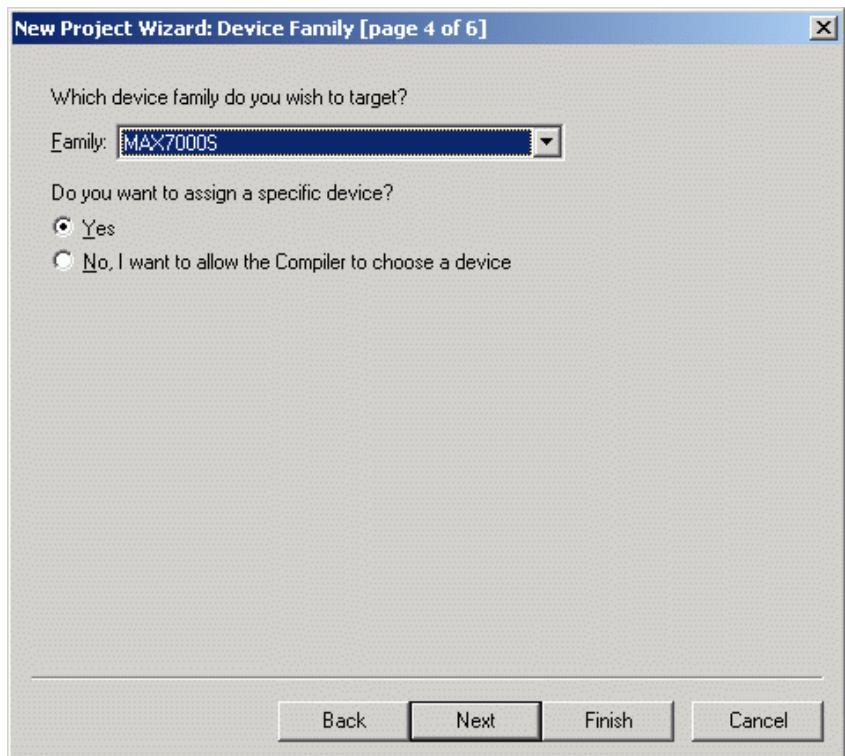
Click “next”.



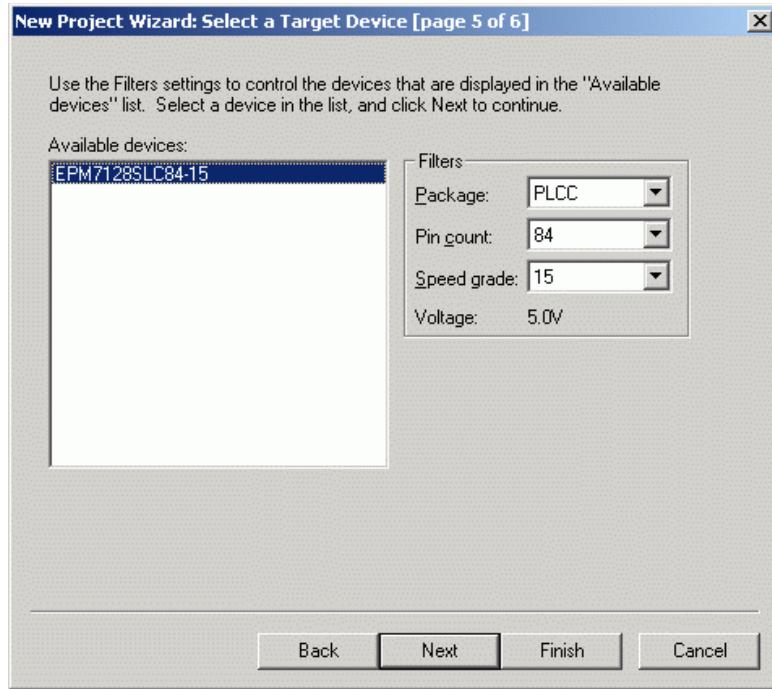
**Step 3:** Click “next” button till you reach EDA tool settings window, there keep all options as none, which in default will select the inbuilt design tools and softwares for the design processing. **Click “next”.**



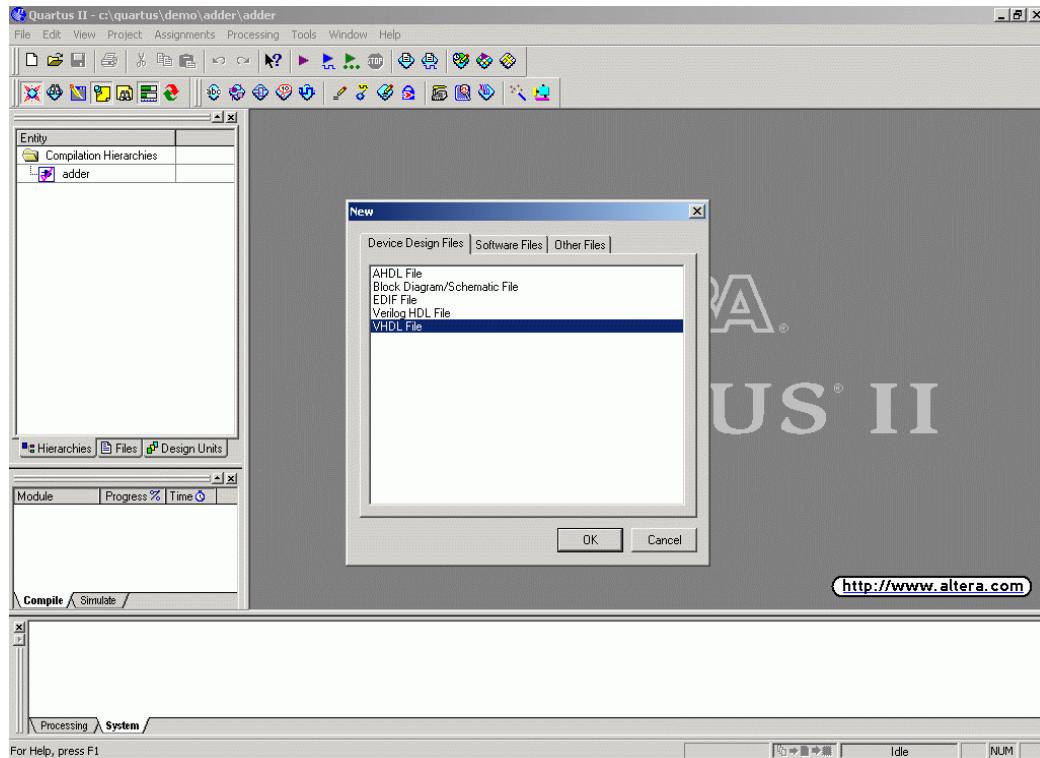
**Step 3:** Select MAX700S device family in the next window. **Click “next”.**



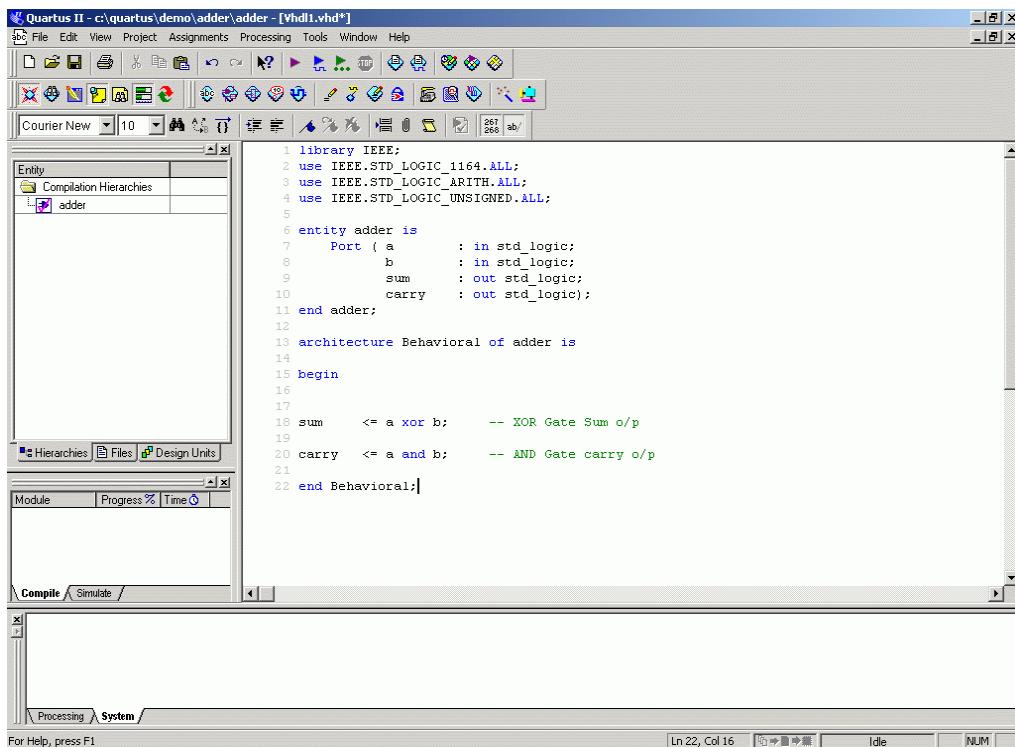
**Step 4:** In the next window, select the device as **EPM7128SLC84-15**. Click “next”.



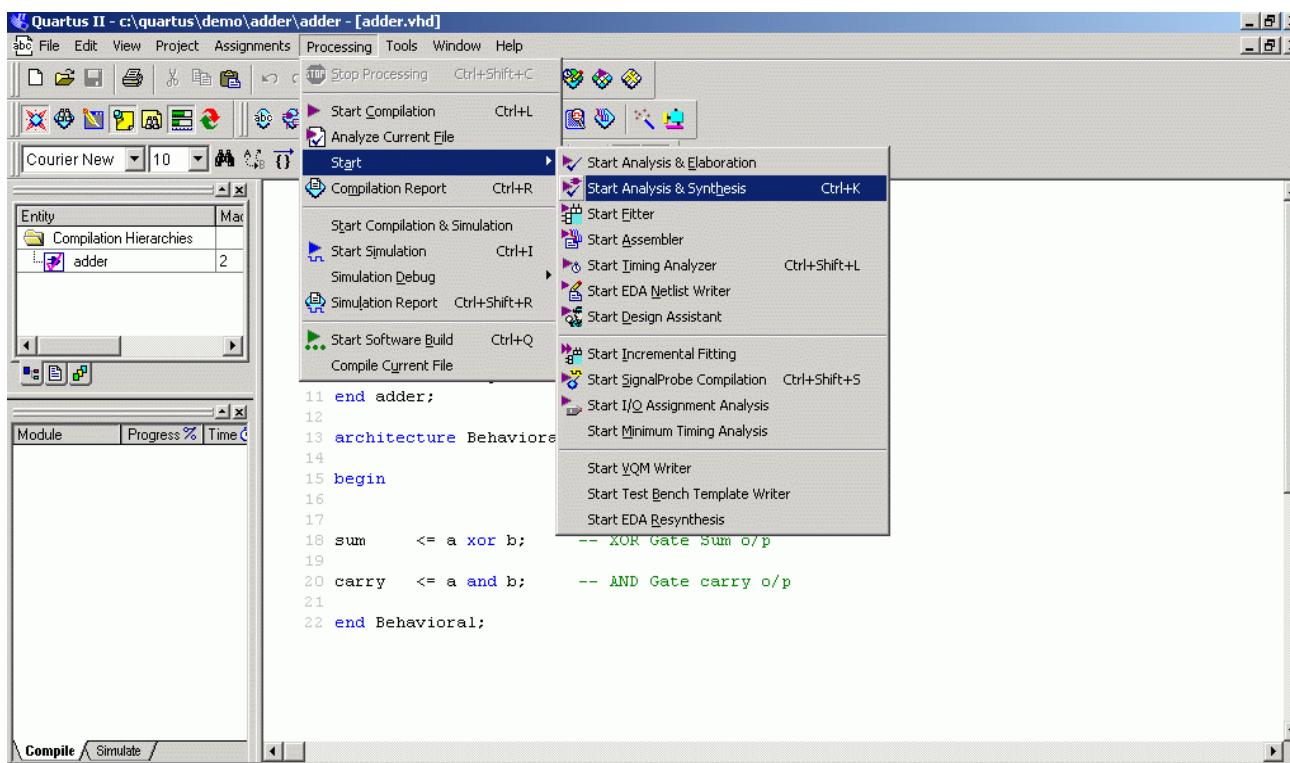
**Step 5:** Click “Finish”. And the new project would be created. Now we need to make and add new design file in the project. So goto “File” menu, and click “New”, and select **VHDL File**, in the “device design files” tab. Click “OK”.



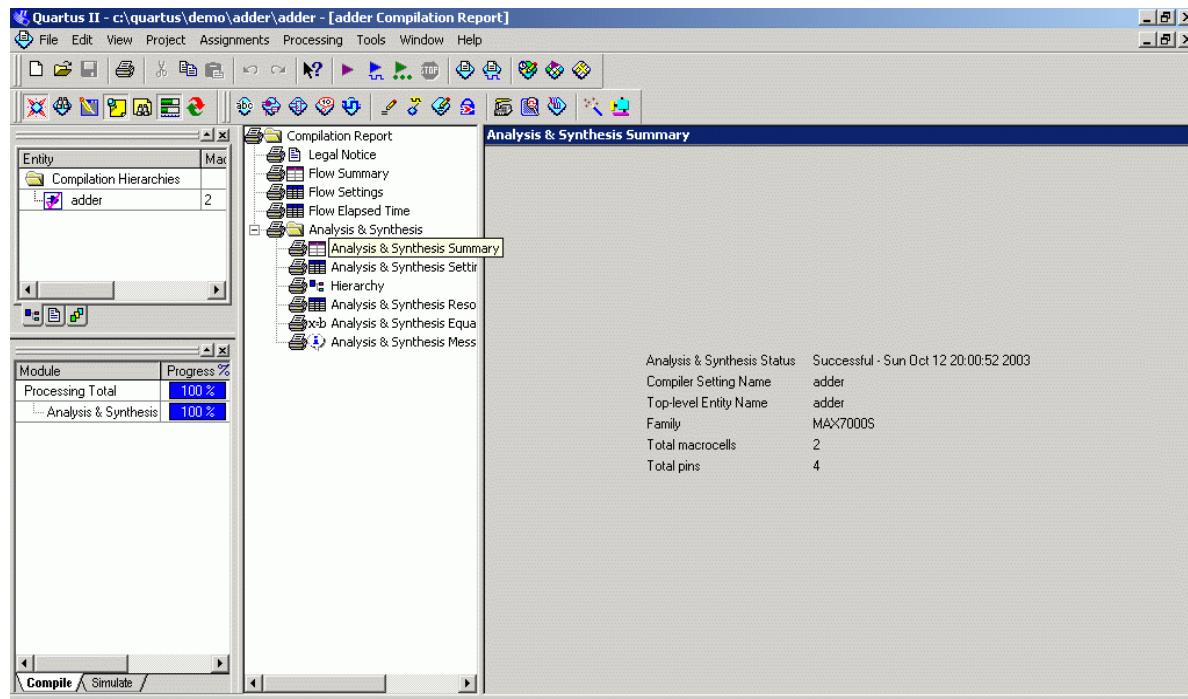
**Step 6:** Write the VHDL code for half adder design, and save the file as “**adder.vhd**”



**Step 7:** Now goto processing menu- then – start - , click start analysis and synthesis.

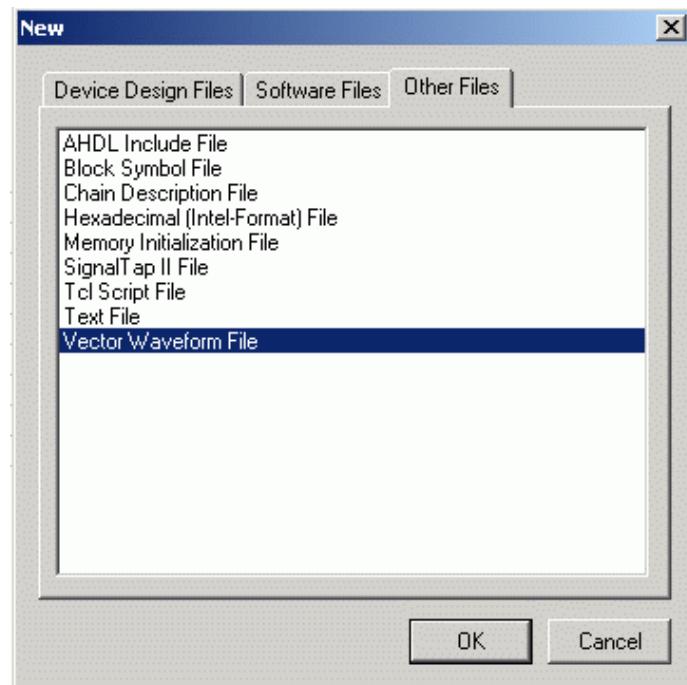


## Step 8: Read the synthesis reports

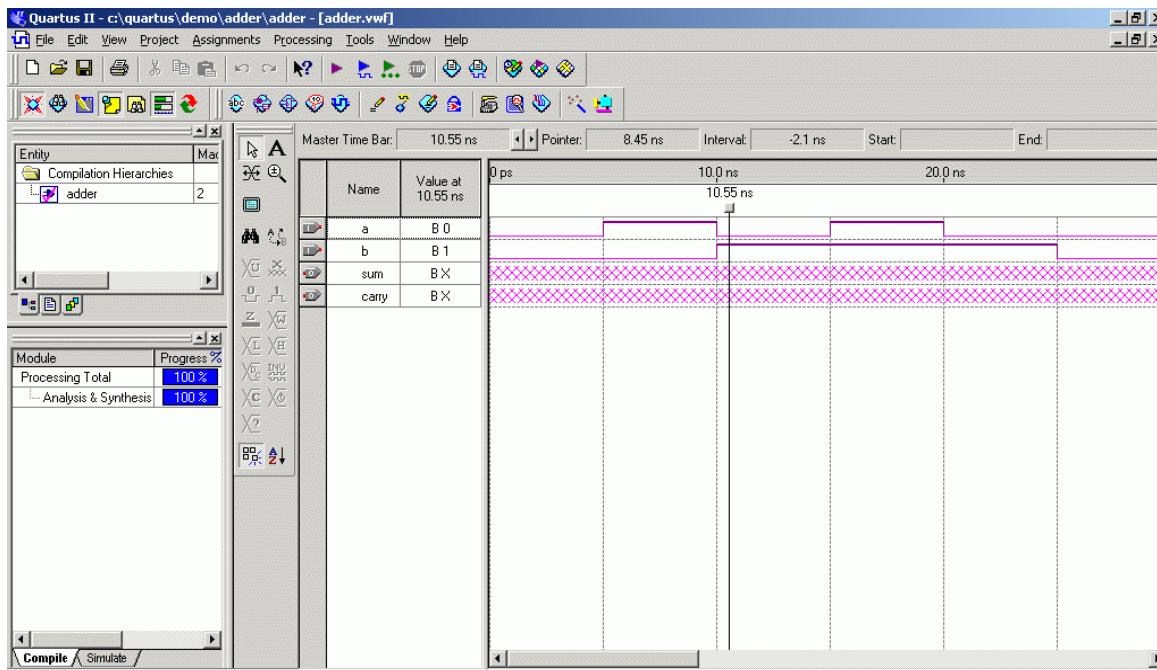


**Step 9:** For performing simulation, we need to create stimuli file from where we can apply input signals and watch the o/p waveforms.

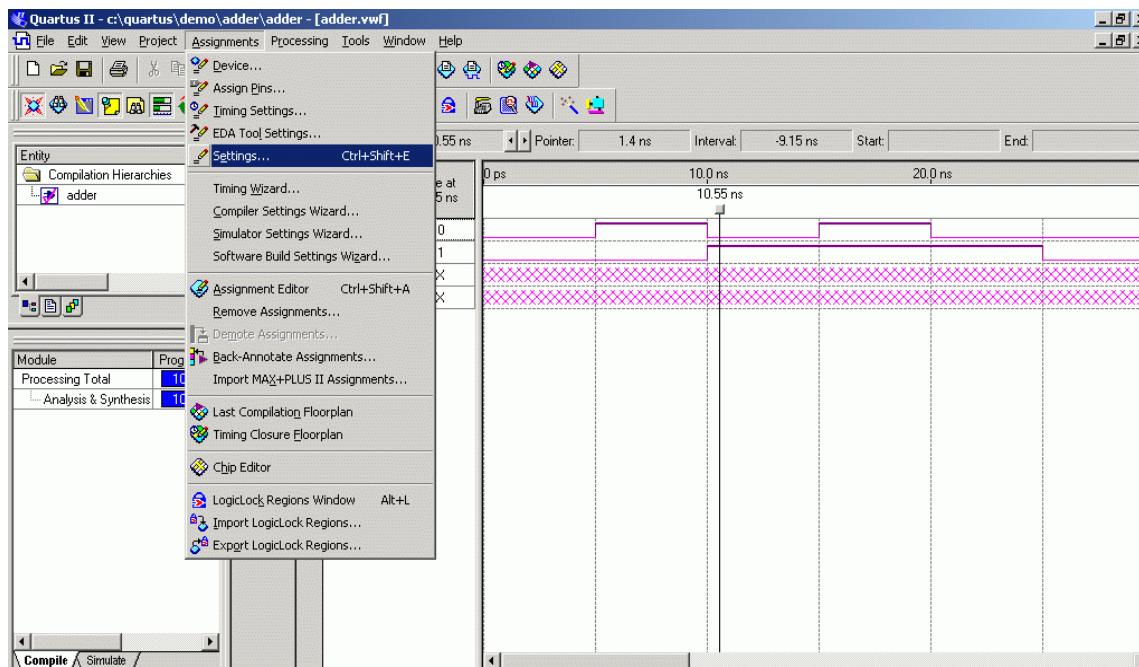
Goto **file** menu, and click **new** file, goto **other files** tab, and select "**vector waveform file**" option.



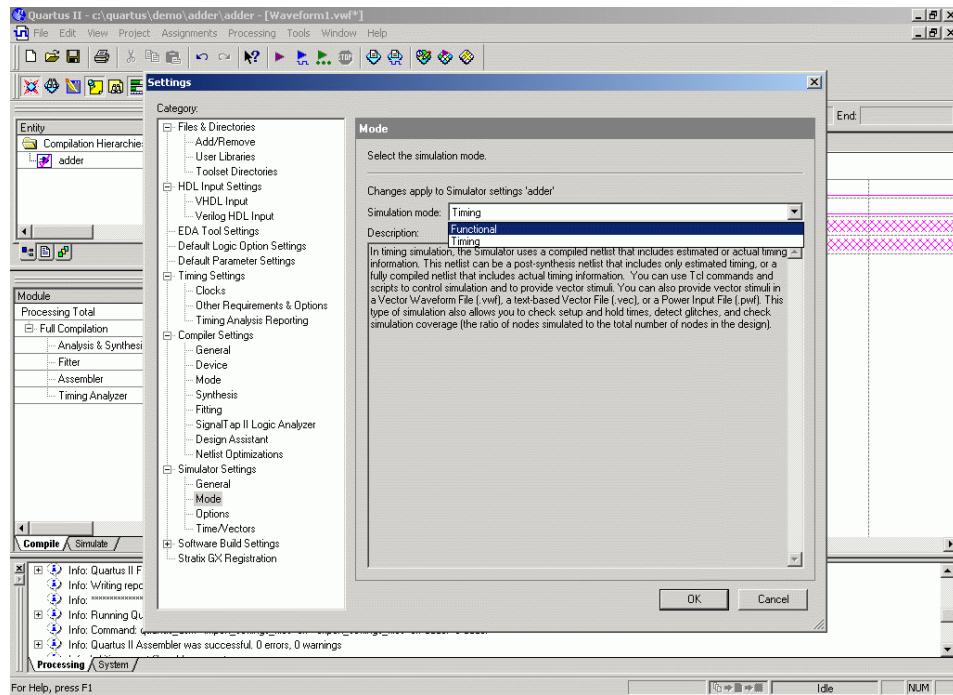
**Step 10:** Add the entity signals in the waveform window, and apply different sets of value to check the functionality. Save the file as the same name of entity, **adder.vwf**



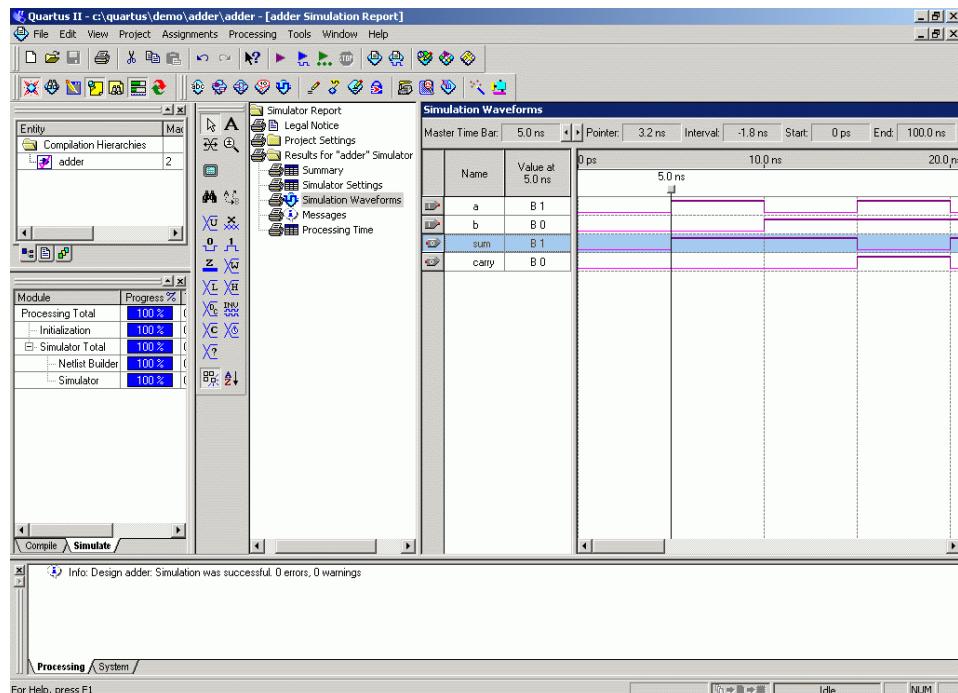
**Step 11:** In Altera Quartus-II software you can perform **Functional and Timing** simulation. For simulation mode settings, go to **assignments** menu, and click **settings**.



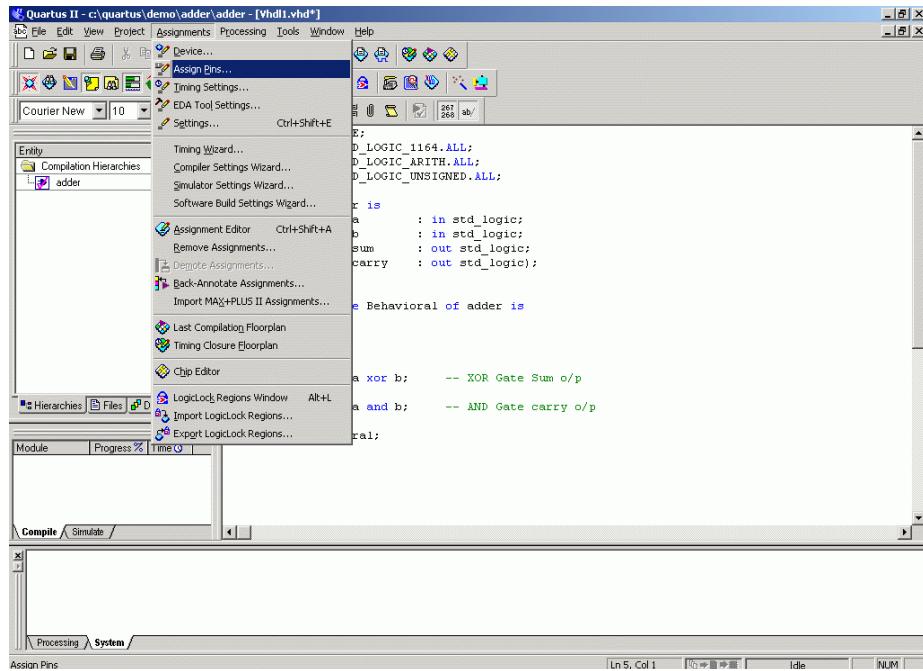
**Step 12:** Now goto **simulator setting**, then to **mode**, and in the right-hand side window, select the simulation mode to **Functional**.



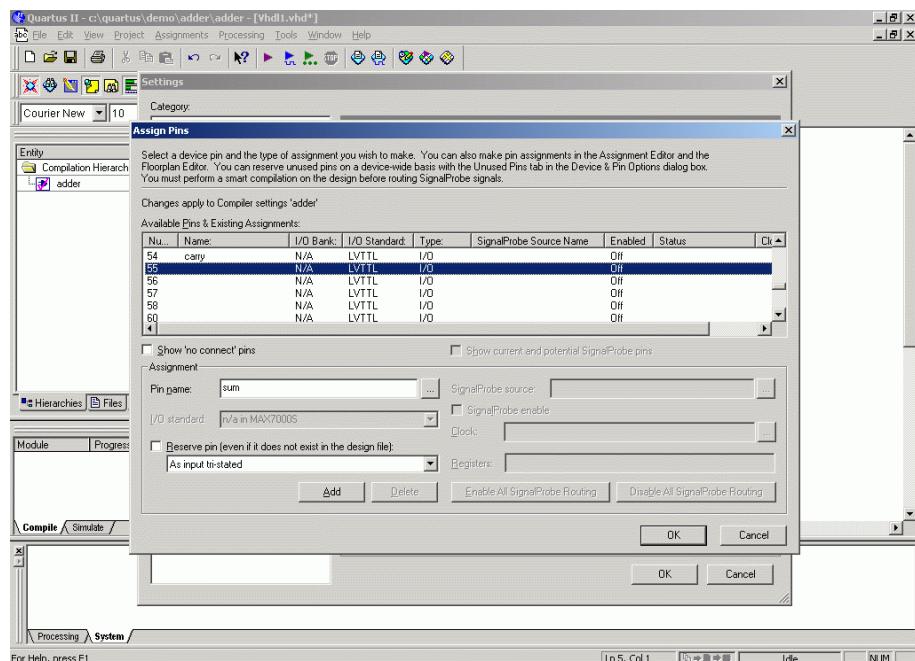
**Step 13:** After clicking OK, come back to main window, and goto processing window, and click start simulation, the Quartus-II will start the simulation the result would appear in couple of minutes. Observe the results, if found bugs, then change VHDL code and start simulation again.



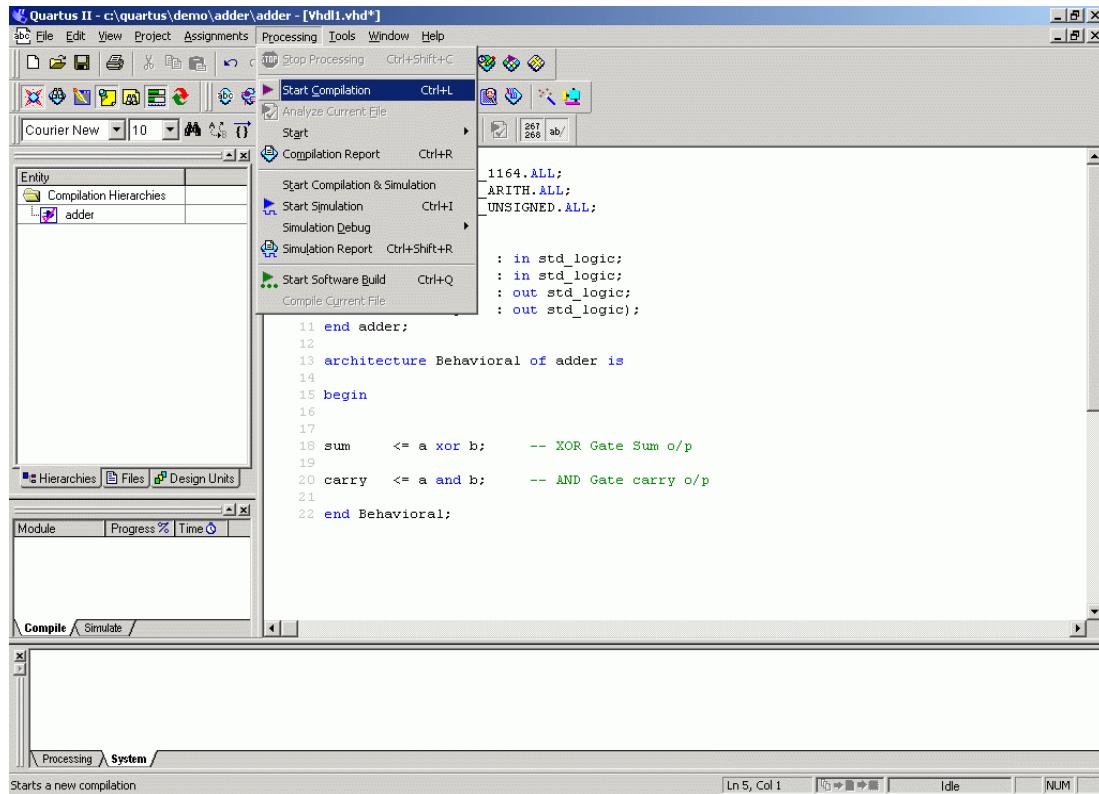
**Step 14:** Once the simulation results found correct, then we need to implement the design in the target device. For this we need to lock our design I/Os with the Kit I/O pin details. Goto **assignment** menu, click “**assign pin**” option.



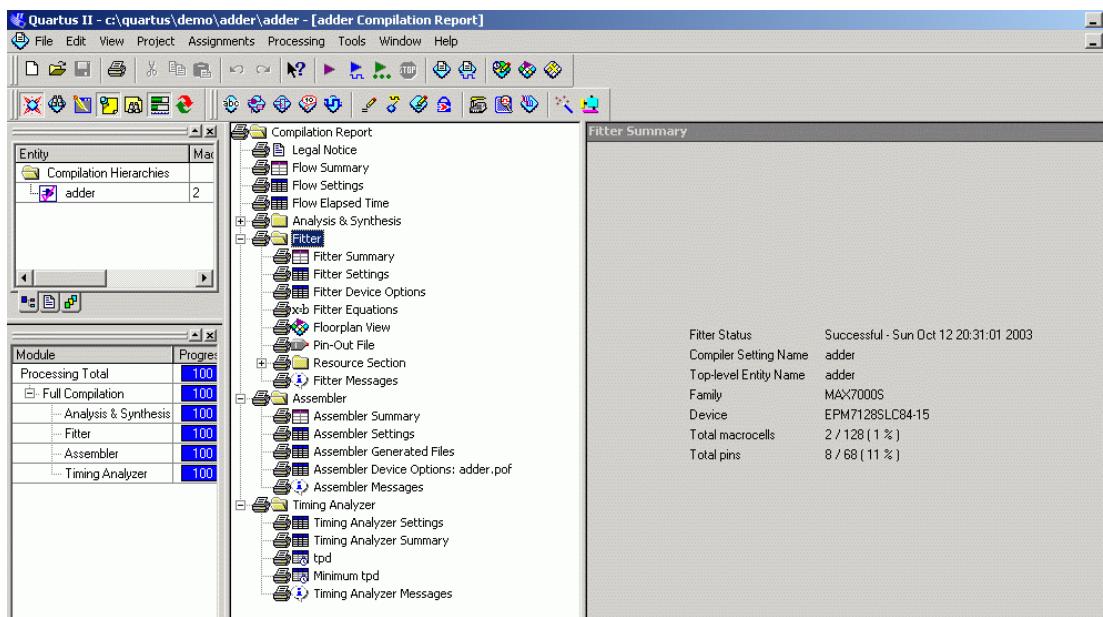
**Step 15:** Looking at the pin assignment chapter, lock the MAX7000s CPLD I/O with the particular pin no., for this select the I/O number on the LHS, name the design I/O in the bottom pin name option, and then click add, the particular signal will be locked to that pin number



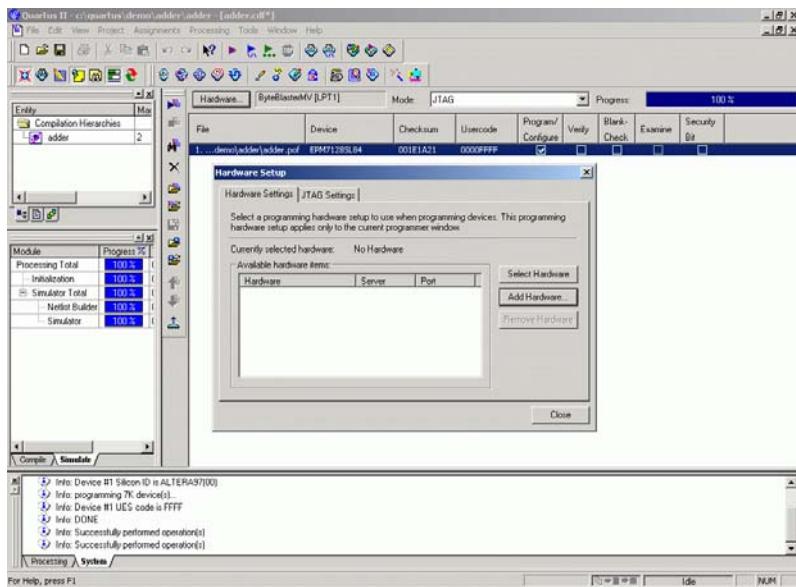
**Step 16:** Once the pin assignment is over, come back to main window. Now we need to implement the design on the particular device. So goto **processing** menu, and click **start compilation** process. Which will fit the design in CPLD and generate the programming file.



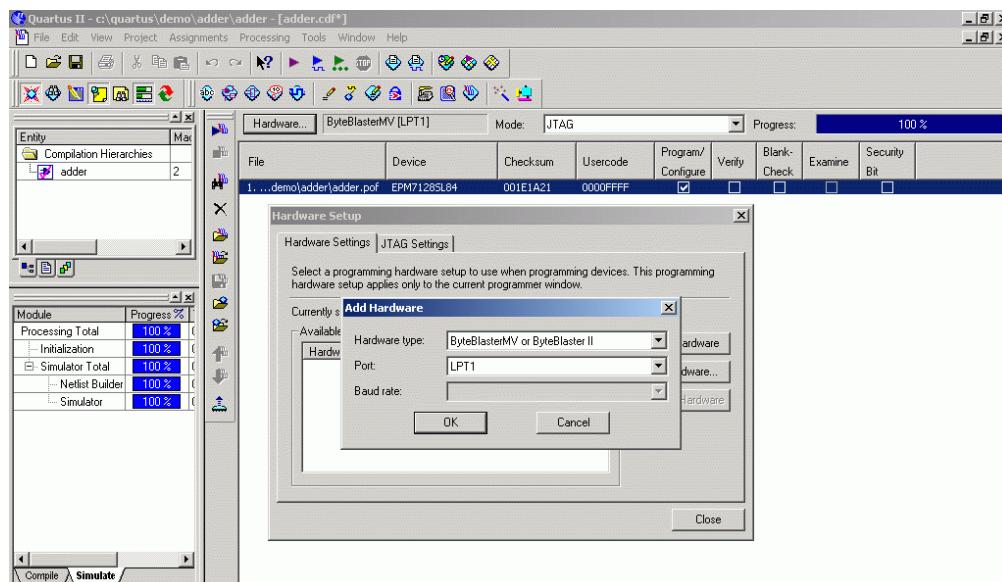
**Step 17:** Once the **compilation** process is over, user can check the **reports** and see the floorplan window.



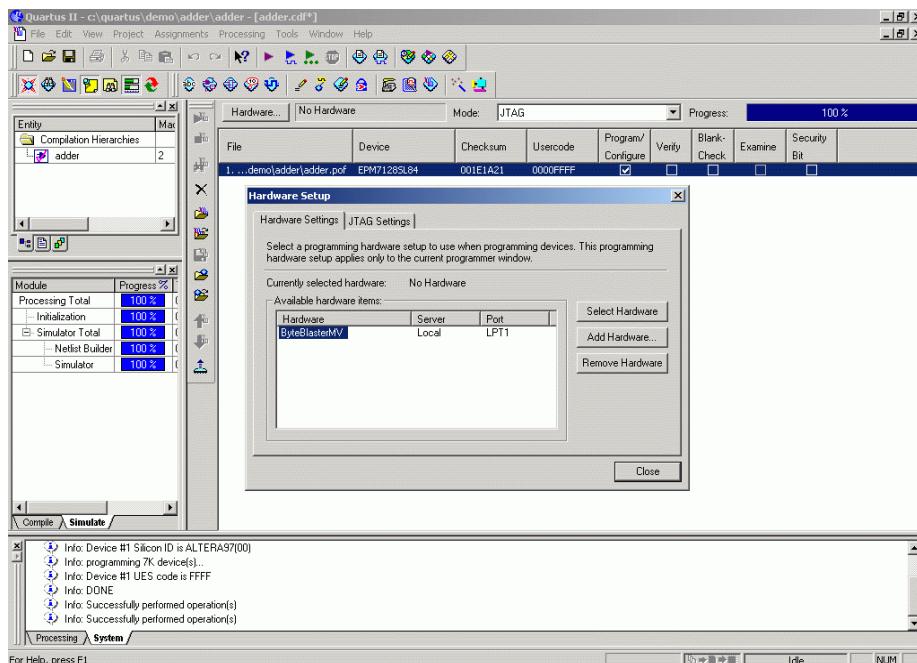
**Step 18:** Now we need to program CPLD, for this goto tools menu, and click programmer. Which will open the programmer; the software will automatically add the programming file (adder.pof). In the opened window select the **program/configure** option. Now we need to select the programming hardware, for which click the hardware tab on the top LHS of programming window.



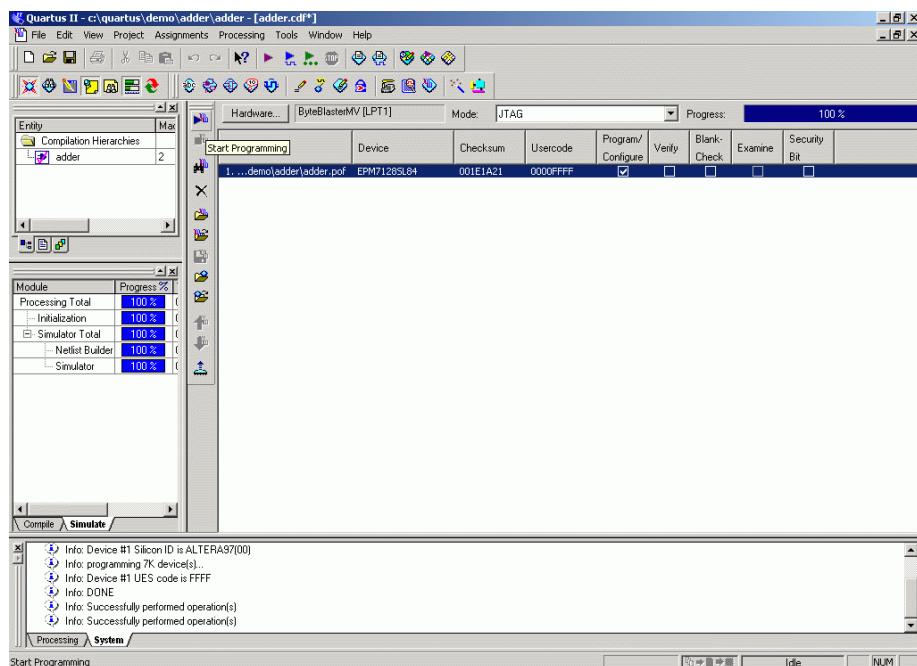
**Step 19:** In the opened window Click **add hardware** tab, and select the hardware type as **ByteBlasterMV** or **ByteBlaster II** and port as **LPT1**.



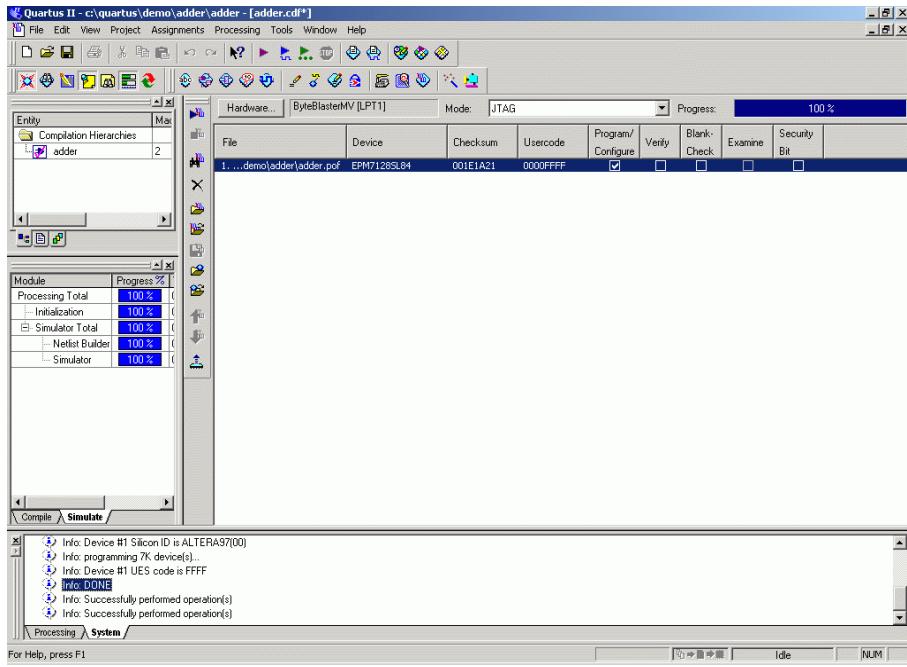
**Step 20:** Come back to hardware setup window, and click the **select hardware** tab and close the window.



**Step 21:** Now click the **start-programming** button (play symbol) on the top LHS of the programming window (keep the program/configure option selected).



**Step 22:** The programmer will start programming and in couple of seconds the device would be configured. Check the DONE indication in the bottom console window.



**Step 23:** Check the design functionality on the board, by applying signal from switches or other points.

## Using Keil Compiler

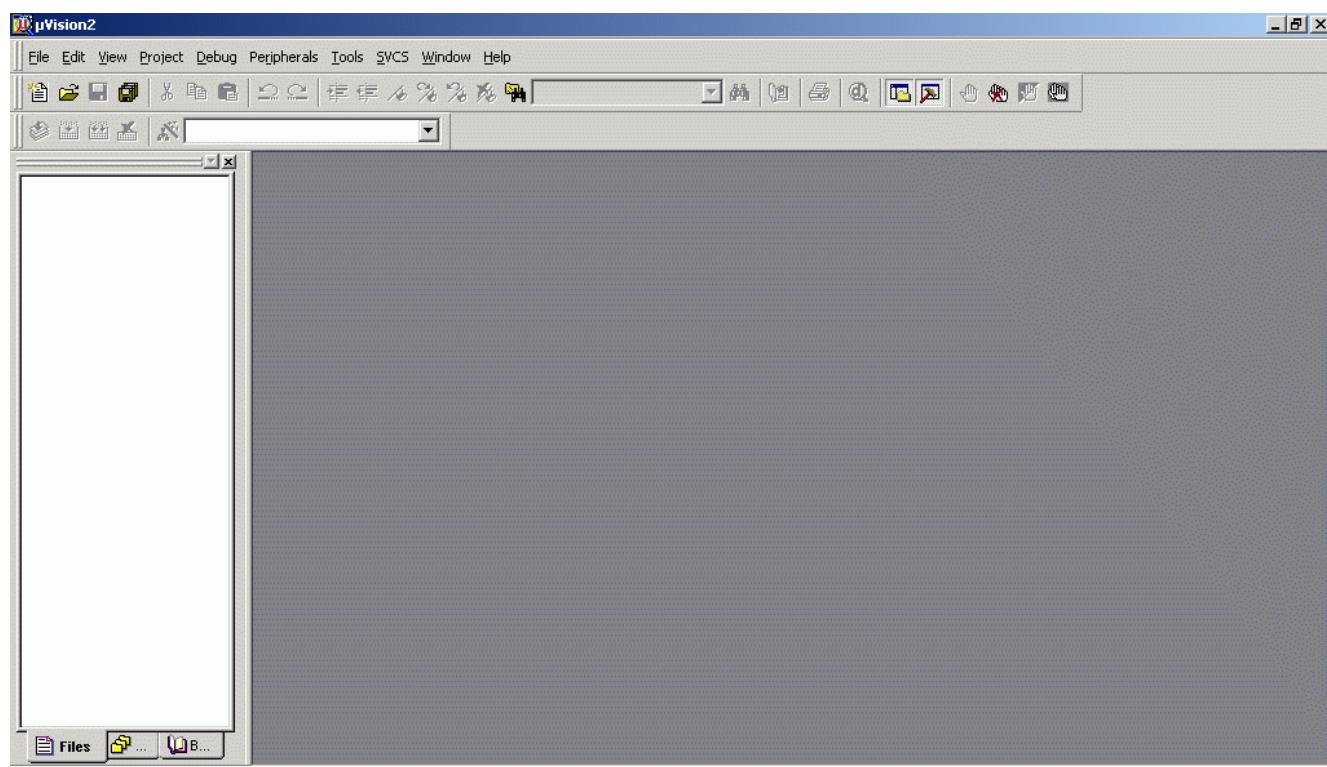
Designers can use compiler from **Keil**, one of the compilers available in market. **Keil** is a cross compiler supporting the 8051 based architecture controllers from various vendors.

Compiler support the source codes written in 'C' and assembly languages.

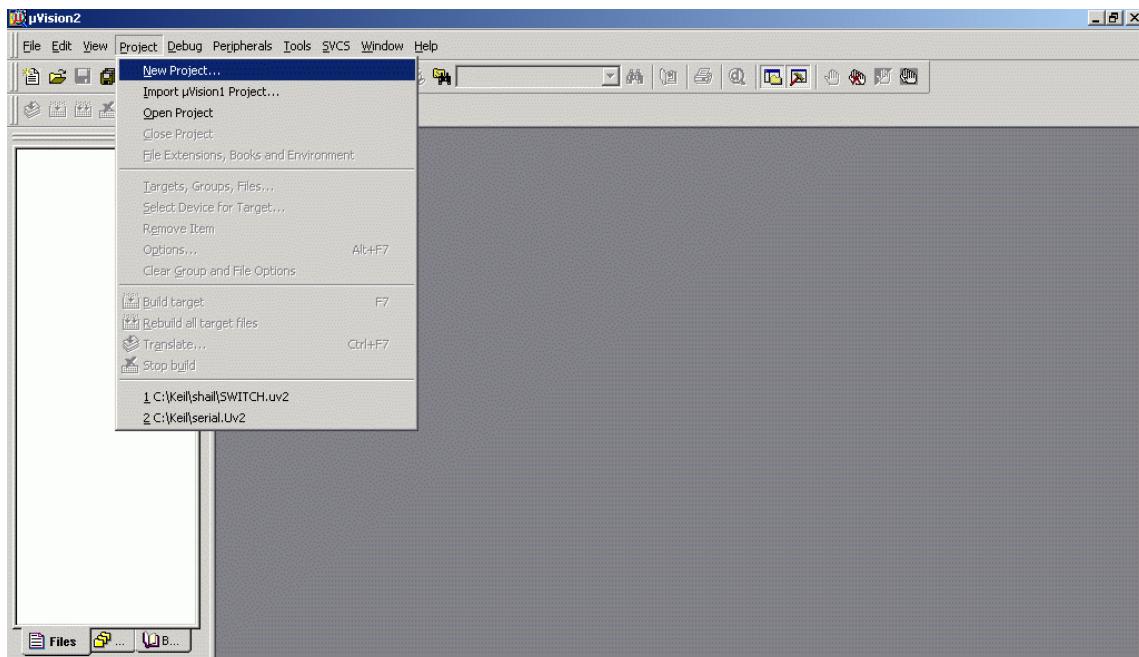
For starting up with MATrix, we have made a design flow guide for using the **keil** compiler, but for more information users can surf the help index of keil compiler,

Install the **Keil** compiler from provided CD-ROM; you can use the evaluation version at start up, which supports the program code upto 2KB, which is sufficient for small development purposes.

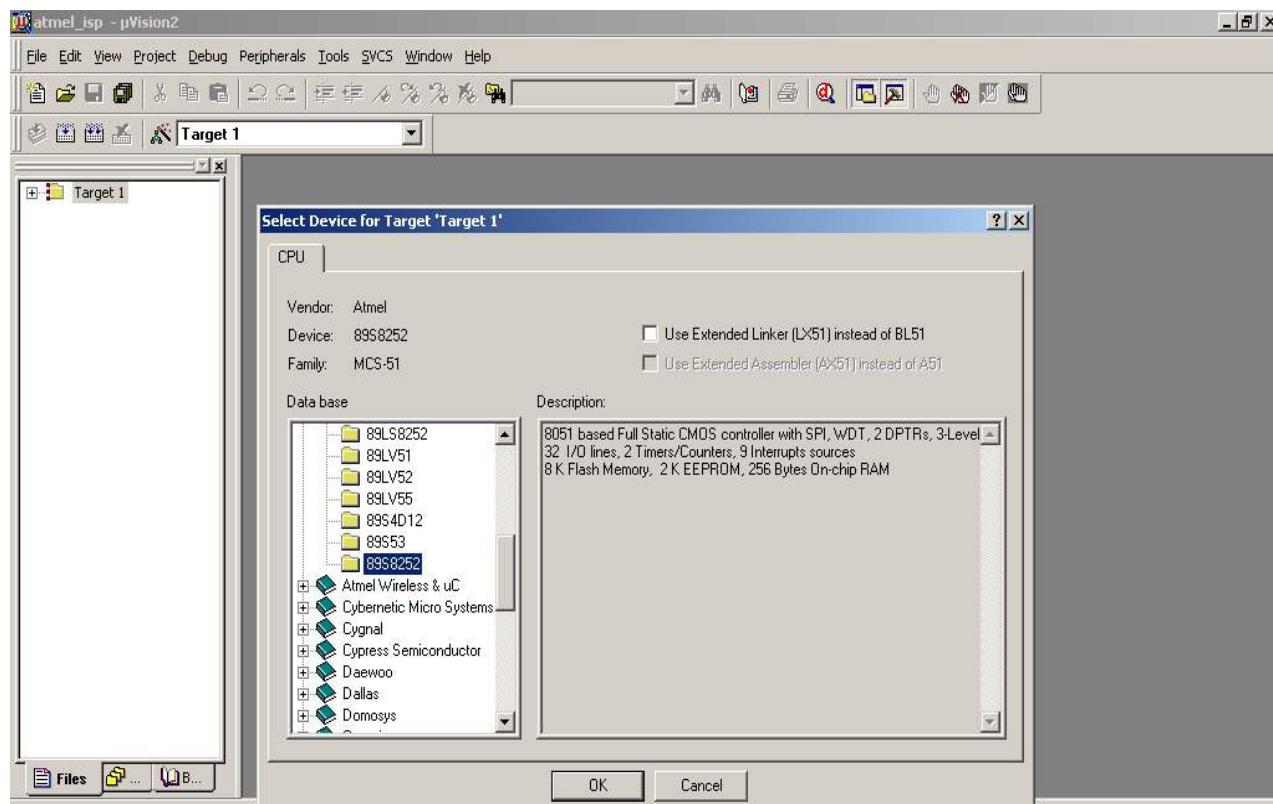
**Step 1:** Run the **keil** compiler EXE, which in turn will open the compiler.



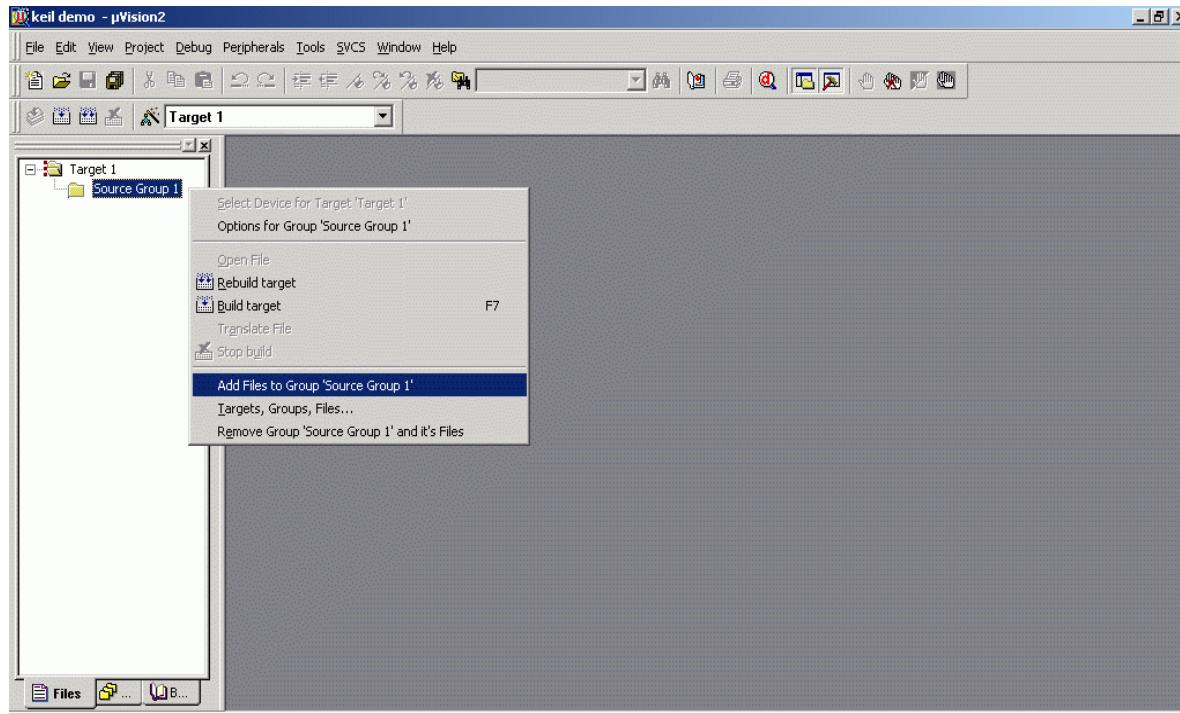
**Step 2:** Go to project menu and left click the new project option.



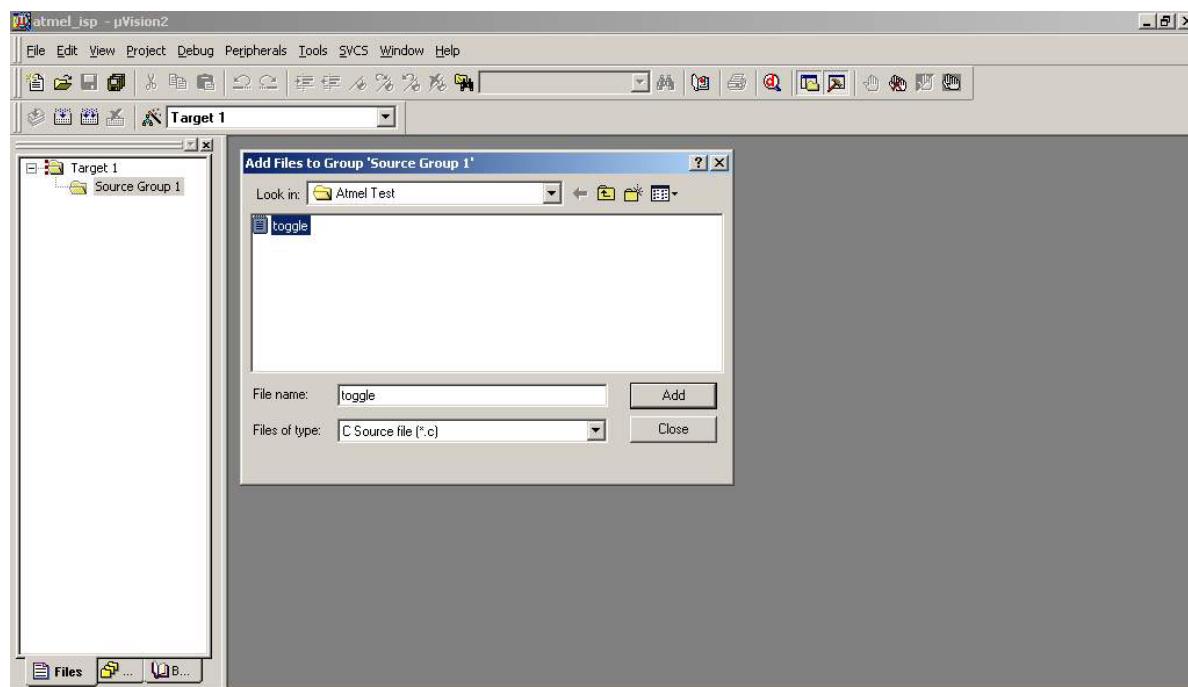
**Step 3:** In the opened window user has to give project name and select the folder for the project creation, for example, name it **keil demo**. Click OK, and in the next window, you have to select the device to work on, the vendor name is **Atmel**, and the device number is **89S8252**. The window will look like as below.



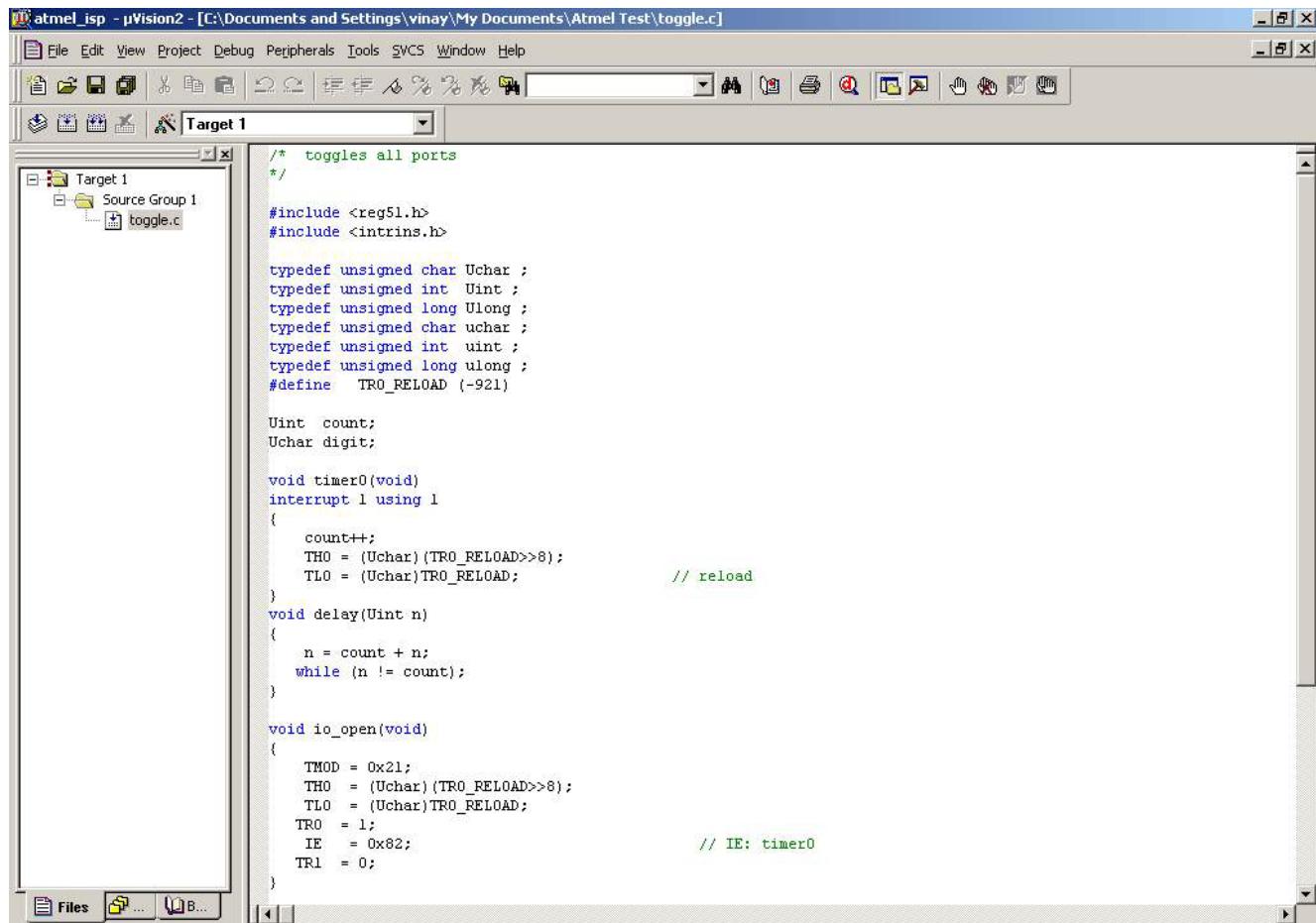
**Step 4:** Click ok on the above window, and the project would be created. Now the user has to add or create the source code for the controller. For example for now designer can use the source code provided along with the code examples. So right click on **source group** and click **add files** to group.



**Step 5:** Browse for the demo source code **toggle.c** and click add to insert it in the project.



**Step 6:** User can have a look on the demo source code, which toggle Port 0, Port 1, and Port 2 after every 10ms. The source code is in 'C' language; users can also use assembly code for designing.



The screenshot shows the Atmel ISP - µVision2 IDE interface. The title bar reads "atmel\_isp - µVision2 - [C:\Documents and Settings\vinay\My Documents\Atmel Test\toggle.c]". The menu bar includes File, Edit, View, Project, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations like Open, Save, Print, and Build. The main window is titled "Target 1" and displays the C source code "toggle.c". The code is as follows:

```

/*
 * toggles all ports
 */

#include <reg51.h>
#include <intrins.h>

typedef unsigned char Uchar ;
typedef unsigned int UInt ;
typedef unsigned long Ulong ;
typedef unsigned char uchar ;
typedef unsigned int uint ;
typedef unsigned long ulong ;
#define TRO_RELOAD (-921)

Uint count;
Uchar digit;

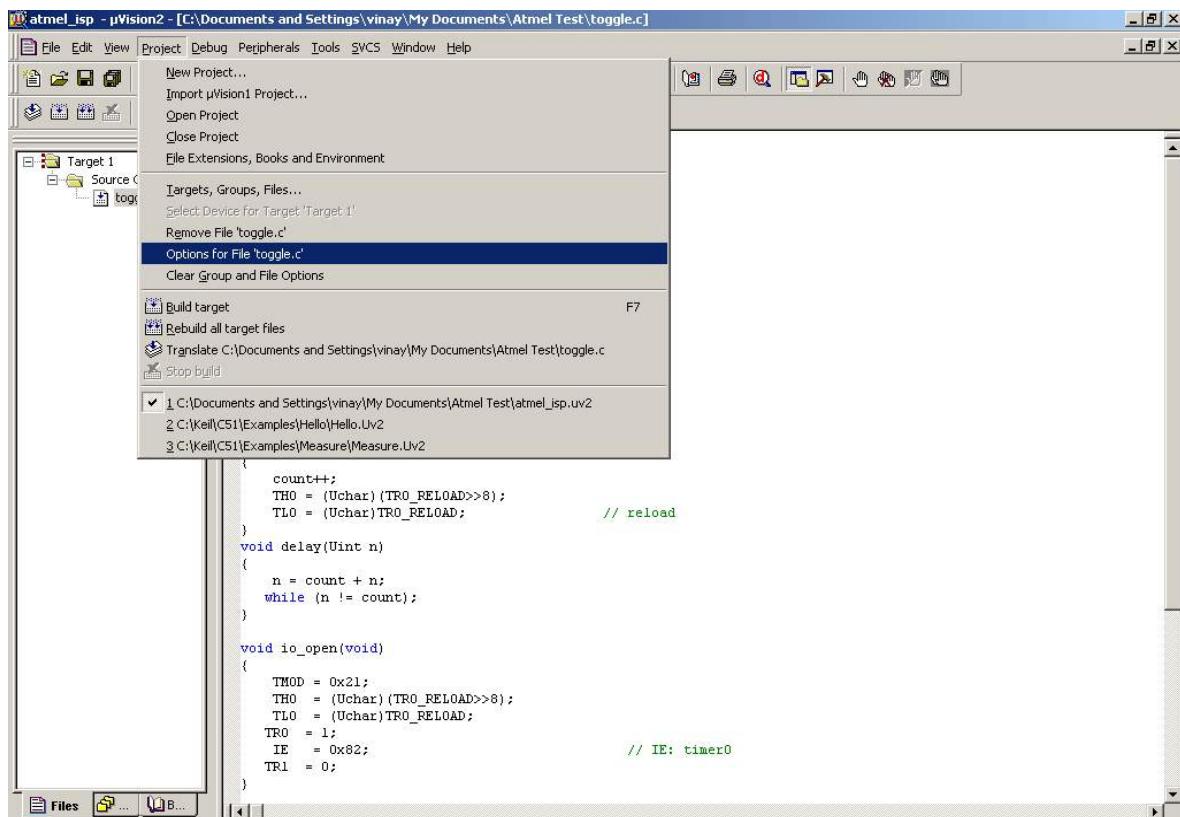
void timer0(void)
interrupt 1 using 1
{
    count++;
    TH0 = (Uchar)(TRO_RELOAD>>8);
    TL0 = (Uchar)TRO_RELOAD;           // reload
}
void delay(Uint n)
{
    n = count + n;
    while (n != count);
}

void io_open(void)
{
    TMOD = 0x21;
    TH0 = (Uchar)(TRO_RELOAD>>8);
    TL0 = (Uchar)TRO_RELOAD;
    TR0 = 1;                           // IE: timer0
    IE = 0x82;
    TR1 = 0;
}

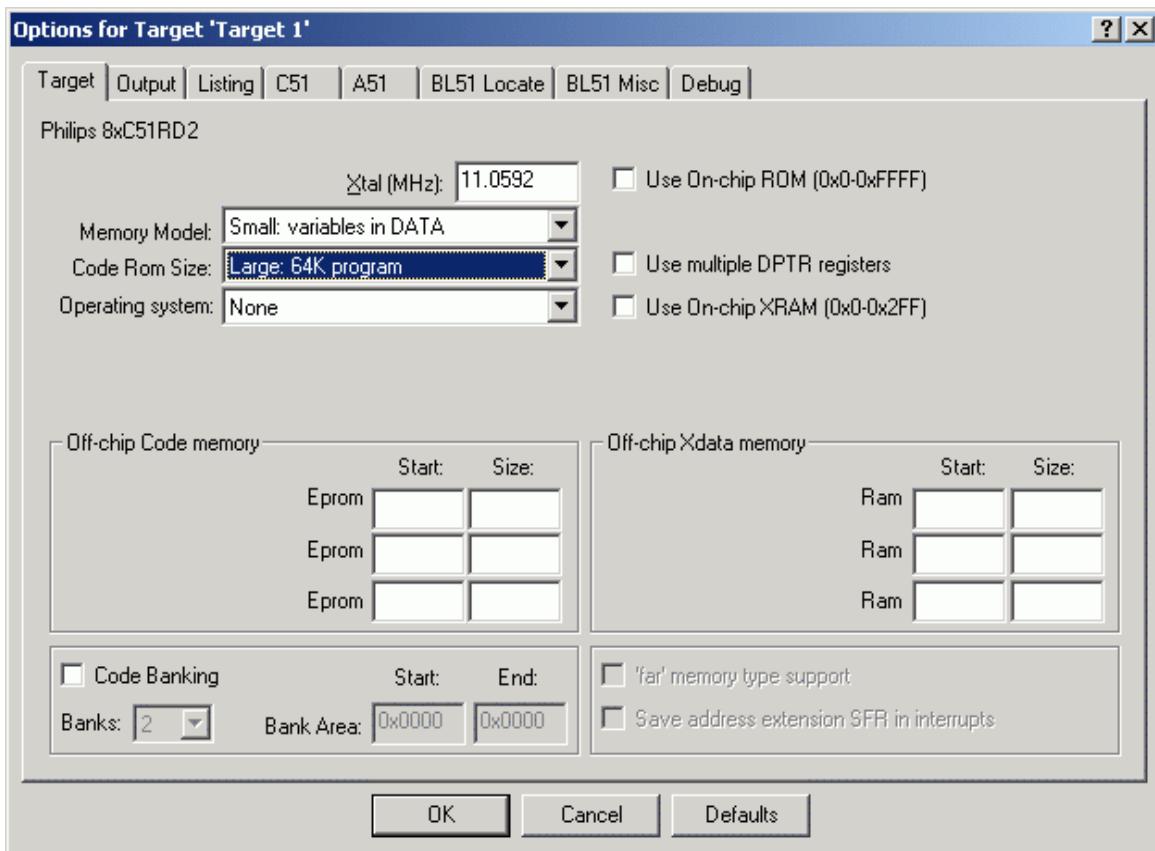
```

**Step 7:** For building the program code or HEX code, we need to set some parameters in the compiler.

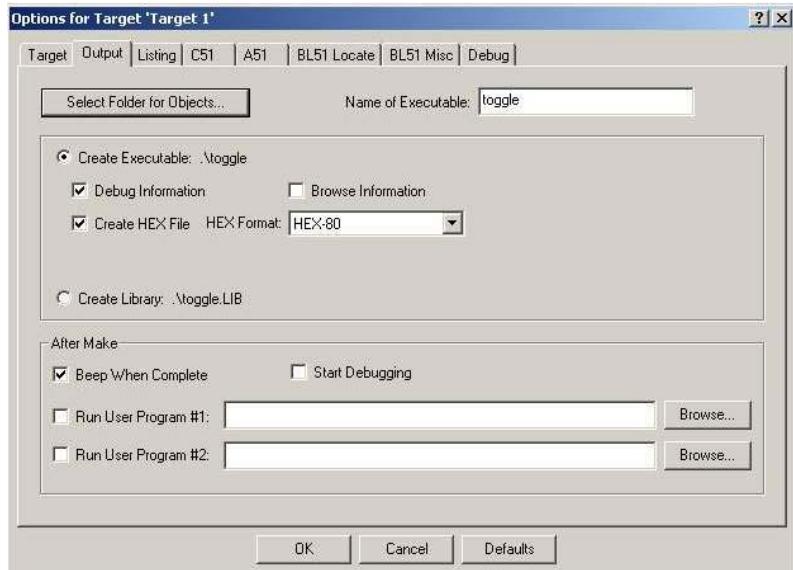
Go to project options, and click option for target.



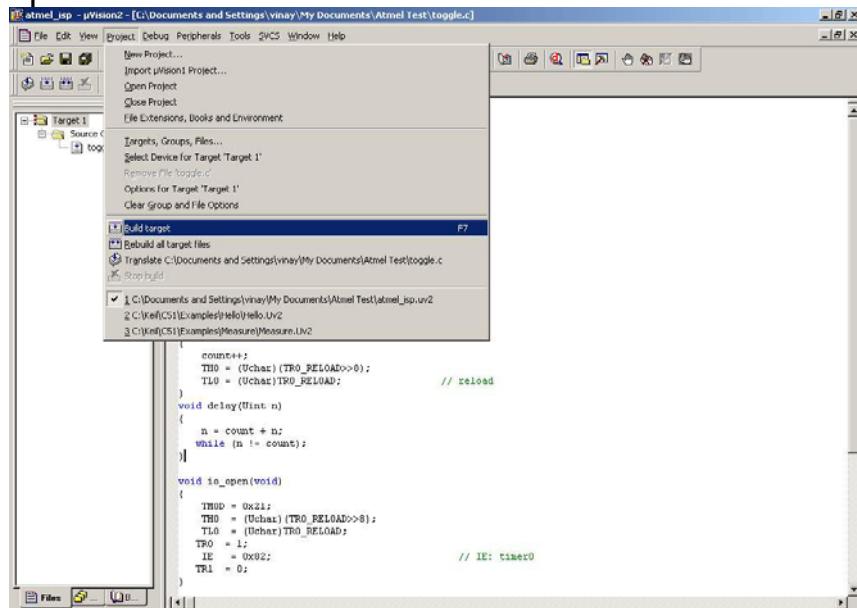
**Step 8:** In the opened window, go to target tab, and in the window, set the Xtal frequency as 11.0592 MHz.



**Step 9:** Now goto **output** tab and set the **name of executable file** as **toggle**; set the option **create executable**; and set **create HEX file** option. Finally click **OK** on the bottom side and come out of the window.



**Step 10:** Now to build the HEX file, goto **project** menu, and click the **build target** option.



This will generate the HEX file in the project folder, which you can use to program the controller using the provided **Atmel ISP** programming software.

Source code for **toggle** program.

89c51 Source code: toggle.c  
HEX file : toggle.hex

## Chapter 6: Configuration/Downloading

For Xilinx devices.

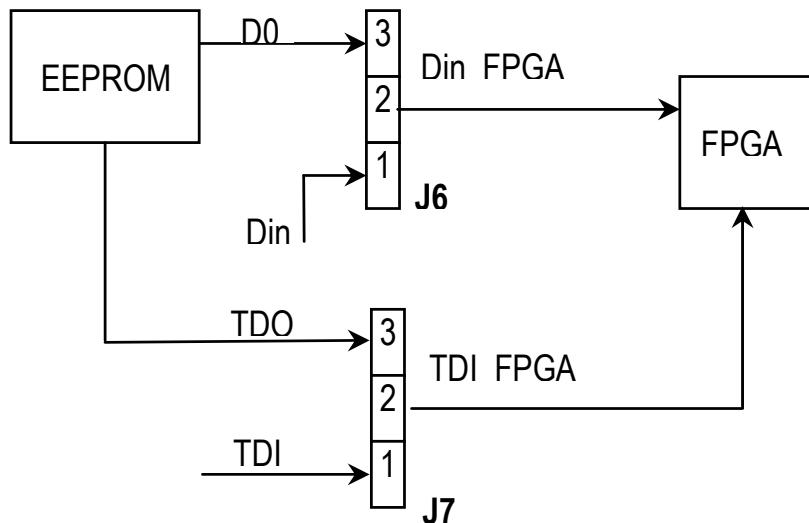
<b>Mode Selection Header J8 – J11</b>	
Slave Serial Short 1-2	JTAG Short 2-3
DIN	TDI
CCLK	TCK
PROG	TMS
DONE	TDO

<b>Mode Selection Switch</b>			
	M0	M1	M2
JTAG	1	0	1
Slave Serial	1	1	1
Master Serial	0	0	0

The FLASH PROM is connected with FPGA through jumpers. While configuring the PLD from PC, the PROM has to be bypassed (JTAG & Slave serial modes).

### Jumper selection for PROM

	Bypass PROM	Use PROM
<b>J6</b>	Short 1-2	Short 2-3
<b>J7</b>	Short 1-2	Short 2-3



**Note:** Altera device adaptors can be programmed only in JTAG mode.  
For more programming details, refer the respective device datasheet.

## Using Variable Frequency Generator

MATrix-II comes along with Variable Frequency Generator which users can use for low frequency applications, thus saving area and power.

Also this Frequency Generator can be used for applying low frequency signals for time based applications.

To use this signal as clock to PLD, do the following

- Short 1-2 pins of jumpers J4 & J5.
- Rotating the pot R50 in clockwise direction will decrease the frequency and vice-versa.
- To use it as signal, connect pin no. 1 of J4 header to any of the user I/O, then pin lock that I/O to the PLD and use in your application design.

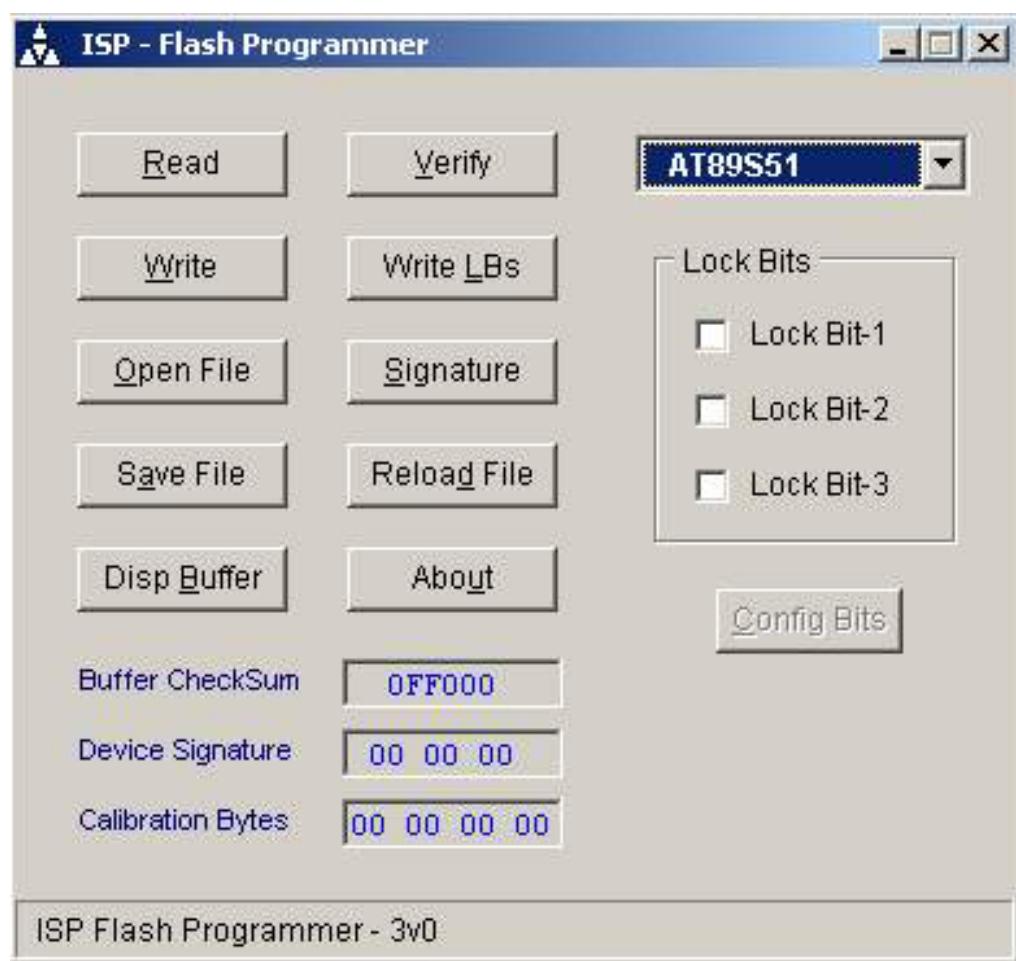
## Programming Atmel Microcontroller

Using the ISP programmer, designers can program the AT89S8252 controller. After building the HEX file from compiler, designers can refer the below shown steps to program the controller.

Insert the controller module on **MATrix baseboard**; connect the cable provided to module and PC's parallel port. Turn ON the power supply; now run the EXE of ISP programmer on your PC.

**Note:** For win98 OS run **ISP-Pgm3v0.exe**, and for win NT/2000/XP OS run **ISP-XP.bat** batch file.

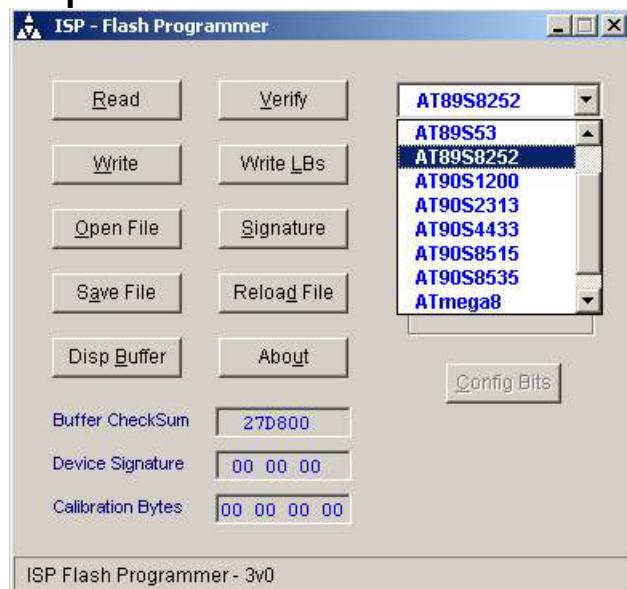
The below shown window will open up.



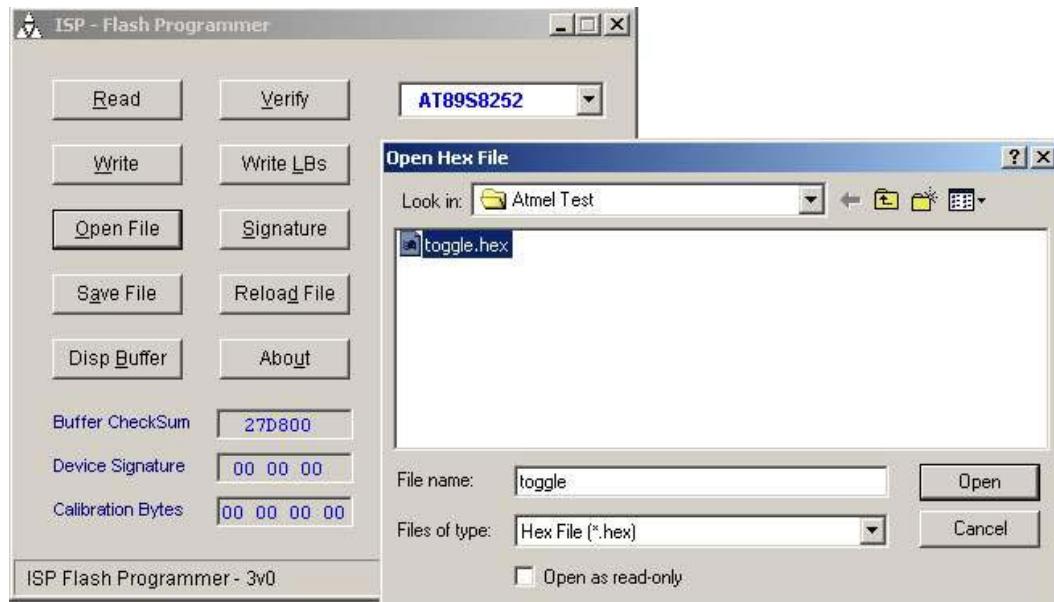
### Features of Programmer:

- Supports AVR & 89s series of controllers.
- Security bit locking facility.
- Write & Read facility.
- Verification included during programming.
- Easy to use

**Step 1:** Select the controller as **AT89S8252** on the RHS of opened window.



**Step 2:** Now Click the **Open file** button and browse to your project folder to open the currently created HEX in the programmer.



**Step 3:** Now Click the **Write** button to program the device. User can also **Write** couple of times to ensure the write process.



As this microcontroller module is connected directly with FPGA, so user has have to program the FPGA for feed through net program to get the signals from microcontroller and display on LEDs or some where else.

For using this controller module on MATrix kit, users have to use FPGA as switch in between.

*Source code for **toggle** program.*

FPGA Source code: toggle\_feed.vhdl

Pin Lock file : toggle\_feed\_UCF.ucf

#### **Header Pin Details:**

J4 (Timer/Interrupts)

Pin No.	Signal
1	Int 0
2	Int 1
3	T0
4	T1
5	Ground

J5 (ISP Header)

Pin No.	Signal
1	SCK
2	MISO
3	MOSI
4	RST
5	Ground

## Chapter 7: Pin Assignment

### For Xilinx Devices

- Spartan-II FPGA (XC2S30PQ208)
- XC9500 CPLD (XC95xx PC84)

Clock and Reset			O/P LEDs						7 segment Display		
	FPGA	CPLD		FPGA	CPLD		FPGA	CPLD		FPGA	CPLD
Reset	206	74	L 15	45	35	L 7	61	45	Seg A	27	34
Clock	80	10	L 14	46	36	L 6	62	46	Seg B	29	33
			L 13	47	37	L 5	63	47	Seg C	30	32
<b>Relay Header</b>			L 12	48	39	L 4	67	48	Seg D	31	31
	FPGA	CPLD	L 11	49	40	L 3	68	50	Seg E	33	26
Relay 1	23	13	L 10	57	41	L 2	69	51	Seg F	34	25
Relay 2	24	11	L 9	58	43	L 1	70	52	Seg G	35	24
			L 8	59	44	L 0	71	53	Seg DP	36	23
<b>RS-232 Port</b>									DISP En 1	37	19
	FPGA	CPLD							DISP En 2	41	17
RXD	193	NA							DISP En 3	42	15
TXD	192	NA							DISP En 4	43	14

NA: Not Available

**Note:** The RS-232 port signals are being shared with FPGAs also, so if user is using RS-232 port along with microcontroller then those FPGA I/Os has to been left floating.

## For Xilinx Devices

- Spartan-II FPGA (XC2S30PQ208)
- XC9500 CPLD (XC95xx PC84)

Configurable Switches								
I/O	FPGA	CPLD	I/O	FPGA	CPLD	I/O	FPGA	CPLD
<b>S23</b>	73	54	<b>S15</b>	87	65	<b>S7</b>	99	75
<b>S22</b>	74	55	<b>S14</b>	88	66	<b>S6</b>	100	76
<b>S21</b>	75	56	<b>S13</b>	89	67	<b>S5</b>	101	77
<b>S20</b>	81	57	<b>S12</b>	90	68	<b>S4</b>	102	79
<b>S19</b>	82	58	<b>S11</b>	94	69	<b>S3</b>	108	80
<b>S18</b>	83	61	<b>S10</b>	95	70	<b>S2</b>	109	81
<b>S17</b>	84	62	<b>S9</b>	96	71	<b>S1</b>	110	82
<b>S16</b>	86	63	<b>S8</b>	98	72	<b>S0</b>	111	83

## For Xilinx Devices

- Spartan-II FPGA (XC2S30PQ208)
- XC9500 CPLD (XC95xx PC84)

Digital I/O & Dot Matrix Rolling display				LCD Header		
Digital I/Os	Dot Matrix	FPGA	CPLD		FPGA	CPLD
B7_0	R1	3	NA	LCD En	148	NA
B7_1	C1	4	NA	LCD RS	150	NA
B7_2	R2	5	NA	LCD7	151	NA
B7_3	C2	8	NA	LCD6	152	NA
B7_4	R3	10	NA	LCD5	153	NA
B7_5	C3	14	NA	LCD4	154	NA
B7_6	R4	15	NA	LCD3	160	NA
B7_7	C4	16	NA	LCD2	161	NA
B7_8	R5	17	NA	LCD1	162	NA
B7_9	C5	18	NA	LCD0	163	NA
B7_10	R6	21	NA			
B7_11	R7	22	NA			
Vref_B7_1		6	NA	Vref_B7_2	20	

**Note:** The digital I/Os are Bank 7 of FPGA. Refer datasheet before using them.

8051 Header			ADC/DAC & Keypad Header			
	FPGA	CPLD		Shared	FPGA	CPLD
P1_7 (Port 1)	164	NA	SL0	ADC_OE	120	4
P1_6 (Port 1)	166	NA	SL1	ADC_CLK	121	5
P1_5 (Port 1)	167	NA	SL2	SOC	122	6
P1_4 (Port 1)	168	NA	SL3	EOC	123	7
P1_3 (Port 1)	172	NA	RL0	ADC_ALE	113	3
P1_2 (Port 1)	173	NA	RI1	SEL0	114	2

P1_1 (Port 1)	174	NA	RL2	SEL1	115	1
P1_0 (Port 1)	175	NA	RL3	SEL2	119	84
P2_7 (Port 2)	176	NA	D0	<i>DAC Data bus</i>	135	NA
P2_6 (Port 2)	178	NA	D1	<i>DAC Data bus</i>	134	NA
P2_5 (Port 2)	179	NA	D2	<i>DAC Data bus</i>	133	NA
P2_4 (Port 2)	180	NA	D3	<i>DAC Data bus</i>	132	NA
P2_3 (Port 2)	181	NA	D4	<i>DAC Data bus</i>	129	NA
P2_2 (Port 2)	187	NA	D5	<i>DAC Data bus</i>	127	NA
P2_1 (Port 2)	188	NA	D6	<i>DAC Data bus</i>	126	NA
P2_0 (Port 2)	189	NA	D7	<i>DAC Data bus</i>	125	NA
P0_7 (Port 0)	194	NA	AD0	<i>ADC Data bus</i>	147	NA
P0_6 (Port 0)	195	NA	AD1	<i>ADC Data bus</i>	146	NA
P0_5 (Port 0)	199	NA	AD2	<i>ADC Data bus</i>	142	NA
P0_4 (Port 0)	200	NA	AD3	<i>ADC Data bus</i>	141	NA
P0_3 (Port 0)	201	NA	AD4	<i>ADC Data bus</i>	140	NA
P0_2 (Port 0)	203	NA	AD5	<i>ADC Data bus</i>	139	NA
P0_1 (Port 0)	204	NA	AD6	<i>ADC Data bus</i>	138	NA
P0_0 (Port 0)	205	NA	AD7	<i>ADC Data bus</i>	136	NA
ALE	185	NA				
WR	182	NA				
RD	191	NA				

NA: Not Available

**Shared:** Some ADC/DAC I/Os are shared with Keypad I/Os, so only one interface can be done at a time.

**Note:** Lock the listed I/Os of entity in **User Constraints File (UCF)** with the above pin numbers before going for implementation process.

## For Altera Devices

- ACEX 1K FPGA (EP1k50 Q144)
- MAX7000S CPLD (EPM7125SLC84)

Clock and Reset			O/P LEDs						7 segment Display		
	FPGA	CPLD		FPGA	CPLD		FPGA	CPLD		FPGA	CPLD
Reset	122	1	L 15	31	35	L 7	43	46	Seg A	17	25
Clock	55	83	L 14	32	36	L 6	44	48	Seg B	18	27
			L 13	33	37	L 5	46	49	Seg C	19	28
<b>Relay Header</b>			L 12	36	39	L 4	47	50	Seg D	20	29
	FPGA	CPLD	L 11	37	40	L 3	48	51	Seg E	21	30
Relay 1	12	22	L 10	38	41	L 2	49	52	Seg F	22	31
Relay 2	13	24	L 9	39	44	L 1	51	54	Seg G	23	33
			L 8	41	45	L 0	59	55	Seg DP	26	34
<b>RS-232 Port</b>									DISP En 1	27	20
	FPGA	CPLD							DISP En 2	28	21
RXD	124	NA							DISP En 3	29	4
TXD	140	NA							DISP En 4	30	5

NA: Not Available

**Note:** The RS-232 port signals are being shared with FPGAs also, so if user is using RS-232 port along with microcontroller then those FPGA I/Os has to been left floating.

## For Altera Devices

- ACEX 1K FPGA (EP1k50 Q144)
- MAX7000S CPLD (EPM7125SLC84)

Configurable Switches								
I/O	FPGA	CPLD	I/O	FPGA	CPLD	I/O	FPGA	CPLD
<b>S23</b>	60	56	<b>S15</b>	70	67	<b>S7</b>	83	77
<b>S22</b>	62	57	<b>S14</b>	72	68	<b>S6</b>	86	79
<b>S21</b>	63	58	<b>S13</b>	73	69	<b>S5</b>	87	80
<b>S20</b>	64	60	<b>S12</b>	78	70	<b>S4</b>	88	81
<b>S19</b>	65	61	<b>S11</b>	79	73	<b>S3</b>	89	18
<b>S18</b>	67	63	<b>S10</b>	80	74	<b>S2</b>	90	17
<b>S17</b>	68	64	<b>S9</b>	81	75	<b>S1</b>	91	16
<b>S16</b>	69	65	<b>S8</b>	82	76	<b>S0</b>	92	15

Digital I/O & Dot Matrix Rolling display		
Dot Matrix	FPGA	CPLD
R1	132	NA
C1	133	NA
R2	135	NA
C2	136	NA
R3	137	NA
C3	138	NA
R4	143	NA
C4	144	NA
R5	8	NA
C5	9	NA
R6	10	NA
R7	11	NA

<b>ADC/DAC &amp; Keypad Header</b>			
	<b>Shared</b>	FPGA	CPLD
SL0	ADC_OE	99	9
SL1	ADC_CLK	100	10
SL2	SOC	101	11
SL3	EOC	102	12
RL0	ADC_ALE	95	84
RI1	SEL0	96	2
RL2	SEL1	97	6
RL3	SEL2	98	8
D0	<i>DAC Data bus</i>	117	NA
D1	<i>DAC Data bus</i>	116	NA
D2	<i>DAC Data bus</i>	114	NA
D3	<i>DAC Data bus</i>	113	NA
D4	<i>DAC Data bus</i>	112	NA
D5	<i>DAC Data bus</i>	111	NA
D6	<i>DAC Data bus</i>	110	NA
D7	<i>DAC Data bus</i>	109	NA
AD0	<i>ADC Data bus</i>	131	NA
AD1	<i>ADC Data bus</i>	130	NA
AD2	<i>ADC Data bus</i>	128	NA
AD3	<i>ADC Data bus</i>	126	NA
AD4	<i>ADC Data bus</i>	121	NA
AD5	<i>ADC Data bus</i>	120	NA
AD6	<i>ADC Data bus</i>	119	NA
AD7	<i>ADC Data bus</i>	118	NA

NA: Not Available

**Shared:** Some ADC/DAC I/Os are shared with Keypad I/Os, so only one adaptor can be done at a time.

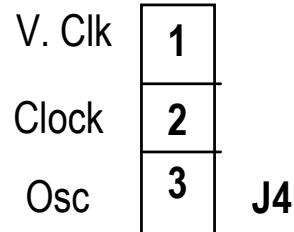
**Note:** Lock the listed I/Os of entity in **Quartus-II** assignment organizer before downloading the configuration file in the respective PLD.

## Chapter 8: Header and Jumper settings

### Clock

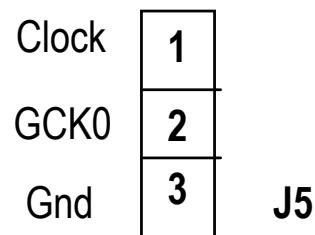
Selecting Clock type **J4**

Jumper Setting	Clock
1-2	Variable Clock (in KHz)
2-3	Oscillator (8 MHz)



Selecting Clock **J5**

Jumper Setting	Clock
1-2	Clock Selected
2-3	GND



**Note:** The GCK0 pin of FPGA is used for applying clock.

### Mode Selection Header

<b>J8 – J11</b>	
<b>Slave Serial Short 1-2</b>	<b>JTAG Short 2-3</b>
DIN	TDI
CCLK	TCK
PROG	TMS
DONE	TDO

### Mode Selection Switch

	M0	M1	M2
JTAG	1	0	1
Slave Serial	1	1	1
Master Serial	0	0	0

**Jumper selection for PROM**

	Bypass PROM	Use PROM
<b>J6</b>	Short 1-2	Short 2-3
<b>J7</b>	Short 1-2	Short 2-3

**Configurable I/Os (J1 – J3)**

I/O number	For Input	For Output
S23-S0	Short 1-2	Keep open

Header Name	Ident
Relay Header	JP5
Digital I/Os and Rolling display	JP3
8051 Header	JP2
LCD Header	JP4
ADC/DAC & Keypad	JP1
PLD Header	JH1, JH2, JH3, JH4
Power supply	JP6

**Relay Header Connections (JP5)**

1	2	3	4	5
Relay 1	Relay 2	+12V	+5V	GND

**Digital I/Os and Rolling display Header Connections (JP3)**

1	3	5	7	9	11	13	15	17	19
C1	C2	C3	C4	C5	R	NC	VREF_B7_1	VREF_B7_2	NC
2	4	6	8	10	12	14	16	18	20
R1	R2	R3	R4	R5	R6	+5V	+12V	GND	GND

**8051 Module Header Connections (JP2)**

2	4	6	8	10	12	14	16	18
+5V	P0_1	P0_3	P0_5	P0_7	Rst	TXD	ALE	P2_0
1	3	5	7	9	11	13	15	17
+5V	P0_0	P0_2	P0_4	P0_6	RXD	NC	P2_1	WR

<b>20</b>	<b>22</b>	<b>24</b>	<b>26</b>	<b>28</b>	<b>30</b>	<b>32</b>	<b>34</b>	<b>36</b>
P2_2	RD	P2_3	P2_5	P2_6	P1_0	P1_2	P1_5	P1_7
<b>19</b>	<b>21</b>	<b>23</b>	<b>25</b>	<b>27</b>	<b>29</b>	<b>31</b>	<b>33</b>	<b>35</b>
P2_4	P2_7	P1_1	P1_3	P1_4	P1_6	GND	GND	GND

**LCD Header Connections (JP4)**

<b>2</b>	<b>4</b>	<b>6</b>	<b>8</b>	<b>10</b>	<b>12</b>	<b>14</b>	<b>16</b>
+3.3V	LCDRS	LCDEN	LCD1	LCD3	LCD6	LCD7	NC
<b>1</b>	<b>3</b>	<b>5</b>	<b>7</b>	<b>9</b>	<b>11</b>	<b>13</b>	<b>15</b>
GND	NC	NC	LCD0	LCD2	LCD4	LCD5	NC

**ADC/DAC & Keypad Header Connections (JP9)**

<b>1</b>	<b>3</b>	<b>5</b>	<b>7</b>	<b>9</b>	<b>11</b>	<b>13</b>	<b>15</b>
+3.3V	SL3/EOC	SL2/SOC	SL1/CLK_ADC	SL0/OE	RL3/SEL2	RL2/SEL1	RL1/SEL0
<b>2</b>	<b>4</b>	<b>6</b>	<b>8</b>	<b>10</b>	<b>12</b>	<b>14</b>	<b>16</b>
AD0	AD1	AD2	AD3	AD4	AD5	AD6	AD7

<b>17</b>	<b>19</b>	<b>21</b>	<b>23</b>	<b>25</b>	<b>27</b>	<b>29</b>	<b>31</b>
RL0/ADC_ALE	GND						
<b>18</b>	<b>20</b>	<b>22</b>	<b>24</b>	<b>26</b>	<b>28</b>	<b>30</b>	<b>32</b>
D7	D6	D5	D4	D3	D2	D1	D0

<b>33</b>	<b>35</b>	<b>37</b>	<b>39</b>
GND	GND	-5V	+5V
<b>34</b>	<b>36</b>	<b>38</b>	<b>40</b>
NC	+12V	-5V	+5V

## General I/O Header

These are copy of PLD headers which can be used for probing or external circuit interfacing.

JH6 (Left)				JH5 (Right)			
Pin No.	Signal	Pin No.	Signal	Pin No.	Signal	No.	Signal
1	GND	2	TMS	1	+3.3V	2	CCLK
3	B7_0	4	B7_1	3	LCD4	4	LCD5
5	B7_2	6	VREF_B7_1	5	LCD6_DIN	6	LCD7
7	+1.2V	8	B7_3	7	LCDRS	8	NC
9	+1.2V	10	B7_4	9	LCDEN	10	AD0
11	GND	12	+3.3V	11	AD1	12	GND
13	+2.5V	14	B7_5	13	+3.3V	14	+2.5V
15	B7_6	16	B7_7	15	AD2	16	AD3
17	B7_8	18	B7_9	17	AD4	18	AD5
19	GND	20	VREF_B7_2	19	AD6	20	GND
21	B7_10	22	B7_11	21	AD7	22	D0
23	RELAY1	24	RELAY	23	D1	24	D2
25	GND	26	+3.3V	25	D3	26	GND
27	SEGA	28	+2.5V	27	+3.3V	28	D4
29	SEGB	30	SEGC	29	+2.5V	30	D5
31	SEGD	32	GND	31	D6	32	D7
33	SEGE	34	SEGF	33	GND	34	SL3_EOC
35	SEGG	36	SEGDP	35	SL2_SOC	36	SL1_CLK_ADC
37	EN1	38	+2.5V	37	SL0_OE	38	RL3_SEL2
39	+3.3V	40	GND	39	+2.5V	40	+3.3V
41	EN2	42	EN3	41	GND	42	RL2_SEL1
43	EN4	44	NC	43	RL1_SEL0	44	RL0_ADC_ALE
45	OP15	46	OP14	45	NC	46	SW1_0
47	OP13	48	OP12	47	SW1_1	48	SW1_2
49	OP11	50	M1	49	SW1_3	50	INIT
51	GND	52	M0	51	PROG	52	+3.3V

### Note:

- Signal “OP” is denoted for O/P LEDs. So **OP15** indicates **L15** & **OP0** indicates **L0** on board.
- Signal “SW1\_0” is denoted for configurable I/Os. So **SW1\_0** indicates **S0** & **SW1\_7** indicates **S7** on board.
- **SW2\_0** indicates **S8** & **SW2\_7** indicates **S15** on board.
- **SW3\_0** indicates **S16** & **SW3\_7** indicates **S23** on board.

## General I/O Header

JH7 (Bottom)				JH8 (Top)			
Pin No.	Signal	Pin No.	Signal	Pin No.	Signal	Pin No.	Signal
1	+3.3V	2	M2	1	+3.3V	2	TCK
3	+5V	4	+5V	3	RST	4	P0_0
5	OP10	6	OP9	5	P0_1	6	P0_2
7	OP8	8		7	+1.2V	8	P0_3
9	OP7	10	OP6	9	P0_4	10	P0_5
11	OP5	12	GND	11	GND	12	+3.3V
13	+3.3V	14	+2.5V	13	+2.5V	14	P0_6
15	OP4	16	OP3	15	P0_7	16	RXD
17	OP2	18	OP1	17	TXD	18	RD
19	OP0	20	GND	19	GND	20	P2_0
21	SW3_7	22	SW3_6	21	P2_1	22	P2_2
23	SW3_5	24	+2.5V	23	+2.5V	24	ALE
25	NC	26	+3.3V	25	+3.3V	26	GND
27	GND	28	GCK0	27	WR	28	P2_3
29	SW3_4	30	SW3_3	29	P2_4	30	P2_5
31	SW3_2	32	SW3_1	31	P2_6	32	GND
33	GND	34	SW3_0	33	P2_7	34	P1_0
35	SW2_7	36	SW2_6	35	P1_1	36	P1_2
37	SW2_5	38	SW2_4	37	P1_3	38	+2.5V
39	+2.5V	40	+3.3V	39	+3.3V	40	GND
41	GND	42	SW2_3	41	P1_4	42	P1_5
43	SW2_2	44	SW2_1	43	P1_6	44	NC
45	NC	46	SW2_0	45	P1_7	46	LCD0
47	SW1_7	48	SW1_6	47	LCD1	48	LCD2
49	SW1_5	50	SW1_4	49	LCD3	50	TDI_FP
51	GND	52	DONE	51	GND	52	TDO

## Power Header (JP6)

Pin No.	Signal
1.	Gnd
2.	+5V
3.	-5V
4.	+12V

## Chapter 9: Using External Adaptors

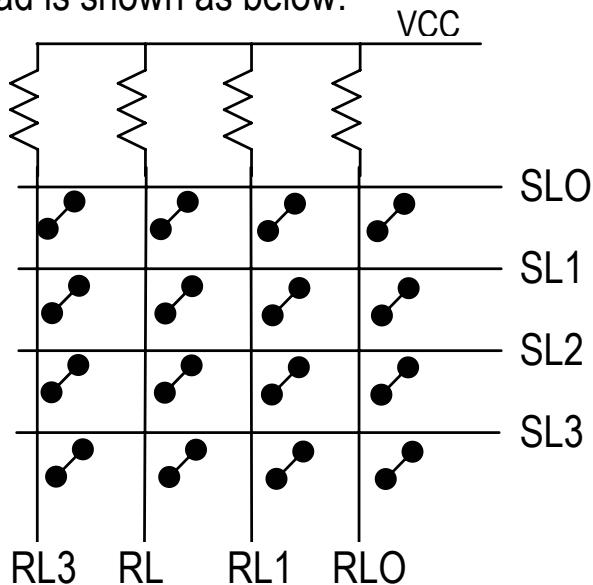
MATrix comes along with the following optional adaptors/modules

1. Keypad adaptor
2. LCD adaptor
3. Dot matrix rolling display card module
4. Relay card module
5. 89C51 adaptor
6. ADC/DAC adaptor

Users can use these adaptors for their solving their design problems and needs.  
Here is the ways and methods to use these adaptors/modules.

### Keypad adaptor

The structure of keypad is shown as below.



User has to scan the “scan lines” (SL0 to SL3) by placing logic ‘0’ one by one on every line and if a key is pressed during scanning particular SL is enabled then the return value will be ‘0’ for that return line too (RL0 to RL3).

With the combination of scan line and return lines, user can come to know about the key hit.

Please refer samples codes in VHDL provided in CD-ROM for further details.

**Inserting Module:** Insert the keypad adaptor by aligning **first** pin nos. of adaptor and baseboard header. Also insertion indication is marked on board.

### LCD module

A 16x2 character LCD display is been provided (optional) with MATrix kit, which can use for message display.

Here is the list of signals used for LCD (can be used only in write mode).

**LCD En**                   **--- LCD enable**

**LCD RS**

**LCD7 - LCD0**           **--- 8-bit LCD data bus**

As there is dedicated routine and protocol to use LCD displays, hence kindly refer the data sheet of oriole 16x2 LCD displays before using it.

It is recommend to use microcontroller for controlling LCD display as it is more easy to use and implement. In either case **ni logic** provides sample codes for microcontroller and FPGA to control LCD display.

*Note: The sample codes are provided along with the kit.*

### Dot matrix rolling display card

Four 5x7 dot matrix displays have been provided on the external module to design and develop to display rolling messages.

The vertical lines are called **columns** and horizontal as **rows**.

There are 7 rows and 20 columns in total, giving 140 LED indications in pixels.

The users have to put character values on rows and select the particular line of a column to display the character line.

User has to put encoded value of columns in binary on the column data bus, as all the columns are decoded onboard of module.

For col= "00000", **C1** would be enabled and for col="10011", **C20** would be enabled.

In the figure 9.1 below shown is complete matrix of LEDs, to use complete display user has to send 20 character set for every columns.

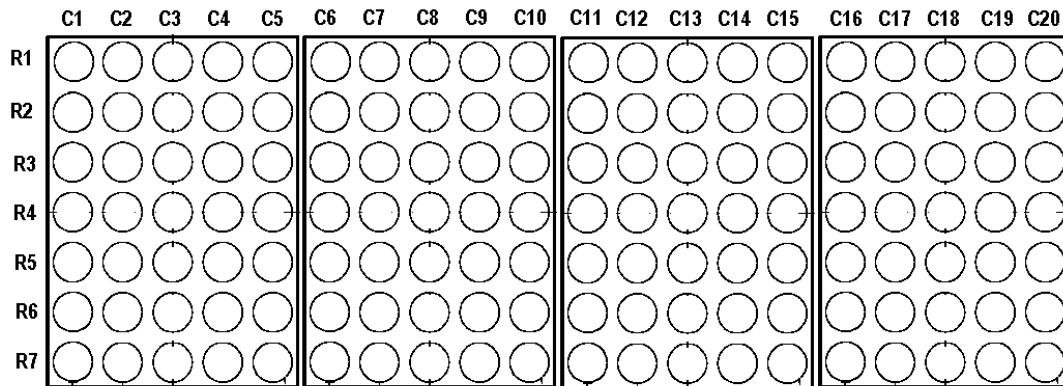
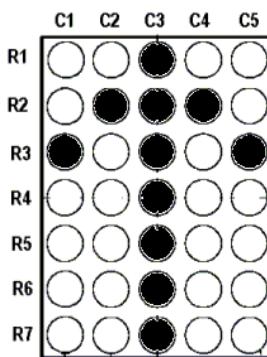


Figure 9.1  
Let's take  
an  
example  
for  
displaying  
a

character/symbol on one matrix shown as below in figure 9.2



Now to display this arrow symbol, user has to send following **row** values for 5 **column** lines.

10h, 20h, 7Fh, 20h, 10h

in HEX

"0010000", "0100000", "1111111", "0100000", "0010000"

in Binary

For generating set of characters, user can use demo software given along with the kit.

Note: The sample codes are provided along with the kit.  
Also character generator software is provided in CD-ROM.

## **Relay card**

A separate module is been provided for interfacing optically isolated relays. Two relays are been provided onboard with its three I/Os on separate header, known as Normally Closed (NC), Normally Open (NO) and Pole. Relays are energized at +12v and are of 7amps rating. To energize the relays user can apply logic '1' on their respective pin nos., which will turn on the relay and would be indicated by LEDs on board.

## **89C51 Adaptor**

AT89s8252 microcontroller from Atmel is provided as additional adaptor with MATrix-II. User can write programs in assembly or 'C' and compile in any of the compliers, for eg. Keil, etc.

The program file in Intel 'HEX' format can be downloaded with the programmer provided.

All ports of AT89s8252 are brought on the FPGA header with some control signals. The timer and interrupts signals are brought on separate jumper of adaptor.

The RS-232 lines (RXD & TXD) are connected with RS-232 port and also with FPGA.

User has to take care while using RXD & TXD lines; only one of the controller (FPGA or AT89s8252) can use these lines as they are shorted to each other.

For more information on microcontroller, visit [www.atmel.com](http://www.atmel.com)

**Inserting Module:** Insert the 89C51 adaptor by aligning **first** pin nos. of adaptor and baseboard header. The component side of 89C51 adaptor should face towards PLD headers.

## ADC/DAC adaptor (JP9)

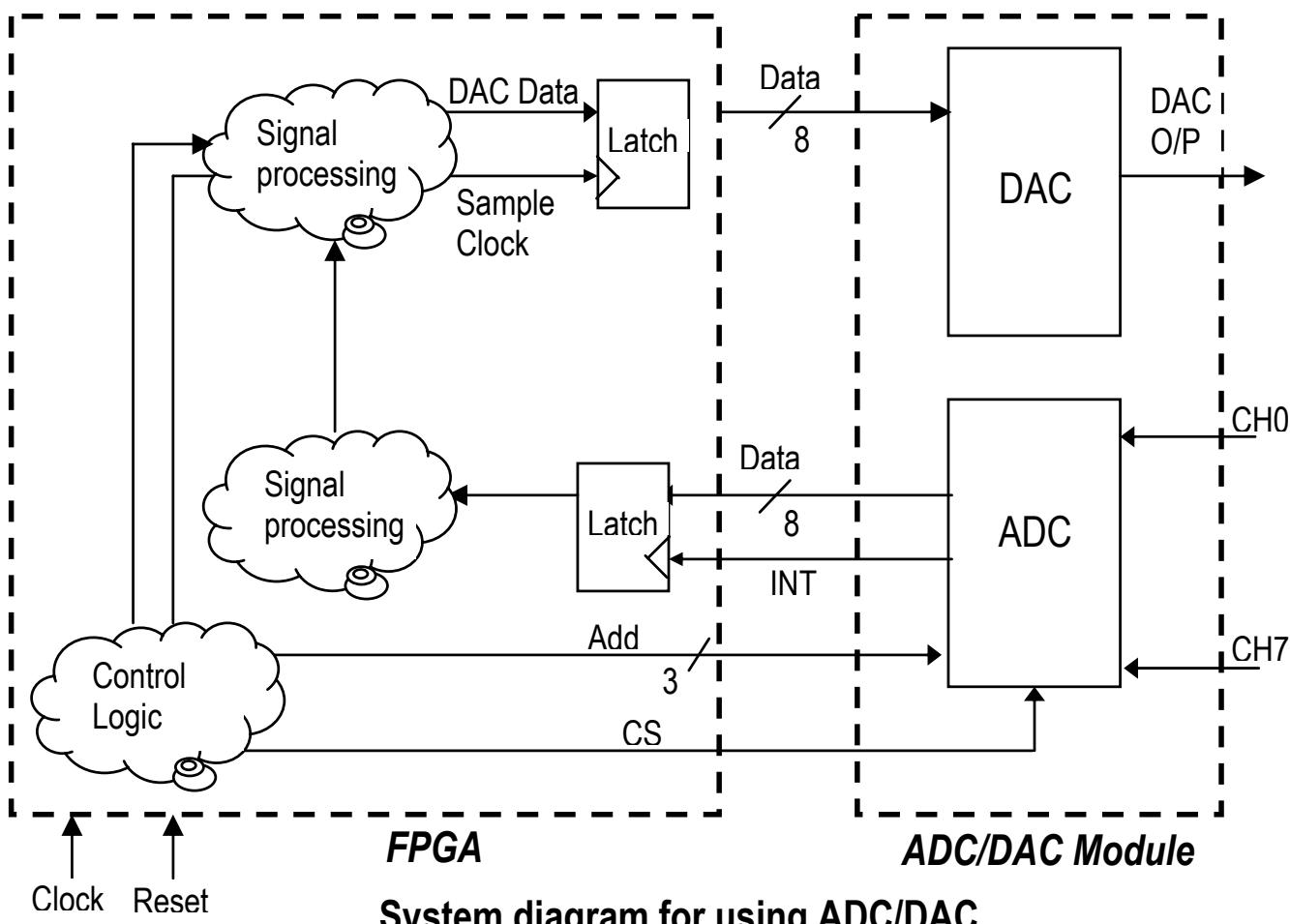
8 channels **ADC 0809** with sampling speed of 20KHz on single channel, and single channel **DAC0800** with 150ns settling time are provided on adaptor.

The I/Os of ADC & DAC directly connected with FPGA, user can control the ADC/DAC from FPGA.

Onboard reference voltage adjustment and gain adjustment in DAC are provided on board, user can set the values for setting desired voltage ranges and values.

In case of other o/p voltage ranges and values, user has to interface external circuitry with the adaptor.

To use the ADC and DAC, use the following logic shown in the below block diagram.



For further information kindly refer datasheets/application notes from, [www.national.com](http://www.national.com), [www.semiconductor.philips.com](http://www.semiconductor.philips.com)

**Inserting Module:** Insert the ADC/DAC adaptor by aligning **first** pin nos. of adaptor and baseboard header. The component side of ADC/DAC adaptor should face towards PLD headers.

## Chapter 10: Sample Code

User can use sample codes to work on MATrix trainer board, provided along with the kit.

The list of sample codes provided is mentioned below, for more information and getting new examples and code, user can visit our website [www.ni2designs.com](http://www.ni2designs.com)

- Digital logic code
  - Combination circuits
    - Basic logic gates
    - Binary to gray converter
    - 7 segment decoder
    - 3:8 decoder
    - Demultiplexer
    - Multiplexer
    - Parity generator
    - Full adder – behavioral model
    - Full adder- structural model
    - Half adder
    - 4 bit ALU
    - Model of IC 74xx245
    - Model of IC 74181 (4 bit ALU)
  - Sequential Logic
    - 4 bit binary counter
    - 4 bit universal binary counter
    - D F/F with asynchronous reset
    - D F/F with synchronous reset
    - SRAM model (16 bytes)
    - 4 bit shift register with enable, load and parallel o/p's

- Add on modules codes
  - ADC/DAC controller
  - 89c51 multiplier controller
  - Dot matrix display module
  - Keypad controller
  - Running LEDs
  - LCD
  - And many more....

All these sample codes and pin assignment (UCF) files are provided in the MATrix user CD, in case of any doubts or queries, visit [www.ni2designs.com](http://www.ni2designs.com)

## Chapter 11: Glossary of Terms

### **ASIC (Application Specific Integrated Circuit)**

A custom integrated circuit designed specifically for one end product or a closely related family of end products.

### **Concurrency**

The ability of an electronic circuit to do several (or at least two) different things at the same time. Contrast with computer programs, which usually execute only one instruction at a time unless the program is running on a processor with multiple, concurrent execution units.

### **CPLD (Complex Programmable Logic Device)**

A programmable IC which is more complex than the original Programmable Logic Devices such as AMD's (originally MMI's) PALs but somewhat less complex than Field Programmable Logic Arrays.

### **EDIF (Electronic Design Interchange Format)**

A standard representation format for describing electronic circuits, used to allow the interchange of circuit design information between EDA tools.

### **FPGA (Field Programmable Gate Array)**

An integrated circuit containing a large number of logic cells or gates that can be programmably configured after the IC has been manufactured. Some FPGAs use fuses for this programming and others store the configuration in an on chip EEPROM or RAM memory. Fuse programmed parts cannot be reprogrammed so they can only be configured once. EEPROM based FPGAs can be erased and reprogrammed so they can be configured many times. RAM based FPGAs can be reconfigured quickly, even while the circuit is in operation.

### **HDL (Hardware Description Language)**

A synthetic computer based language used for the formal description of electronic circuits. An HDL can describe a circuit's operation, its design, and a set of tests to verify circuit operation through simulation. The two most popular digital HDLs are VHDL and Verilog. An analog HDL called AHDL is under development by many vendors. HDLs make it easier to develop very large designs through formal software engineering methods that define ways to divide a large team project into smaller pieces that can be implemented by individual team members.

### **Moore's Law**

An empirical law developed and later revised by Intel's Gordon Moore which predicts that the IC industry is capable of doubling the number of transistors on a silicon chip

every 18 months (originally every year) resulting in declining IC prices and increasing performance. Most design cycles in the electronics industry including embedded system development firmly rely on Moore's law.

### **Net List (or Netlist)**

A computer file (sometimes a printed listing) containing a list of the signals in an electronic design and all of the circuit elements (transistors, resistors, capacitors, ICs, etc.) connected to that signal in the design.

### **PLCC (Plastic Leaded Chip Carrier)**

A low cost IC package (usually square). PLCCs have interconnection leads on either two (usually only for memory chips) or all four sides (for logic and ASIC chips).

### **PLD (Programmable Logic Device)**

The generic term for all programmable logic ICs including PLAs (programmable logic arrays), PALs, CPLDs (complex PLDs), and FPGAs (field programmable gate arrays).

### **PROM (Programmable Read Only Memory)**

An integrated circuit that stores programs and data in many embedded systems. PROM stores retains information even when the power is off but it can only be programmed or initialized once.

### **RTL (Register Transfer Level or Register Transfer Logic)**

A register level description of a digital electronic circuit. Registers store intermediate information between clock cycles in a digital circuit, so an RTL description describes what intermediate information is stored, where it is stored within the design, and how that information moves through the design as it operates.

### **Simulation**

Modeling of an electronic circuit (or any other physical system) using computer based algorithms and programming. Simulations can model designs at many levels of abstraction (system, gate, transistor, etc.). Simulation allows engineers to test designs without actually building them and thus can help speed the development of complex electronic systems. However, the simulations are only as good as the mathematical models used to describe the systems; inaccurate models lead to inaccurate simulations. Therefore, accurate component models are essential for accurate simulations.

### **Synthesis (also Logic Synthesis)**

A computer process that transforms a circuit description from one level of abstraction to a lower level, usually towards some physical implementation. Synthesis is to

hardware design what compilation is to software development. In fact, logic synthesis was originally called hardware compilation.

### **User Constraints File (UCF)**

A user created ASCII file for storing timing constraints and location constraints for a design implementation.

### **Verilog**

A hardware description language developed by Gateway Design Automation (now part of Cadence) in the 1980s which became very popular with ASIC and IC designers.

### **VHDL (VHSIC Hardware Description Language)**

A hardware description language developed in the 1980s by IBM, Texas Instruments, and Intermetrics under US government contract for the Department of Defense's VHSIC (Very High Speed Integrated Circuit ) program. VHDL enjoys a growing popularity with ASIC designers as VHDL development tools mature.

## Chapter 15 : Troubleshooting

### **Errors while programming FPGA**

There may be errors while programming FPGA, this may be due to many reasons, kindly check the following steps to recover the error.

- FPGA modules should be properly inserted.
- Check the jumper settings of FPGA module.
- Check the programming mode selection settings.
- See that the programming cable is properly inserted.
- Ground the clock (GCK0) I/P during programming; re-connect it after programming the FPGA.
- The parallel port of PC should be in ECP/EPP mode; else there would be connection errors.

### **Errors while programming 89c51**

There may be errors while programming 89c51, kindly check the following steps to rectify the error.

- Module should be properly inserted.
- The programming cable should be properly inserted.
- Turn off the power supply and re-start the operation.
- Check the Atmel ISP programmer settings (refer chapter configuration).
- Check the buffer of Atmel ISP programmer's buffer memory is properly filled.

### **ADC/DAC Module not working properly**

- Insert the module properly.
- Check the analog signal connections.
- Check the channel is properly selected.
- Check the ADC & DAC logic.
- For ADC debugging, connect its o/p to LEDs to check that's its is working or not.
- For DAC, connect switch I/Ps to DAC data I/P. Now vary switch values and check DC voltage at DAC o/p header.

## Disclaimer

**ni logic pvt. Ltd., Pune** takes the liability to replace the module/product for any design fault from our side.

This user manual is for supporting designers to develop their application. All configuration, usage, design examples, pin assignments are provided to help it's users. **ni logic pvt. Ltd., Pune** does not take any responsibility for typological errors, misprints, and damages caused with this manual.

**ni logic pvt. Ltd., Pune** does not take any responsibility of failures or damages caused to product due to incorrect design practices, misuse, improper handling or not following the datasheet specifications of devices.

This Product has been **tested successfully OK** for all modules and specification of the product.

Dated: \_\_\_\_\_

Ser. No.: \_\_\_\_\_

Production In-charge

Head - Design & Development

**Notes:**