Exploring FPDs For Design of Digital Hardware

Sayak Haldar(111205038)
Shuvam Bosana(111205042)
Subham Banerjee(111205014)
contact info:-sayakhaldar@ymail.com
shuvambosana0705@gmail.com
subhambansphs@gmail.com
Bengal Engineering & Science University,Shibpur
Department Name Computer Science And Technology

Project Outline

- Why FPDs?
- Steps of a typical FPGA based design
- An example of a simple digital design
- Implementing the proposed hardware through software

Project Related minor Work :

- Familiarisaton with VHDL
- Learning of A PCB Designing software
- Documentation of the Project and Presentation using appropri ate software and open source tools

Definitions of Relevant Terminology

- Field-Programmable Device (FPD):-a general term that refers to any type of integrated circuit used for implementing digital hardware, where the chip can be configured by the end user to realize dierent designs. Programming of such a device often involves placing the chip into a special programming unit, but some chips can also be configured "in-system". Another name for FPDs is programmable logic devices (PLDs); although PLDs encompass the same types of chips as FPDs, we prefer the term FPD because historically the word PLD has referred to relatively simple types of devices.

- FPGA : a Field-Programmable Gate Array is an FPD featuring a general structure that allows very high logic capacity. Whereas CPLDs feature logic resources with a wide number of inputs (AND planes), FPGAs offer more narrow logic resources. FPGAs also offer a higher ratio of flip-flops to logic resources than do CPLDs.

- VHDL :-(VHSIC Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field programmable gate arrays and integrated circuits. VHDL can also be used as a general purpose parallel programming lan- guage.

- PCB :-A printed circuit board (PCB) mechanically supports and electrically connects electronic components using conductive tracks, pads and other features etched from copper sheets laminated onto a non-conductive substrate. PCB's can be single sided (one copper layer), double sided (two copper layers) or multi-layer. Conductor on dierent layers are connected with plated-through holes called vias.

Why FPDs?

1 Evolution of Programmable Logic Device

The first type of user-programmable chip that could implement logic circuits was the Programmable Read-Only Memory (PROM), in which address lines can be used as logic circuit inputs and data lines as outputs. Logic functions, however, rarely require more than a few product terms, and a PROM contains a full decoder for its address inputs. PROMS are thus an inecient architecture for realizing logic circuits, and so are rarely used in practice for that purpose. The rst device developed later specically for implementing logic circuits was the Field-Programmable Logic Array (FPLA), or simply PLA for short. A PLA consists of two levels of logic gates: a programmable ”wired” AND-plane followed by a programmable ”wired” OR-plane. A PLA is structured so that any of its inputs (or their complements) can be AND'ed together in the AND-plane; each AND-plane output can thus correspond to any product term of the inputs. Similarly, each ORplane output can be congured to produce the logical sum of any of the AND-plane outputs. With this structure, PLAs are well-suited for implementing logic functions in sum-of-products form. They are also quite versatile, since both the AND terms and OR terms can have many inputs (this feature is often referred to as wide AND and OR gates).

1.1 Using a ROM as a PLD

Before PLDs were invented, read-only memory (ROM) chips were used to create arbitrary combinational logic functions of a number of inputs. Consider a ROM with m inputs (the address lines) and n outputs (the data lines). When used as a memory, the ROM contains 2m words of n bits each. Now imagine that the inputs are driven not

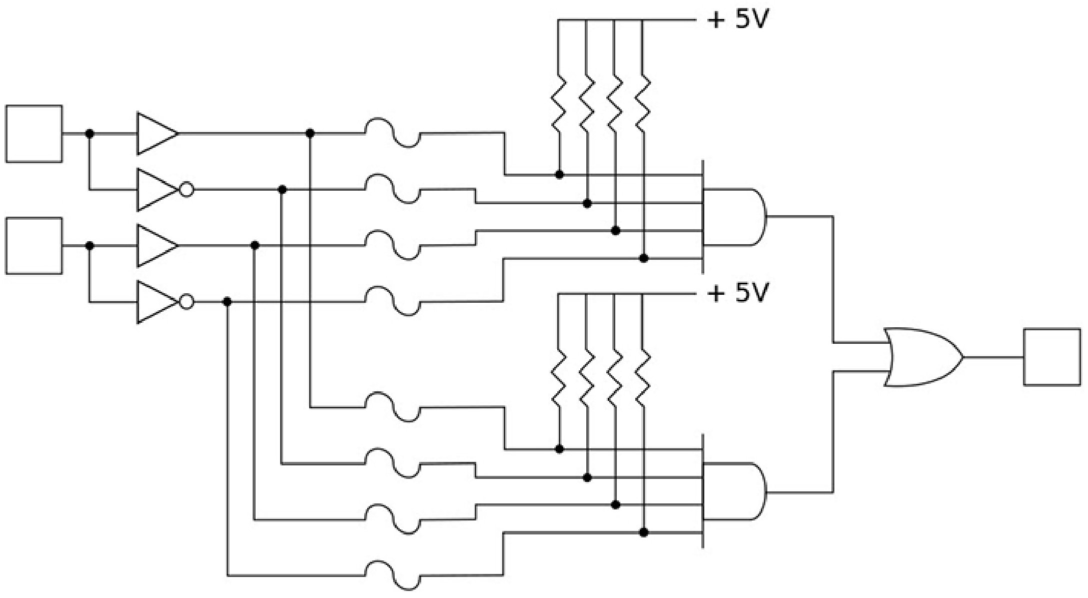


Figure 1: A simplified PAL device

by an m-bit address, but by m independent logic signals.

Theoretically, there are 22m possible Boolean functions of these m input signals. By Boolean function in this context is meant a single function that maps each of the 2m possible combinations of the m Boolean inputs to a single Boolean output. There are 22m possible distinct ways to map each of 2m inputs to a Boolean value, which explains why there are 22m such Boolean functions of m inputs. Now, consider that each of the n output pins acts, independently, as a logic device that is specially selected to sample just one of the possible 22m such functions. At any given time, only one of the 2m possible input values can be present on the ROM, but over time, as the input values span their full possible domain, each output pin will map out its particular function of the 2m possible input values, from among the 22m possible such functions. Note that the structure of the ROM allows just n of the 22m possible such Boolean functions to be produced at the output pins. The ROM therefore becomes equivalent to n separate logic circuits, each of which generates a chosen function of the m inputs. The advantage of using a ROM in this way is that any conceivable function of all possible combinations of the m inputs can be made to appear at any of the n outputs, making this the most general-purpose combinational logic device available for m input pins and n output pins. Also, PROMs (programmable ROMs), EPROMs (ultraviolet-erasable PROMs) and EEPROMs (electrically erasable PROMs) are available that can be programmed using a standard PROM programmer without requiring specialised hardware or software. However, there are several disadvantages:

- they are usually much slower than dedicated logic circuits,
- they cannot necessarily provide safe "covers" for asynchronous logic transitions so the PROM's outputs may glitch as the inputs switch,
- they consume more power,
- they are often more expensive than programmable logic, especially if high speed is required.

Since most ROMs do not have input or output registers, they cannot be used stand-alone for sequential logic. An external TTL register was often used for sequential designs such as state machines. Common EPROMs, for example the 2716, are still sometimes used in this way by hobby circuit designers, who often have some lying around. This use is sometimes called a 'poor man's PAL'.

1.2 Early programmable logic

In 1969, Motorola oered the XC157, a mask-programmed gate array with 12 gates and 30 uncommitted input/output pins. In 1970, Texas Instruments developed a mask-programmable IC based on the IBM read-only associative memory or ROAM. This device, the TMS2000, was programmed by altering the metal layer during the production of the IC. The TMS2000 had up to 17 inputs and 18 outputs with

8 JK flip flop for memory. TI coined the term Programmable Logic Array for this device. In 1971, General Electric Company (GE) was developing a programmable logic device based on the new PROM technology. This experimental device improved on IBM's ROAM by allowing multilevel logic. Intel had just introduced the oating-gate UV erasable PROM so the researcher at GE incorporated that technology. The GE device was the rst erasable PLD ever developed, predating the Altera EPLD by over a decade. GE obtained several early patents on programmable logic devices. In 1973 National Semiconductor introduced a mask-programmable PLA device (DM7575) with 14 inputs and 8 outputs with no memory registers. This was more popular than the TI part but cost of making the metal mask limited its use. The device is signicant because it was the basis for the eld programmable logic array produced by Signetics in 1975, the 82S100. (Intersil actually beat Signetics to market but poor yield doomed their part.) In 1974 GE entered into an agreement with Monolithic Memories to develop a mask- programmable logic device incorporating the GE innovations. The device was named the 'Programmable Associative Logic Array' or PALA. The MMI 5760 was completed in 1976 and could implement multilevel or sequential circuits of over 100 gates. The device was supported by a GE design environment where Boolean equations would be converted to mask patterns for conguring the device. The part was never brought to market.

1.3 PLA

In 1970, Texas Instruments developed a mask-programmable IC based on the IBM read-only associative memory or ROAM. This device, the TMS2000, was programmed by altering the metal layer during the production of the IC. The TMS2000 had up to 17 inputs and 18 outputs with 8 JK ip op for memory. TI coined the term Programmable Logic Array for this device. A programmable logic array (PLA) has

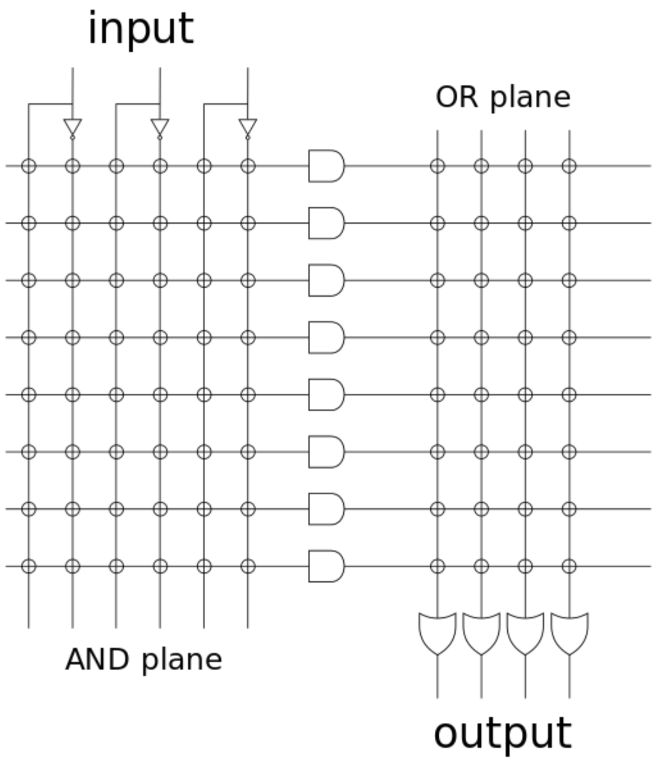


Figure 2: PLA schematic example

a programmable AND gate array, which links to a programmable OR gate array, which can then be conditionally complemented to produce an output.

1.4 PAL

PAL devices have arrays of transistor cells arranged in a "xed-OR, programmable-AND" plane used to implement "sum-of-products" binary logic equations for each of the outputs in terms of the inputs and either synchronous or asynchronous feedback from the outputs. MMI introduced a breakthrough device in 1978, the Programmable Array Logic or PAL. The architecture was simpler than that of Signetics FPLA because it omitted the programmable OR array. This made the parts faster, smaller and cheaper. They were available in 20 pin 300 mil DIP packages while the FPLAs came in 28 pin 600 mil packages. The PAL Handbook demystied the design process. The PALASM design software (PAL Assembler) converted the engineers' Boolean equations into the fuse pattern required to program the part. The PAL devices were soon second-sourced by National Semiconductor, Texas Instruments and AMD. After MMI succeeded with the 20-pin PAL parts, AMD introduced the 24-pin 22V10 PAL with additional features. After buying out MMI (1987), AMD spun o a consolidated operation as Vantis, and that business was acquired by Lattice Semiconductor in 1999.

1.5 GALs

An innovation of the PAL was the generic array logic device, or GAL, invented by Lattice Semiconductor in 1985. This device has the same logical properties as the PAL but can be erased and reprogrammed. The GAL is very useful in the prototyping stage of a design, when any bugs in the logic can be corrected by reprogramming. GALs are programmed and reprogrammed using a PAL programmer, or by using the in-circuit programming technique on supporting chips. Lattice GALs combine CMOS and electrically erasable (E2) oating gate technology for a high-speed, low-power logic device. A similar device called a PEEL (programmable electrically erasable logic) was introduced by the International CMOS Technology (ICT) corporation.

1.6 CPLDs

PALs and GALs are available only in small sizes, equivalent to a few hundred logic gates. For bigger logic circuits, complex PLDs or CPLDs can be used. These contain the equivalent of several PALs linked by programmable interconnections, all in one integrated circuit. CPLDs can replace thousands, or even hundreds of thousands, of logic gates. Some CPLDs are programmed using a PAL program-

Figure 3: Lattice GAL 16V8 and 20V8

mer, but this method becomes inconvenient for devices with hundreds of pins. A second method of programming is to solder the device to its printed circuit board, then feed it with a serial data stream from a personal computer. The CPLD contains a circuit that decodes the data stream and congures the CPLD to perform its specied logic function. Some manufacturers (including Altera and Microsemi) use JTAG to programme CPLD's in-circuit from .JAM les.

1.7 FPGA

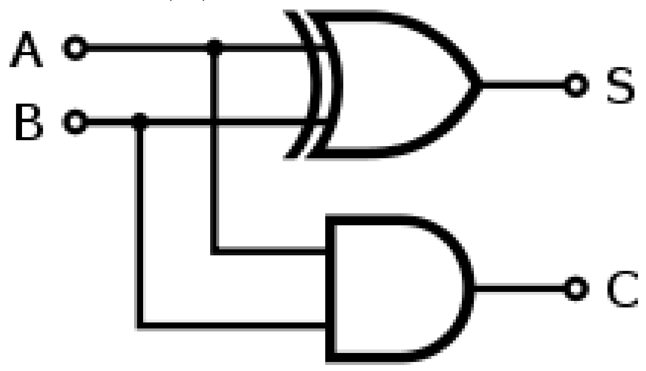
While PALs were busy developing into GALs and CPLDs (all discussed above), a separate stream of development was happening. This type of device is based on gate array technology and is called the eld-programmable gate array (FPGA). Early examples of FPGAs are the 82s100 array, and 82S105 sequencer, by Signetics, introduced in the late 1970s. The 82S100 was an array of AND terms. The 82S105 also had flip flop functions. FPGAs use a grid of logic gates,and once stored, the data doesn't change, similar to that of an ordinary gate array. The term "eld-programmable" means the device is programmed by the customer, not the manufacturer. FPGAs are usually programmed after being soldered down to the circuit board, in a manner similar to that of larger CPLDs. In most larger FPGAs the conguration is volatile, and must be re-loaded into the device whenever power is applied or dierent functionality is required. Conguration is typically stored in a conguration PROM or EEPROM. EEPROM versions may be in-system programmable (typically via JTAG). The dierence between FPGAs and CPLDs is that FPGAs are internally based on Look-up tables (LUTs) whereas CPLDs form the logic functions with sea-of-gates (e.g. sum of products). CPLDs are meant for simpler designs while FPGAs are meant for more complex designs. In general, CPLDs are a good choice for wide combinational logic applications, whereas FPGAs are more suitable for large state machines (i.e. microprocessors).

Steps of a typical FPGA based design

- Architectural Design
- Choice of Language(Vertilg,VHDL)
- Editing Programs
- Compiling Programs
- Synthesizing programs(.EDIF)
- Placing and routing programs (.VO, .SDF, .TTF)
- Loading programs to FPGA
- Debugging FPGA programs
- Documenting programs
- Delivering programs

An example of a simple digital design

Here we are going to discuss about the Half adder which is a combinational circuit. The half adder adds two single binary digits A and B. It has two outputs, sum (S) and carry (C). The half-adder adds two input bits and generate carry and sum which are the two outputs of half-adder.The input variables of a Half adder are called the Augend(A) and addend(B) bits.The output variables are the Sum(S) and Carry(C).



The Truth table for the Half adder is :

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Implementing of Proposed Hardware through Software
In this mini project we have used Quartus II : 9th edition to synthesize combinational circuits in an FPGA. The FPGA we used is of the Cyclone family (EP1C12Q240CB) . Quartus II is a tool which provides us with not only Schematic Block Diagram method of designing a circuit but also provides us with a platform for designing the same using a hardware accelerating language such as VHDL and VERILOG. This hardware accelerating software, Quartus is of great use as it helps generate functions, compile our designs, add clock pulses with various duty cycle , count and frequency.Further it allows us to have an idea of the shape of the curve generated. If correct pins of a FPGA interface board is assigned in this software by the user, we can get the required function generated in the LCD display of the interface board or even in an oscillator.

Schematic Design Entry Method with Quartus Graphical Simulation

Here we describe the basic steps in detail in order to design a combinational logic circuit in the Quartus 2 software. It will be easier to demonstrate as well as to understand these steps by taking an example. Note that these steps are totally basic and generalised. Any circuit can be designed following them. Just to make the steps more understandable we present these steps for developing and simulating a Half Adder using Schematic Design Entry Method.

Launch Quartus II software

1. Launch the Quartus II software.
2. Go to File - Project Wizard (on the File menu, click New Project Wizard)
3. New Project Wizard: Introduction windows appears. It shows you the five steps you may need to go through in setting up a new project. Click Next.
4. New Project Wizard: Directory, Name, Top-Level Entity window appears.

a. Enter the working project directory

b. Enter project name: HalfAdder. As you type HalfAdder name, Top-Level Entity name will be automatically filled with HalfAdder.

c. Click next. When asked to create the directory, click Yes.

5. New Project Wizard: Add Files window appears. You have no files to add now. Hence, just click Next.
6. New Project Wizard: Family and device settings window appears. Scroll up/down the family choices and choose Cyclone I family. Scroll up/down the available devices and choose the

FPGA number you are working with.Leave other options as they are. Click Finish.

Design the circuit

1. File New. New file option window appears.
2. Select Block Diagram/ Schematic File. Click OK. BDF Editor window appears.
3. Double click anywhere on the window.Select Symbol window appears.
4. Type xor in the Name box.Click Ok.Drag the XOR symbol and put it in the middle of the window.
5. Repeat steps 3 and 4 but with and2 symbol name to put a 2-input AND symbol at the bottom of the XOR symbol.
6. Input ports and output ports are symbols with names ”input” and ”output”, respectively.Thus,repeat steps 3 and 4 with symbol name ”input” and ”output” to put two input ports and two output ports on the window. Click and drag the symbols.
7. Click the orthogonal node tool in the list of tool on the left of the BDF window .To connect point A to point B, position the cursor on point A, click and hold the left button of the mouse, drag the line and drop it at point B.

   Example: To connect input symbol to the XOR symbol input, position the cursor on the right end point of the input symbol, click and hold the left mouse button, drag the line and drop it on the input of the XOR symbol. Follow the steps above connect the input symbols and the output symbols to the XOR and AND symbols.
8. Click the cursor in the list of tool on the left of the BDF window . Double click the input symbol and change the input name to

X. Double click the other input symbol and change the input name to Y. Repeat the steps to name the output names to SUM and CARRY, respectively.

9. Double click anywhere on the window. Select Symbol window appears.
10. Type title in the Name box. Click Ok. Drag the title symbol and put it the lower right- hand part of the window
11. Double click the title symbol, the title block properties window appears. Click any field that you want to modify and change the value.
12. Save the bdf file as HalfAdder.bdf, the same name you used as the project name with bdf extension.

Compile the half adder

Go to Processing Start Compilation. If there is errors in the message window, fix the errors and compile the design again. Ignore the warnings.

Simulate the half adder (Timing Simulation)

1. File - New.New file option window appears.
2. Click Other Files tab and select Vector Waveform File. Click Ok. Waveform Vector File (wvf) windows appears.
3. Left click anywhere in the Name pane (left of vwf window) and click Insert Insert Node or Bus. Insert Node or Bus window appears.
4. Click Node Finder. Node Finder window appears.
5. Select Pins: all in the Filter selection box and click List. Move all signals from the left pane (Nodes Found) to the right pane (Nodes Selected) by clicking . Click OK. On the Insert Node or Bus window click OK again. The vwf window appears.

6. Arrange the signals in the Name pane in logical order. This arrangement allows you to read the output easier.

7. Right click on Y and under Value Count Value. Count Value window appears. Click Timing tab and change Count Every value to 20 ns. Click OK.

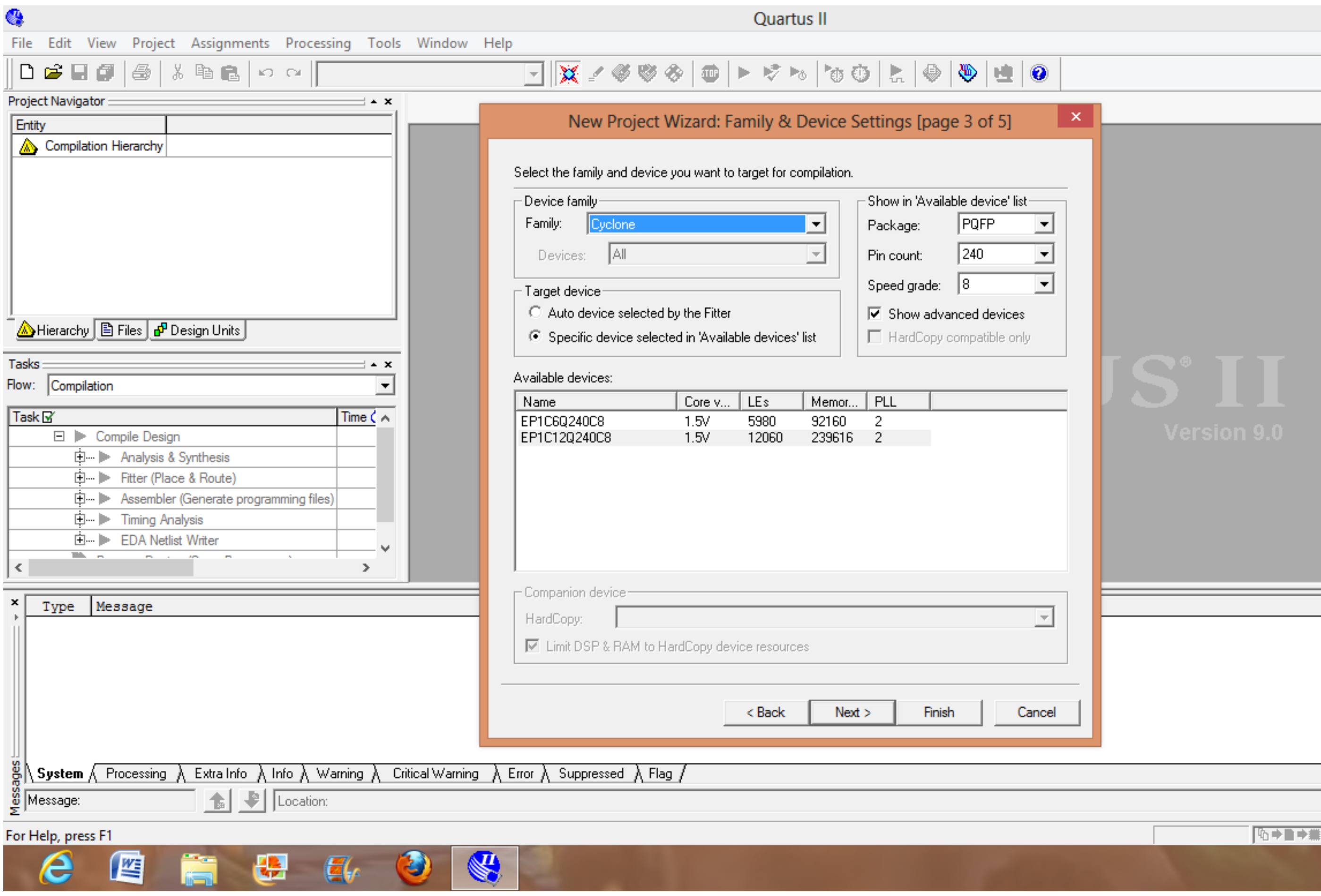
8. Right click on X and under Value Count Value. Count Value window appears. Click Timing tab and change the Multiplied by value to 2. Click OK. If you have another input to set, say in a design that has four inputs, the multiplied by values must be set 2 times higher that the lesser significant input.

9. View Zoom Out as many times as necessary so that you could see all possible combinations of X and Y up to 80 ns.

10. Save your HalfAdder.vwf file.

11. First, you will do the timing simulation where you can see the signal delay from inputs to outputs (Simulation mode: Timing).Go to Processing Simulator Tool. Simulator Tool window appears. Check the Overwrite simulation input file with simulation results box. Click Start. If you do not see the Start button, close the message window at the bottom.

12. If the simulation is successful, click Open to open the vwf file and see the result of simulation.



New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family
Family: Cyclone
Devices: All

Show in 'Available device' list
Package: PQFP
Pin count: 240
Speed grade: 8
[✓] Show advanced devices
[ ] HardCopy compatible only

Target device
( ) Auto device selected by the Fitter
(●) Specific device selected in 'Available devices' list

Available devices:

Name	Core v...	LEs	Memor...	PLL
EP1C6Q240C8	1.5V	5980	92160	2
EP1C12Q240C8	1.5V	12060	239616	2

Companion device
HardCopy:
[✓] Limit DSP & RAM to HardCopy device resources

Figure 4: This is how we choose the family and the model no. of the FPGA we are working with

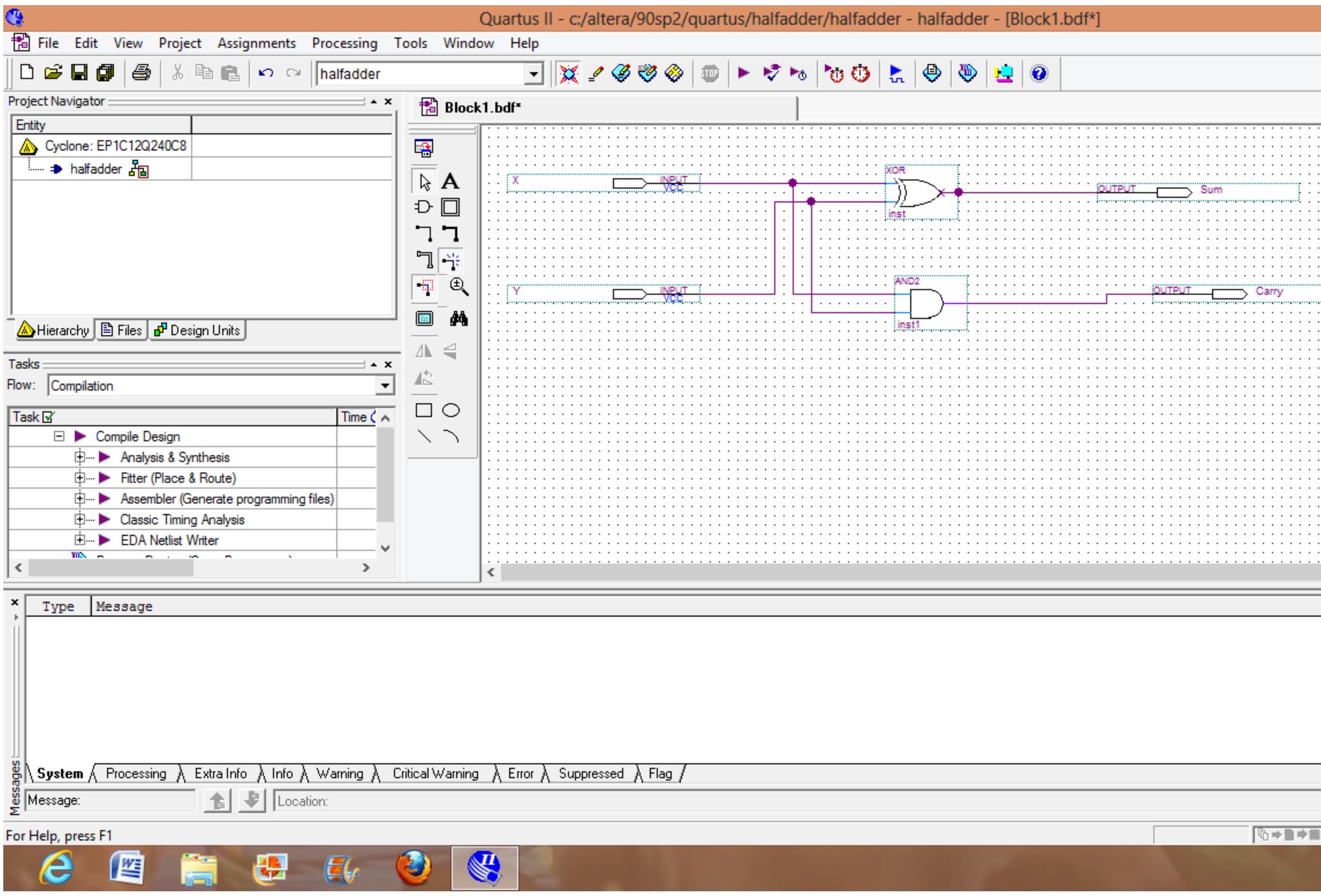
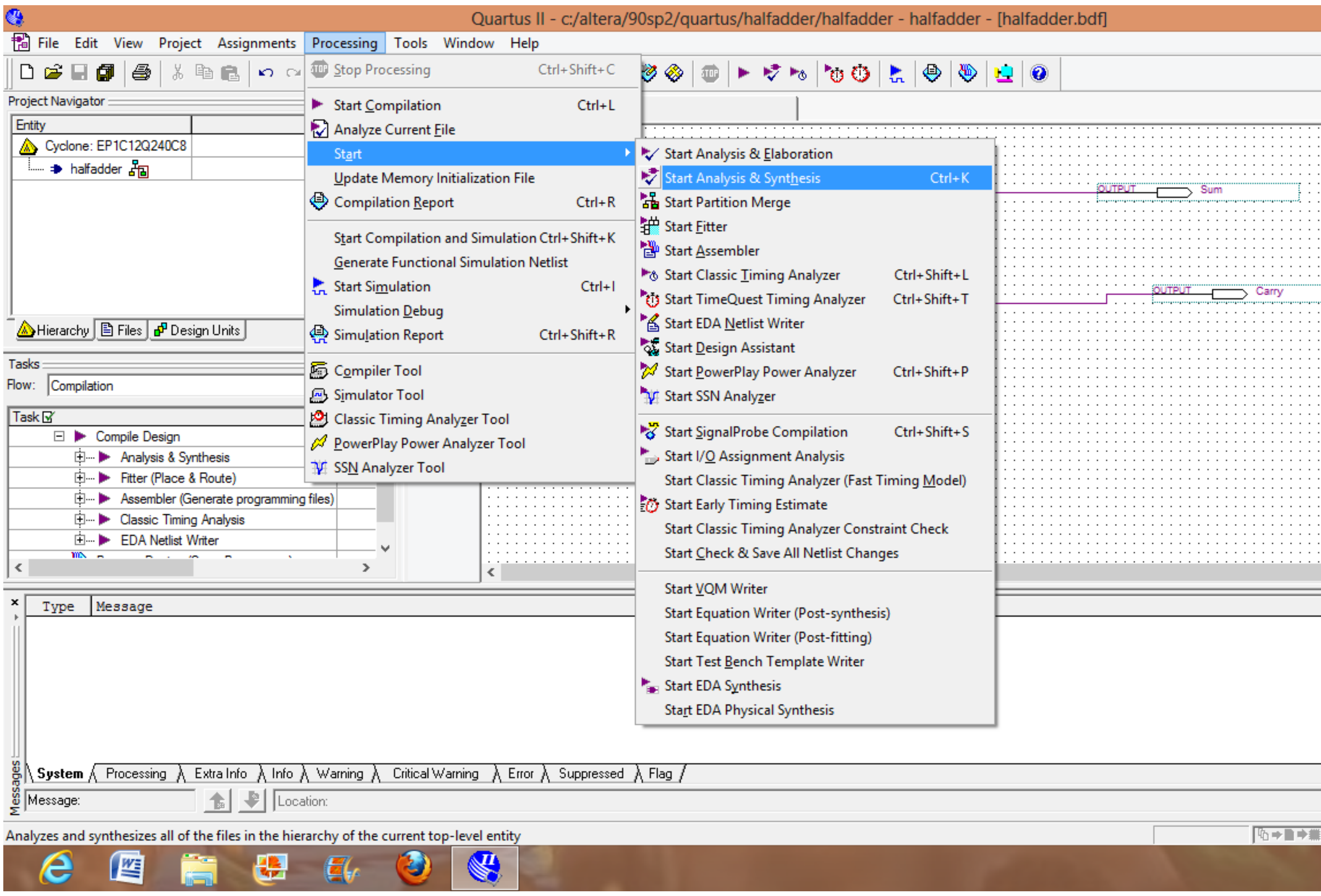
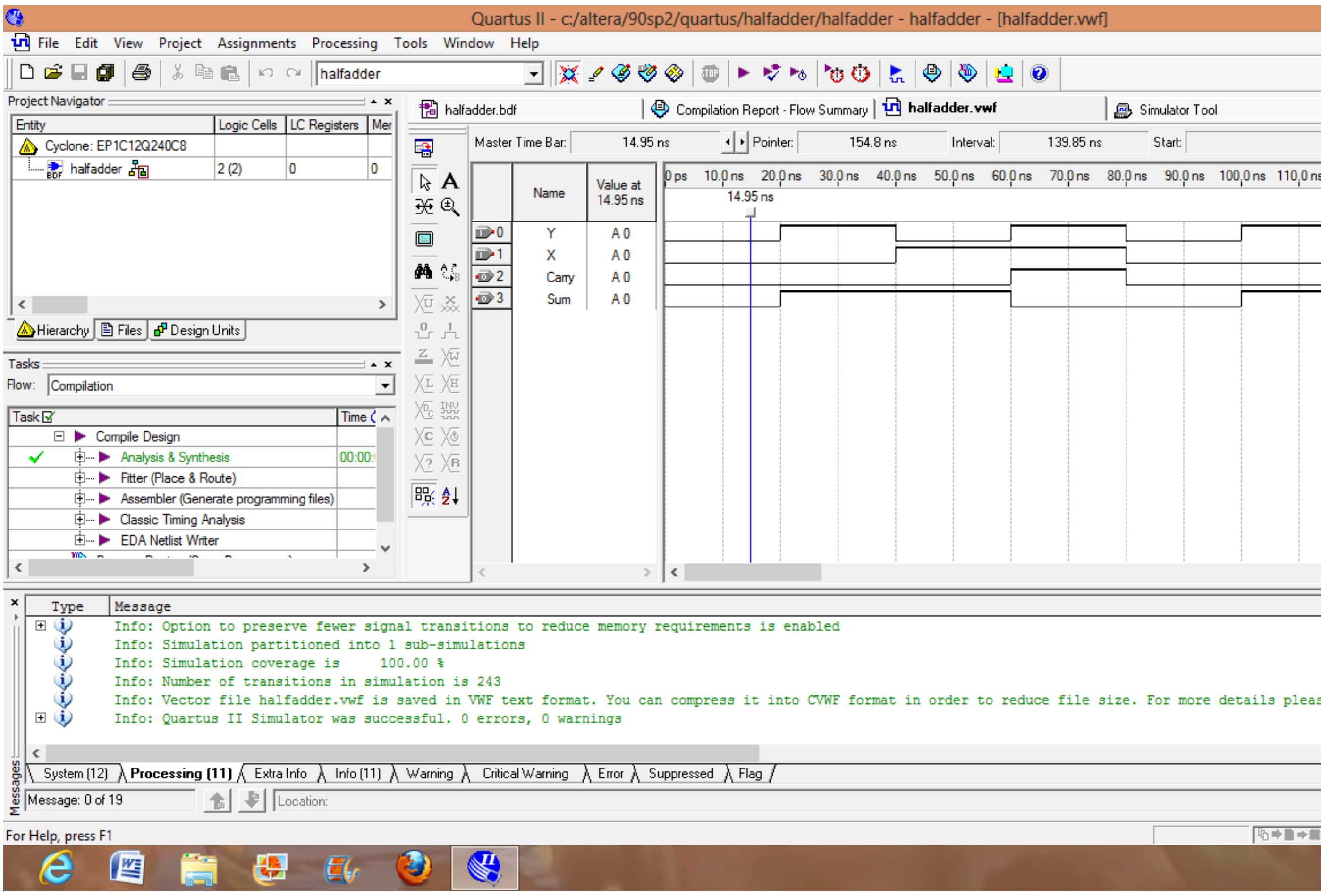


Figure 5: This is how we bulid the block diagram/schematic file.


Quartus II - c:/altera/90sp2/quartus/halfadder/halfadder - halfadder - [halfadder.bdf]
File Edit View Project Assignments Processing Tools Window Help
Stop Processing Ctrl+Shift+C
Start Compilation Ctrl+L
Analyze Current File
Start
Update Memory Initialization File
Compilation Report Ctrl+R
Start Compilation and Simulation Ctrl+Shift+K
Generate Functional Simulation Netlist
Start Simulation Ctrl+I
Simulation Debug
Simulation Report Ctrl+Shift+R
Compiler Tool
Simulator Tool
Classic Timing Analyzer Tool
PowerPlay Power Analyzer Tool
SSN Analyzer Tool
Start Analysis & Elaboration
Start Analysis & Synthesis Ctrl+K
Start Partition Merge
Start Fitter
Start Assembler
Start Classic Timing Analyzer Ctrl+Shift+L
Start TimeQuest Timing Analyzer Ctrl+Shift+T
Start EDA Netlist Writer
Start Design Assistant
Start PowerPlay Power Analyzer Ctrl+Shift+P
Start SSN Analyzer
Start SignalProbe Compilation Ctrl+Shift+S
Start I/O Assignment Analysis
Start Classic Timing Analyzer (Fast Timing Model)
Start Early Timing Estimate
Start Classic Timing Analyzer Constraint Check
Start Check & Save All Netlist Changes
Start VQM Writer
Start Equation Writer (Post-synthesis)
Start Equation Writer (Post-fitting)
Start Test Bench Template Writer
Start EDA Synthesis
Start EDA Physical Synthesis
Project Navigator
Entity
Cyclone: EP1C12Q240C8
halfadder
Hierarchy Files Design Units
Tasks
Flow: Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programming files)
Classic Timing Analysis
EDA Netlist Writer
OUTPUT Sum
OUTPUT Carry
Type Message
System Processing Extra Info Info Warning Critical Warning Error Suppressed Flag
Messages
Message: Location:

Analyzes and synthesizes all of the files in the hierarchy of the current top-level entity

Figure 6: This is how to process it after the block diagram/Schematic file is drawn.



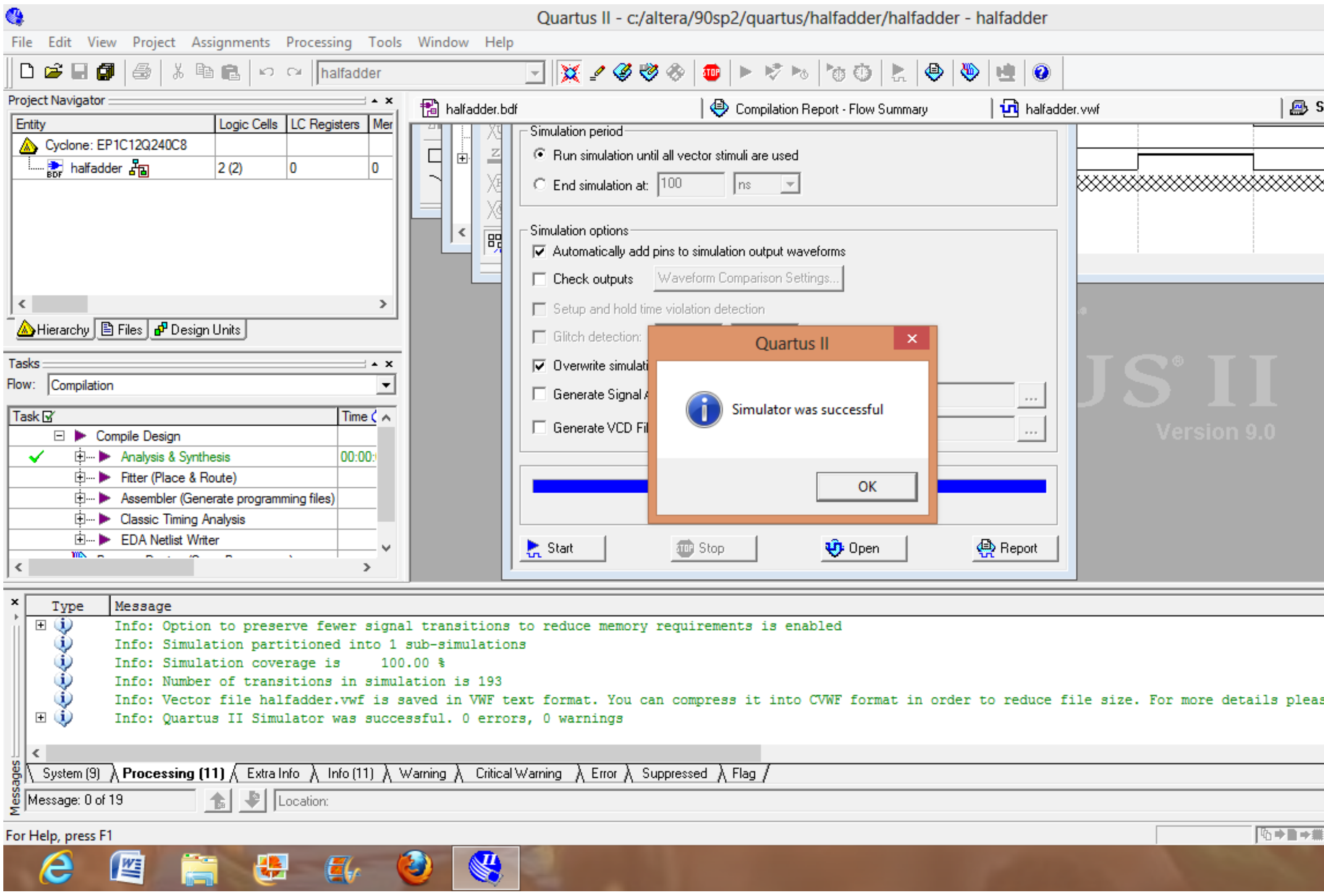
Entity	Logic Cells	LC Registers	Mer
Cyclone: EP1C12Q240C8			
halfadder	2 (2)	0	0

Task	Time
Compile Design	
✓ Analysis & Synthesis	00:00:
Fitter (Place & Route)	
Assembler (Generate programming files)	
Classic Timing Analysis	
EDA Netlist Writer	

	Name	Value at 14.95 ns
0	Y	A 0
1	X	A 0
2	Carry	A 0
3	Sum	A 0

```
Type Message
Info: Option to preserve fewer signal transitions to reduce memory requirements is enabled
Info: Simulation partitioned into 1 sub-simulations
Info: Simulation coverage is     100.00 %
Info: Number of transitions in simulation is 243
Info: Vector file halfadder.vwf is saved in VWF text format. You can compress it into CVWF format in order to reduce file size. For more details pleas
Info: Quartus II Simulator was successful. 0 errors, 0 warnings
```

Figure 7: After we have done our work with HalfAdder.vwf file ,its time for simulation.


Quartus II - c:/altera/90sp2/quartus/halfadder/halfadder - halfadder
File Edit View Project Assignments Processing Tools Window Help
halfadder
Project Navigator
Hierarchy
Files
Design Units
Tasks
halfadder.bdf
Compilation Report - Flow Summary
halfadder.vwf
Start
Stop
Open
Report
Version 9.0
OK

Entity	Logic Cells	LC Registers	Mer
Cyclone: EP1C12Q240C8			
halfadder	2 (2)	0	0

Flow: Compilation

Task	Time
Compile Design	
✓ Analysis & Synthesis	00:00:
Fitter (Place & Route)	
Assembler (Generate programming files)	
Classic Timing Analysis	
EDA Netlist Writer	

Simulation period
[✓ Run simulation until all vector stimuli are used]
End simulation at: 100 ns

Simulation options
[✓] Automatically add pins to simulation output waveforms
[ ] Check outputs
[ ] Setup and hold time violation detection
[ ] Glitch detection:
[✓] Overwrite simulati
[ ] Generate Signal A
[ ] Generate VCD Fil

Quartus II
Simulator was successful

```
Type   Message
       Info: Option to preserve fewer signal transitions to reduce memory requirements is enabled
       Info: Simulation partitioned into 1 sub-simulations
       Info: Simulation coverage is      100.00 %
       Info: Number of transitions in simulation is 193
       Info: Vector file halfadder.vwf is saved in VWF text format. You can compress it into CVWF format in order to reduce file size. For more details pleas
       Info: Quartus II Simulator was successful. 0 errors, 0 warnings
```

System (9) | **Processing (11)** | Extra Info | Info (11) | Warning | Critical Warning | Error | Suppressed | Flag

Message: 0 of 19   Location:

For Help, press F1

Figure 8: Compilation is successful

Project Related Minor Work

Getting Familiar with VHDL

VHDL is a language for describing electronic systems. It arouse out of the United States Government's Very High Speed Integrated Circuits (VHSIC) program, initiated in 1980. In the course of this program, it became clear that there was a need for a standard language for describing the structure and function of integrated circuits (ICs). Hence the VHSIC Hardware Description Language (VHDL) was developed, and subsequently adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE) in the US. There are some important differences between VHDL and a conventional procedural language. VHDL is inherently parallel, i.e ; commands, which correspond to logic gates, are executed in parallel, as soon as a new input arrives.

Basic structure of a VHDL file

A digital system in VHDL consists of a design entity that can contain other entities that are then considered components of the top level entity. Each entity is modelled by an entity declaration and an architecture body. Each entity can be considered as the interface to the outer world that defines the input and output signals. The architecture body contains the description of the entity and is composed of interconnected entities, processes and components, all operating concurrently.

Entity Declaration

The entity declaration defines the NAME of the entity and lists the input and output ports. The general form is as follows:- `entity NAME_OF_ENTITY is [ generic generic_declarations);]`

```
 port (signal_names :  mode type;
    signal_names :  mode type
     :
       signal_names:  mode type)
         end[name_of_entity];
```

An entity always starts with the keyword entity, followed by its name and the keyword is. Next are the port declarations using the keyword port. An entity declaration always ends with the keyword end, optionally [] followed by the name of the entity.

- `The NAME_OF_ENTITY is a user-selected identifier.`
- `signal_names consists of a comma separated list of one or more user-selected identifiers that specify external interface signals.`
- mode: is one of the reserved words to indicate the signal direction :

a). In:-Indicates that the signal is an input

b). out:-indicates that the signal is an output of the entity whose value can only be read by other entities that use it.

c). buffer:–Indicates that the signal is an output of the entity whose value can be read inside the entity's architecture.

d). inout:–The signal can be an input or an output.

- type:- a built-in or user-defined signal type. Examples of types are bit, bit_vector, Boolean, character, std_logic, and std_ulogic.

a). Bit:-can have the value 0 and 1

b). Bit_vector:- is a vector of bit values (e.g. bit_vector (0 to 7)

c). std_logic, std_ulogic, std_logic_vector, std_ulogic_vector: can have 9 values to indicate the value and strength of a signal. Std_ulogic and std_logic are preferred over the bit or bit_vector types

d). boolean:– can have the value TRUE and FALSE

e). integer:– can have a range of integer values

f). real:– can have a range of real values

g). character:– any printing character

h). time:– to indicate time

Generic

generic declarations are optional and determine the local constants used for timing and sizing (e.g. bus widths) the entity. A generic can have a default value. The syntax for a generic follows:-
generic (

```
constant_name:  type [:=value] ;
constant_name:  type [:=value] ;
:
constant_name:  type [:=value] );
```

Four-to-one multiplexer of which each input is an 8-bit word.

```
entity mux4_to_1 is
  port (I0,I1,I2,I3: in std_logic_vector(7 downto 0);
    SEL: in std_logic_vector (1 downto 0);
      OUT1: out std_logic_vector(7 downto 0));
        end mux4_to_1;
```

An example of the entity declaration of a D flip-flop with set and reset inputs is:

```
entity dff_sr is
   port (D,CLK,S,R: in std_logic;
      Q,Qnot: out std_logic);
         end dff_sr;
```

Architecture Body

The architecture body specifies how the circuit operates and how it is implemented. As discussed earlier, an entity or circuit can be specified in a variety of ways, such as behavioral, structural (interconnected components), or a combination of the above.

The architecture body looks as follows:-

```
architecture architecture_ name of NAME_OF ENTITY is
   ——declaration
      ——components declarations
      ——signal declarations
      ——constant declarations
      ——function declarations
      ——procedure declarations
      ——type declarations
Begin
——statements




      :
end architecture_name;
```

The architecture body described at the behavioral level, is given below:-

```
architecture behavioral of BUZZER is
begin
WARNING <= ( not DOOR and IGNITION) or (not SBELT and IGNITION);
end behavioral;
```

The behavioral description of a two-input AND gate:

```
entity AND2 is
   port (in1, in2:in std_logic;
      out1: out std_logic);
         end AND2;
         architecture  behavioral_2 of AND2 is
         begin
            out1 <= in1 and in2;
               end behavioral_2;
```

Circuits can also be described using a structural model that specifies what gates are used and how they are interconnected. The following example illustrates it.

```
architecture structuralof BUZZER is
   —— Declarations
   component AND2
      port in1, in2: in std_logic;
         out1:out std_logic);
   end component;
   componentOR2
      port (in1, in2: in  std_logic;
         out1: out  std_logic);
   end component;
    component NOT1
      port (in1: in std_logic; out1: out std_logic);
end component;


——Declaration of signals used to interconnect gates


         signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;


begin
```

——Component instantiations statements

```
signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;
   begin
      —— Component instantiations statements
      U0: NOT1 port map (DOOR, DOOR_NOT);
      U1: NOT1 port map (SBELT, SBELT_NOT);
      U2: AND2 port map (IGNITION, DOOR_NOT, B1);
      U3: AND2 port map (IGNITION, SBELT_NOT, B2);
      U4: OR2 port map  (B1, B2, WARNING);
end structural;
```

Learning of a PCB Designing software

We have just started our work with PCB designing softwares.We hope that in our next semester, we will continue our work with it. We have used two softwares for our elementary work with PCBs.

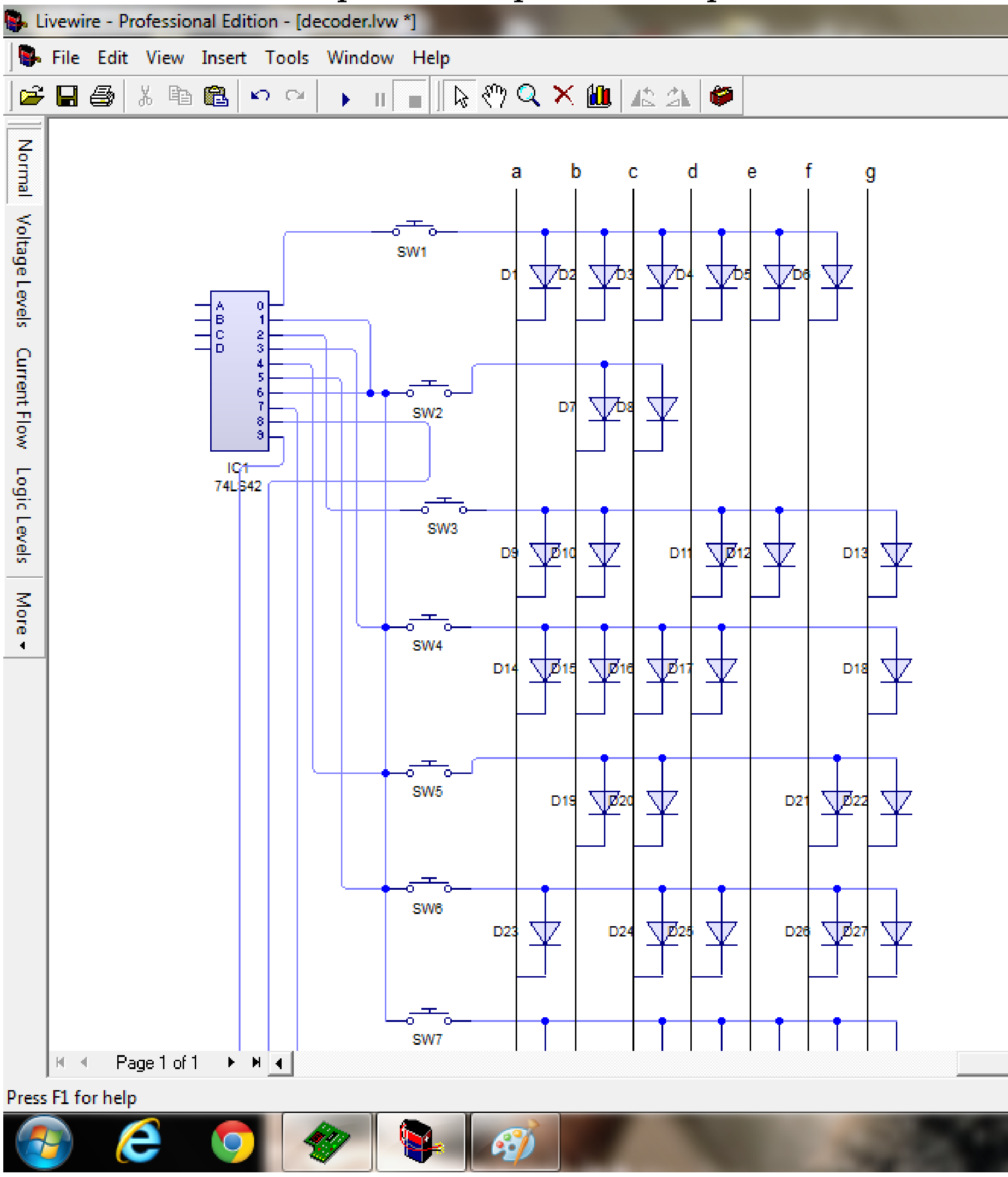
- Liveware:- A sophisticated software package for designing and simulating electronic circuits. Switches, transistors, diodes, integrated circuits and hundreds of other components can all be connected together to investigate the behaviour of a circuit. There are no limits to what can be designed and no loose connections or faulty components to worry about. However, if the maximum ratings for any components are exceeded, they will explode on screen!

- PCB Wizard:- a powerful package for designing single-sided and double-sided printed circuit boards (PCBs). It provides a comprehensive range of tools covering all the traditional steps in PCB production, including schematic drawing, schematic capture, component placement, automatic routing, Bill of Materials reporting and le generation for manufacturing. In addition, PCB Wizard oers a wealth of clever new features that do away with the steep learning curve normally associated with PCB packages. What we do that we design our circuit in liveware. Then we simulate the circuit. Then by using PCB wizard, we convert the designed circuit to a PCB layout. We have just started our work with PCB designing softwares.So far we have designed some basic circuits using 555 timer ic and a decoder matrix ROM.

Steps for designing a PCB using those two softwares

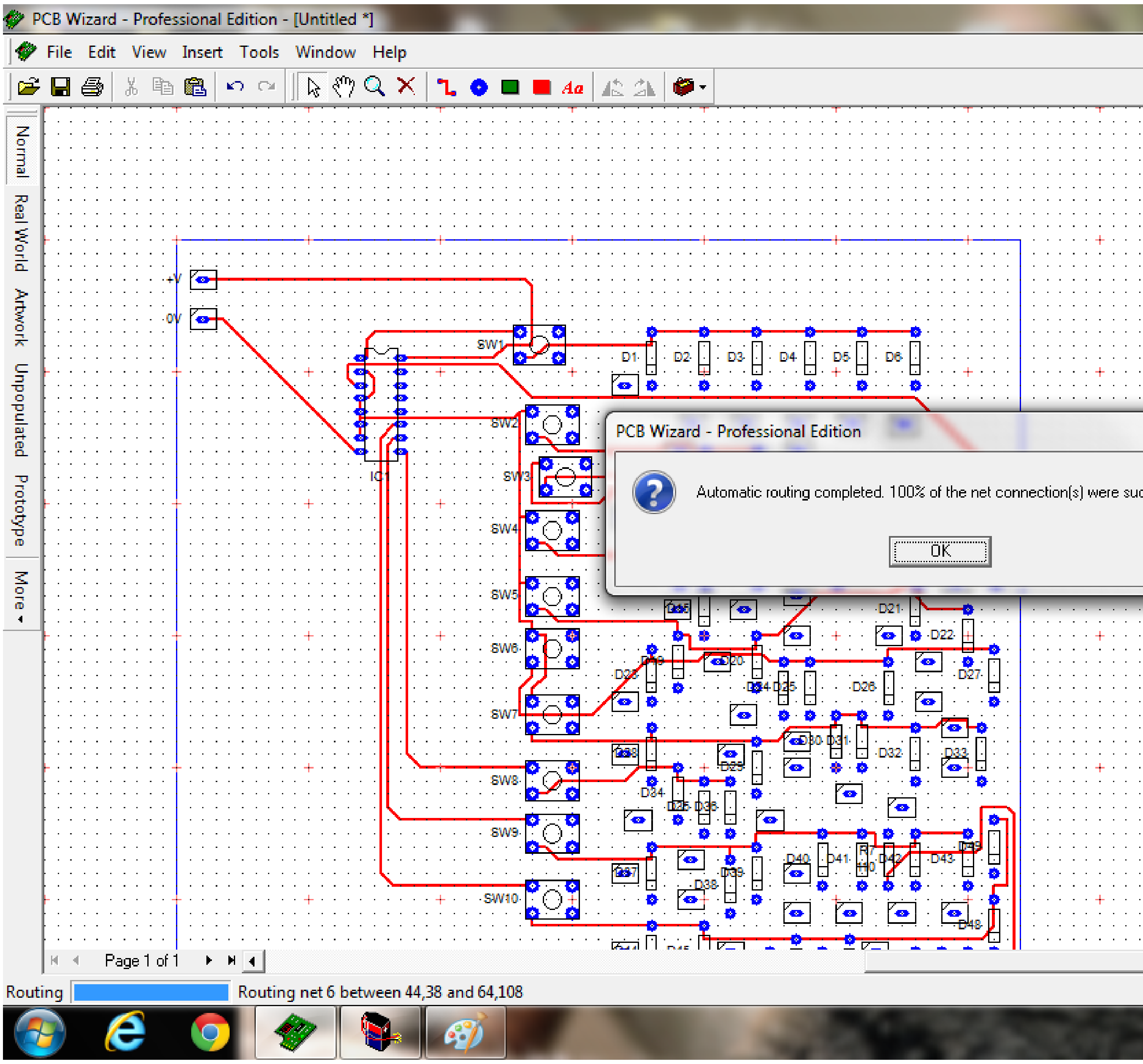
1. **Open the liveware software. The components to draw the desired circuit is available in the liveware. Draw the circuit diagram by choosing the right ic , resistor properties , diode properties etc.**
2. **After you have drawn the circuit, simulate it using the "run" button or by pressing F9.**
3. **Before converting it into PCB, make sure that the PCB wizard window is open.**
4. **Then choose tool - Convert - Design to Printed Circuit Board from Liveware.**
5. **After the "Convert to Printed Circuit Board" is open, choose the option "No, let liveware specify those options for me".**
6. **Then you will have to design the respective PCB in the PCB wizard window.**

Next,we designed a Decoder Matrix ROM using Liveware and PCB Wizard. we would like to provide a pictorial rep-



resentation of it.

Figure 9 :The circuit diagram of the decoder matrix ROM we have drawn with the help of liveware.


PCB Wizard - Professional Edition - [Untitled *]
File Edit View Insert Tools Window Help
Normal
Real World
Artwork
Unpopulated
Prototype
More
+V
0V
IC1
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8
SW9
SW10
D1
D2
D3
D4
D5
D6
PCB Wizard - Professional Edition
OK
Page 1 of 1

Automatic routing completed. 100% of the net connection(s) were suc

Routing Routing net 6 between 44,38 and 64,108

Figure 10 :Creating the PCB with respect to the circuit diagram you have created in Liveware.

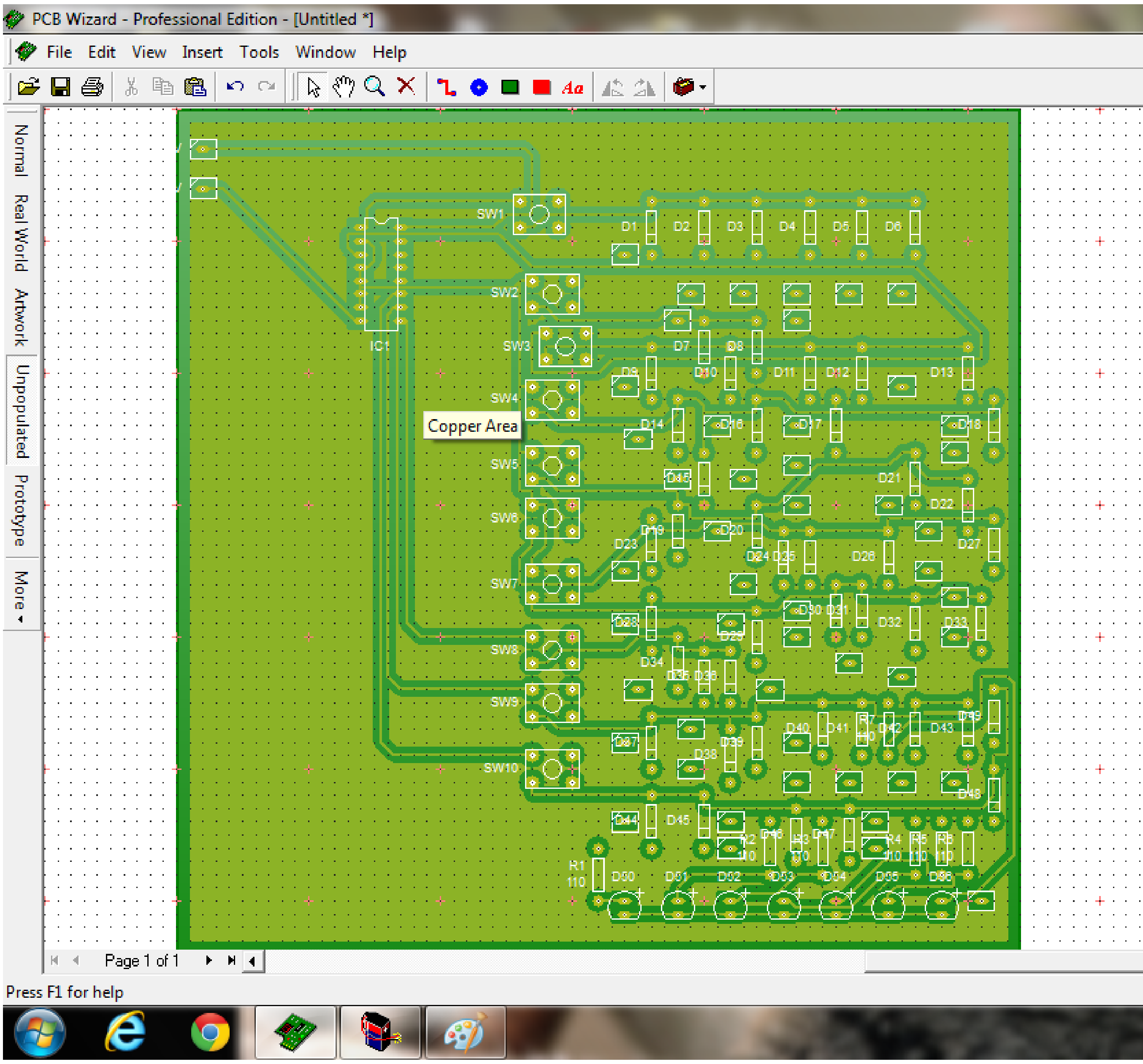


Figure 11 :The Final design of the PCB.

Along with the technical work we have also planned to learn the modern documentation/presentation methodologies which would be helpful in pursuing our future professional or academic career.
So the office suit (word, PPT, Excel), excluding the database package Access, is explored and used. Moreover. Latex is extensively learnt —- e.g., the project report is in latex as well as the presentation (through Beamer).

Future Scope Of Work

After studying the Quartus 2 software and working with FPGA , we are confident of being able to design more complex circuits and other required applications. We are now capable of studying these devices in further depths and actually using or implementing these using practical applications. The future scope of work is immense. Our goal is to study the methods of designing a PCB using a PCB drawing software and build a board similar to that of an FPGA containing interface that we used in this mini project.

References

1. **VHDL primer by Jan Van der Spiegel. University of Pennsylvania**
2. **VHDL cookbook by Peter J Ashenden**
3. **Latex- A document preparation system, User's guide and reference manual-Leslie Lamport**
4. **www.google.co.in**
5. **www.wikipedia.org**
6. **www.wikibooks.org**

Our Special Thanks to...................

- **Professor Amit Kumar Das , Department of Computer Science and Technology**

- **Professor K. Mukherjee, Department of Electrical Engineering**

- **Debrajda, P.G student, Department of Electrical Engineering**