# NRI INSTITUTE OF INFORMATION SCIENCE & TECHNOLOGY, BHOPAL


**Session-2015**

## DEPARTMENT OF ELECTRONICS & COMMUNICAITON ENGINEERING

# LAB MANUAL

# OF

# SOFTWARE LAB-IV

# EC-606

# LIST OF EXPERIMENT

1) To Write a program for basic logic gates ( NOT , AND, OR, NAND NOR XOR & XNOR) and Synthesize & simulate using VHDL Coding.
2) To Write a program for Half adder and full adder and Synthesize & simulate using VHDL Coding.
3) To Write a program for Half sub stractactor and full substractor and Synthesize & simulate using VHDL Coding.
4) To Write a program for 4-bit Parallel Binary Adder and Synthesize & simulate using VHDL Coding
5) To Write a program for 4-to-1 Multiplexer and Synthesize & simulate using VHDL Coding..
6) To Write a program for 1-to-4 Demultiplexer and Synthesize & simulate using VHDL Coding.
7) To Write a program for 3-to-8 Decoder and Synthesize & simulate using VHDL Coding.
8) To Write a program for RS, J-K, D Flip Flop and Synthesize & simulate using VHDL Coding.
9) To Write a program for 4-bit up Counter and Synthesize & simulate using VHDL Coding
10) To Write a program for 4-bit Down Counter and Synthesize & simulate using VHDL Coding.

# Introduction to Xilinx ISE

- **Xilinx software**

The Xilinx ISE 6.5i software will be used in this text. . All menus structures and screen shots are taken from the ISE 6.5i version.

**Introduction:**

A hardware description language (HDL) is a language that describes the hardware of digital systems in a textural form.

It can be used to represent

- Logic diagrams,
- Boolean expressions,
- Digital circuits.

**Features:**

- The HDL **represents digital systems in the form of documentation** which can be understood by human as well as computers.
- It makes **easy to exchange the ideas** between the designers.
- It provides **access to computer-aided design tools** to aid in the design process.
- It resembles a programming language, but the orientation of the HDL is specifically towards describing hardware structures and behavior.
- The **storage, retrieval and processing of programs** written using HDL can be **performed easily and efficiently**.
- They are used to **describe hardware for the purpose of simulation, modeling, testing, design and documentation**.

**Major HDLs:**

There are two standard HDL's that are supported by IEEE –

- VHDL,
- Verilog HDL.

**VHDL:**

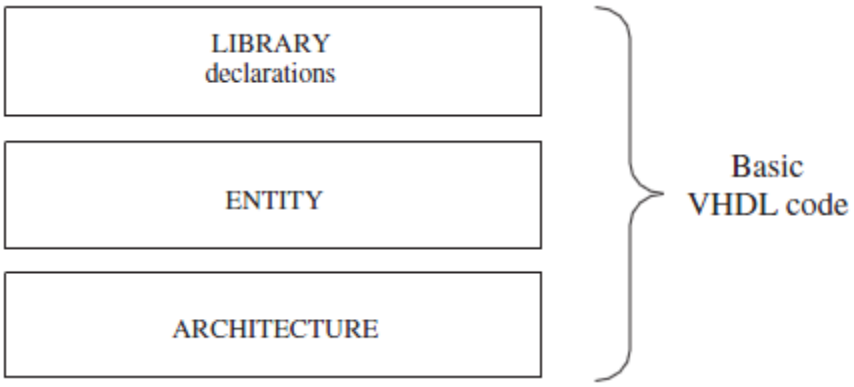
VHDL stands for **VHSIC Hardware Description Language**. VHSIC is itself an abbreviation for **Very High Speed Integrated Circuits**. VHDL was the original and first hardware description language to be standardized by the Institute of Electrical and Electronics Engineers (IEEE 1076 standard).

VHDL can be used in three different approaches for describing the hardware. These three different approaches are –

1. Data flow description
2. Structural description
3. Behavioral description

**DATA FLOW DESCRIPTION**:

To make the designs more modular and effective, a design is typically decomposed into several blocks. These blocks are connected together to form a complete design. As depicted in figure below, a standalone piece of VHDL code is composed of three fundamental sections:




**Fundamental sections of a basic VHDL code**

- **LIBRARY declarations**: Contains a list of all libraries to be used in the design. For Example: ieee, std, work, etc.
- **ENTITY**: Specifies the I/O pins of the circuit.
- **ARCHITECTURE**: Contains the VHDL code, which describes how the circuit should behave (function).

**1. LIBRARY Declarations**:

The Libraries can be declared in VHDL using two lines of code, one containing the name of the library and the other line containing a use clause as follows.

```
LIBRARY library_name;
USE library_name. package_name .all;
```

At least three packages, from three different libraries, are usually needed in a design:

- *ieee.std_logic_1164* (from the *ieee* library),
- *std.standard* (from the *std* library), and
- *work* (*work* library).

The library declarations for the above different packages are given as follows:

**LIBRARY** ieee;
**USE** *ieee.std_logic_1164*. all;

**LIBRARY** std;
**USE** *std.standard*. all;

**LIBRARY** work;
**USE** *work*. all;

The libraries std and work shown above are made visible by default, so there is no need to declare them; only the ieee library must be explicitly written.

**2. ENTITY**:

An ENTITY is a list with specifications of all inputs and output pins (PORTS) of the circuit. Its syntax is shown below.

```
ENTITY entity_name IS
    PORT (
     port_name : signal_mode  signal_type;
     port_name : signal_mode  signal_type;
       ……);
END entity_name;
```

The ***mode*** specifies whether this is an input (in), output (out), or both (inout). The ***type*** specifies the kind of values the signal can carry.

Comments can be added at the end of a VHDL statement or as an individual line by itself preceded by the '—' symbol.

**Example**: Let us consider the AND gate.


**AND gate**

Its ENTITY can be specified as:

```
ENTITY and_gate IS
    PORT (
        A, B: in std_logic;
```

The meaning of the ENTITY above is the following:

The signals A and B are of mode *in* (inputs) and type *std_logic* (standard logic). The signal Y is declared as mode *out* (output) and of the type *std_logic*. The name chosen for the entity was ***and_gate***.

### 3. ARCHITECTURE:

The ARCHITECTURE is a description of how the circuit should behave. The syntax is of the following:

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

- **FPGA development board**

  The Digilent Digilab IIE board (available from www.digilentinc.com) will be used in this tutorial, however, other boards utilizing a Xilinx FPGA can easily be substituted. The Digilab IIE board contains a Xilinx Spartan-IIE device (XC2S200E) with the equivalent of approximately 200,000 gates. The Digilab IIE board contains a minimal number of prototyping devices, but it contains 6 40-pin I/O headers for attaching daughter boards with additional functionality.

  The Spartan-IIE devices are identical to those in Xilinx's Virtex-E family of FPGAs. In fact, the product model stored on the device itself is the Virtex-E model number (for the FPGA on the Digilab board the on-chip device name is XCV200E).

- **Example Project:**

  VHDL entry, compilation, and downloading of a simple inverter.

  To test the design, the pushbutton on the Diglab IIE board will be used as the input to the

inverter and the LED on the board will be used as the output.

- **VHDL Design Entry**

  The Xilinx ISE tools allow the design to be entered several ways including graphical schematics, state machine diagrams, VHDL, and Verilog. This tutorial will focus on VHDL entry, but the other methods are similar and can be easily explored once the reader is comfortable with the ISE software.

- **Starting a project**

  Start the Xilinx ISE Project Navigator. Choose File ) New Project. A popup dialog box will appear. Enter tutor1 for Project Name. For the Project Location, select the directory where the project will be stored (i.e., Z:nXProjntutor1) for your project.

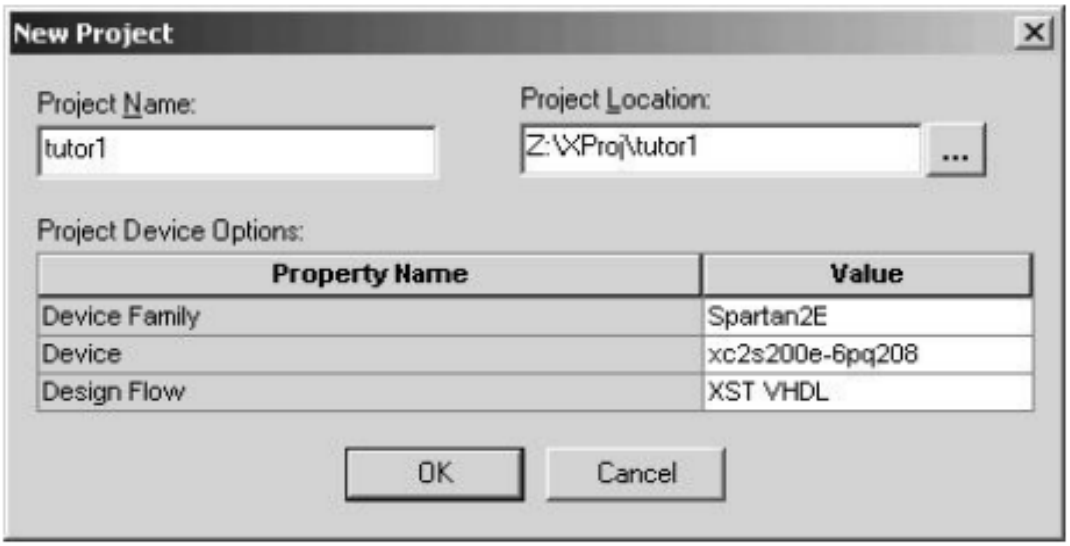



**Figure L1.1:** The New Project dialog box is used to enter the project and FPGA device information required to create a new project.

Next, the FPGA that will be used with this project needs to be specified. The compilation process is device specific, so the complete device specification (including package type) must be entered when creating a new project. Look on the top of the FPGA you are using. The de vice model, package type, and speed grade will be printed on it. The Spartan-IIE device used on the Digilab IIE board is shown in Fig ure L1.2. For the Digilab IIE board, set the Device Family to Spar tan2E. The device model is XC2S200E, the package type is PQ208, and the speed grade is 6, so set the Device to xc2s200e-6pq208. Finally, set the Design Flow to XST VHDL and click OK.

- **Creating a design file:**

  Choose Project ) New Source. . . . A popup dialog box will appear. Select VHDL Module from the list on the left and enter top level for the File Name. Make sure the Add to Project option is checked and click Next. The next screen of options allows you to enter the input and output signals for this module. Leave the default values for Entity Name and Architecture Name. For this project, only one input and one output are needed. For the input, enter pb for the Port Name and in for the Direction. Likewise, for the output, enter led1 and out respectively. Since both of these signals are only one bit, the MSB and LSB fields can be left blank. If a multi-signal bus was being specified, then the bus width would be specified in these fields. Click Next. A summaryof the new ¯le to be created will be displayed next. Verify that

  all of the information is correct. Use the Back button to return to any previous screen and make corrections. When you have verified that everything is correct, click Finish. The design ¯le top level.vhd will be created, added to the current project, and opened for editing as shown in Figure L1.5.

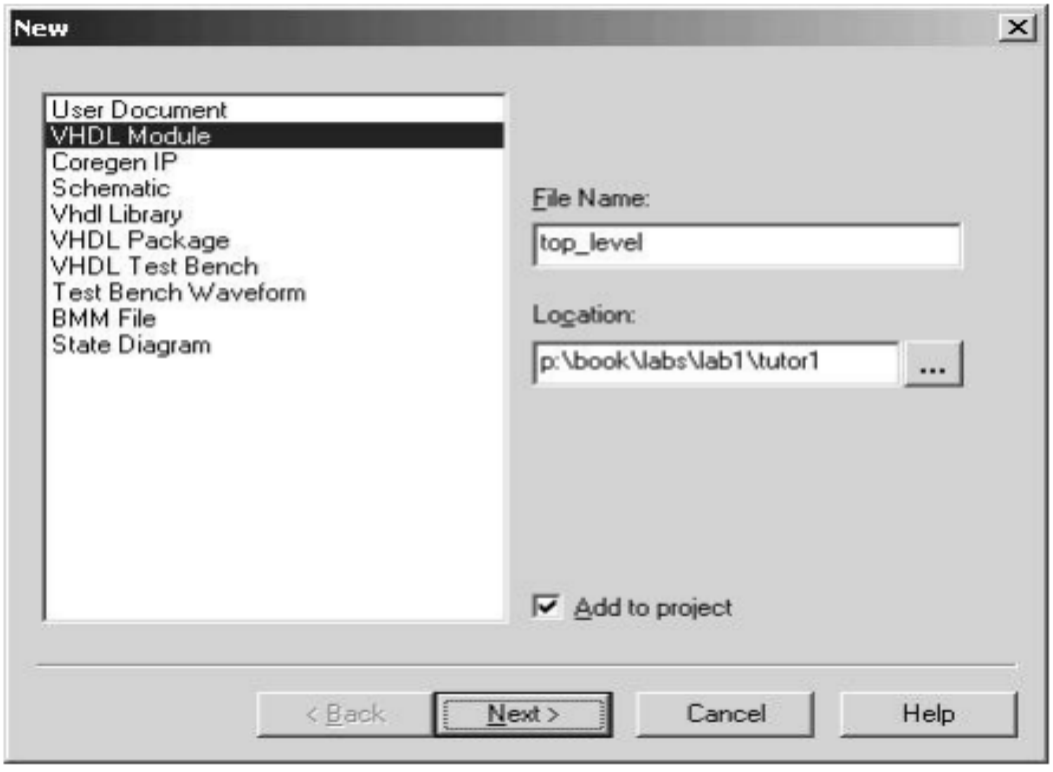



**Figure L1.3:** Specify the name and type of the new source file in the New Source dialog box.

**Getting to know the Project Navigator**

Take a few minutes to familiarize yourself with the Project Navigator layout. As shown in Figure L1.5, there are four window panes in the default layout. All text ¯les (design ¯les, report ¯les, etc.)

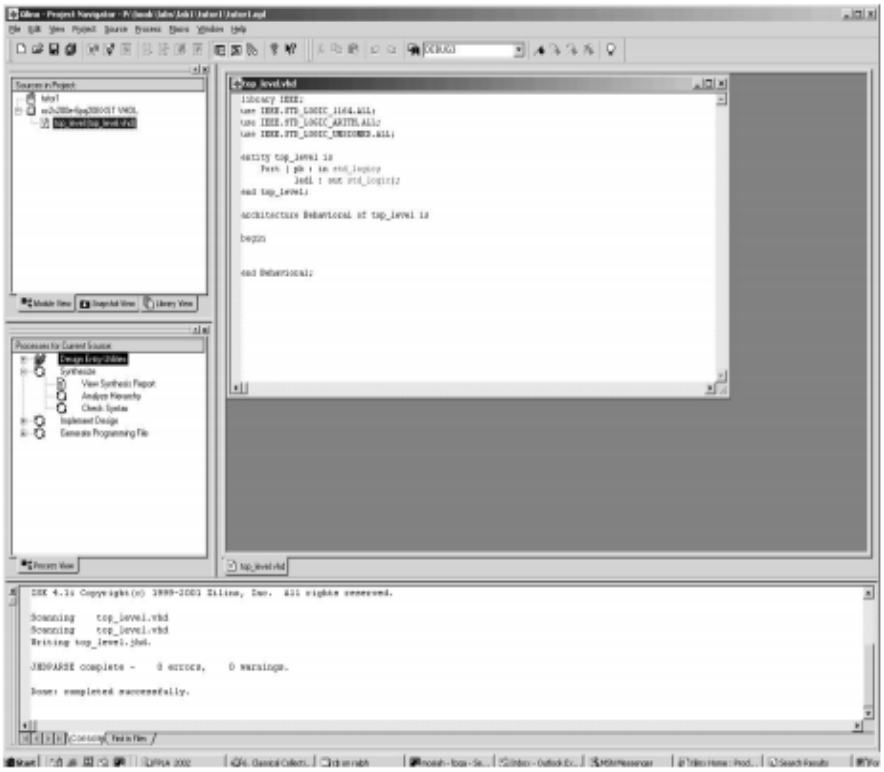
are displayed in the main window on right-hand side of the screen. In the upper left hand pane, you may select the Module View, Snapshot View, or Library View. Click on each of these tabs and look at the information provided. Then click on Module View to return to the default. Module View displays the design files in the current project and shows the relationship between the ¯les. Double-clicking on any design file in the Module View window will open up that ¯le in the main window.

The Process View window is displayed in the lower left-hand side. This window provides access to the pin assignment file, simulator, compiler settings, compilation reports, compiler controls, and FPGA programmer. Click on the + sign beside each item in the Process View window to view the different options available.

Finally, the bottom pane is the Console window. All text output from the Project Navigator, including warning and error messages, are displayed here.

Writing the VHDL model

Look at the top _level.vhd ¯le that you generated earlier. The first four lines of code indicate which VHDL libraries and pack ages are needed for this VHDL module. IEEE.STD LOGIC 1164, IEEE.STD LOGIC ARITH, and IEEE.STD LOGIC UNSIGNED are the basic VHDL packages from the IEEE library that define the key words and operators for the language. These should be included at the top of all VHDL ¯les that you create, unless you have a specific need to use a different library and/or package



.

==========
=

## LIST OF EXPERIMENT

1) To Write a program for basic logic gates ( NOT , AND, OR, NAND NOR XOR & XNOR) and Synthesize & simulate using VHDL Coding.
2) To Write a program for Half adder and full adder and Synthesize & simulate using VHDL Coding.
3) To Write a program for Half sub stractactor and full substractor and Synthesize & simulate using VHDL Coding.
4) To Write a program for 4-bit Parallel Binary Adder and Synthesize & simulate using VHDL Coding
5) To Write a program for 4-to-1 Multiplexer and Synthesize & simulate using VHDL Coding..
6) To Write a program for 1-to-4 Demultiplexer and Synthesize & simulate using VHDL Coding.
7) To Write a program for 3-to-8 Decoder and Synthesize & simulate using VHDL Coding.
8) To Write a program for RS, J-K, D Flip Flop and Synthesize & simulate using VHDL Coding.
9) To Write a program for 4-bit up Counter and Synthesize & simulate using VHDL Coding
10) To Write a program for 4-bit Down Counter and Synthesize & simulate using VHDL Coding.

# EXPERIMENT NO: 1

**AIM:** To Write a program for basic logic gates ( NOT , AND, OR, NAND NOR XOR & XNOR) and Synthesize & simulate using VHDL Coding.

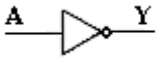
- **SOFTWARE USED:** Xilinx ISE 6.5 software.
- APPARATUS REQUIRED: SPARTAN 2.
- **PROCEDURE:**

  - Open XILINX ISE
  - Open new VHDL-file
  - Choose File ) New Project
  - Project Name & Project Location
  - Creating a design file, Project --- New Source
  - Select VHDL Module
  - Add to Project option is checked and click Next
  - Compile and Run the program
  - For the output see command window\ Figure window

- **PROGRAM :**

The VHDL program for logic gates are written following the data flow approach.

1. **Implementation of NOT logic.**
   The VHDL program for the NOT gate shown in the figure below can be written as follows:




**NOT gate**

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


entity not11 is
   Port ( a : in  STD_LOGIC;
          b : out  STD_LOGIC);
end not11;

architecture Behavioral of not11 is
```

```
begin
b <= not a ;

end Behavioral;
```

## 2. Implementation of AND logic.

The VHDL program for the AND gate shown in the figure below can be written as follows:


**AND gate**

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity nitinand is
    Port ( p,q : in  STD_LOGIC;
           r : out  STD_LOGIC);
end nitinand;

architecture Behavioral of nitinand is

begin
r <= p and q ;

end Behavioral;
```

### 3. Implementation of OR logic.

The VHDL program for the OR gate shown in the figure below can be written as follows:


**OR gate**

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity or11 is
    Port ( a,b : in  STD_LOGIC;
           x : out  STD_LOGIC);
end or11;

architecture Behavioral of or11 is

begin
x <= a or b ;

end Behavioral;
```

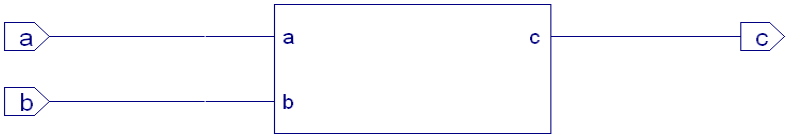
### 4. Implementation of NAND logic

The VHDL program for the NAND gate shown in the figure below can be written as follows:

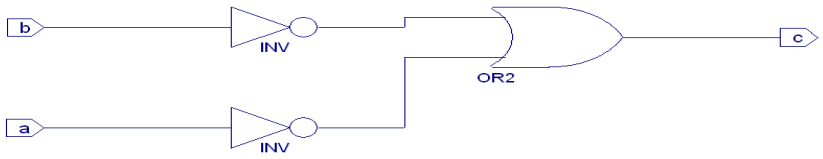

**NAND gate**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity nandgate is
   Port ( a : in std_logic;
          b : in std_logic;
          c : out std_logic);
end nandgate;
architecture Behavioral of nandgate is
begin
c<= a nand b;
end Behavioral;
```

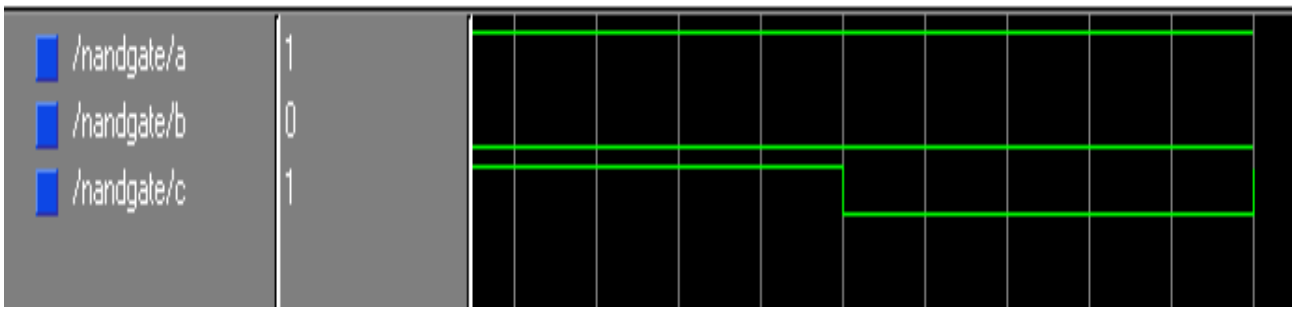
- RTL SCHEMATIC

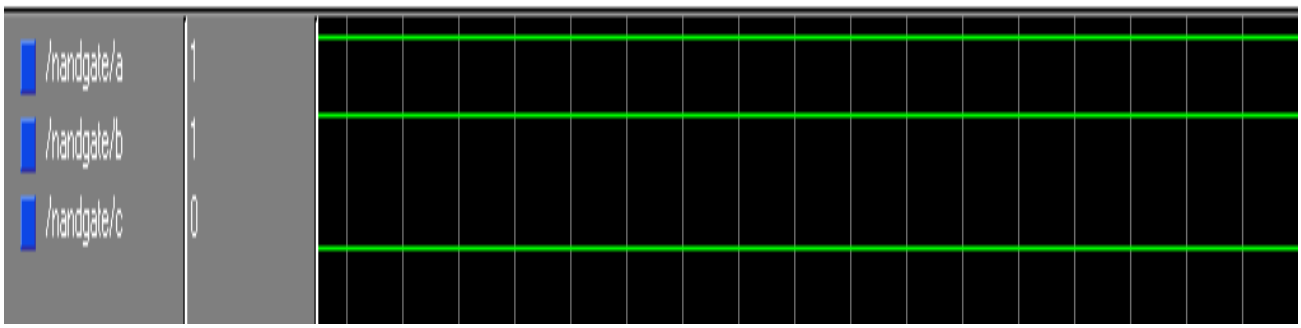



SOFTWARE GENERATED INTERNAL ARCHITECTURE




- SIMULATION WAVEFORM



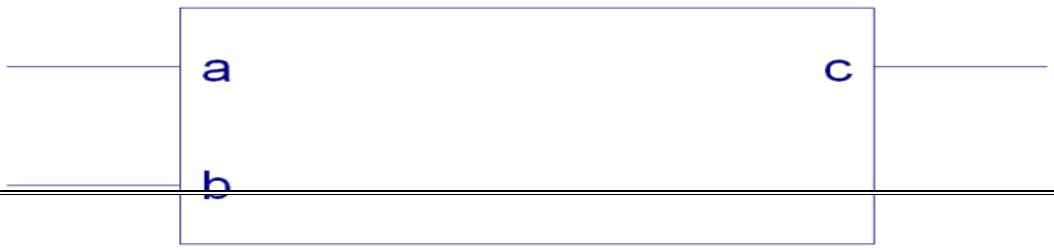
## 5. Implementation of NOR logic

The VHDL program for the NOR gate shown in the figure below can be written as follows:

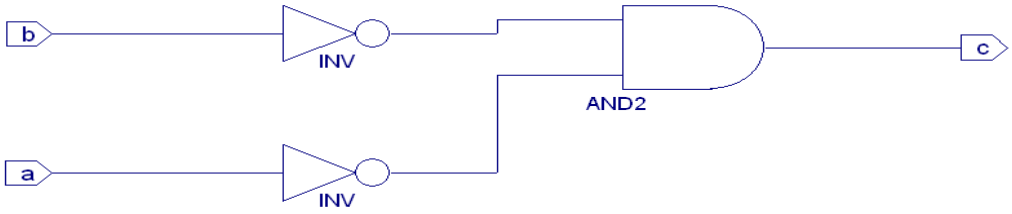

**NOR gate**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity norgate is
    Port ( a : in std_logic;
           b : in std_logic;
           c : out std_logic);
end norgate;
architecture Behavioral of norgate is
begin
 c<= a nor b;
end Behavioral;
```

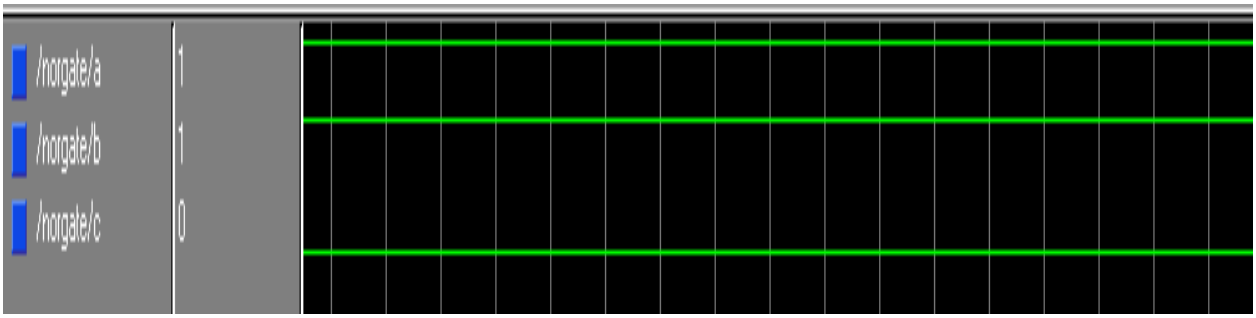
- RTL SCHEMATIC

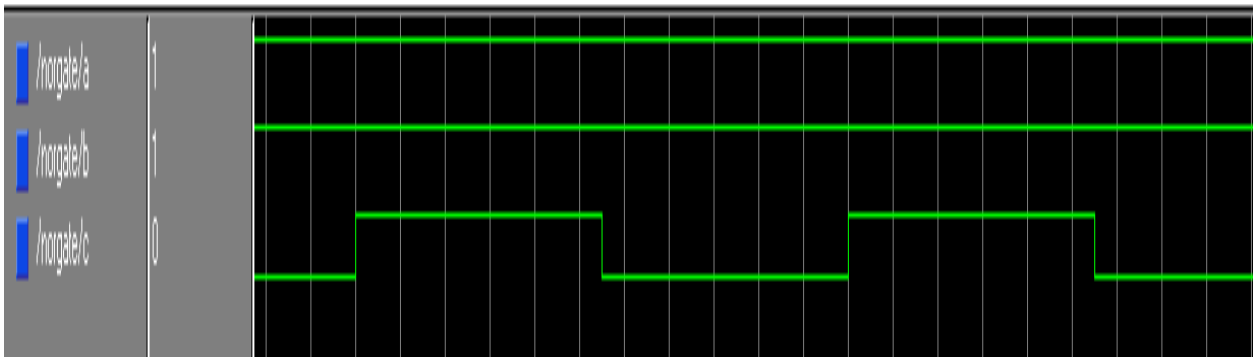


- SOFTWARE GENERATED INTERNAL ARCHITECTURE




- SIMULATION WAVEFORM






## 6. IMPLEMENTATION OF XOR GATE

The VHDL program for the XOR gate shown in the figure below can be written as follows:

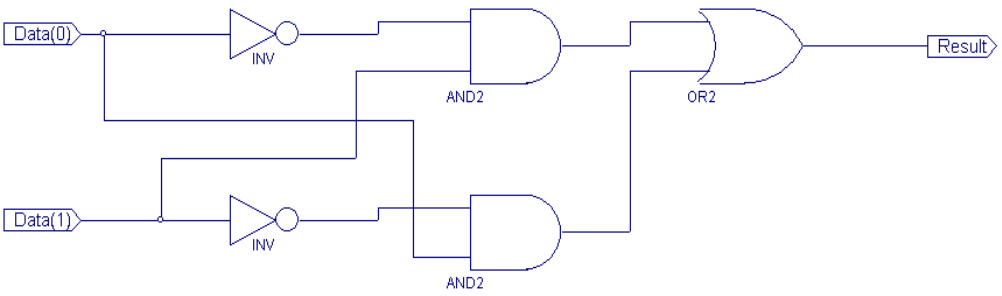

**XOR gate**

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

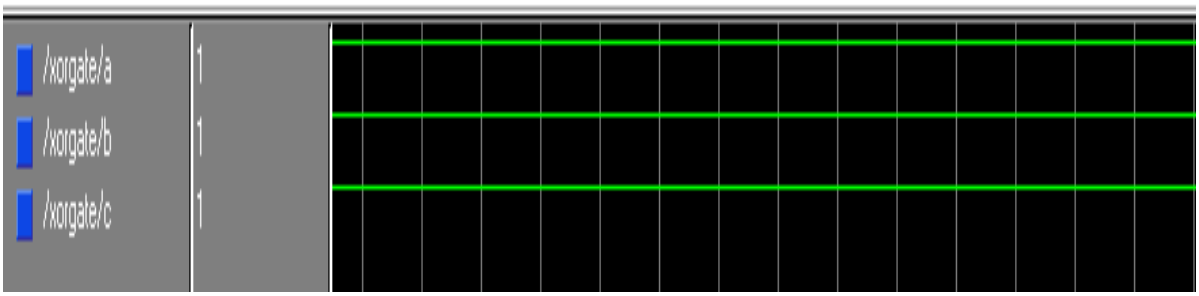
```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity xorgate is
    Port ( a : in std_logic;
           b : in std_logic;
           c : out std_logic);
end xorgate;
architecture Behavioral of xorgate is
begin
   c<= a xor b;
end Behavioral;
```

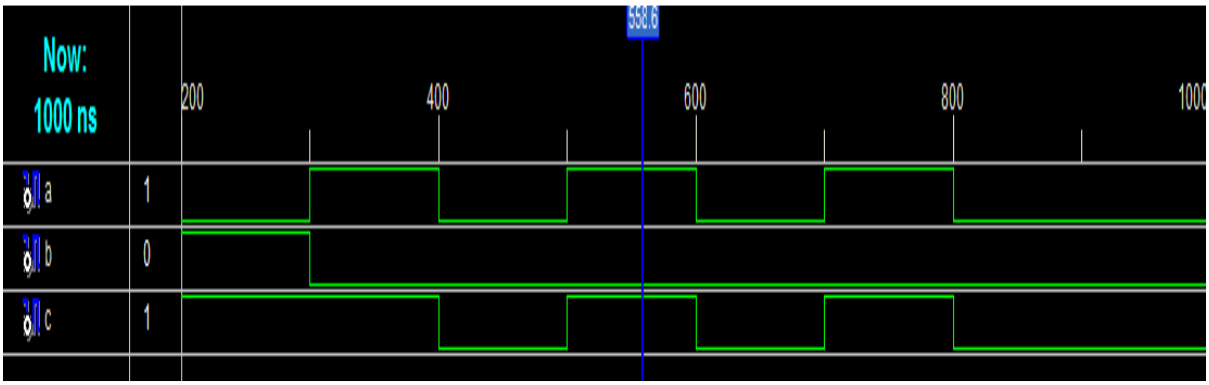
- RTL SCHEMATIC


- SOFTWARE GENERATED INTERNAL ARCHITECTURE




- SIMULATION WAVEFORM




## 7. IMPLEMENTATION OF XNOR GATE

The VHDL program for the XNOR gate shown in the figure below can be written as follows:

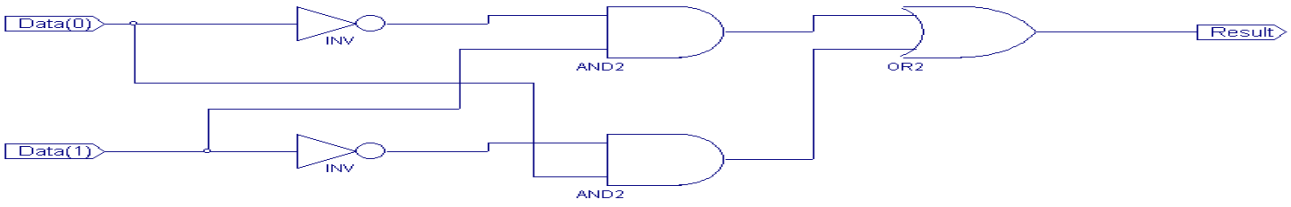

**XNOR gate**

library IEEE;

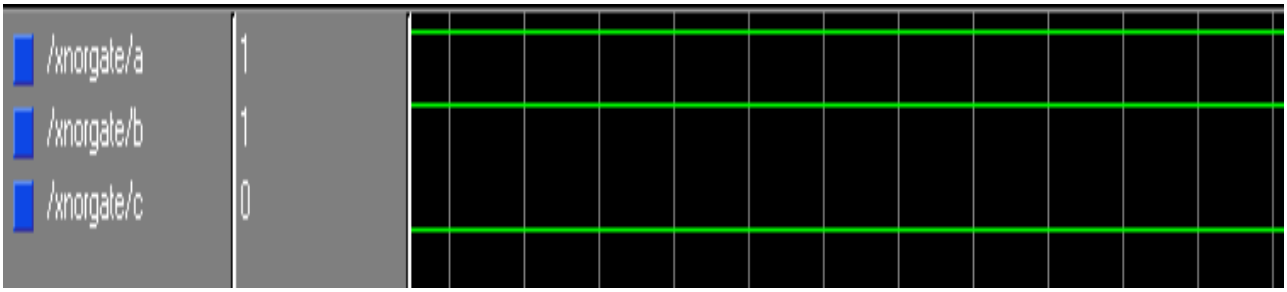
```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity xnorgate is
    Port ( a : in std_logic;
           b : in std_logic;
           c : out std_logic);
end xnorgate;
architecture Behavioral of xnorgate is
begin
 c<= a xnor b;
end Behavioral;
```

- **RTL SCHEMATIC**


- SOFTWARE GENERATED INTERNAL ARCHITECTURE




- SIMULATION WAVEFORM




- **CONCLUSION:** The simulation result of elementary RTL, SOFTWARE GENERATED INTERNAL ARCHITECTURE, SIMULATION WAVEFORM logic gate.

==========

## EXPERIMENT NO: 2

**AIM:** To Write a program for Half adder and full adder and Synthesize & simulate using VHDL Coding.

- **SOFTWARE USED:** Xilinx ISE 6.5 software.
- **APPARATUS REQUIRED**: SPARTAN 2.
- **THEORY:**

**Half Adder:**

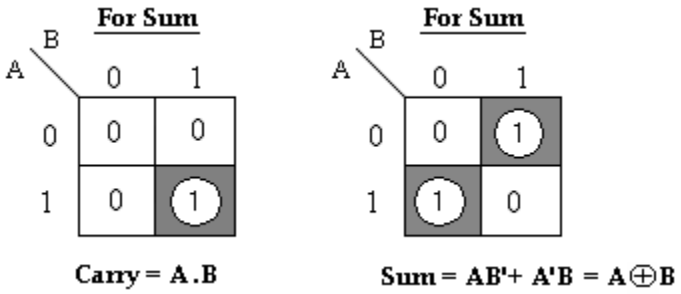
A *half- adder* is a combinational circuit that can be used to add two binary bits. It has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY.

The truth table of a half-adder, showing all possible input combinations and the corresponding outputs are shown below.

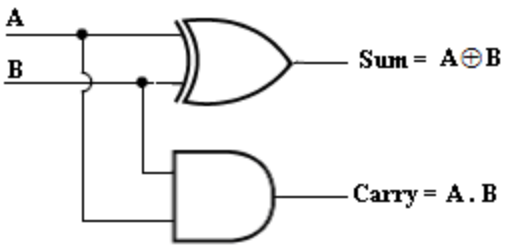
| Inputs | | Outputs | |
|---|---|---|---|
| **A** | **B** | **Carry (C)** | **Sum (S)** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**Truth table of half-adder**

**K-map Simplification:**




**The logic diagram of the half adder is,**




**Logic Diagram of Half Adder**

## Full Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. Two of the input variables, represent the significant bits to be added. The third input represents the carry from previous lower significant position.

The truth table is shown below,

**Truth Table:**

| Inputs | | | Outputs | |
|---|---|---|---|---|
| **A** | **B** | **$C_{in}$** | **Sum (S)** | **Carry ($C_{out}$)** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

To derive the simplified Boolean expression from the truth table, the Karnaugh map method is adopted as,

**For Carry**

| A \ $BC_{in}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

**Carry, $C_{out}$ = $AB + AC_{in} + BC_{in}$**

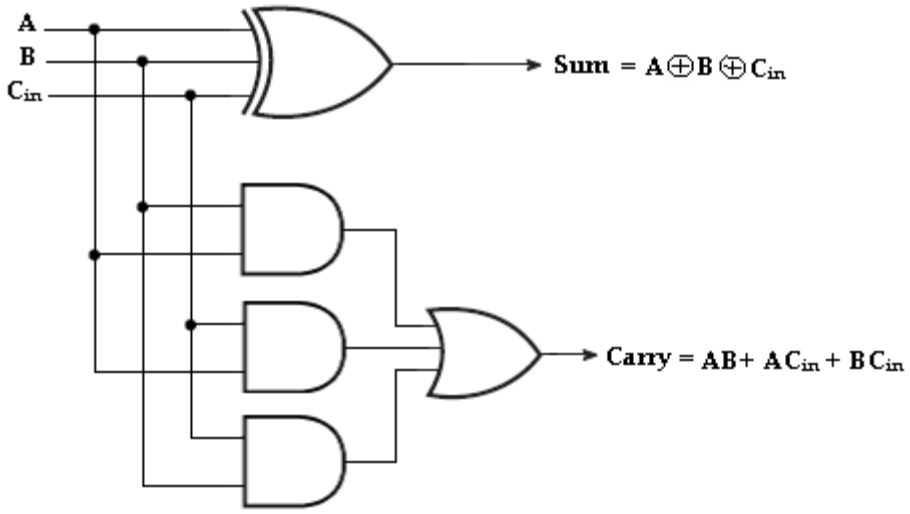
**For Sum**

| A \ $BC_{in}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

**Sum, $S = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$**

**$= A \oplus B \oplus C_{in}$**

The logic diagram of the full adder is,




**Logic Diagram of Full Adder**

## ➢ PROCEDURE:

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

## ➢ PROGRAM :

**Implement half adder using VHDL.**

**Data Flow**

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

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;


entity half2 is
    Port ( a,b : in  STD_LOGIC;
           s,c : out  STD_LOGIC);
end half2;
architecture Behavioral of half2 is
begin
s <= a xor b ;
c <=a and b ;
end Behavioral;
```

## Half Adder by Structural

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
```

```
--library UNISIM;
--use UNISIM.VComponents.all;

entity halfadders is
    Port ( p,q : in  STD_LOGIC;
           r,s : out  STD_LOGIC);
end halfadders;
architecture Behavioral of halfadders is
component xor2
port (a,b:in std_logic; x:out std_logic);
end component;
component nitinand
port(a,b:in std_logic; x:out std_logic);
end component;
begin
g : xor2 port map (p,q,r) ;
h : nitinand port map (p,q,s) ;
end Behavioral;
```

## Half Adder by Behavior

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;
entity halfadderb is
    Port ( a,b : in  STD_LOGIC ;
           sum,carry : out  STD_LOGIC) ;
end halfadderb;
architecture Behavioral of halfadderb is
begin
process (a,b)
variable x1,x2 : std_logic ;
begin
x1 := not a ;
x2 := not b ;
sum <=(a and x2)or (b and x1) ;
carry <= a and b ;
end process;
end Behavioral;
```

## IMPLEMENT FULL ADDER USING VHDL.

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;
entity fulladder1 is
   Port ( a,b,c : in  STD_LOGIC;
          sum ,carry : out  STD_LOGIC);
end fulladder1;
architecture Behavioral of fulladder1 is
begin
sum <= a xor b xor c ;
carry <= (a and b)or (b and c)or (c and a) ;
end Behavioral;
```

## EXPERIMENT NO: 3

**AIM: To Write a program for Half substractor and full substractor and Synthesize & simulate using VHDL Coding.**

- **SOFTWARE USED:** Xilinx ISE 6.5 software.
- APPARATUS REQUIRED: SPARTAN 2.
- **THEORY:**

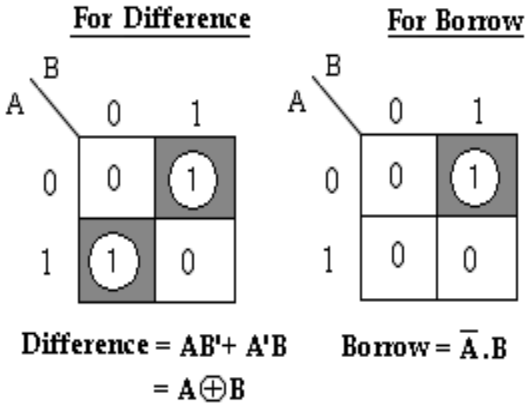
### Half Subtractor

A *half- subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction.

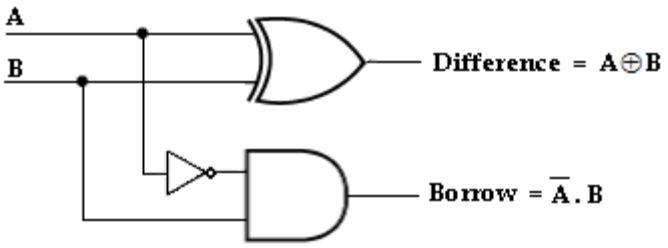
The truth table of half-subtractor, showing all possible input combinations and the corresponding outputs are shown below.

| Input | | Output | |
|---|---|---|---|
| **A** | **B** | **Difference (D)** | **Borrow ($B_{out}$)** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

**K-map simplification:**




The logic diagram of the half subtractor is,




i. **Logic Diagram of Half Subtractor**

**ii. Full Subtractor:**

A *full subtractor* performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.

As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as $B_{in}$. There are two outputs, namely DIFFERENCE, D and BORROW, $B_{out}$. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit.

The truth table for full-subtractor is,

**Truth Table:**

| **Inputs** | | | **Outputs** | |
|---|---|---|---|---|
| **A** | **B** | **$B_{in}$** | **Difference(D)** | **Borrow($B_{out}$)** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**K-map Simplification:**

**For Difference**

| A \ $BB_{in}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

**For Borrow**

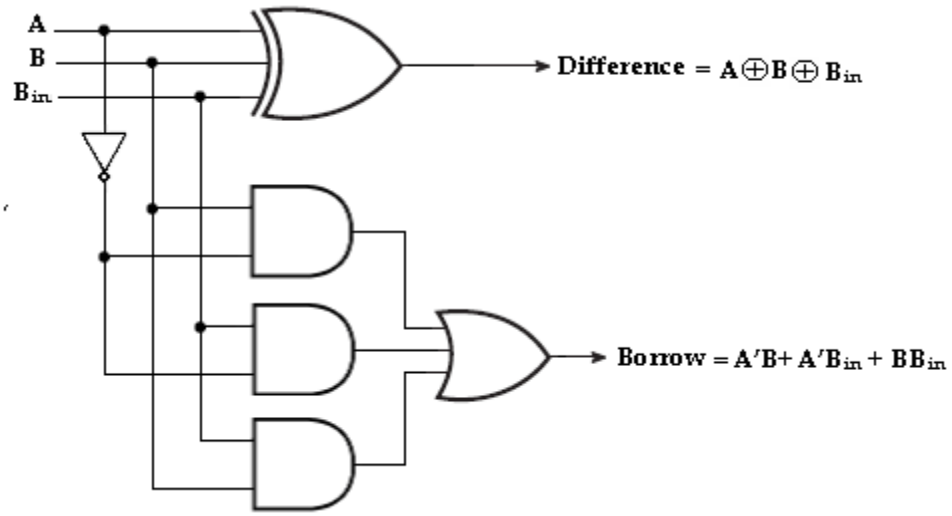
| A \ $BB_{in}$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |

$$\text{Difference, } D = A'B'B_{in} + A'BB'_{in} + AB'B'_{in} + ABB_{in}$$

$$= A \oplus B \oplus B_{in}$$

$$\text{Borrow, } B_{out} = A'B + A'B_{in} + BB_{in}$$

The logic diagram of the full subtractor is,




**Logic Diagram of Full Subtractor**

## ➢ PROCEDURE:

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

## ➢ PROGRAM :

## Implement Half subtractor using VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;


entity halfsub is
    Port ( a,b : in  STD_LOGIC;
               dif,bro : out  STD_LOGIC);
end halfsub;
architecture Behavioral of halfsub is
begin
dif <=  a xor b;
bro <=  (not a)and b;
end Behavioral;
```

## Implement Full Subtractor using VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;


entity fullsub is
    Port ( a,b,c : in  STD_LOGIC;
           dif,bro : out  STD_LOGIC);
end fullsub;
architecture Behavioral of fullsub is
component halfsub
port(a,b:in std_logic; dif,bro:out std_logic);
end component;
component or1
port (a,b:in std_logic; c:out std_logic);
end component;
signal d1,b1,b2:std_logic;
begin
x1: halfsub port map (a,b,d1,b1);
x2: halfsub port map (c,d1,dif,b2);
x3: or1 port map (b1,b2,bro);
end Behavioral;
```

## EXPERIMENT NO: 4

**AIM:** To Write a program for 4-bit Parallel Binary Adder and Synthesize & simulate using VHDL Coding.

➢ **SOFTWARE USED:** Xilinx ISE 6.5 software.

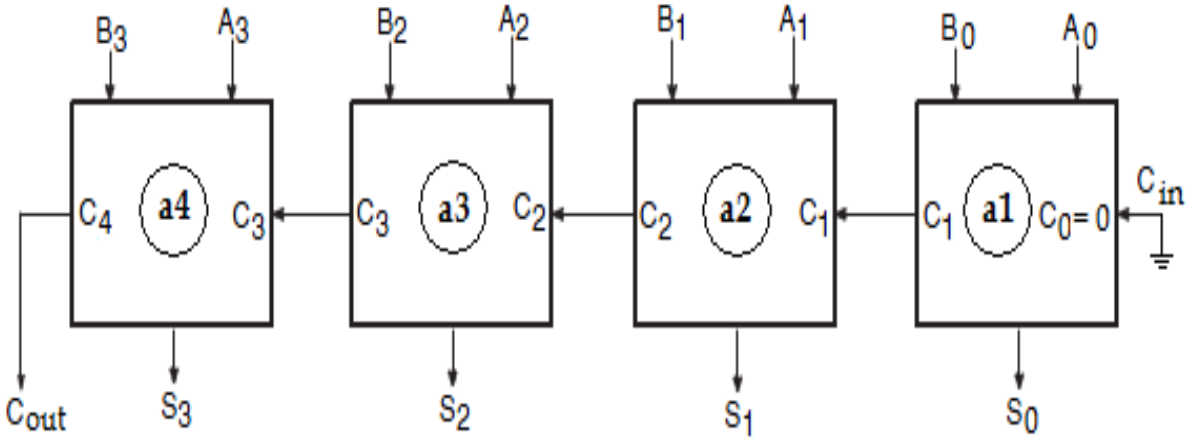
APPARATUS REQUIRED: SPARTAN 2

➢ **THEORY :**

**4-bit Parallel Binary Adder:**

The 4-bit binary adder using full adder circuits is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output. The bits are added with full adders, starting from the least significant position, to form the sum it and carry bit. The input carry $C_0$ in the least significant position must be 0. The carry output of the lower order stage is connected to the carry input of the next higher order stage.

The VHDL program for the 4-bit parallel binary adder can be written as follows. This program follows ***structural approach***.




**4-bit Parallel Binary Adder**

➢ **PROCEDURE:**

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

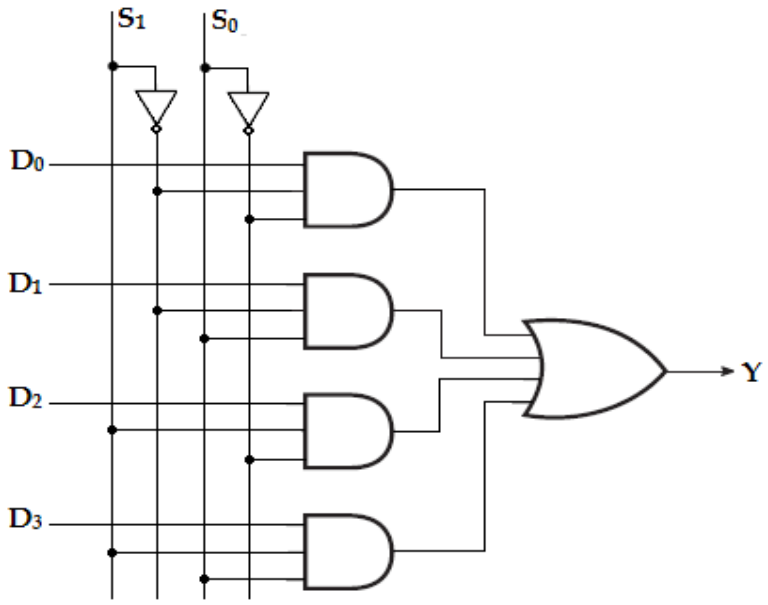
➢ **PROGRAM :**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY 4bit_adder IS
      PORT (
         A, B: in std_logic_vector (0 to 3);
         S: out std_logic_vector (0 to 3);
         Cout: out std_logic);
END 4bit_adder;
ARCHITECTURE arch_4bitadder OF 4bit_adder IS
   Signal Cin, C1, C2, C3, C4: Std_logic;
   □ Holds current value and a set of □ possible future values
COMPONENT fulladder IS
   PORT ( A, B, C: in std_logic);
         Sum, Carry: out std_logic);
         Cout: out std_logic);
END COMPONENT;
BEGIN
         Cin<='0';
         a1: full adder port map (A(0),B(0),Cin,S(0),C1);
         a2: full adder port map (A(1),B(1),C1,S(1),C2);
         a3: full adder port map (A(2),B(2),C2,S(2),C2);
         a4: full adder port map (A(3),B(3),C3,S(3),C3);
         Cout <= C4;
END subtractor _arch;
```

- **AIM: To Write a program for 4-to-1 Multiplexer and Synthesize & simulate using VHDL Coding.**
- **SOFTWARE USED:** Xilinx ISE 6.5 software.
- APPARATUS REQUIRED: SPARTAN 2
- **THEORY :**

**4-to-1 Multiplexer:**

Each of the four inputs $D_0$ through $D_3$, is applied to one input of AND gate. Selection lines $S_1$ and $S_0$ are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output.




**4-to-1 Multiplexer**

.

## PROCEDURE:

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

## PROGRAM :

**Multiplexer 4 x 1 Using IF**

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

entity mux41 is
   Port ( i0,i1,i2,i3 : in  STD_LOGIC;
          z : out  STD_LOGIC;
          s0,s1 : in std_logic);
end mux41;
architecture Behavioral of mux41 is
begin
process (i0,i1,i2,i3,s0,s1)
begin
if (s0 ='0' and s1 ='0') then
z<=i0;
elsif (s0 ='0' and s1 ='1')then
z<= i1;
elsif (s0 ='1' and  s1 ='0')then
z<= i2;
else
z<= i3;
end if ;
end process;
end Behavioral;
```

## Multiplexer 4 x 1 Using CASE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity muxcase is
    Port ( i : in  STD_LOGIC_VECTOR (3 downto 0);
           s: in  STD_LOGIC_vector (1 downto 0);
           z : out  STD_LOGIC);
end muxcase;
architecture Behavioral of muxcase is
begin
process (s)
begin
case s is
when "00"=> z<=i(0);
when "01"=> z<=i(1);
when "10"=> z<=i(2);
when others => z<= i(3);
end case;
end process;
end Behavioral;
```

## MULTIPLEXER 4 X 1 USING WITH SELECT

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity muxslt41 is
    Port ( i : in  STD_LOGIC_VECTOR (03 downto 0);
           s : in  STD_LOGIC_VECTOR (01 downto 0);
           z : out  STD_LOGIC);
end muxslt41;
architecture Behavioral of muxslt41 is
begin
with s select
z <= i(0) when "00",
      i(1) when "01",
      i(2) when "10",
      i(3) when others ;
end Behavioral;
```

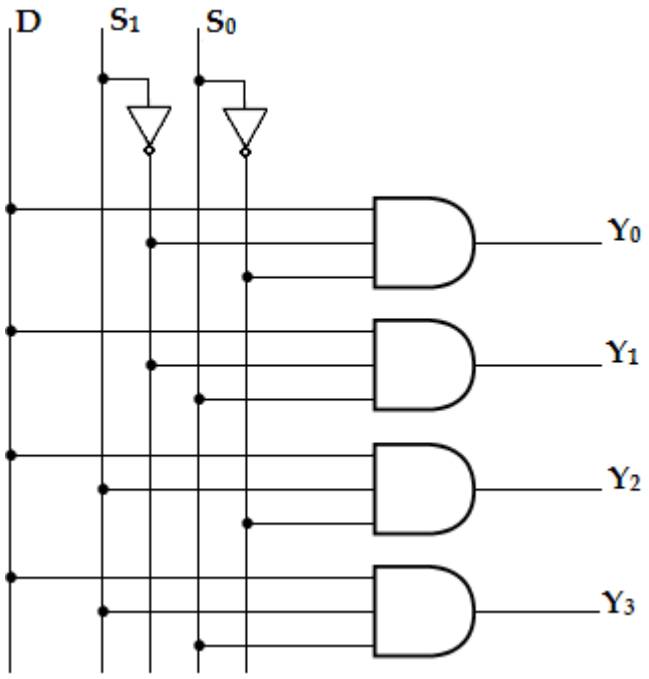
**AIM: To Write a program for 1-to-4 Demultiplexer and Synthesize & simulate using VHDL Coding**.

➢ **SOFTWARE USED:** Xilinx ISE 6.5 software.

APPARATUS REQUIRED: SPARTAN 2

**1-to-4 Demultiplexer:**

A 1-to-4 demultiplexer has a single input, $\mathbf{D_{in}}$, four outputs ($\mathbf{Y_0}$ **to** $\mathbf{Y_3}$) and two select inputs ($\mathbf{S_1}$ **and** $\mathbf{S_0}$). It can be implemented using four 3-input AND gates and two NOT gates. Here, the input data line D, is connected to all the AND gates. The two select lines $S_1$, $S_0$ enable only one gate at a time and the data that appears on the input line passes through the selected gate to the associated output line.




**1-to-4 line Demultiplexer**

➢ **PROCEDURE:**

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

➢ **PROGRAM :**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY demux IS
    PORT (
      D, S1, S0: in std_logic;
      Y0, Y1, Y2, Y3: out std_logic);
END demux;
ARCHITECTURE arch_demux OF demux IS
BEGIN
    Y0 <= D and (not S1) and (not S0);
    Y1 <= D and (not S1) and S0;
    Y2 <= D and S1 and (not S0);
    Y3 <= D and S1 and S0;
END arch_demux;
```

**AIM:** To Write a program for 3-to-8 Decoder and Synthesize & simulate using VHDL Coding.

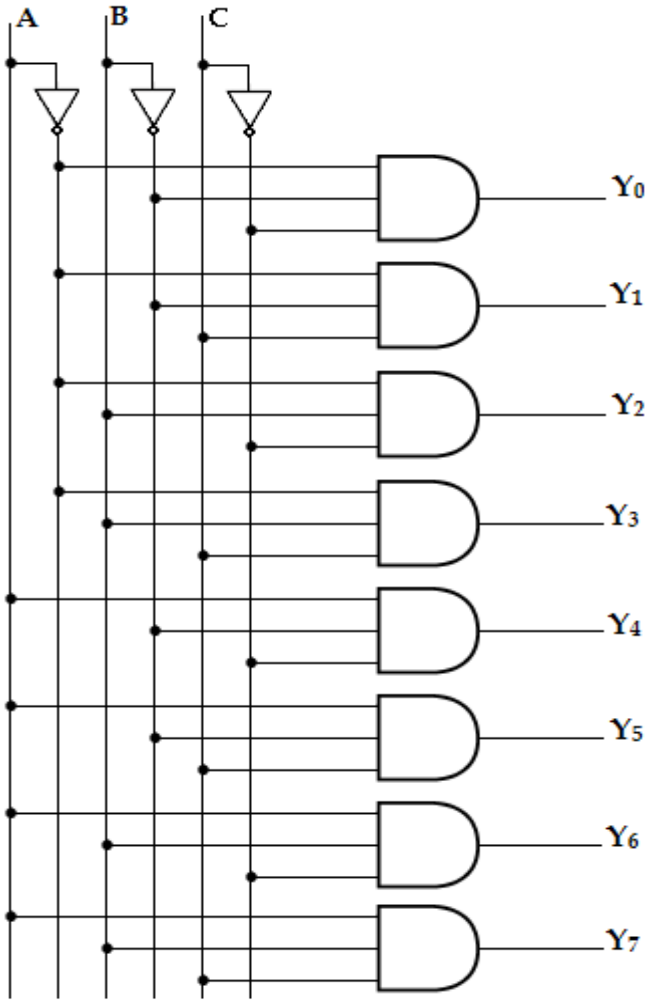
- **SOFTWARE USED:** Xilinx ISE 6.5 software.

APPARATUS REQUIRED: SPARTAN 2

**3-to-8 Decoder:**

A 3-to-8 line decoder has three inputs (A, B, C) and eight outputs ($Y_0$- $Y_7$). Based on the 3 inputs one of the eight outputs is selected.

The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. The input variables may represent a binary number and the outputs will represent the eight digits in the octal number system. The output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 represents the min term equivalent of the binary number presently available in the input lines.




**3-to-8 line decoder**

## ➢ PROCEDURE:

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

## ➢ PROGRAM : Implement Decoder using VHDL

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder161 is

   Port ( i : in  STD_LOGIC_VECTOR (15 downto 0);

          a,b,c,d : in  STD_LOGIC;

          z : out  STD_LOGIC);

end decoder161;

architecture Behavioral of decoder161 is

begin

process (a,b,c,d,i)

begin

if (a='0' and b='0' and  c='0' and d ='0')then

z<= i(0);

elsif a='0' and b='0' and c='0' and d='1' then

z<= i(1);

elsif a='0' and b='0' and c='1' and d='0' then

z<= i(2);

elsif a='0' and b='0' and c='1' and d='1' then
```

```
z<= i(3);
elsif a='0' and b='1' and c='0' and d='0' then
z<= i(4);
elsif a='0' and b='1' and c='0' and d='1' then
z<= i(5);
elsif a='0' and b='1' and c='1' and d='0' then
z<= i(6);
elsif a='0' and b='1' and c='1' and d='1' then
z<= i(7);
elsif a='1' and b='0' and c='0' and d='0' then
z<= i(8);
elsif a='1' and b='0' and c='0' and d='1' then
z<= i(9);
elsif a='1' and b='0' and c='1' and d='0' then
z<= i(10);
elsif a='1' and b='0' and c='1' and d='1' then
z<= i(11);
elsif a='1' and b='1' and c='0' and d='0' then
z<= i(12);
elsif a='1' and b='1' and c='0' and d='1' then
z<= i(13);
elsif a='1' and b='1' and c='1' and d='0' then
z<= i(14);
else
z<= i(15);
end if;
end process;
end Behavioral;
```

# EXPERIMENT NO: 8

**AIM:** To Write a program for RS,J-K, D Flip Flop and Synthesize & simulate using VHDL Coding.

- **SOFTWARE USED:** Xilinx ISE 6.5 software.

APPARATUS REQUIRED: SPARTAN 2

## THEORY

## JK Flip-flop:

The VHDL program for the JK Flip-flop can be written as follows. This program follows ***behavioral approach*** as per truth table.

| CLK | Inputs | | Output | State |
|---|---|---|---|---|
| | J | K | $Q_{n+1}$ | |
| 1 | 0 | 0 | $Q_n$ | No Change |
| 1 | 0 | 1 | 0 | Reset |
| 1 | 1 | 0 | 1 | Set |
| 1 | 1 | 1 | $Q_n$' | Toggle |

## D Flip-flop:

The VHDL program for the D Flip-flop can be written as follows. This program follows ***behavioral approach*** as per truth table.

| Clock | D | $Q_{n+1}$ | State |
|---|---|---|---|
| 1 | 0 | 0 | Reset |
| 1 | 1 | 1 | Set |
| 0 | x | $Q_n$ | No Change |

## ➢ PROCEDURE:

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

## ➢ PROGRAM :

### Implement R-S flip-flop using VHDL.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating

---- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity rsflipflop is

    Port ( r,s,clk,reset : in  STD_LOGIC;

           q,qbar : out  STD_LOGIC);

end rsflipflop;

architecture Behavioral of rsflipflop is

signal  s1,s2:std_logic;

begin

process( clk,reset)

begin

if falling_edge(clk)then
```

```
if reset='1' then
s1<='0';
elsif(r='0' and s='0')then
s1<=s1;
s2<=s2;
elsif (r='0' and s='1')then
s1<='1';
s2<='0';
elsif(r='1' and s='0')then
s1<='0';
s2<='1';
end if;
end if;
end process;
q<=s1;
qbar<=s2;
end Behavioral;
```

## Implement J-K flip-flop using VHDL.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity jkff is
    Port ( j,k,clk,reset : in  STD_LOGIC;
             q,qbar : out  STD_LOGIC);
end jkff;
architecture Behavioral of jkff is
signal s1 : std_logic;
begin
process(clk,j,k,reset)
begin
 if (reset = '1')then
           s1 <= '0';
           elsif rising_edge (clk)then
            if (j ='0' and k= '0')then
            s1 <= s1;
            elsif (j='0' and k='1')then
            s1<= '0';
            elsif (j='1' and k='0')then
            s1<= '1';
            elsif (j='1' and k='1')then
            s1<= not s1;
            end if;
            end if;
            end process;
 q<= s1;
 qbar<= not s1;
end Behavioral;
```

## Implement D flip-flop using VHDL

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


entity dff is
    Port ( d, clk : in  STD_LOGIC;
           q, qbar : out  STD_LOGIC);
end dff;
architecture Behavioral of dff is
signal x: std_logic;
begin
process (clk)
begin
if (clk' event and clk = '1') then
x<= d;
else
x <= x;
end if;
end process;
q<= x; qbar<= not x;
end Behavioral;
```

# EXPERIMENT NO: 9

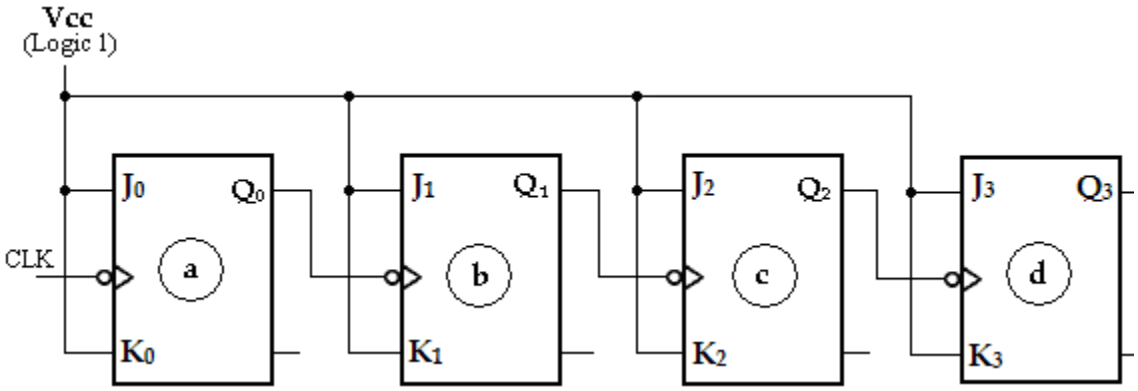
**AIM:** To Write a program for 4-bit Up Counter and Synthesize & simulate using VHDL Coding.

➢ **SOFTWARE USED:** Xilinx ISE 6.5 software.

APPARATUS REQUIRED: SPARTAN 2

## 4-bit Asynchronous/ Ripple Counter:

The VHDL program for the 4-bit Asynchronous/ Ripple Counter shown in the figure below can be written as follows. This program follows ***structural approach***.




➢ **PROCEDURE:**

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

➢ **PROGRAM :**

## Implement 4-bit Up Counter using VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counter is
generic (n:natural:=4) ;
   Port ( reset,clk: in  STD_LOGIC;
          q : out  STD_LOGIC_vector (3 downto 0));
end counter;
architecture Behavioral of counter is
signal state:std_logic_vector (n-1 downto 0);
begin
process (reset,clk)
begin
if (reset='1')then
state <= "0000";
elsif rising_edge (clk)then
state <= state (n-1 downto 0) + '1';
else state <=  state (n-1 downto 0);
end if ;
end process;
q<= state (n-1 downto 0);
end Behavioral;
```

# EXPERIMENT NO: 10

**AIM:** To Write a program for 4-bit Down Counter and Synthesize & simulate using VHDL Coding.

- **SOFTWARE USED:** Xilinx ISE 6.5 software.

APPARATUS REQUIRED: SPARTAN 2

- **PROCEDURE:**

- Open XILINX ISE
- Open new VHDL-file
- Choose File ) New Project
- Project Name & Project Location
- Creating a design file, Project --- New Source
- Select VHDL Module
- Add to Project option is checked and click Next
- Compile and Run the program
- For the output see command window\ Figure window

➢ **PROGRAM :**

## Implement 4-bit Down Counter using VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counterdown is
generic (n:natural:=4) ;
   Port ( reset,clk: in  STD_LOGIC;
          q : out  STD_LOGIC_vector (3 downto 0));
end counterdown;
architecture Behavioral of counterdown is
signal state : std_logic_vector (n-1 downto 0);
begin
process (reset,clk)
begin
if (reset='1')then
state <= "0000";
elsif rising_edge (clk)then
state <= state (n-1 downto 0) - '1';
else state <=  state (n-1 downto 0);
end if ;
end process;
q<= state (n-1 downto 0);
end Behavioral;
```

10)