Abu Dhabi University

CEN - 466 Advanced Digital Design

Lab Report 1

Introduction to VHDL: gates using VHDL, components and port map

Author:
Muhammad Obaidullah 1030313
Mohammed Farooq 1007778
Mehdi Ismail 1005689

Supervisor:
Dr. Mohammed Assad Ghazal

Section 1

December 8, 2012

Abstract

We used VHDL ¡hardware description language¿ to work on the CycloneII board of Altera. Our course of ¡advanced digital system design¿ aimed at getting us to apply the logic gates and other equipments and techniques learnt of digital systems applied via VHDL coding. This lab will show us through each of the features and language basics to implement logic gates and see their output on the boards.

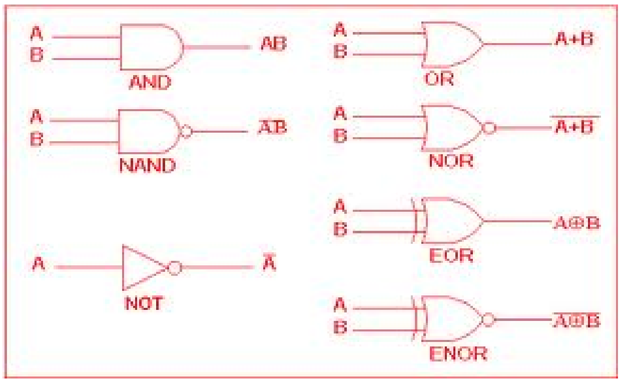
1 Introduction

VHDL that stands for <VHSIC (very-high-speed-integrated-circuit) Hardware Description Language>, a language developed initially by US department of defense to to have a standard in equipments documentation. The language is similar to the very first language of the kind, Verilog, although verilog is case sensitive and weakly typed language, VHDL is not case sensitive but is a strongly typed language and also influenced verilog to adopt open standard after noticing the success of VHDL. We learnt and used VHDL on QuartusII, software of altera for its boards that the lab of university were equiped with. These boards belonged to the CycloneII family of FPGA boards, more specifically the EP2C35F672C6, which are named as EP2C35 identifies the family, 672 after that is the pins on it. FPGA's are Field programmable gate array, meaning they can be programmably designed to create circuits that we previously bought specific chips and implemented.

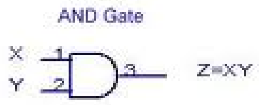
Within the course objectives was to familiarize ourselves with digital logics application into the boards, so as to be able to create any simulation of hardware we might need within a short time frame. To that respect we implemented codes of each of the gates (NOT, AND, OR, XOR, NAND, NOR, XNOR) simulation on the same board simultaneously.

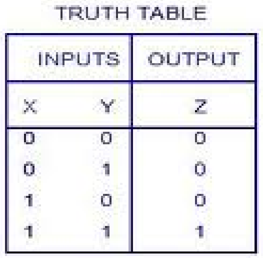
2 Experiment Set-up

The Experiment was set up by opening and setting up the VHDL coding Integrated Development Environment Quartus II. The code was written in the Quartus IDE and then we debugged it. After debugging, We assigned the outputs to the pins on the ALTERA board. These outputs can by anything ranging from LEDs to Buzzers. Additionally, there were some inputs to be assigned to or how else could we get the input from the user. These all pins were assigned by referring to the Datasheet of the ALTERA Cyclone II and were assigned by the Pin Planner inside the Quartus.



2 Input AND Gate

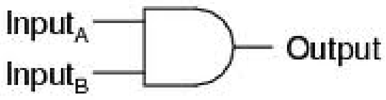


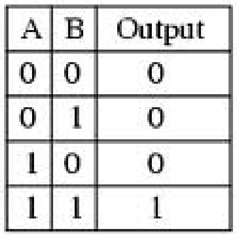


TRUTH TABLE

INPUTS		OUTPUT
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

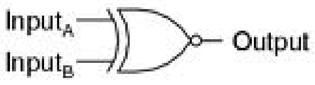
2-input AND gate

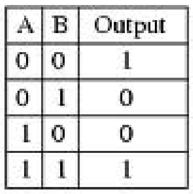




A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

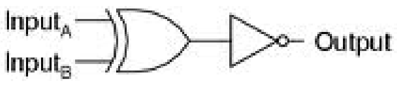
Exclusive-NOR gate



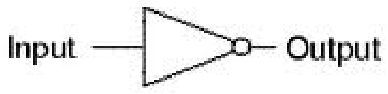


A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Equivalent gate circuit

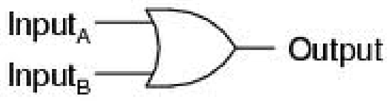


INVERTER



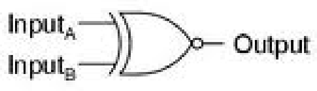
Input	Output
1	0
0	1

2-input OR gate



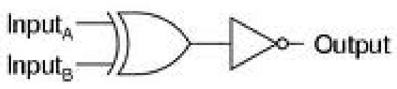
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive-NOR gate

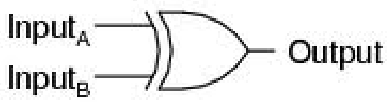


A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

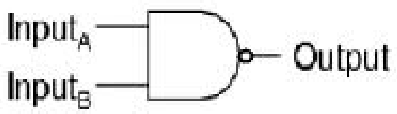
Equivalent gate circuit



Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



Logic Circuit OfNAND Gate

3 List of Equipment used

Equipments and materials used during the experiment include

- Computers with Quartus Software.
- CycloneII, Altera Boards
- Power Cable and USB cable to the board

4 Procedure

As the first and introductory lab work on the VHDL of the board, we have detailed the process of running the VHDL code and implementing the design. End of this lab report also has the datasheet that the board pins were implemented on.

- Open Quartus.
- Click on Create a new project.
- Click next.
- Create a folder on a directory of your choice and select it as a working directory for you project.
- Name your project and click next.
- Now choose the board to be “EP2C35F672C6”.
- Click next and finish.
- Now go to File ¿ New ¿ VHDL File.
- Start writing the code.

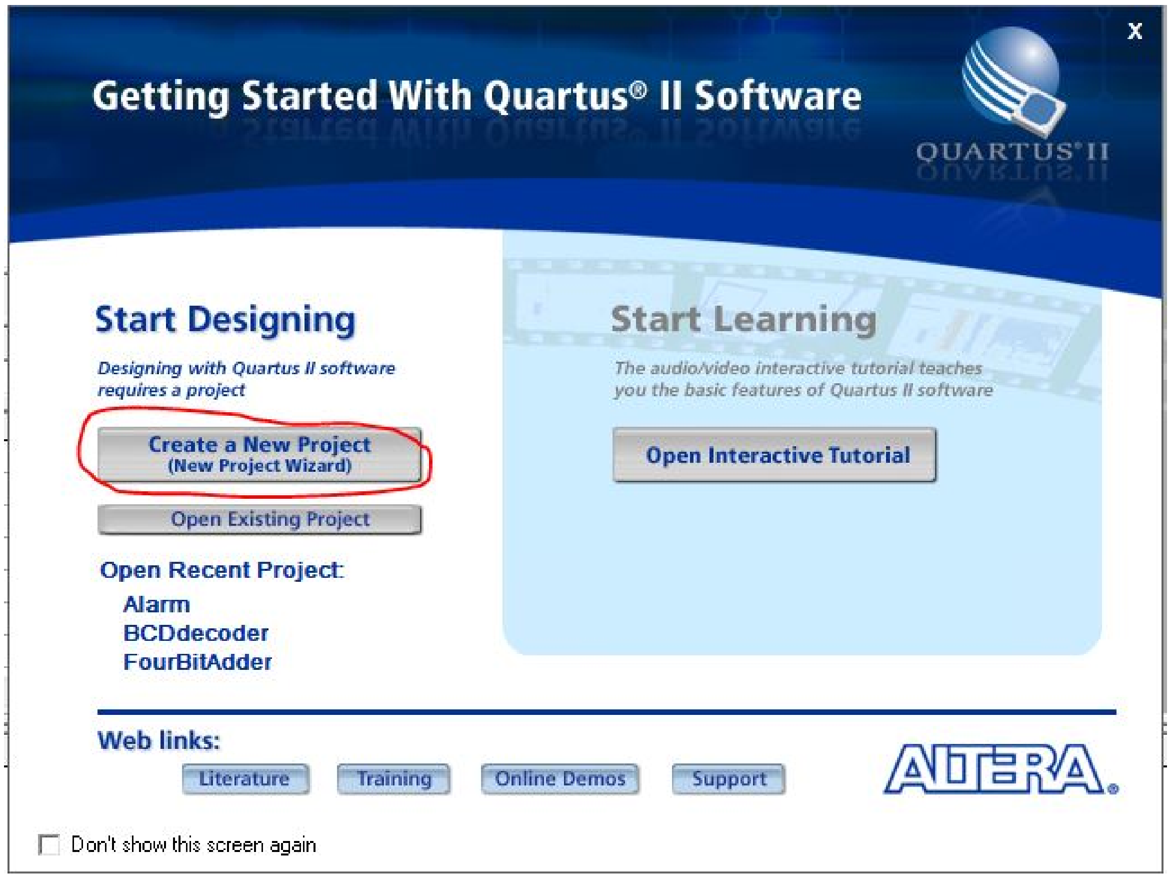
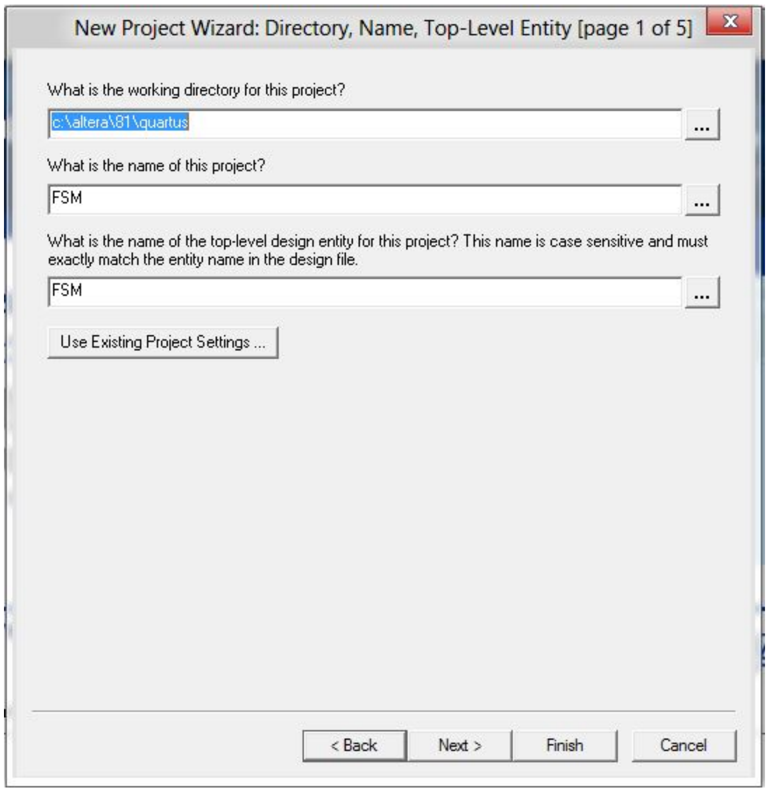

Getting Started With Quartus® II Software
QUARTUS®II
Start Designing
Designing with Quartus II software requires a project
Create a New Project (New Project Wizard)
Open Existing Project
Open Recent Project:
Alarm
BCDdecoder
FourBitAdder
Start Learning
The audio/video interactive tutorial teaches you the basic features of Quartus II software
Open Interactive Tutorial
Web links:
Literature
Training
Online Demos
Support
ALTERA
Don't show this screen again

Figure 1: Click on “Create a New Project”.


New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

c:\altera\81\quartus

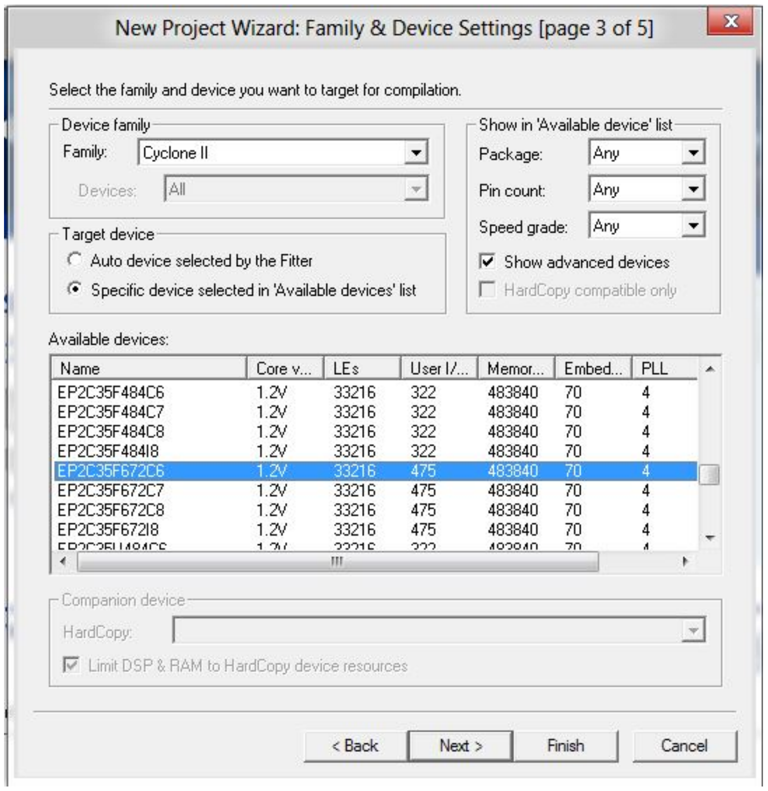
What is the name of this project?

FSM

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

FSM

Figure 2: Choose the directory and name it without spaces and special characters.


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 II

Devices: All

Target device

Auto device selected by the Fitter [✓ Specific device selected in 'Available devices' list]

Show in 'Available device' list

Package: Any

Pin count: Any

Speed grade: Any

✓ Show advanced devices

HardCopy compatible only

Available devices:

Name	Core v...	LEs	User I/...	Memor...	Embed...	PLL
EP2C35F484C6	1.2V	33216	322	483840	70	4
EP2C35F484C7	1.2V	33216	322	483840	70	4
EP2C35F484C8	1.2V	33216	322	483840	70	4
EP2C35F484I8	1.2V	33216	322	483840	70	4
EP2C35F672C6	1.2V	33216	475	483840	70	4
EP2C35F672C7	1.2V	33216	475	483840	70	4
EP2C35F672C8	1.2V	33216	475	483840	70	4
EP2C35F672I8	1.2V	33216	475	483840	70	4
EP2C35U484C6	1.2V	33216	322	483840	70	4

Companion device

HardCopy:

✓ Limit DSP & RAM to HardCopy device resources

Figure 3: Choose the highlighted board and the Family should be “Cyclone II”.

5 VHDL Code for AND Gate

```
--------------------------------- -- Lines
-- VHDL code for AND gate       -- starting with    --
--------------------------------- -- are comments

library ieee;          -- Include all the libraries
use ieee.std_logic_1164.all; -- & other built-in namespaces
                 -- that are typically used in the code
--------------------------------- -- Implementation

entity AND_ent is               -- Entity is used to declare I/O used in the code
port( x: in std_logic;    -- port x is declared as input which accepts when
    y: in std_logic;   -- a logic input/Output value. i.e. 0 or 1
    F: out std_logic      -- F is declared as Output
);
end AND_ent;

-----------------------------------------------------------

architecture AND_arch1 of AND_ent is
                 -- architecture is where actual description lies.
                 -- Two approaches can be used to describe
                 -- how our code should be working.
begin

    process(x, y)       -- process states that perform the below set of code
                 -- when either of x or y changes
    begin
        -- compare to truth table
        if ((x='1') and (y='1')) then
                 -- if x and y are equal to 1
      F <= '1';       -- Assign 1 to F
  else
      F <= '0';       -- Assign 0 to F
  end if;
    end process;

end AND_arch1;          -- end of Architecture

architecture AND_arch2 of AND_ent is
begin

    F <= x and y;       -- and is a reserved keyword which perform AND

end AND_arch2;

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

6 VHDL Code for NAND Gate

```
------------------------------------ -- Lines
-- VHDL code for NAND gate      -- starting with    --
------------------------------------ -- are comments
```

```
                    -- Include all the libraries
use ieee.std_logic_1164.all;    -- & other built-in namespaces
                    -- that are typically used in the code
------------------------------- -- Implementation

entity NAND_ent is
port( x: in std_logic;    -- Entity is used to declare I/O used in the code
    y: in std_logic;    -- port x is declared as input which accepts when
    F: out std_logic    -- F is declared as Output
);
end NAND_ent;
-------------------------------------------

architecture NAND_arch1 of NAND_ent is
                  -- architecture is where actual description lies.
                  -- Two approaches can be used to describe
                  -- how our code should be working.
begin

    process(x, y)       -- process states that perform the below set of code
                  -- when either of x or y changes
    begin
        -- compare to truth table
        if (x='1' and y='1') then -- here the 'and' compares X and Y
      F <= '0';          -- Assign 0 to F
  else
      F <= '1';          -- Assign 1
  end if;
    end process;

end NAND_arch1;
-------------------------------------------

architecture NAND_arch2 of NAND_ent is
begin

  F <= x nand y;          -- nand is a reserved keyword which perform NAND

end NAND_arch2;
-------------------------------------------
```

7 VHDL Code for NOR Gate

```
---------------------------- -- Lines
-- VHDL code for NOR gate    -- starting with    --
---------------------------- -- are comments


library ieee;        -- Include all the libraries
use ieee.std_logic_1164.all; -- & other built-in namespaces
                -- that are typically used in the code
---------------------------- -- Implementation

entity NOR_ent is             -- Entity is used to declare I/O used in the code
```

```
port( x: in std_logic;   -- port x and y is declared as input which accepts when
    y: in std_logic;   -- a logic input/Output value. i.e. 0 or 1
    F: out std_logic     -- F is declared as Output
);
end NOR_ent;

------------------------------------------

architecture NOR_arch1 of NOR_ent is
               -- architecture is where actual description lies.
               -- Two approaches can be used to describe
               -- how our code should be working.
begin

    process(x, y)      -- process states that perform the below set of code
               -- when either of x or y changes
    begin
        -- compare to truth table
  if (x='0' and y='0') then
               -- if x and y is equal to 0, '=' is comparison operator
            F <= '1';    -- Assign 1 to F
  else
      F <= '0';      -- Assign 0 to F, => is assignment operator
  end if;
    end process;

end NOR_arch1;        -- end of Architecture

architecture NOR_arch2 of NOR_ent is
begin

    F <= x nor y;       -- nor is a reserved keyword which perform NOR

end NOR_arch2;

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

8 VHDL Code for NOT Gate

```
---------------------------- -- Lines
-- VHDL code for NOT gate    -- starting with   --
---------------------------- -- are comments


library ieee;        -- Include all the libraries
use ieee.std_logic_1164.all; -- & other built-in namespaces
               -- that are typically used in the code
---------------------------- -- Implementation

entity NOT_ent is            -- Entity is used to declare I/O used in the code
port( x: in std_logic;   -- port x is declared as input which accepts when
               -- a logic input/Output value. i.e. 0 or 1
    F: out std_logic     -- F is declared as Output
);
end NOT_ent;
```

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

architecture NOT_arch1 of NOT_ent is
                -- architecture is where actual description lies.
                -- Two approaches can be used to describe
                -- how our code should be working.
begin

    process(x)          -- process states that perform the below set of code
                -- when x changes
    begin
        -- compare to truth table
  if (x='1') then       -- If x = 1
            F <= '0';     -- Assign 0 to F which shows the opposite of input
  else
      F <= '1';
  end if;
    end process;

end NOT_arch1;          -- end of Architecture

architecture NOT_arch2 of NOT_ent is
begin

    F <= not x;         -- not is a reserved keyword which perform NOT

end NOT_arch2;

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

9 VHDL Code for OR Gate

```
------------------------------ -- Lines
-- VHDL code for OR gate       -- starting with    --
------------------------------ -- are comments


library ieee;        -- Include all the libraries
use ieee.std_logic_1164.all; -- & other built-in namespaces
                -- that are typically used in the code
------------------------------ -- Implementation

entity OR_ent is               -- Entity is used to declare I/O used in the code
port( x: in std_logic;   -- port x is declared as input which accepts when
    y: in std_logic;   -- a logic input/Output value. i.e. 0 or 1
    F: out std_logic     -- F is declared as Output
);
end OR_ent;

----------------------------------------

architecture OR_arch1 of OR_ent is
                -- architecture is where actual description lies.
                -- Two approaches can be used to describe
                -- how our code should be working.
begin

    process(x, y)       -- process states that perform the below set of code
                -- when either of x or y changes
    begin
       -- compare to truth table
        if ((x='0') and (y='0')) then
                -- if x and y is equal to 0 '=' is comparison operator
      F <= '0';       -- Assign 0 to F '=>' is assignment operator in VHDL
  else
      F <= '1';
  end if;
    end process;        -- end of Architecture

end OR_arch1;

architecture OR_arch2 of OR_ent is
begin

    F <= x or y;        -- or is a reserved keyword which perform OR

end OR_arch2;
```

10 VHDL Code for XNOR Gate

```
------------------------------ -- Lines
-- VHDL code for XNOR          -- starting with    --
------------------------------ -- are comments

library ieee;                  -- Include all the libraries
use ieee.std_logic_1164.all;   -- & other built-in namespaces
                -- that are typically used in the code
------------------------------ -- Implementation


entity XNOR_ent is             -- Entity is used to declare I/O used in the code
port( x: in std_logic;     -- port x is declared as input which accepts when
    y: in std_logic;     -- a logic input/Output value. i.e. 0 or 1
    F: out std_logic       -- F is declared as Output
);
end XNOR_ent;

------------------------------------------

architecture XNOR_arch1 of XNOR_ent is
                               -- architecture is where actual description lies.
                -- Two approaches can be used to describe
                -- how our code should be working.
begin

    process(x, y)       -- process states that perform the below set of code
                -- when either of x or y changes

    begin
```

```
        -- compare to truth table
        if (x/=y) then        -- if x is not equal to 0
      F <= '0';       -- Assign 0 to F
  else
      F <= '1';       -- Assign 1
  end if;
    end process;

end XNOR_arch1;         -- end of Architecture

architecture XNOR_arch2 of XNOR_ent is
begin

    F <= x xnor y;          -- xnor is a reserved keyword which perform XNOR

end XNOR_arch2;

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

11 VHDL Code for XOR Gate

```
----------------------------- -- Lines
-- VHDL code for XOR gate     -- starting with    --
----------------------------- -- are comments

library ieee;         -- Include all the libraries
use ieee.std_logic_1164.all; -- & other built-in namespaces
               -- that are typically used in the code
----------------------------- -- Implementation

entity XOR_ent is             -- Entity is used to declare I/O used in the code
port( x: in std_logic;   -- port x is declared as input which accepts when
    y: in std_logic;   -- a logic input/Output value. i.e. 0 or 1
    F: out std_logic     -- F is declared as Output
);
end XOR_ent;

----------------------------------------

architecture XOR_arch1 of XOR_ent is
               -- architecture is where actual description lies.
               -- Two approaches can be used to describe
               -- how our code should be working.
begin

    process(x, y)      -- process states that perform the below set of code
               -- when either of x or y changes
    begin
        -- compare to truth table
  if (x/=y) then        -- if x and y not equal to eachother
        F <= '1';       -- Assign 1 to F
  else
      F <= '0';       -- else 0 to F
  end if;
    end process;
end XOR_arch1;          -- end of Architecture

architecture XOR_arch2 of XOR_ent is
begin

    F <= x xor y;          -- xor is a reserved keyword which perform AND

end XOR_arch2;

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

12 Results and Discussions

- After compiling the code successful uploading and running of the Altera Board was achieved.

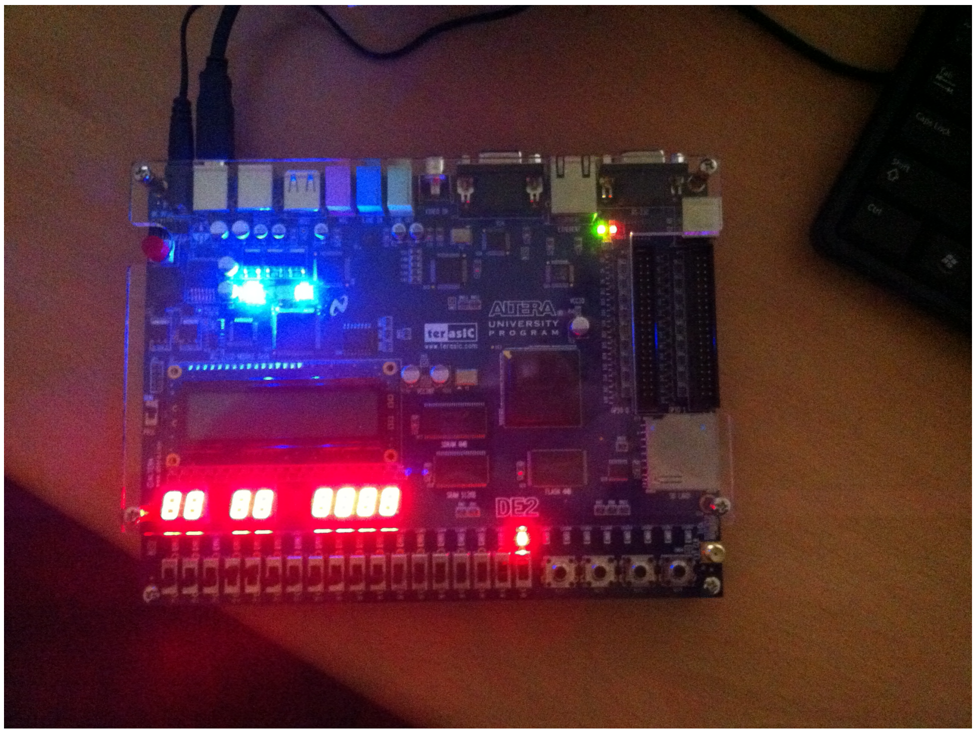


Figure 4: The Board working on AND GATE

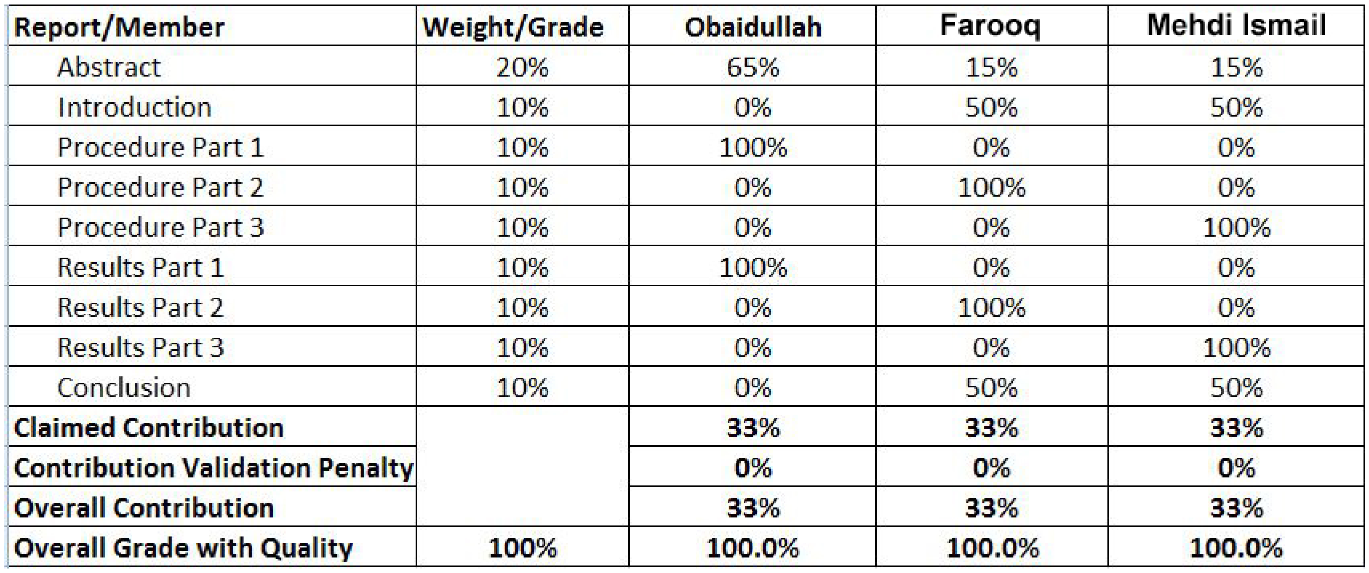
13 Conclusion

- These were just the basic gates, many gates can be joined together to make up combinational or sequential blocks.
- Solving a problem using FSM is much easier than doing it combinationally.
- FSM allows us to think of the outputs and inputs and build them into a number of stages.

14 References

http://www.altera.com/devices/fpga/cyclone2/overview/cy2-overview.html
Details on the CycloneII boards, pins, I/O pins availibility.
http://esd.cs.ucr.edu/labs/tutorial/
Website that built an idea through the class teaching/showing VHDL coding.

15 Team Dynamics



Report/Member	Weight/Grade	Obaidullah	Farooq	Mehdi Ismail
Abstract	20%	65%	15%	15%
Introduction	10%	0%	50%	50%
Procedure Part 1	10%	100%	0%	0%
Procedure Part 2	10%	0%	100%	0%
Procedure Part 3	10%	0%	0%	100%
Results Part 1	10%	100%	0%	0%
Results Part 2	10%	0%	100%	0%
Results Part 3	10%	0%	0%	100%
Conclusion	10%	0%	50%	50%
Claimed Contribution		**33%**	**33%**	**33%**
Contribution Validation Penalty		**0%**	**0%**	**0%**
Overall Contribution		**33%**	**33%**	**33%**
Overall Grade with Quality	**100%**	**100.0%**	**100.0%**	**100.0%**