# Computer Organization &Architecture Lab

# Paper Code: PCCCS492

# Work Book

IEM

INSTITUTE OF ENGINEERING & MANAGEMENT

श्रद्धावान लभते ज्ञानम्

Good Education, Good Jobs

## Department of Computer Science & Engineering

## B. Tech

## 2nd year 4th Semester

**NAME:** Aditya Bose

**SECTION:** A

**CLASS ROLL NO.:** 1 **ENROLLMENT NO.:** 12020002002001

# INSTITUTE OF ENGINEERING & MANAGEMENT

## COMPUTER ARCHITECTURE LAB

## PAPER CODE: PCCCS 492

**LAB ASSIGNMENTS**

**1.Implementation of AND, OR, NOT, XOR, NANAD, NOR gates using Xilinx ISE.**

**2. Implementation of Half Adder, Half Subtractor, Full Adder, Full Subtractor using Xilinx ISE.**

**3. Implementation of 8 bit Adder, 8bit Subtractor, 8 bit Multiplier and 4 bit parallel Adder and 4 bit parallel subtractor using Xilinx ISE.**

**4. Implementation of Binary to Gray and Gray to binary using Xilinx ISE.**

**5. Implementation of 4 bit comparator, 2:4 Decoder, 3:8 Decoder using Xilinx ISE.**

**6. Implementation of 2:4 Decoder (using case), 4:2 Encoder (Data flow and using case), 8:3 Encoder (using Loop).**

**7. Implementation of 2:1 MUX (Data flow), 4:1 MUX (Dataflow, if-else, using case), 1:4 DEMUX (using case and data flow) using Xilinx ISE.**

**8. Implementation of Full Adder using Half Adders (Structural Method), 4 bit Parallel Adder (using Structural Method).**

**9. Implementation of 2:1 MUX using basic gates(Structural Method), 4:1 MUX using 2:1 MUX (using Structural Method).**

**10. Implementation of SR Flip-Flop, JK Flip Flop, D Flip Flop and T (Toggle) Flip Flop using Xilinx ISE.**

## Computer Architecture Laboratory Projects (PCCCS 492)

1 Design Control Unit (CU) based on 8-bit word size processor. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

2 Design Arithmetic & Logical Unit(CU) based on 8-bit word size processor.(CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

3 Design a Signed Multiplier Unit. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

4 Design a *Customized* Encrypter with key size of 4-bit. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

5 Design a *Customized* Decrypter with key size of 4-bit. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

6 Design a 16-bit Cyclic Redundancy Code Checker. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

7 Design a Hamming Code Generator Unit. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

8 Design a Digital Clock Unit with HH:MM:SS output. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

9 Design a Stack Unit with Memory Control. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

10 Design a Instruction Queue Unit of word size *n*. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

11 Design a Register Unit which contains *x* number of registers (x is a multiple of 2). It must allow register pair operations. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

12 Design Memory (RAM) Unit with *n* word-size &*m* number of words. (CO1, CO2, CO3, CO4, PO1-PO5, PO9, PO12)

**Assignment 1:** Implementation of AND, OR, NOT, XOR, NAND, NOR gates using Xilinx ISE.
**Aim:**

**Objective:** To implement OR gate

**Software used:**

| Property Name | Xilinx ISE |
|---|---|
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | VHDL |

**Theory:**

OR gate is a logical circuit which performs the logical OR operation on 2 or more Boolean variables.

**Truth Table:**

| **INPUT** | | **OUTPUT** |
|---|---|---|
| **A** | **B** | **Y** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

## Behavioral Model:

Here, the gate is designed, using the **truth table**. It simulates the behavior of the output of the gate, upon taking different Boolean variables as inputs. The **Behavioral Architecture** is used in this case.

## Data flow Model:

Here, the gate is constructed, using the **Dataflow Architecture**. The gate is designed, using it's **Boolean function**.

## Code:

1. **Behavioral Model Code:**

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_OR_gate is
end tb_OR_gate;

architecture behavior of tb_OR_gate is
 component OR_gate is
  port(x: in std_logic;
       y: in std_logic;
       z: out std_logic;
       );
 end component;
signal x,y: std_logic := '0';
signal z: std_logic;

begin
```

```
 uut: OR_gate port map(x,y,z);

stim: process
 begin
  x<='0';y<='0'; wait for 100 ns;
  x<='0';y<='1'; wait for 100 ns;
  x<='1';y<='0'; wait for 100 ns;
  x<='1';y<='1'; wait for 100 ns;
 end process;
end;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity OR_gate is
 port(x: in std_logic;
      y: in std_logic;
      z: out std_logic;
      );
end OR_gate;

architecture Behavioral of OR_gate is
 begin
  process(x,y)

   begin

    if(x='0' and y='0') then z<='0';
     else z<='1';
    end if;


  end process;
end Behavioral;
```

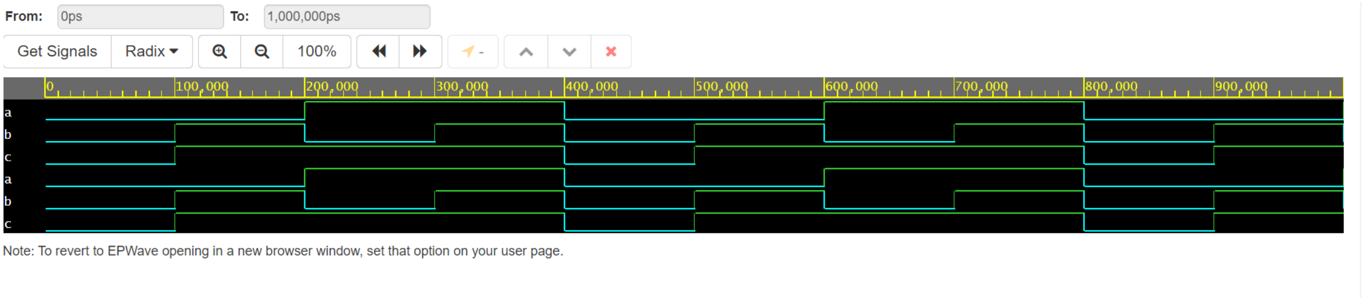
**<u>2. Data flow Model Code</u>**

```
-- Code your testbench here

library IEEE;
```

```
use IEEE.std_logic_1164.all;

entity tb_OR_gate is
end tb_OR_gate;

architecture myarch of tb_OR_gate is
 component OR_gate is
  port(a: in std_logic;
       b: in std_logic;
       c: out std_logic;
       );
 end component;
 signal a,b: std_logic:='0';
 signal c: std_logic;
begin
 uut: OR_gate port map(a,b,c);

stim: process
 begin
  a<='0';b<='0'; wait for 100 ns;
  a<='0';b<='1'; wait for 100 ns;
  a<='1';b<='0'; wait for 100 ns;
  a<='1';b<='1'; wait for 100 ns;
 end process;
end myarch;
```

```
-- Code your design here

library IEEE;

use IEEE.std_logic_1164.all;


entity OR_gate is

 port(a: in std_logic;

      b: in std_logic;

      c: out std_logic;

      );

end OR_gate;


architecture myarch of OR_gate is

 begin

  c<=a or b;

end myarch;
```

**Output:**



From: 0ps To: 1,000,000ps

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

**Objective:** To implement AND gate

**Software used:**

| Property Name | Xilinx ISE |
|---|---|
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | VHDL |

**Theory:**

The **AND Gate** performs the **logical AND** operation, between 2 or more different Boolean variables. It simulates the **arithmetic multiplication operation**.

**Truth Table:**

| **INPUT** | | **OUTPUT** |
|---|---|---|
| **A** | **B** | **Y** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## Behavioral Model:

Here, the gate is designed, using the **truth table**. It simulates the behavior of the output of the gate, upon taking different Boolean variables as inputs. The **Behavioral Architecture** is used in this case.

## Data flow Model:

Here, the gate is constructed, using the **Dataflow Architecture**. The gate is designed, using it’s **Boolean function**.

## Code:

### 1.Behavioral Model Code:

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_AND_gate is
end tb_AND_gate;

architecture behavior of tb_AND_gate is
 component AND_gate is
  port(x: in std_logic;
       y: in std_logic;
       z: out std_logic;
       );
 end component;
signal x,y: std_logic := '0';
signal z: std_logic;

begin
 uut: AND_gate port map(x,y,z);

stim: process
 begin
  x<='0';y<='0'; wait for 100 ns;
  x<='0';y<='1'; wait for 100 ns;
  x<='1';y<='0'; wait for 100 ns;
  x<='1';y<='1'; wait for 100 ns;
 end process;
end;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity AND_gate is
 port(x: in std_logic;
      y: in std_logic;
      z: out std_logic;
      );
```

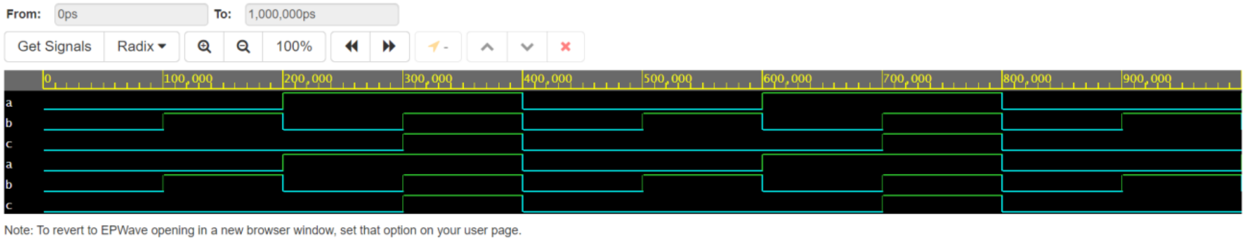
```
end AND_gate;

architecture Behavioral of AND_gate is
 begin
  process(x,y)

   begin

    if(x='1' and y='1') then z<='1';
     else z<='0';
    end if;


 end process;
end Behavioral;
```

**2. Data flow Model Code:**

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_AND_gate is
end tb_AND_gate;

architecture myarch of tb_AND_gate is
 component AND_gate is
```

```
  port(a: in std_logic;
       b: in std_logic;
       c: out std_logic;
       );
 end component;
 signal a,b: std_logic:='0';
 signal c: std_logic;
begin
 uut: AND_gate port map(a,b,c);

stim: process
 begin
  a<='0';b<='0'; wait for 100 ns;
  a<='0';b<='1'; wait for 100 ns;
  a<='1';b<='0'; wait for 100 ns;
  a<='1';b<='1'; wait for 100 ns;
 end process;
end myarch;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity AND_gate is
 port(a: in std_logic;
      b: in std_logic;
      c: out std_logic;

      );

end AND_gate;


architecture myarch of AND_gate is

 begin

  c<=a and b;

end myarch;
```

**Output:**



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

**Objective:** To implement NOT gate

**Software used:**

| Property Name | Xilinx ISE |
|---|---|
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | VHDL |

**Theory:**

The **NOT Gate** inverts a Boolean variable, i.e., alters its value from 0 to 1 and from 1 to 0. Thus, it performs the **logical NOT** operation, on one Boolean variable.

**Truth Table:**

| **INPUT** | **OUTPUT** |
|---|---|
| **A** | **Y** |
| 0 | 1 |
| 1 | 0 |

**Behavioral Model:**

Here, the gate is designed, using the **truth table**. It simulates the behavior of the output of the gate, upon taking different Boolean variables as inputs. The **Behavioral Architecture** is used in this case.

**Data flow Model:**

Here, the gate is constructed, using the **Dataflow Architecture**. The gate is designed, using it's **Boolean function**.

**Code:**

**1. Behavioral Model Code:**

```
entity tb_NOT_gate is
end tb_NOT_gate;

architecture behavior of tb_NOT_gate is
 component NOT_gate is
  port(x: in std_logic;
       y: out std_logic;
       );
 end component;
signal x: std_logic := '0';
signal y: std_logic;

begin
 uut: NOT_gate port map(x,y);

stim: process
 begin
  x<='0'; wait for 100 ns;
  x<='1'; wait for 100 ns;
 end process;
end;

-- Code your design here
```

```
library IEEE;

use IEEE.std_logic_1164.all;


entity NOT_gate is

 port(x: in std_logic;

      y: out std_logic;

      );

end NOT_gate;


architecture Behavioral of NOT_gate is

 begin

  process(x)


   begin


    if(x='0') then y<='1';

     else y<='0';

    end if;




  end process;

end Behavioral;
```

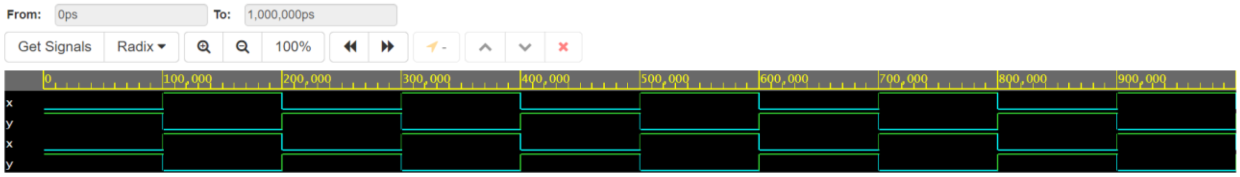
**2. Data flow Model Code:**

```
-- Code your testbench here
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_NOT_gate is
end tb_NOT_gate;

architecture myarch of tb_NOT_gate is
 component NOT_gate is
  port(a: in std_logic;
       b: out std_logic;
       );
 end component;
 signal a: std_logic:='0';
 signal b: std_logic;
begin
 uut: NOT_gate port map(a,b);

stim: process
 begin
  a<='0'; wait for 100 ns;
  a<='1'; wait for 100 ns;
 end process;
end myarch;

-- Code your design here
library IEEE;
```

```
use IEEE.std_logic_1164.all;


entity NOT_gate is
 port(a: in std_logic;
      b: out std_logic;
      );
end NOT_gate;


architecture myarch of NOT_gate is
 begin
  b<=not a;
end myarch;
```

**Output:**



From: 0ps To: 1,000,000ps

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

**Objective:** To implement NOR gate

## Software used:

| Property Name | Xilinx ISE |
|---|---|
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | VHDL |

## Theory:

The **NOR Gate** represents a **NOT Gate**, followed by an **OR Gate**, i.e., it negates the OR Gate, for 2 or more Boolean variables.

## Truth Table:

| **INPUT** | | **OUTPUT** |
|---|---|---|
| **A** | **B** | **Y** |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

## Behavioral Model:

Here, the gate is designed, using the **truth table**. It simulates the behavior of the output of the gate, upon taking different Boolean variables as inputs. The **Behavioral Architecture** is used in this case.

## Data flow Model:

Here, the gate is constructed, using the **Dataflow Architecture**. The gate is designed, using it’s **Boolean function**.

## Code:

### 1.Behavioral Model Code:

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;


entity tb_NOR_gate is
end tb_NOR_gate;


architecture behavior of tb_NOR_gate is
 component NOR_gate is
  port(x: in std_logic;
       y: in std_logic;
       z: out std_logic;
       );
 end component;
signal x,y: std_logic := '0';
signal z: std_logic;


begin
 uut: NOR_gate port map(x,y,z);
```

```
stim: process
 begin
  x<='0';y<='0'; wait for 100 ns;
  x<='0';y<='1'; wait for 100 ns;
  x<='1';y<='0'; wait for 100 ns;
  x<='1';y<='1'; wait for 100 ns;
 end process;
end;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity NOR_gate is
 port(x: in std_logic;
      y: in std_logic;
      z: out std_logic;
      );
end NOR_gate;

architecture Behavioral of NOR_gate is
 begin
  process(x,y)

   begin
```

```
    if(x='0' and y='0') then z<='1';
     else z<='0';
    end if;



  end process;
end Behavioral;
```

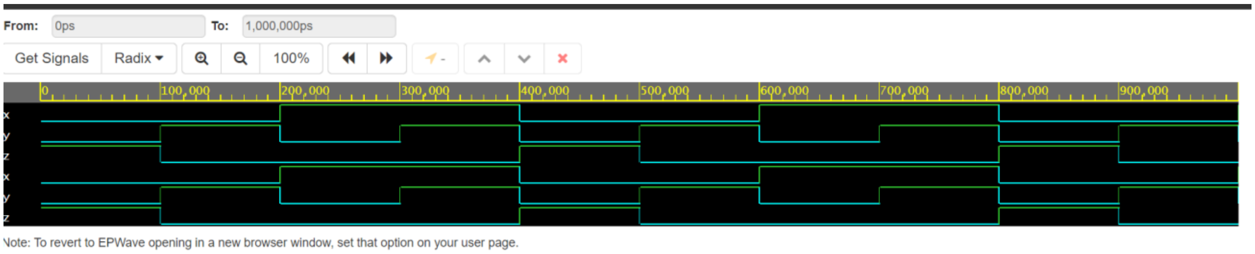
**2.Data flow Model Code:**

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_NOR_gate is
end tb_NOR_gate;

architecture myarch of tb_NOR_gate is
 component NOR_gate is
  port(a: in std_logic;
       b: in std_logic;
       c: out std_logic;
       );
 end component;
 signal a,b: std_logic:='0';
 signal c: std_logic;
```

```
begin
 uut: NOR_gate port map(a,b,c);

stim: process
 begin
  a<='0';b<='0'; wait for 100 ns;
  a<='0';b<='1'; wait for 100 ns;
  a<='1';b<='0'; wait for 100 ns;
  a<='1';b<='1'; wait for 100 ns;
 end process;
end myarch;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity NOR_gate is
 port(a: in std_logic;
      b: in std_logic;
      c: out std_logic;
      );
end NOR_gate;

architecture myarch of NOR_gate is
 begin
  c<=a nor b;
```

```
end myarch;
```

## Output:



From: 0ps To: 1,000,000ps

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

**Objective:** To implement NAND gate

**Software used:**

| Property Name | Xilinx ISE |
|---|---|
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | VHDL |

**Theory:**

The **NAND Gate** represents an AND Gate, followed by a NOT Gate. It negates the logical AND operation, for 2 or more Boolean variables.

**Truth Table:**

| **INPUT** | | **OUTPUT** |
|---|---|---|
| **A** | **B** | **Y** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Behavioral Model:**

Here, the gate is designed, using the **truth table**. It simulates the behavior of the output of the gate, upon taking different Boolean variables as inputs. The **Behavioral Architecture** is used in this case.

## Data flow Model:

Here, the gate is constructed, using the **Dataflow Architecture**. The gate is designed, using it's **Boolean function**.

## Code:

### 1. Behavioral Model Code:

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_NAND_gate is
end tb_NAND_gate;

architecture behavior of tb_NAND_gate is
 component NAND_gate is
  port(x: in std_logic;
       y: in std_logic;
       z: out std_logic;
       );
 end component;
signal x,y: std_logic := '0';
signal z: std_logic;

begin
```

```
 uut: NAND_gate port map(x,y,z);

stim: process
 begin
  x<='0';y<='0'; wait for 100 ns;
  x<='0';y<='1'; wait for 100 ns;
  x<='1';y<='0'; wait for 100 ns;
  x<='1';y<='1'; wait for 100 ns;
 end process;
end;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity NAND_gate is
 port(x: in std_logic;
      y: in std_logic;
      z: out std_logic;
      );
end NAND_gate;

architecture Behavioral of NAND_gate is
 begin
  process(x,y)
  begin


   if(x='1' and y='1') then z<='0';
    else z<='1';
   end if;



 end process;
end Behavioral;
```

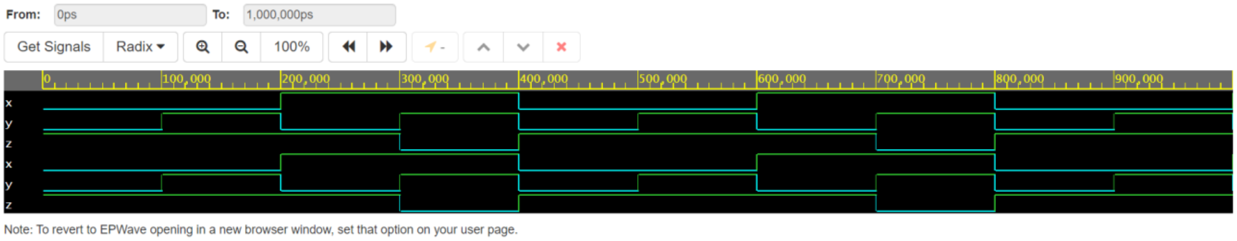
**2. Data flow Model Code:**

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_NAND_gate is
end tb_NAND_gate;

architecture myarch of tb_NAND_gate is
 component NAND_gate is
  port(a: in std_logic;
       b: in std_logic;
       c: out std_logic;
       );
 end component;
 signal a,b: std_logic:='0';
 signal c: std_logic;
begin
 uut: NAND_gate port map(a,b,c);

stim: process
 begin
  a<='0';b<='0'; wait for 100 ns;
  a<='0';b<='1'; wait for 100 ns;
  a<='1';b<='0'; wait for 100 ns;
  a<='1';b<='1'; wait for 100 ns;
 end process;
end myarch;
```

```
-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity NAND_gate is
 port(a: in std_logic;
      b: in std_logic;
      c: out std_logic;
      );
end NAND_gate;

architecture myarch of NAND_gate is
```

```
 begin
  c<=a nand b;
end myarch;
```

## Output:



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

**Objective:** To implement XOR gate

**Software used:**

| Property Name | Xilinx ISE |
|---|---|
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | VHDL |

**Theory:**

This gate is used to implement the logical XOR operation, between 2 or more Boolean variables. It produces an output 1 when the values of the variables are different and 0 when they are the equal.

**Truth Table:**

| **INPUT** | | **OUTPUT** |
|---|---|---|
| **A** | **B** | **Y** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Behavioral Model:**

Here, the gate is designed, using the **truth table**. It simulates the behavior of the output of the gate, upon taking different Boolean variables as inputs. The **Behavioral Architecture** is used in this case.

## Data flow Model:

Here, the gate is constructed, using the **Dataflow Architecture**. The gate is designed, using it’s **Boolean function**.

## Code:

### 1. Behavioral Model Code:

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;


entity tb_XOR_gate is
end tb_XOR_gate;


architecture behavior of tb_XOR_gate is
 component XOR_gate is
  port(x: in std_logic;
       y: in std_logic;
       z: out std_logic;
       );
 end component;
signal x,y: std_logic := '0';
signal z: std_logic;


begin
 uut: XOR_gate port map(x,y,z);
```

```
stim: process
 begin
  x<='0';y<='0'; wait for 100 ns;
  x<='0';y<='1'; wait for 100 ns;
  x<='1';y<='0'; wait for 100 ns;
  x<='1';y<='1'; wait for 100 ns;
 end process;
end;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity XOR_gate is
 port(x: in std_logic;
      y: in std_logic;
      z: out std_logic;
      );
end XOR_gate;

architecture Behavioral of XOR_gate is
 begin
  process(x,y)

   begin
```

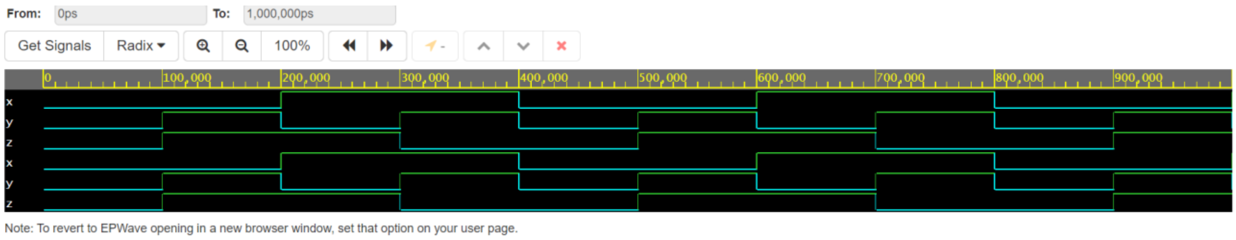
```
    if(x=y) then z<='0';
     else z<='1';
    end if;



 end process;
end Behavioral;
```

**2.Data flow Model Code:**

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_XOR_gate is
end tb_XOR_gate;

architecture myarch of tb_XOR_gate is
 component XOR_gate is
  port(a: in std_logic;
       b: in std_logic;
       c: out std_logic;
       );
 end component;
 signal a,b: std_logic:='0';
 signal c: std_logic;
begin
```

```
 uut: XOR_gate port map(a,b,c);

stim: process
 begin
  a<='0';b<='0'; wait for 100 ns;
  a<='0';b<='1'; wait for 100 ns;
  a<='1';b<='0'; wait for 100 ns;
  a<='1';b<='1'; wait for 100 ns;
 end process;
end myarch;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity XOR_gate is
 port(a: in std_logic;
      b: in std_logic;
      c: out std_logic;
      );
end XOR_gate;

architecture myarch of XOR_gate is
 begin
  c<=a xor b;
end myarch;
```

## Output:



From: 0ps To: 1,000,000ps

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

## Discussion :

**Assignment:2** Implementation of Half Adder, Half Subtractor, Full Adder, Full Subtractor using Xilinx ISE.

**Aim:**

**Software used:**

| Property Name | Xilinx ISE |
|---|---|
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | VHDL |

**Half adder:**

**Theory:**

The **half-adder** is a type of electronic circuit, that is able to add 2 binary digits (of size 1 bit each) and provide the output along with the carry value. The expressions for the sum and carry bits are as follows:-

- **Sum:** S=A XOR B
- **Carry:** C=A AND B

**Truth table:**

| **INPUT** | | **OUTPUT** | |
|---|---|---|---|
| **A** | **B** | **Sum** | **Carry** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

## Behavioral Model:

Here, the circuit is designed using the **truth table**. The **Behavioral Architecture** is used in this case, which simulates the output behaviour of the circuit, considering various inputs.

## Data flow Model:

Here, the circuit is designed using the **Dataflow Architecture**, which takes into account, the **Boolean function**, of the circuit.

## Code:

### 1. Behavioral Model Code:

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_HALF_adder is
end tb_HALF_adder;

architecture behavior of tb_HALF_adder is
 component HALF_adder is
  port(a: in std_logic;
       b: in std_logic;
       s: out std_logic;
       c: out std_logic;
       );
 end component;
```

```
signal x,y: std_logic := '0';
signal s,c: std_logic;

begin
 uut: HALF_adder port map(x,y,s,c);

stim: process
 begin
  x<='0';y<='0'; wait for 100 ns;
  x<='0';y<='1'; wait for 100 ns;
  x<='1';y<='0'; wait for 100 ns;
  x<='1';y<='1'; wait for 100 ns;
 end process;
end;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity HALF_adder is
 port(a: in std_logic;
      b: in std_logic;
      s: out std_logic;
      c: out std_logic;
      );
end HALF_adder;
```

```
architecture Behavioral of HALF_adder is
 begin
  process(a,b)

   begin

    if(a=b) then s<='0';
     else s<='1';
    end if;

    if(a='0' and b='1') then c<='1';
     else c<='0';
    end if;

  end process;
end Behavioral;
```

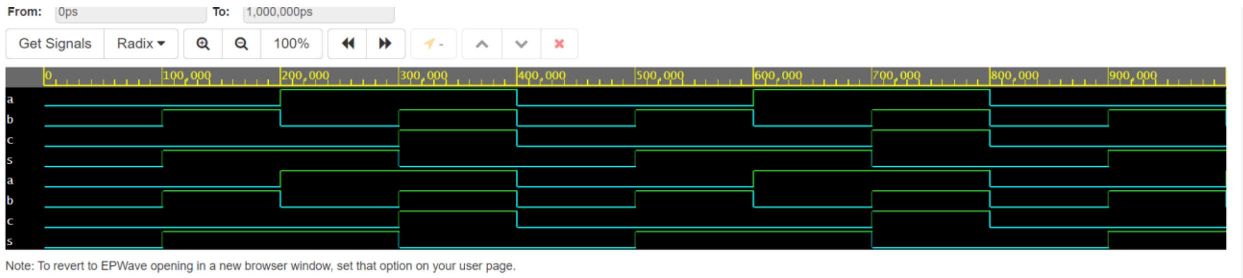
**2.Data flow Model Code:**

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_HALF_adder is
end tb_HALF_adder;
```

```
architecture myarch of tb_HALF_adder is
 component HALF_adder is
  port(
       a: in std_logic;
       b: in std_logic;
       s: out std_logic;
       c: out std_logic;
       );
 end component;
 signal a,b: std_logic :='0';
 signal s,c: std_logic;
begin
 uut: HALF_adder port map(a,b,s,c);

stim: process
 begin
  a<='0';b<='0'; wait for 100 ns;
  a<='0';b<='1'; wait for 100 ns;
  a<='1';b<='0'; wait for 100 ns;
  a<='1';b<='1'; wait for 100 ns;
 end process;
end myarch;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity HALF_adder is
 port(a: in std_logic;
      b: in std_logic;
      s: out std_logic;
      c: out std_logic;
      );
end HALF_adder;

architecture myarch of HALF_adder is
 begin
  s<=a xor b;
  c<=a and b;
end myarch;
```

## Output:



From: 0ps To: 1,000,000ps

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

## Discussion:

## Half Subtractor:

### Theory:

A Half Subtractor is an electronic circuit, which subtracts 2 binary digits (each of size 1 Bit) and is able to provide both difference and borrow, as outputs. The expressions for difference and borrow are as follows:

- **Difference:** D=X XOR Y
- **Borrow:** B=(NOT X) AND Y

### Truth table:

| **INPUT** | | **OUTPUT** | |
|---|---|---|---|
| **A** | **B** | **Difference** | **Bout** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

### Behavioral Model:

Here, the circuit is designed using the **truth table**. The **Behavioral Architecture** is used in this case, which simulates the output behaviour of the circuit, considering various inputs.

### Data flow Model:

Here, the circuit is designed using the **Dataflow Architecture**, which takes into account, the **Boolean function**, of the circuit.

### Code:

### 1. Behavioral Model Code:

```
-- Code your testbench here

library IEEE;

use IEEE.std_logic_1164.all;
```

```
entity tb_HALF_subtractor is
end tb_HALF_subtractor;

architecture behavior of tb_HALF_subtractor is
 component HALF_subtractor is
  port(x: in std_logic;
       y: in std_logic;
       d: out std_logic;
       b: out std_logic;
       );
 end component;
signal x,y: std_logic := '0';
signal d,b: std_logic;

begin
 uut: HALF_subtractor port map(x,y,d,b);

stim: process
 begin
  x<='0';y<='0'; wait for 100 ns;
  x<='0';y<='1'; wait for 100 ns;
  x<='1';y<='0'; wait for 100 ns;
  x<='1';y<='1'; wait for 100 ns;
 end process;
end;
```

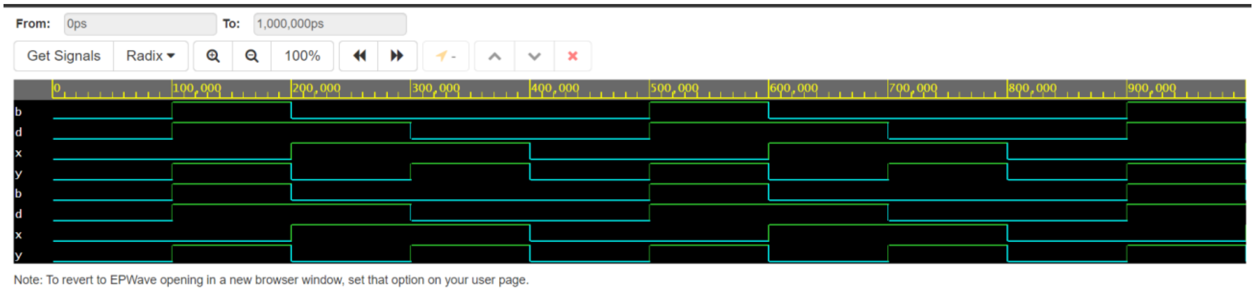
```
-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity HALF_subtractor is
 port(x: in std_logic;
      y: in std_logic;
      d: out std_logic;
      b: out std_logic;
      );
end HALF_subtractor;

architecture Behavioral of HALF_subtractor is
 begin
  process(x,y)

   begin

    if(x=y) then d<='0';
     else d<='1';
    end if;

    if(x='0' and y='1') then b<='1';
     else b<='0';
    end if;
  end process;
end Behavioral;
```

**2. Data flow Model Code:**

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_HALF_subtractor is
end tb_HALF_subtractor;

architecture myarch of tb_HALF_subtractor is
 component HALF_subtractor is
  port(
       x: in std_logic;
       y: in std_logic;
       d: out std_logic;
       b: out std_logic;
       );
 end component;
 signal x,y: std_logic :='0';
 signal d,b: std_logic;
begin
 uut: HALF_subtractor port map(x,y,d,b);

stim: process
```

```
 begin
  x<='0';y<='0'; wait for 100 ns;
  x<='0';y<='1'; wait for 100 ns;
  x<='1';y<='0'; wait for 100 ns;
  x<='1';y<='1'; wait for 100 ns;
 end process;
end myarch;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity HALF_subtractor is
 port(x: in std_logic;
      y: in std_logic;
      d: out std_logic;
      b: out std_logic;
      );
end HALF_subtractor;

architecture myarch of HALF_subtractor is
 begin
  d<=x xor y;
  b<=(not x) and y;
end myarch;
```

## Output:



From: 0ps To: 1,000,000ps

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

## Discussion :

## Full Adder:

### Theory :

A **Full Adder** is an electronic circuit, which adds 3 binary digits (each of size 1 Bit) and is able to provide both the sum and carry, as outputs.

### Truth table:

| **INPUT** | | | **OUTPUT** | |
|---|---|---|---|---|
| **A** | **B** | **Cin** | **Sum** | **Cout** |
| 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 |

### Behavioral Model:

Here, the circuit is designed using the **truth table**. The **Behavioral Architecture** is used in this case, which simulates the output behaviour of the circuit, considering various inputs.

### Data flow Model:

Here, the circuit is designed using the **Dataflow Architecture**, which takes into account, the **Boolean function**, of the circuit.

### Code:

### 1. Behavioral Model Code:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity FULLADDER_BEHAVIORAL_SOURCE is

Port ( A : in  STD_LOGIC_VECTOR (2 downto 0);

       O : out  STD_LOGIC_VECTOR (1 downto 0));

end FULLADDER_BEHAVIORAL_SOURCE;


architecture Behavioral of FULLADDER_BEHAVIORAL_SOURCE is


begin

process (A)

begin


---for SUM

if (A = "001" or A = "010" or A = "100" or A = "111")
then

O(1) <= '1';

---single inverted commas used for assigning to one bit

else

O(1) < = '0';
```

```
end if;



---for CARRY

if (A = "011" or A = "101" or A = "110" or A = "111")
then

O(0) <= '1';

else

O(0) <= '0';

end if;



end process;

end Behavioral;

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity FullAdder_tb is

end entity;

```

```
architecture tb of FullAdder_tb is
component FULLADDER_BEHAVIORAL_SOURCE is
Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
O : out STD_LOGIC_VECTOR (1 downto 0));
end component;

signal A : STD_LOGIC_VECTOR(2 downto 0);
signal O : STD_LOGIC_VECTOR(1 downto 0);

begin

uut: FULLADDER_BEHAVIORAL_SOURCE port map(
A => A, O => O);

stim: process
begin

A <= "000";
wait for 20 ns;
```

```
A <= "001";
wait for 20 ns;

A <= "010";
wait for 20 ns;

A <= "011";
wait for 20 ns;

A <= "100";
wait for 20 ns;

A <= "101";
wait for 20 ns;

A <= "110";
wait for 20 ns;
```

```
A <= "111";

wait for 20 ns;



wait;



end process;

end tb;
```

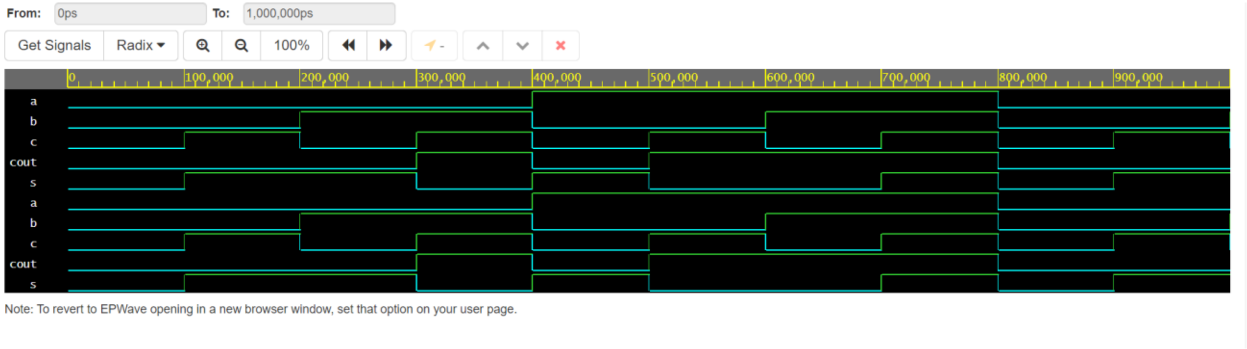
**2. Data flow Model Code:**

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_FULL_adder is
end tb_FULL_adder;

architecture myarch of tb_FULL_adder is
 component FULL_adder is
  port(
       a: in std_logic;
       b: in std_logic;
       c: in std_logic;
```

```
      s: out std_logic;
      cout: out std_logic;
      );
 end component;
 signal a,b,c: std_logic :='0';
 signal s,cout: std_logic;
begin
 uut: FULL_adder port map(a,b,c,s,cout);

stim: process
 begin
  a<='0';b<='0';c<='0'; wait for 100 ns;
  a<='0';b<='0';c<='1'; wait for 100 ns;
  a<='0';b<='1';c<='0'; wait for 100 ns;
  a<='0';b<='1';c<='1'; wait for 100 ns;
  a<='1';b<='0';c<='0'; wait for 100 ns;
  a<='1';b<='0';c<='1'; wait for 100 ns;
  a<='1';b<='1';c<='0'; wait for 100 ns;
  a<='1';b<='1';c<='1'; wait for 100 ns;
 end process;
end myarch;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity FULL_adder is
 port(a: in std_logic;
      b: in std_logic;
      c: in std_logic;
      s: out std_logic;
      cout: out std_logic;
      );
end FULL_adder;


architecture myarch of FULL_adder is
 begin
  s<=a xor b xor c;
  cout<=(a and b) or (b and c) or (c and a);
end myarch;
```

## Output:



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

## Discussion :

**Full Subtractor:**

**Theory :**

A **Full Subtractor** is an electronic circuit, which performs the subtraction of 3 Binary digits (each of size 1 Bit) and is able to provide both the borrow and difference as the outputs.

**Truth table:**

| INPUT | | | OUTPUT | |
|---|---|---|---|---|
| **A** | **B** | **Bin** | **Difference** | **Bout** |
| 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 |

**Behavioral Model:**

Here, the circuit is designed using the **truth table**. The **Behavioral Architecture** is used in this case, which simulates the output behaviour of the circuit, considering various inputs.

**Data flow Model:**

Here, the circuit is designed using the **Dataflow Architecture**, which takes into account, the **Boolean function**, of the circuit.

**Code:**

**1. Behavioral Model Code:**

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity FULLSUBTRACTOR_BEHAVIORAL_SOURCE is

Port ( A : in  STD_LOGIC_VECTOR (2 downto 0);

Y : out  STD_LOGIC_VECTOR (1 downto 0));

end FULLSUBTRACTOR_BEHAVIORAL_SOURCE;


architecture Behavioral of
FULLSUBTRACTOR_BEHAVIORAL_SOURCE is
```

```
begin

process (A)

begin

if (A = "001" or A = "010" or A = "111") then

Y <= "11";

elsif (A = "011") then

Y <= "01";

elsif (A = "100") then
```

```
Y <= "10";


else


Y <= "00";


end if;


end process;


end Behavioral;
```

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

entity full_sub_tb is
end entity;
```

```
architecture tb of full_sub_tb is
component FULLSUBTRACTOR_BEHAVIORAL_SOURCE is
Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
Y : out STD_LOGIC_VECTOR (1 downto 0));
end component;

signal A: STD_LOGIC_VECTOR(2 downto 0);
signal Y: STD_LOGIC_VECTOR(1 downto 0);

begin

uut: FULLSUBTRACTOR_BEHAVIORAL_SOURCE port map(
A => A, Y => Y);

stim:process
begin

A <= "000";
wait for 20 ns;
```

```
A <= "001";

wait for 20 ns;


A <= "010";

wait for 20 ns;


A <= "011";

wait for 20 ns;


A <= "100";

wait for 20 ns;


A <= "101";

wait for 20 ns;


A <= "110";

wait for 20 ns;
```

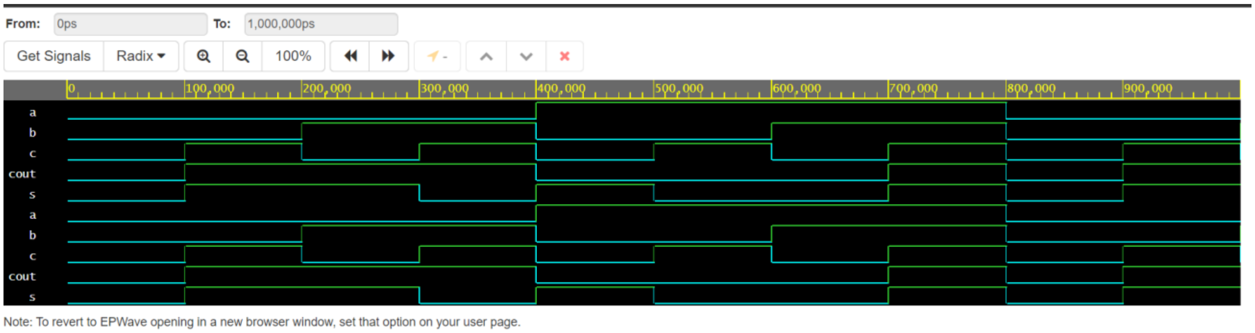
```
A <= "111";

wait for 20 ns;

wait;



end process;



end tb;
```

**2. Data flow Model Code:**

```
-- Code your testbench here

library IEEE;

use IEEE.std_logic_1164.all;



entity tb_FULL_subtractor is

end tb_FULL_subtractor;



architecture myarch of tb_FULL_subtractor is

 component FULL_subtractor is

  port(

       a: in std_logic;

       b: in std_logic;

       c: in std_logic;

       s: out std_logic;
       cout: out std_logic;
       );
 end component;
 signal a,b,c: std_logic :='0';
 signal s,cout: std_logic;
begin
 uut: FULL_subtractor port map(a,b,c,s,cout);

stim: process
 begin
  a<='0';b<='0';c<='0'; wait for 100 ns;
  a<='0';b<='0';c<='1'; wait for 100 ns;
  a<='0';b<='1';c<='0'; wait for 100 ns;
  a<='0';b<='1';c<='1'; wait for 100 ns;
  a<='1';b<='0';c<='0'; wait for 100 ns;
  a<='1';b<='0';c<='1'; wait for 100 ns;
  a<='1';b<='1';c<='0'; wait for 100 ns;
  a<='1';b<='1';c<='1'; wait for 100 ns;
 end process;
end myarch;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

entity FULL_subtractor is
```

```
 port(a: in std_logic;
      b: in std_logic;
      c: in std_logic;
      s: out std_logic;
      cout: out std_logic;
      );
end FULL_subtractor;


architecture myarch of FULL_subtractor is
 begin
  s<=a xor b xor c;
  cout<=(b and c) or ((not a) and ( b or c));
end myarch;
```

## Output:



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

## Discussion:

**Assignment 3: Implementation of 8 bit adder & 8 bit subtractor and 4 bit parallel Adder and Subtractor using XILINX ISE**

**Aim :**

**Software used:**

| Property Name | Xilinx ISE |
|---|---|
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | VHDL |

## 8 bit Adder:

**Theory:**

An **8 Bit Adder** is an electronic circuit, which adds 2 Binary values, each of size 8 Bits.

**Data flow Model:**

Here, the circuit is designed using the **Dataflow Architecture**, which takes into account, the **Boolean function**, of the circuit.

**Code:**

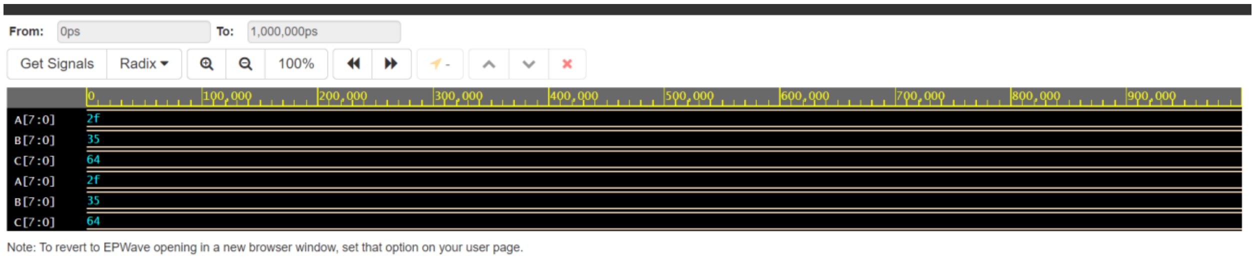
**Data flow model code:**

```
-- Code your testbench here

library IEEE;

use IEEE.std_logic_1164.all;
```

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;


-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--USE ieee.numeric_std.ALL;


ENTITY tb_uadder IS

END tb_uadder;


ARCHITECTURE behavior OF tb_uadder IS


   -- Component Declaration for the Unit Under Test (UUT)


   COMPONENT uadder

   PORT(

        A : IN  std_logic_vector(7 downto 0);

        B : IN  std_logic_vector(7 downto 0);

        C : OUT  std_logic_vector(7 downto 0)

       );

   END COMPONENT;



  --Inputs

  signal A : std_logic_vector(7 downto 0) := (others => '0');

  signal B : std_logic_vector(7 downto 0) := (others => '0');
```

```
        --Outputs
   signal C : std_logic_vector(7 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: uadder PORT MAP (
          A => A,
          B => B,
          C => C
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
                --<clock> <= '0';
                --wait for <clock>_period/2;
                --<clock> <= '1';
                --wait for <clock>_period/2;
   --end process;
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      A<="00101111";B<="00110101"; wait for 100 ns;

      --wait for <clock>_period*10;

      -- insert stimulus here

      wait;
   end process;

END;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

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 using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

entity uadder is
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
           B : in  STD_LOGIC_VECTOR (7 downto 0);
           C : out  STD_LOGIC_VECTOR (7 downto 0));
end uadder;

architecture Dataflow of uadder is

begin
 C <= A + B;



end Dataflow;
```

**Output:**



From: 0ps To: 1,000,000ps

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

## 8 bit Subtractor:

### Theory :

An **8-Bit Subtractor** is an electronic circuit which subtracts 2 Binary digits, each of size 8 Bits.

### Data flow model:

Here, the circuit is designed using the **Dataflow Architecture**, which takes into account, the **Boolean function**, of the circuit.

### Code: (Dataflow model code)

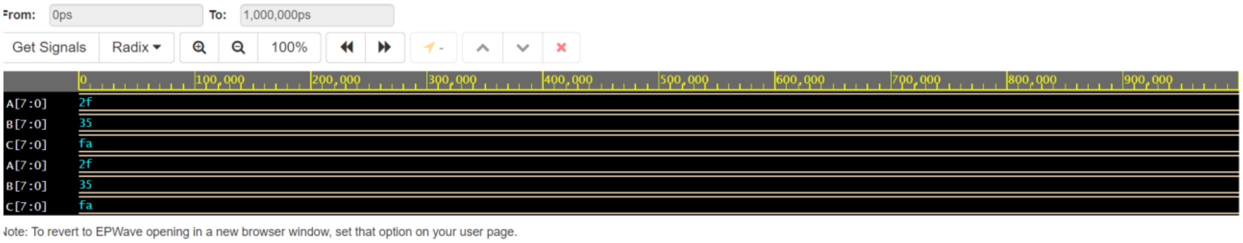
```
-- Code your testbench here

library IEEE;

use IEEE.std_logic_1164.all;


LIBRARY ieee;

USE ieee.std_logic_1164.ALL;


-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values
```

```
--USE ieee.numeric_std.ALL;


ENTITY tb_uadder IS

END tb_uadder;


ARCHITECTURE behavior OF tb_uadder IS


   -- Component Declaration for the Unit Under Test (UUT)


   COMPONENT uadder

   PORT(

        A : IN  std_logic_vector(7 downto 0);

        B : IN  std_logic_vector(7 downto 0);

        C : OUT  std_logic_vector(7 downto 0)

       );

   END COMPONENT;




  --Inputs

  signal A : std_logic_vector(7 downto 0) := (others => '0');

  signal B : std_logic_vector(7 downto 0) := (others => '0');


         --Outputs

  signal C : std_logic_vector(7 downto 0);

  -- No clocks detected in port list. Replace <clock> below with

  -- appropriate port name
```

```
   --constant <clock>_period : time := 10 ns;


BEGIN


            -- Instantiate the Unit Under Test (UUT)
   uut: uadder PORT MAP (
          A => A,
          B => B,
          C => C
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
                  --<clock> <= '0';
                  --wait for <clock>_period/2;
                  --<clock> <= '1';
                  --wait for <clock>_period/2;
   --end process;



   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      A<="00101111";B<="00110101"; wait for 100 ns;

      --wait for <clock>_period*10;

      -- insert stimulus here

      wait;
   end process;

END;

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

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 using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

```
--library UNISIM;

--use UNISIM.VComponents.all;


entity uadder is

    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);

           B : in  STD_LOGIC_VECTOR (7 downto 0);

           C : out  STD_LOGIC_VECTOR (7 downto 0));

end uadder;


architecture Dataflow of uadder is


begin

 C <= A - B;



end Dataflow;
```

## Output:



ꟳrom: 0ps To: 1,000,000ps

```
A[7:0]  2f
B[7:0]  35
C[7:0]  fa
A[7:0]  2f
B[7:0]  35
C[7:0]  fa
```

ote: To revert to EPWave opening in a new browser window, set that option on your user page.

## 4 bit Parallel Adder:

### Theory:

A **4 Bit Parallel Adder** is an electronic circuit, which adds 2 Binary values, each of size 4 Bits and provides the sum as the output.

### Dataflow model:

Here, the circuit is designed using the **Dataflow Architecture**, which takes into account, the **Boolean function**, of the circuit.

### Code:

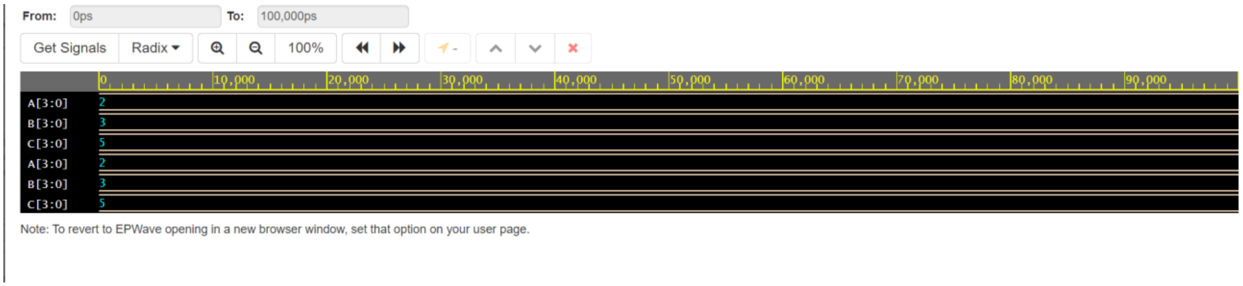
```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY tb_uadder IS
END tb_uadder;

ARCHITECTURE behavior OF tb_uadder IS

   -- Component Declaration for the Unit Under Test (UUT)
```

```
 COMPONENT uadder
 PORT(
      A : IN  std_logic_vector(3 downto 0);
      B : IN  std_logic_vector(3 downto 0);
      C : OUT  std_logic_vector(3 downto 0)
     );
 END COMPONENT;



--Inputs
signal A : std_logic_vector(3 downto 0) := (others => '0');
signal B : std_logic_vector(3 downto 0) := (others => '0');


         --Outputs
signal C : std_logic_vector(3 downto 0);
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name


--constant <clock>_period : time := 10 ns;


BEGIN



         -- Instantiate the Unit Under Test (UUT)
uut: uadder PORT MAP (
       A => A,
      B => B,
      C => C
    );

-- Clock process definitions
--<clock>_process :process
--begin
		--<clock> <= '0';
		--wait for <clock>_period/2;
		--<clock> <= '1';
		--wait for <clock>_period/2;
--end process;



-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   A<="0010";B<="0011"; wait for 100 ns;

   --wait for <clock>_period*10;

   -- insert stimulus here

   wait;
end process;
```

```
END;
-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;

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 using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

entity uadder is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           C : out  STD_LOGIC_VECTOR (3 downto 0));
end uadder;
```

```
architecture Dataflow of uadder is


begin

 C <= A + B;



end Dataflow;
```

**Output:**



From: 0ps To: 100,000ps

```
A[3:0]  2
B[3:0]  3
C[3:0]  5
A[3:0]  2
B[3:0]  3
C[3:0]  5
```

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

## 4 bit Parallel Subtractor:

### Theory:

A **4 Bit Parallel Subtractor** is an electronic circuit, which subtracts 2 Binary values, each of size 4 Bits and provides the difference, as output.

### Dataflow model:

Here, the circuit is designed using the **Dataflow Architecture**, which takes into account, the **Boolean function**, of the circuit.

### Code :

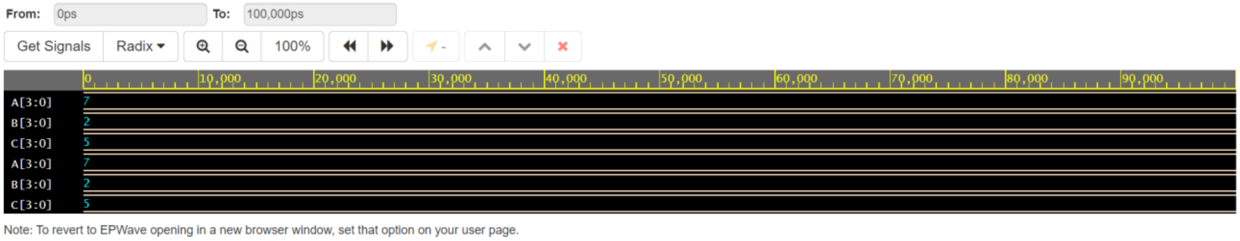
```
-- Code your testbench here

library IEEE;

use IEEE.std_logic_1164.all;


LIBRARY ieee;

USE ieee.std_logic_1164.ALL;


-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--USE ieee.numeric_std.ALL;


ENTITY tb_uadder IS

END tb_uadder;


ARCHITECTURE behavior OF tb_uadder IS


   -- Component Declaration for the Unit Under Test (UUT)
```

```
 COMPONENT uadder
 PORT(
      A : IN  std_logic_vector(3 downto 0);
      B : IN  std_logic_vector(3 downto 0);
      C : OUT  std_logic_vector(3 downto 0)
     );
 END COMPONENT;



--Inputs
signal A : std_logic_vector(3 downto 0) := (others => '0');
signal B : std_logic_vector(3 downto 0) := (others => '0');

          --Outputs
signal C : std_logic_vector(3 downto 0);
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name

--constant <clock>_period : time := 10 ns;

BEGIN

          -- Instantiate the Unit Under Test (UUT)
uut: uadder PORT MAP (
        A => A,
        B => B,
        C => C
     );

-- Clock process definitions
--<clock>_process :process
--begin
        --<clock> <= '0';
        --wait for <clock>_period/2;
        --<clock> <= '1';
        --wait for <clock>_period/2;
--end process;



-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
  A<="0111";B<="0010"; wait for 100 ns;
  --wait for <clock>_period*10;

  -- insert stimulus here

  wait;
end process;
```

```
END;


-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;


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 using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;


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


entity uadder is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           C : out  STD_LOGIC_VECTOR (3 downto 0));
end uadder;
```

```
architecture Dataflow of uadder is


begin
 C <= A - B;
end Dataflow;
```

## Output :



From: 0ps To: 100,000ps

A[3:0] 7
B[3:0] 2
C[3:0] 5
A[3:0] 7
B[3:0] 2
C[3:0] 5

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

## Discussion :

**Assignment 4: Implementation of binary to Gray converter and Gray to Binary converter using XILINX ISE**

**Aim:**

**Software used:**

| | |
|---|---|
| Property Name | |
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | |

**Gray to Binary:**

**Theory :**

**Truth table: (Gray to Binary)**

| Gray | | | | Binary | | | |
|---|---|---|---|---|---|---|---|
| G3 | G2 | G1 | G0 | B3 | B2 | B1 | B0 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

## Dataflow Model:

## Code:

## Behavioral Model code:

## Dataflow Model Code:

**<u>Output :</u>**

**<u>Binary to Gray:</u>**

**<u>Theory :</u>**

**<u>Truth table:</u> (Binary to Gray)**

| Binary | | | | Gray | | | |
|---|---|---|---|---|---|---|---|
| **B3** | **B2** | **B1** | **B0** | **G3** | **G2** | **G1** | **G0** |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Dataflow Model:**

**Code:**

**Behavioral Model code:**

**Dataflow Model Code:**

**Output :**

**Discussion:**

**Assignment 5: Implementation of 4bit comparator, 2:4 Decoder,3:8 Decoder, 8bit Multiplier using XILINX ISE**

**Aim:**

**Software used:**

| | |
|---|---|
| Property Name | |
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | |

**4 bit Comparator:**

**Theory:**

**Code:**

## Behavioral Model Code:

## Output:

## 2:4 Decoder:

## Truth table:

| I1 | I0 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Dataflow Model:

**Code:**

**Output :**

**3:8 Decoder:**

**Truth table:**

| I2 | I1 | I0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

## Dataflow Model:

## Code:

**Output :**

**8 Bit Multiplier:**

**Code:**

**Output:**

**Discussion:**

**Assignment 6: Implementation of 2:4 Decoder (using case), 4:2 Encoder (Dataflow & using case), 8:3 Encoder (using loop)**

**Aim:**

**Software used:**

| | |
|---|---|
| Property Name | |
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | |

**2:4 Decoder:**

**Theory:**

## Truth table:

| I1 | I0 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## Behavioral Model: (using case)

## Output :

## 4:2 Encoder:

### Theory :

### Truth Table:

| I3 | I2 | I1 | I0 | E1 | E0 |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

### Dataflow Model:

### Dataflow Code:

**Output :**

**Behavioral Code:**

## Output :

## 8:3 Encoder:

## Truth Table:

| I7 | I6 | I5 | I4 | I3 | I2 | I1 | I0 | E2 | E1 | E0 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

## Behavioral Code:

## Output :

## Discussion :

## **Assignment 7:** Implementation of 2:1 MUX(Dataflow), 4:1 MUX (Dataflow, if-else and using Case) and 1:4 Demux (Using Dataflow and case) using XILINX ISE.

**Software used:**

| | |
|---|---|
| Property Name | |
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | |

## Theory :

## 2:1 MUX:

## Truth table:

| INPUT |  |  | OUTPUT |
|---|---|---|---|
| S | I1 | I0 | M |
|  |  |  |  |
|  |  |  |  |

## Dataflow Model:

## Code:

## Dataflow Model:

**Output :**

**4:1 MUX:**

**Truth Table:**

| INPUT | | | | | | OUTPUT |
|---|---|---|---|---|---|---|
| S1 | S0 | I3 | I2 | I1 | I0 | M |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

**Dataflow Model:**

**Code:**

**Dataflow Model Code:**

**Output :**

**Behavioral Model code: (Using CASE)**

**Behavioral Model code: (Using if-else)**

**Output:**

## 1:4 De mux:

## Theory :

## Truth Table:

| Data Input | Select Input | | Output | | | |
|---|---|---|---|---|---|---|
| D | S1 | S0 | Y3 | Y2 | Y1 | Y0 |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

## Dataflow Model:

## Dataflow Code:

## Behavioral code (using Case):

**Output:**

**Discussion:**

**Assignment 8:** **Implementation of Full Adder using Half Adder (using Structural Method), 4 bit Parallel Adder (using Structural Method).**

**Aim :**

**Software used:**

| | |
|---|---|
| Property Name | |
| Device family | |
| Device | |
| Package | |
| Speed | |

| | |
|---|---|
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | |

**Full Adder:**

**Theory:**

**Truth Table:**

| INPUT | | | OUTPUT | |
|---|---|---|---|---|
| **A** | **B** | **Cin** | **S** | **Cout** |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

**Structural Model**

**Code:**

**Output:**

## 4 Bit Parallel Adder:

## Theory:

## Structural Model:

## Code:

**<u>Output:</u>**

**<u>Discussion:</u>**

**<u>Assignment 9:</u> Implementation of 2:1 MUX (using Structural Method) & 4:1 MUX using 2:1 MUX (using Structural Method) in XILINX ISE**

**<u>Aim :</u>**

**<u>Software used:</u>**

| | |
|---|---|
| Property Name | |
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | |

**Theory :**

**2:1 MUX**

**Truth table:**

| **INPUT** | | | **OUTPUT** |
|---|---|---|---|
| **S** | **I1** | **I0** | **Y** |
| | | | |
| | | | |
| | | | |
| | | | |

**Structural Model:**

**Code : (Structural method)**

**Output :**

**4:1 MUX**

**Truth table:**

| S1 | S0 | Y |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

**Structural Model:**

**Code : (Structural)**

**Output :**

**Discussion :**

**Assignment 10: Implementation of SR flip flop, JK flip flop, D flip flop, T flip flop (using if –else) in XILINX ISE**

**Aim :**

## Software used:

| | |
|---|---|
| Property Name | |
| Device family | |
| Device | |
| Package | |
| Speed | |
| Top-level source type | |
| Synthesis Tool | |
| Simulator | |
| Preferred Language | |

## Theory :

## SR flip flop:

## Truth table:

| **CLK** | **S** | **R** | **Q** | **Q’** |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## Dataflow model:

**Code :**

**Output :**

**JK flip flop:**

## Truth table:

| CLK | J | K | Q | Q’ |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## Dataflow model:

## Code :

## Output :

**D flip flop:**

**Truth table:**

| CLK | D | Q | Q’ |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Dataflow model:**

**Code :**

**Output :**

## T flip flop:

## Truth table:

| CLK | T | Q | Q’ |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## Dataflow model:

## Code :

## Output :

## Discussion: