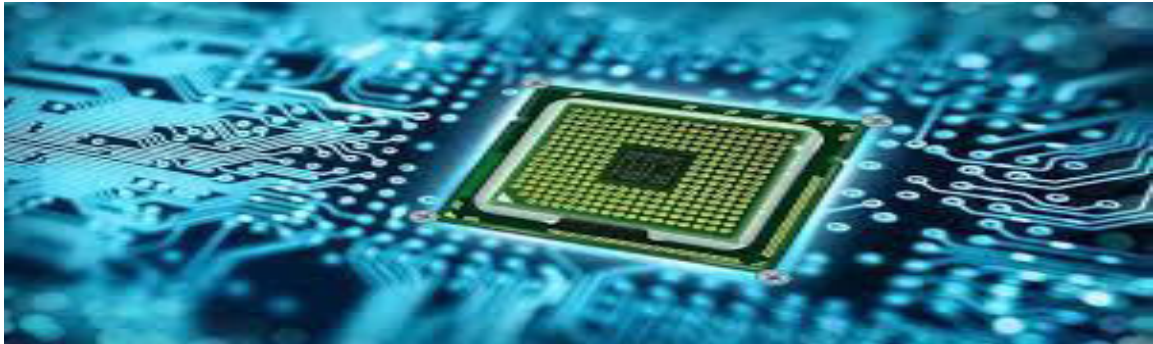


4-bit Arithmetic Logic Unit



Submitted by:

Soham Maiti

Nilak Patra

Aniruddha Bag

Supriyo Seni

Nirmalya Sasmal

In partial fulfilment for the degree of

B. Tech in

Electronics and Communication Engineering

at

Ardent Computech Pvt. Ltd.



CERTIFICATE

This is Certified that this project work was carried out
under my supervision

“DESIGN OF 4-BIT ALU” is the bonafide work of

SOHAM MAITI:

SUPRIYO SENI:

ANIRUDDHA BAG:

NIRMALYA SASMAL:

NILAK PATRA:

.....

SIGNATURE

Name:

PROJECT MANAGER

.....

SIGNATURE

.....

Ardent Original Seal

Acknowledgement

*We would like to express our special thanks and
Gratitude to our project instructor” Sri. **Souvik Sarkar**” for
his immense support and guidance in completing our
project.*

*We are also grateful to all the faculty members of Ardent
and our colleagues for their support.*

INDEX

Sl. No.	CONTENTS	PAGE NO.
1	Objective	05
2	Abstract	05
3	Introduction	06-09
4	Tool and Environment Used	09
5	Design	10
6	Block Diagram	10
7	Input and Output of ALU	11
8	Functional Table	11
9	RTL Design view	12-13
10	Simulation and Synthesis	13-38
11	Overall Implementation	39-45
12	Result	46
13	Conclusion and Future Scope	47
14	References	48

1.OBJECTIVE

To Design a 4-bit Arithmetic logic Unit using VHDL code, which can perform 16 different operations.

Logical Operations: OR, AND, NAND, NOR, XOR, XNOR, SHIFT, NOT

Arithmetic Operations: ADDITION, SUBSTRUCTION, MULTIPLICATION, ROTATION

2.ABSTRACT

In this modern day technology, there is an immense need of developing suitable data communication for embedded systems. An arithmetic unit, or ALU, enables computers to perform mathematical operations on binary numbers. They can be found at the heart of every digital computer and are one of the most important parts of a CPU (Central Processing Unit).

VHDL (VHSIC hardware description language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. It became IEEE standard 1076 in 1987. It was updated and is known today as "IEEE standard 1076 1993".

This report proposes a technique to design and implement a 4-bit ALU which is digital circuit to perform arithmetic and logical operations on XILINX ISE 9.2i using VHDL.

3.INTRODUCTION

3.1 ARITHMATIC LOGIC UNIT (ALU):

An **arithmetic logic unit (ALU)** is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the **central processing unit (CPU)** of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A **register** is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

An ALU performs basic arithmetic and logic operations. Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR.

All information in a computer is stored and manipulated in the form of **binary numbers**, i.e. 0 and 1. **Transistor** switches are used to manipulate binary numbers since there are only two possible states of a switch: open or closed. An open transistor, through which there is no current, represents a 0. A closed transistor, through which there is a current, represents a 1.

Operations can be accomplished by connecting multiple transistors. One transistor can be used to control a second one - in effect, turning the transistor switch on or off depending on the state of the second transistor. This is referred to as a **gate** because the arrangement can be used to allow or stop a current.

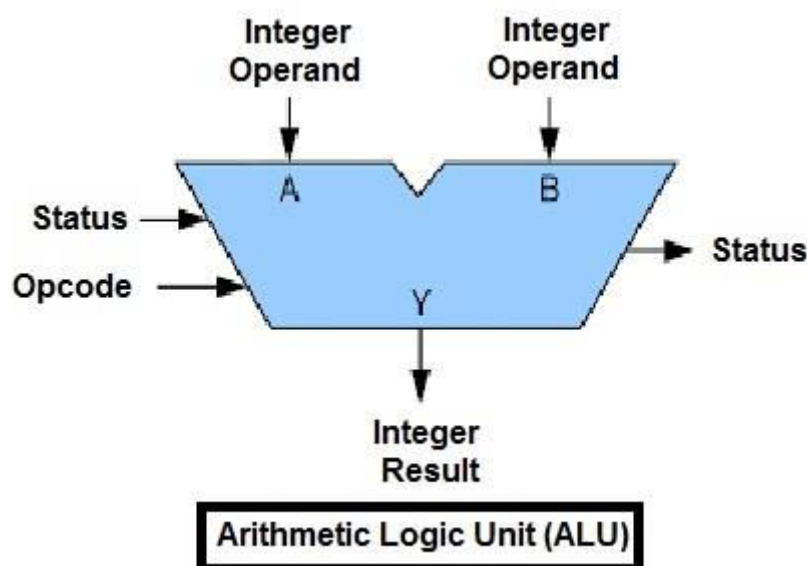


Figure: Symbol of ALU

Different operation as carried out by ALU can be categorized as follows –

- logical operations – These include operations like AND, OR, NOT, XOR, NOR, NAND, etc.
- Bit-Shifting Operations – This pertains to shifting the positions of the bits by a certain number of places either towards the right or left, which is considered a multiplication or division operations.
- Arithmetic operations – This refers to bit addition and subtraction. Although multiplication and division are sometimes used, these operations are more expensive to make. Multiplication and subtraction can also be done by repetitive additions and subtractions respectively.

3.2 HDL:

- It is high level abstraction languages oriented to describe/model Hardware.
- Understandable by computers and humans.
- HDLs are used for:
 - First of all to model hardware independently of its technological implementation.
 - To simulate HDL models against specifications.
 - To synthesize HDL models into different implementation technologies.
- Main HDLs :
 - VHDL: VHSIC Hardware Description Language (Std. IEEE 1076 since 1987).
 - Verilog: Cadence simulation language (1985) moved to public HDL.
 - C/C++/SystemC/SystemVerilog: higher abstraction languages.

3.2.1 VHDL:

VHDL (VHSIC (Very High Speed Integrated Circuits) hardware description language) is commonly used as a design-entry language for field-programmable gate arrays and application-specific integrated circuits in electronic design automation of digital circuits. VHDL was originally developed at the behest of the US Department of Defense in order to document the behavior of the ASICs that supplier companies were including in equipment. That is to say, VHDL was developed as an alternative to huge, complex manuals which were subject to implementation-specific details.

.It became IEEE standard 1076 in 1987. It was updated in 1993 and is known today as "IEEE standard 1076 1993.

VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. This collection of simulation models is commonly called a test bench.

There are two aspects to modelling hardware that any hardware description language facilitates; true abstract behavior and hardware structure. A multitude of language or user defined data types can be used. This will make models easier to write, clearer to read and avoid unnecessary conversion functions that can clutter the code. One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE) to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test bench. The key advantage of VHDL, when used for systems design, is that it allows the behavior of the required system to be described (modelled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). Another benefit is that VHDL allows the description of a concurrent system. VHDL is a dataflow language, unlike procedural computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time. VHDL project is multipurpose and portable. Being created for one element base, a computing device project can be ported on another element base, for example VLSI with various technologies.

3.3 INTRODUCTION TO XILINX (INTRODUCTION TO FPGA TECHNOLOGY):

Xilinx (an Electronics Company) is the world's largest supplier of programmable logic devices and the inventor of the field programmable gate array (FPGA). Today Xilinx is the worldwide leader of complete programmable logic solutions.

Xilinx has two main FPGA families – the high performance video Virtex series and the high volume Spartan series. Each model series has been released in multiple generations since its launch. The latest Virtex-6 and Spartan-6 FPGA families are said to consume 50 % less power , cost 20 % less , and have up to twice the logic capacity of previous generations of FPGAs. The Spartan series targets applications with low power footprint, extremes cost sensitivity and high volume e.g. displays, set top boxes wireless routers and other applications.

The ISE Design Suite is the central electronic design automation (EDA) product family sold by Xilinx .The ISE Design Suite features include design entry and synthesis supporting Verilog or VHDL, place and route (PAR), completed verification and Chipscope pro tools , and creation of the bit files that are used to configure the chip .

Xilinx designs, develops and markets programmable logic products including integrated circuits (ICs), software design tools and services . Xilinx sells both FPGAs and CPLDs (Complex Programmable Logic Devices) for electronic equipment manufacturers.

3.3.1 XILINX ISE (The Integrated Software Environment):

Xilinx ISE (Integrated Synthesis Environment) is a discontinued software tool from Xilinx for synthesis and analysis of HDL designs, which primarily targets development of embedded firmware for Xilinx FPGA and CPLD integrated circuit (IC) product families. Use of the last released edition from October 2013 continues for in-system programming of legacy hardware designs containing older FPGAs and CPLDs otherwise orphaned by the replacement design tool, Vivado Design Suite.

ISE enables the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and ChipScope Pro.^[4] The Xilinx ISE is primarily used for circuit synthesis and design, while ISIM or the ModelSim logic simulator is used for system-level testing.

As commonly practiced in the commercial electronic design automation sector, Xilinx ISE is tightly-coupled to the architecture of Xilinx's own chips (the internals of which are highly proprietary) and cannot be used with FPGA products from other vendors. Given the highly proprietary nature of the Xilinx hardware product lines, it is rarely possible to use open source alternatives to tooling provided directly from Xilinx, although as of 2020, some exploratory attempts are being made.

4.TOOLS AND ENVIRONMRNT USED:

Minimum Hardware Requirement:

- Computer : IBM or compatible
- Hard disk : 20 GB or higher
- Processor : PENTIUM– IV 2 GHz or above
- Ram : 512 Mb and above

Minimum Software Requirement:

- Operating System : Windows 10
- Development Software : Xilinx ISE 14.7

5.DESIGN:

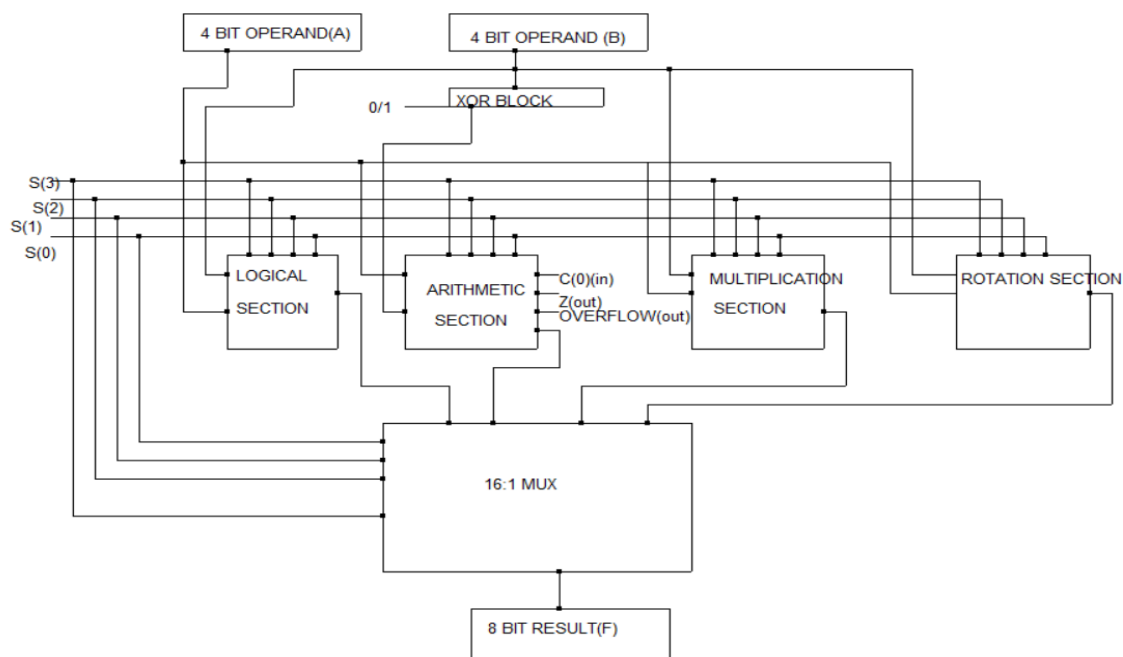
When designing the ALU we will follow the principle "Divide and Conquer" in order to use a **modular design** that consists of smaller, more manageable blocks, some of which can be re-used. In the design of 4-bit ALU we add 4 bit OR, AND, NOR, NOT, XOR, NAND, XNOR, ADDER, SUBTRACTOR, ROTATOR, MULTIPLIER, AND SHIFTING.

The approach used here is to split the ALU into four modules, one Logic, one Arithmetic module, one Multiplication module and one Rotation module. Designing each element of each module separately will be easier than designing a bit-slice as one unit. Then all the 16 elements are joined using a 16:1 MUX. A possible block diagram of the ALU is shown in Figure.

Here the logic unit is made up of a 16:1 MUX. Each input to the logic unit is the output from a separate OR, AND, NOR, NOT, XOR, NAND, XNOR (Logical), and ADDER, SUBTRACTOR, MULTIPLICATION, ROTATION AND SHIFTING.

The desired operation can be performed by selecting the suitable, corresponding select lines (from 0 to 15), to obtain the 4-bit output.

6.BLOCK DIAGRAM OF ALU:



7.INPUT/OUTPUT of ALU:

Input of ALU: Two 4-bit operands A & B

Operation Selection Line: S(4 bit)

Result selection Line: S =1011(Arithmetic operation)

=1101(Arithmetic operation)

=0010(Multiplication operation)

=0100(Rotation operation)

=Else(Logical operations)

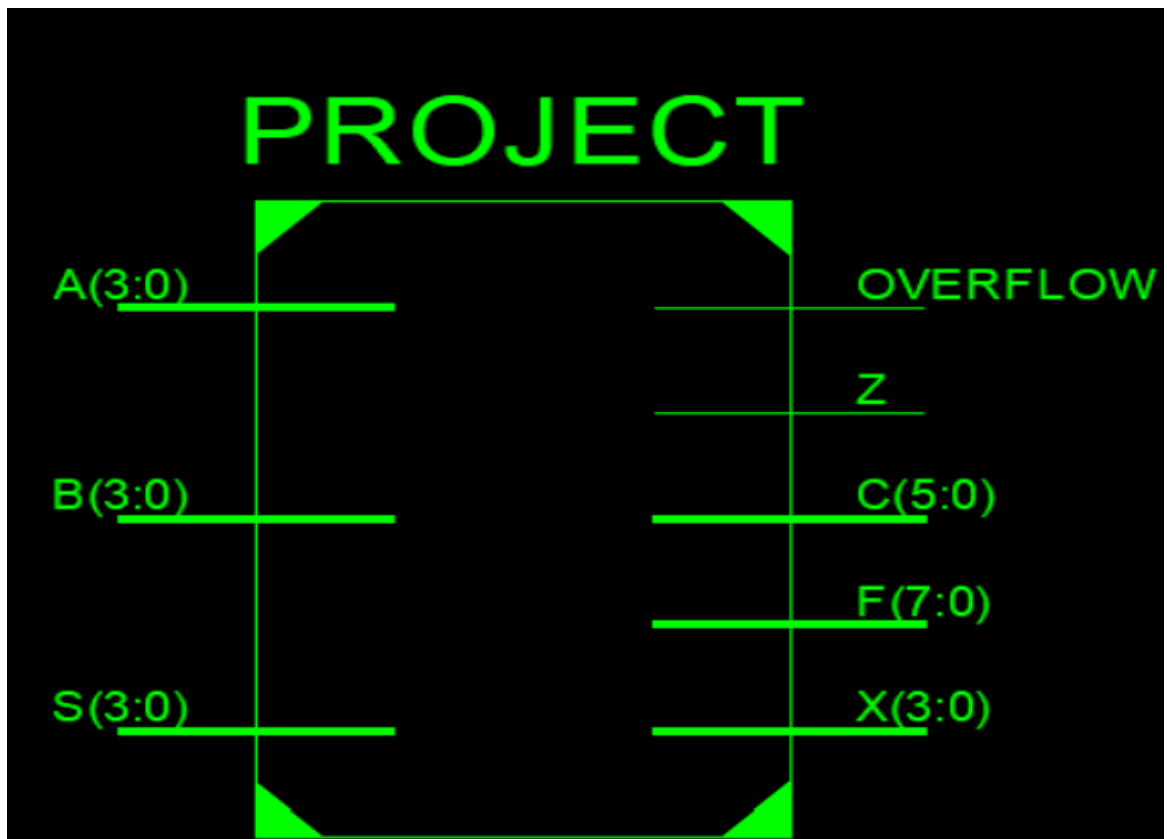
Output of ALU: One 8-bit result operand.

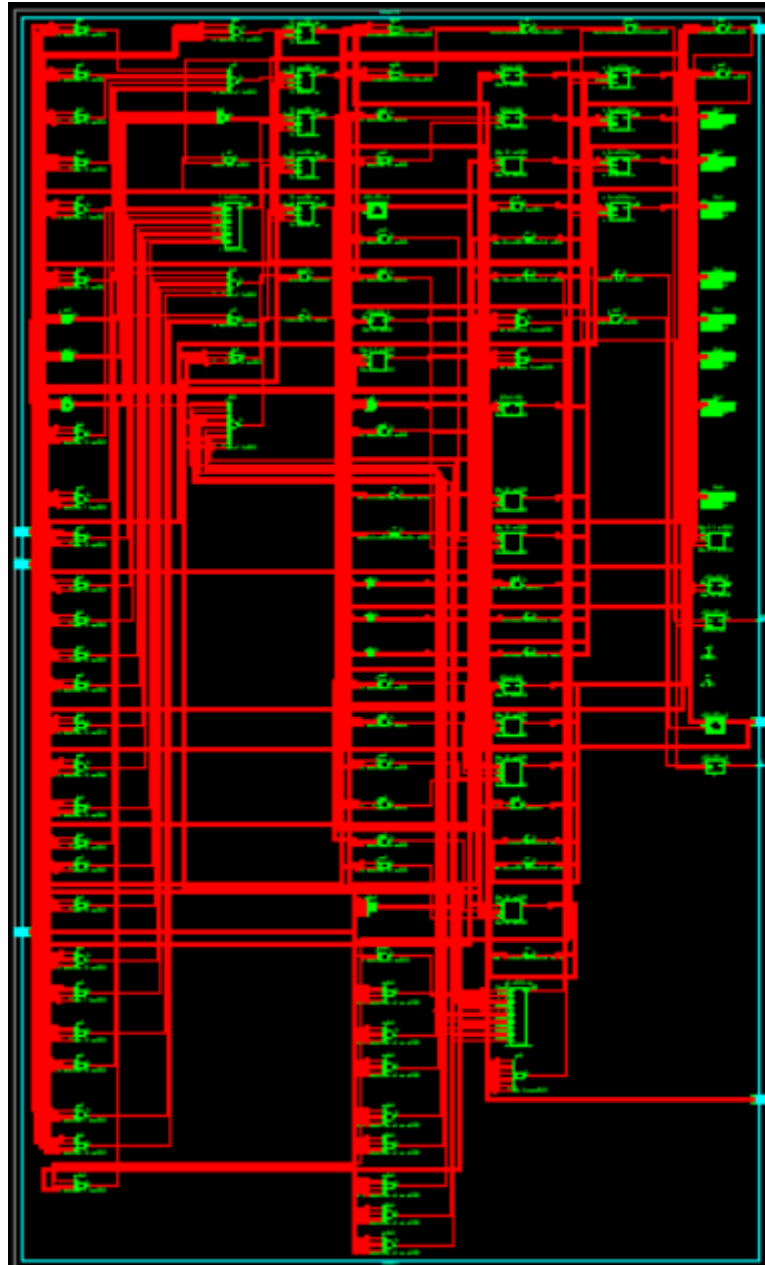
Z (zero), Overflow (overflow check)

8.FUNCTIONAL TABLE OF ALU:

OPCODE	S(3)	S(2)	S(1)	S(0)	OPERATION	FUNCTION
0000	0	0	0	0	F=0	RESET
0001	0	0	0	1	$F=(A \mid B)'$	NOR Operation
0010	0	0	1	0	$F=A*B$	MULTIPLICATION Operation
0011	0	0	1	1	$F=B'$	B Complement
0100	0	1	0	0	F=ROTATE A	ROTATION Operation
0101	0	1	0	1	$F=A'$	A Complement
0110	0	1	1	0	$F=(A \text{ XOR } B)$	XOR Operation
0111	0	1	1	1	$F=(A \& B)'$	NAND Operation
1000	1	0	0	0	$F=(A\&B)$	AND Operation
1001	1	0	0	1	$F=(A \text{ XOR } B)'$	XNOR Operation
1010	1	0	1	0	$F=A*2$	Left Shift A by 1 bit
1011	1	0	1	1	$F=(A+B)$	ADDITION Operation
1100	1	1	0	0	$F=B*2$	Left Shift B by 1 bit
1101	1	1	0	1	$F=(A+B'+1)$	SUBTRUCTION Operation
1110	1	1	1	0	$F=(A \mid B)$	OR Operation
1111	1	1	1	1	F=1	SET

9.RTL DESIGN VIEW:





10.SIMULATION AND SYSNTHESIS ON Xilinx ISE:

In 4 bit ALU logical section can perform 10 logical operation and RESET and SET operation. Logical operation which can be performed are OR Operation, NOR Operation, AND Operation, NAND Operation, XOR Operation, XNOR Operation, A Compliment, B Compliment, LEFT SHIFT A by 1 bit, LEFT SHIFT B by 1 bit. Arithmetic section can work ADDITION and SUBSTRUCTION. In Computer negative number is represented in 2's compliment form. So first 1's compliment is needed and then we have to add 1 to LSB. As we know $(X \text{ XOR } 1) = X'$ & $(X \text{ XOR } 0) = X$ so we use this concept. So, in ADDITION we do the XOR operation with second operand with 0 and give initial carry=0. In, SUBSTRUCTION operation

we do XOR operation with second operand with 1 and give initial carry 1. Multiplication section can do multiplication only and Rotation section can do rotate operation.

1.RESET:

```
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;

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

--library UNISIM;

--use UNISIM.VComponents.all;

entity RESET is

    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          B : in  STD_LOGIC_VECTOR (3 downto 0);
          F : out STD_LOGIC_VECTOR (7 downto 0));

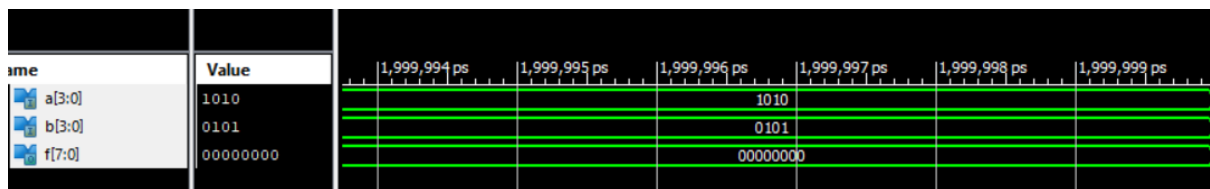
end RESET;

architecture Behavioral of RESET is

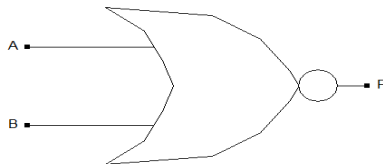
begin

    F(7)<='0';
    F(6)<='0';
    F(5)<='0';
    F(4)<='0';
    F(3)<='0';
    F(2)<='0';
    F(1)<='0';
    F(0)<='0';

end Behavioral;
```



2. NOR operation:



Truth Table

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

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;

-- Uncomment the following library declaration if instantiating

-- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity NOR is

Port (A : in STD_LOGIC_VECTOR (3 downto 0);

B : in STD_LOGIC_VECTOR (3 downto 0);

F : out STD_LOGIC_VECTOR (7 downto 0));

end NOR;

architecture Behavioral of NOR is

```

begin

F(7)<='0';

F(6)<='0';

F(5)<='0';

F(4)<='0';

F(3)<=(A(3) NOR B(3));

F(2)<=(A(2) NOR B(2));

F(1)<=(A(1) NOR B(1));

F(0)<=(A(0) NOR B(0));

end Behavioral;

```

Name	Value	2,999,994 ps	2,999,995 ps	2,999,996 ps	2,999,997 ps	2,999,998 ps	2,999,999 ps
a[3:0]	0110			0110			
b[3:0]	0101			0101			
f[7:0]	00001000			00001000			

3. Multilication:

```

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;

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

entity MULTIPLICATION is

    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          B : in  STD_LOGIC_VECTOR (3 downto 0);
          C : inout STD_LOGIC_VECTOR (5 downto 0);
          F : out  STD_LOGIC_VECTOR (7 downto 0));

```


end MULTIPLICATION;

architecture Behavioral of MULTIPLICATION is

begin

C(0) <=(A(0) AND B(1)) AND (A(1) AND B(0));

C(1) <=((A(2) AND B(0)) AND (A(1) AND B(1))) OR ((A(2) AND B(0)) AND (A(0) AND B(2))) OR ((A(1) AND B(1)) AND (A(0) AND B(2))) OR ((A(2) AND B(0)) AND C(0)) OR ((A(1) AND B(1)) AND C(0)) OR ((A(0) AND B(2)) AND C(0));

C(2) <=((A(3) AND B(0)) AND (A(2) AND B(1))) OR ((A(3) AND B(0)) AND (A(1) AND B(2))) OR ((A(3) AND B(0)) AND (A(0) AND B(3))) OR ((A(3) AND B(0)) AND C(1)) OR ((A(2) AND B(1)) AND C(1)) OR ((A(1) AND B(2)) AND C(1)) OR ((A(0) AND B(3)) AND C(1)) OR ((A(2) AND B(1)) AND (A(1) AND B(2))) OR ((A(2) AND B(1)) AND (A(0) AND B(3))) OR ((A(1) AND B(2)) AND (A(0) AND B(3)));

C(3) <=((A(3) AND B(1)) AND (A(2) AND B(2))) OR ((A(3) AND B(1)) AND (A(1) AND B(3))) OR ((A(2) AND B(2)) AND (A(1) AND B(3))) OR ((A(3) AND B(1)) AND C(2)) OR ((A(1) AND B(3)) AND C(2)) OR ((A(2) AND B(2)) AND C(2));

C(4) <=((A(3) AND B(2)) AND (A(2) AND B(3))) OR ((A(3) AND B(2)) AND C(3)) OR ((A(2) AND B(3)) AND C(3));

C(5) <=((A(3) AND B(3)) AND C(3));

F(0) <=A(0) AND B(0);

F(1) <=(A(1) AND B(0)) XOR (A(0) AND B(1));

F(2) <=(A(2) AND B(0)) XOR (A(1) AND B(1)) XOR (A(0) AND B(2)) XOR C(0);

F(3) <=(A(3) AND B(0)) XOR (A(2) AND B(1)) XOR (A(1) AND B(2)) XOR (A(0) AND B(3)) XOR C(1);

F(4) <=(A(3) AND B(1)) XOR (A(2) AND B(2)) XOR (A(1) AND B(3)) XOR C(2);

F(5) <=(A(3) AND B(2)) XOR (A(2) AND B(3)) XOR C(3);

F(6) <=(A(3) AND B(3)) XOR C(4);

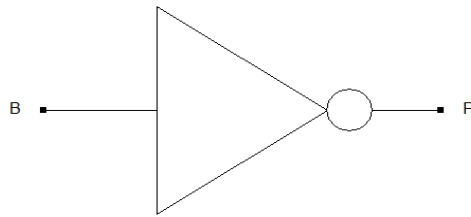
F(7) <=C(5);

end Behavioral;

Name	Value	1,999,994 ps	1,999,995 ps	1,999,996 ps	1,999,997 ps	1,999,998 ps	1,999,999 ps
a[3:0]	0110			0110			
b[3:0]	0101			0101			
c[5:0]	000000			000000			
f[7:0]	00011110			00011110			

4. B's complement:

$$F=B'$$



```
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;
```

```
-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx primitives in this code.
```

```
--library UNISIM;
```

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

```
entity COMPLIMENT_B is
```

```
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
          B : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
          F : out STD_LOGIC_VECTOR (7 downto 0));
```

```
end COMPLIMENT_B;
```

```
architecture Behavioral of COMPLIMENT_B is
```

```
begin
```

```
F(7)<='0';
```

```
F(6)<='0';
```

```
F(5)<='0';
```

```
F(4)<='0';
```

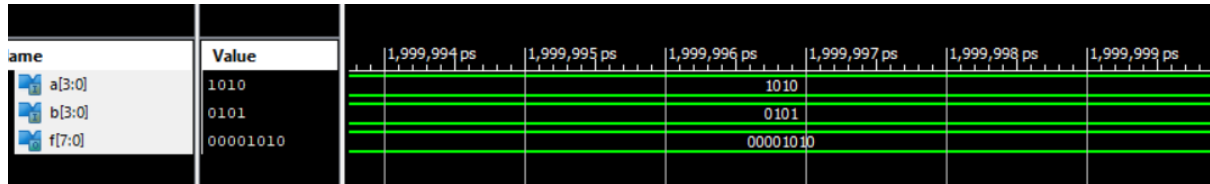
```
F(3)<=(NOT B(3));
```

```
F(2)<=(NOT B(2));
```

```
F(1)<=(NOT B(1));
```

```
F(0)<=(NOT B(0));
```

```
end Behavioral;
```



5:Rotation operation:

```
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;
```

```
-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx primitives in this code.
```

```
--library UNISIM;
```

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

```
entity ROTATION is
```

```
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
```

```
          B : in STD_LOGIC_VECTOR (3 downto 0);
```

```
          F : out STD_LOGIC_VECTOR (7 downto 0));
```

```
end ROTATION;
```

```
architecture Behavioral of ROTATION is
```

```
begin
```

```
F(7)<='0';
```

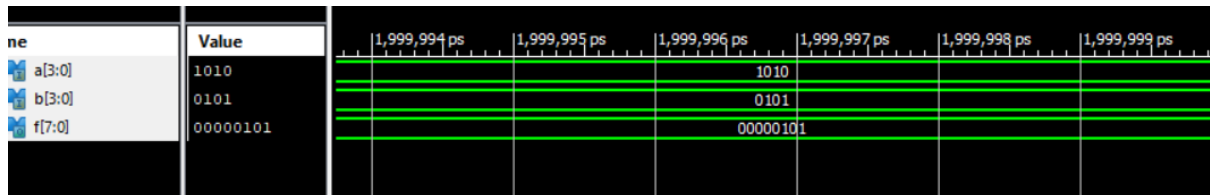
```
F(6)<='0';
```

```
F(5)<='0';
```

```

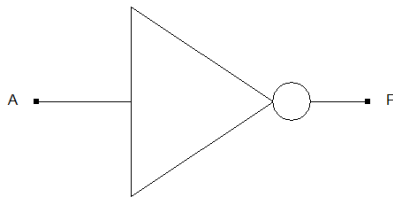
F(4)<='0';
F(3)<=(A(0));
F(2)<=(A(1));
F(1)<=(A(2));
F(0)<=(A(3));
end Behavioral;

```



6. A's complement:

$F=A'$



```

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;

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

entity COMPLIMENT_A is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);

```

```

        B : in STD_LOGIC_VECTOR (3 downto 0);

        F : out STD_LOGIC_VECTOR (7 downto 0));

end COMPLIMENT_A;

architecture Behavioral of COMPLIMENT_A is

begin

F(7)<='0';

F(6)<='0';

F(5)<='0';

F(4)<='0';

F(3)<=(NOT A(3));

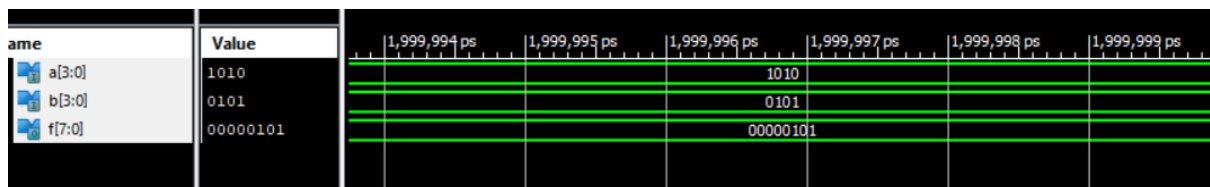
F(2)<=(NOT A(2));

F(1)<=(NOT A(1));

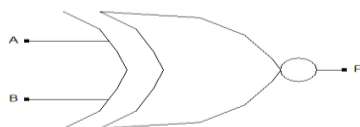
F(0)<=(NOT A(0));

end Behavioral;

```



7.XOR operation:



Truth table

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

```
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;

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

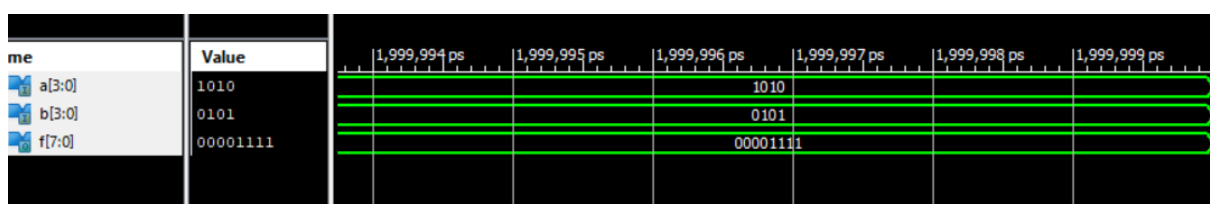
--library UNISIM;

--use UNISIM.VComponents.all;

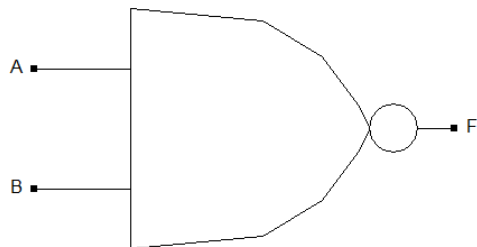
entity XOR is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          F : out STD_LOGIC_VECTOR (7 downto 0));
end XOR;

architecture Behavioral of XOR is
begin
    F(7)<='0';
    F(6)<='0';
    F(5)<='0';
    F(4)<='0';
    F(3)<=(A(3) XOR B(3));
    F(2)<=(A(2) XOR B(2));
    F(1)<=(A(1) XOR B(1));
    F(0)<=(A(0) XOR B(0));
end Behavioral;

```



8.NAND Operation:



Truth Table

Inputs		Output
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

VHDL CODE:

```
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;

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

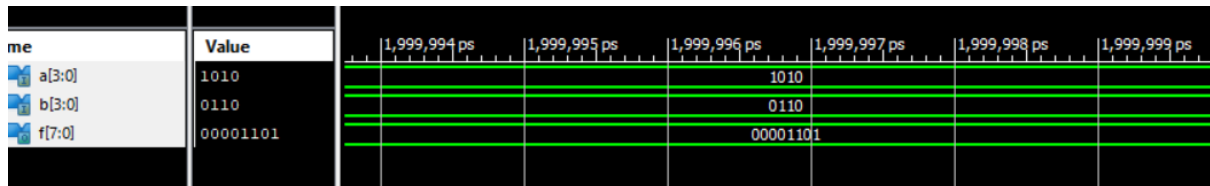
entity NAND is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          F : out STD_LOGIC_VECTOR (7 downto 0));
end NAND;

architecture Behavioral of NAND is
begin
    F(7)<='0';
    F(6)<='0';
    F(5)<='0';
```

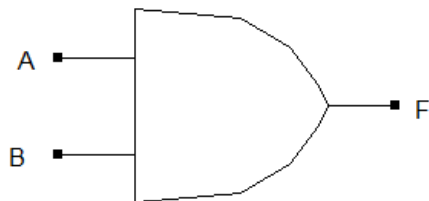
```

F(4)<='0';
F(3)<=NOT(A(3) AND B(3));
F(2)<=NOT(A(2) AND B(2));
F(1)<=NOT(A(1) AND B(1));
F(0)<=NOT(A(0) AND B(0));
end Behavioral;

```



9. AND Operation:



Truth Table

Inputs		Output
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

VHDL CODE:

```

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;

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

entity AND is

```



```

Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
      B : in STD_LOGIC_VECTOR (3 downto 0);
      F : out STD_LOGIC_VECTOR (7 downto 0));

end AND;

```

architecture Behavioral of AND is

begin

```
F(7)<='0';
```

```
F(6)<='0';
```

```
F(5)<='0';
```

```
F(4)<='0';
```

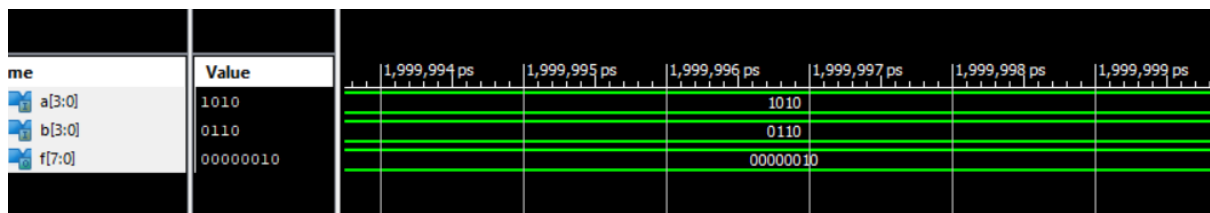
```
F(3)<=(A(3) AND B(3));
```

```
F(2)<=(A(2) AND B(2));
```

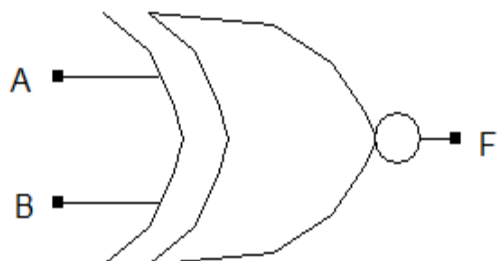
```
F(1)<=(A(1) AND B(1));
```

```
F(0)<=(A(0) AND B(0));
```

end Behavioral;



10. XNOR Operation:



Truth Table

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

```

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;

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

--library UNISIM;

--use UNISIM.VComponents.all;

entity XNOR is

    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);

          B : in  STD_LOGIC_VECTOR (3 downto 0);

          F : out STD_LOGIC_VECTOR (7 downto 0));

end XNOR;

architecture Behavioral of XNOR is

begin

    F(7)<='0';

    F(6)<='0';

    F(5)<='0';

    F(4)<='0';

    F(3)<=NOT(A(3) XOR B(3));

    F(2)<=NOT(A(2) XOR B(2));

    F(1)<=NOT(A(1) XOR B(1));

    F(0)<=NOT(A(0) XOR B(0));

end Behavioral;

```

me	Value	1,999,994 ps	1,999,995 ps	1,999,996 ps	1,999,997 ps	1,999,998 ps	1,999,999 ps
a[3:0]	1001			1001			
b[3:0]	1010			1010			
f[7:0]	00001100			00001100			

11. Left shift A by 1 bit:

```
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;

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

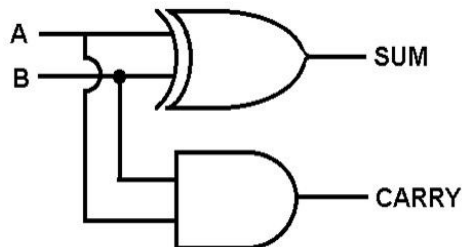
entity SHIFT_A is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          B : in  STD_LOGIC_VECTOR (3 downto 0);
          F : out STD_LOGIC_VECTOR (7 downto 0));
end SHIFT_A;

architecture Behavioral of SHIFT_A is
begin
    F(7)<='0';
    F(6)<='0';
    F(5)<='0';
    F(4)<=A(3);
    F(3)<=A(2);
    F(2)<=A(1);
    F(1)<=A(0);
    F(0)<='0';
end Behavioral;
```

		1,999,994 ps	1,999,995 ps	1,999,996 ps	1,999,997 ps	1,999,998 ps	1,999,999 ps
ne	Value						
 a[3:0]	1010			1010			
 b[3:0]	0101			0101			
 f[7:0]	00010100			00010100			

12. Addition Operation

Half Adder:



Truth Table

Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
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;

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

--use UNISIM.VComponents.all;

entity HALFADDER is
    Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          F : out STD_LOGIC;
          C : out STD_LOGIC);
end HALFADDER;

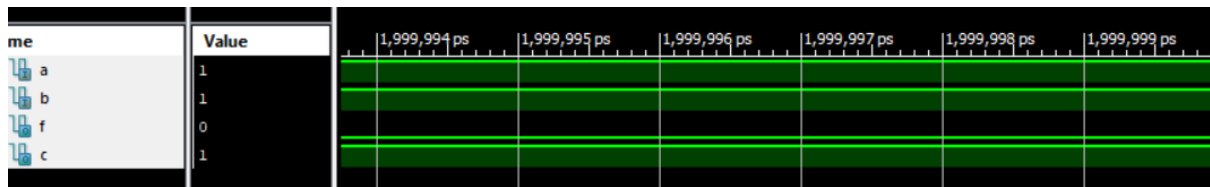
architecture Behavioral of HALFADDER is

begin

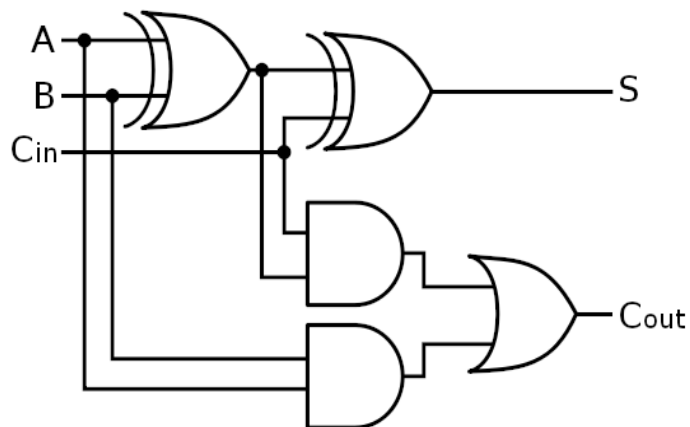
F<=A XOR B;

C<=A AND B;

end Behavioral;
```



Full Adder:



Truth Table

Input			Output	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```

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;

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

entity FULLADDER is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          B : in  STD_LOGIC_VECTOR (3 downto 0);
          C : inout STD_LOGIC_VECTOR (4 downto 0);
          X : inout STD_LOGIC_VECTOR (3 downto 0);
          F: out  STD_LOGIC_VECTOR (3 downto 0));
  
```

```

end FULLADDER;

architecture Behavioral of FULLADDER is

begin

X(0) <=B(0) XOR 0;
X(1) <=B(1) XOR 0;
X(2) <=B(2) XOR 0;
X(3) <=B(3) XOR 0;

C(0) <= 0;

C(1) <=(A(0) AND X(0)) OR (A(0) AND C(0)) OR (X(0) AND C(0));
C(2) <=(A(1) AND X(1)) OR (A(1) AND C(1)) OR (X(1) AND C(1));
C(3) <=(A(2) AND X(2)) OR (A(2) AND C(2)) OR (X(2) AND C(2));
C(4) <=(A(3) AND X(3)) OR (A(3) AND C(3)) OR (X(3) AND C(3));

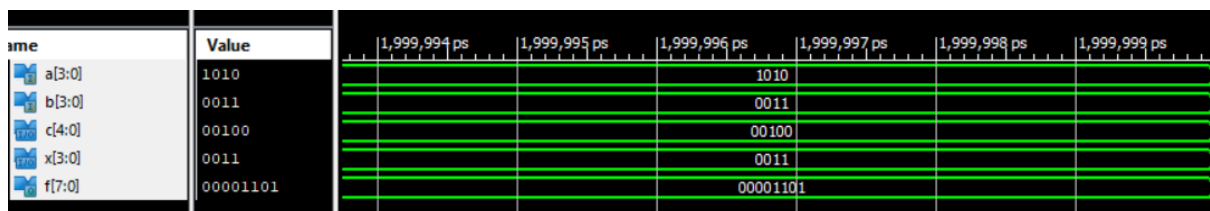
F(0) <=A(0) XOR X(0) XOR C(0);
F(1) <=A(1) XOR X(1) XOR C(1);
F(2) <=A(2) XOR X(2) XOR C(2);
F(3) <=A(3) XOR X(3) XOR C(3);

F(4) <=C(4);

F(5) <=0;
F(6) <=0;
F(7) <=0;

end Behavioral;

```



13. Left Shift B by 1 bit:

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

```

```

UNISIM;

--use UNISIM.VComponents.all;

entity -- 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 SHIFT_B is

    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           F : out STD_LOGIC_VECTOR (7 downto 0));

end SHIFT_B;

architecture Behavioral of SHIFT_B is

begin

F(7)<='0';
F(6)<='0';
F(5)<='0';
F(4)<=B(3);
F(3)<=B(2);
F(2)<=B(1);
F(1)<=B(0);
F(0)<='0';

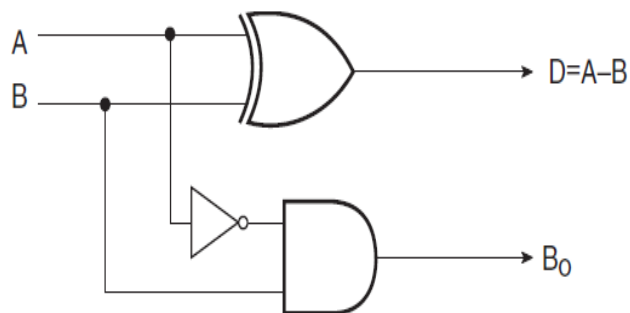
end Behavioral;

```

	Value	1,999,994 ps	1,999,995 ps	1,999,996 ps	1,999,997 ps	1,999,998 ps	1,999,999 ps
a[3:0]	1001			1001			
b[3:0]	0110			0110			
f[7:0]	00001100			00001100			

14. Subtraction Operation:

Half Subtractor:



Truth table

Input		Output	
A	B	D	Bo
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

```
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;

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

entity HALFSUBSTRUCTION is
    Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          F : out STD_LOGIC;
          BORROW : out STD_LOGIC);
end HALFSUBSTRUCTION;
```

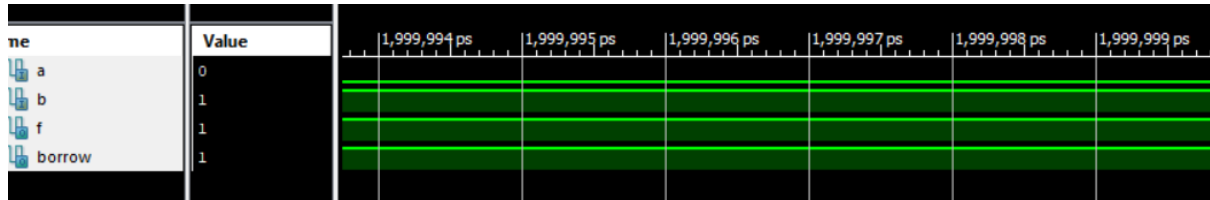

architecture Behavioral of HALFSUBSTRUCTION is

begin

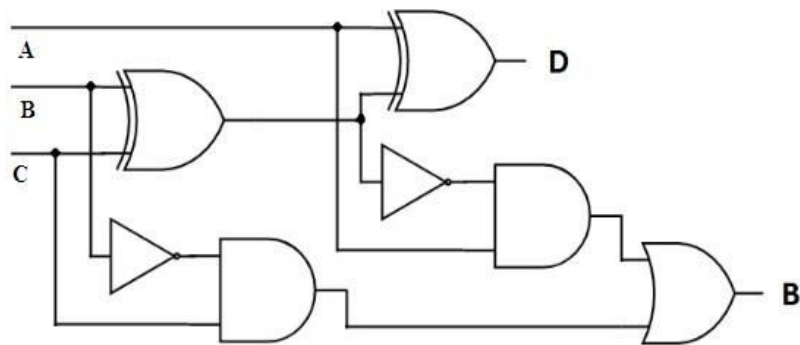
F<=A XOR B;

BORROW<=(NOT A) AND B;

end Behavioral;



Full Subtractor:



Truth Table:

Input			Output	
A	B	C	D	B
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

```

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;

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

--use UNISIM.VComponents.all;

entity FULLSUBSTRUCTION is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          B : in  STD_LOGIC_VECTOR (3 downto 0);
          X : inout STD_LOGIC_VECTOR (3 downto 0);
          C : inout STD_LOGIC_VECTOR (4 downto 0);
          F : out  STD_LOGIC_VECTOR (7 downto 0));
end FULLSUBSTRUCTION;

architecture Behavioral of FULLSUBSTRUCTION is
begin
    X(0) <= B(0) XOR '1';
    X(1) <= B(1) XOR '1';
    X(2) <= B(2) XOR '1';
    X(3) <= B(3) XOR '1';
    C(0) <= '1';
    C(1) <= (A(0) AND X(0)) OR (A(0) AND C(0)) OR (X(0) AND C(0));
    C(2) <= (A(1) AND X(1)) OR (A(1) AND C(1)) OR (X(1) AND C(1));
    C(3) <= (A(2) AND X(2)) OR (A(2) AND C(2)) OR (X(2) AND C(2));
    C(4) <= (A(3) AND X(3)) OR (A(3) AND C(3)) OR (X(3) AND C(3));

```

F(0) <=A(0) XOR X(0) XOR C(0);

F(1) <=A(1) XOR X(1) XOR C(1);

F(2) <=A(2) XOR X(2) XOR C(2);

F(3) <=A(3) XOR X(3) XOR C(3);

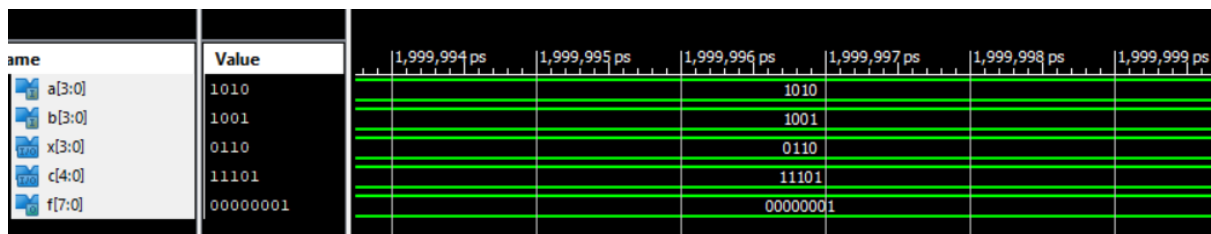
F(4) <=NOT C(4);

F(5) <='0';

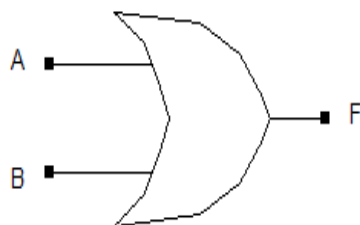
F(6) <='0';

F(7) <='0';

end Behavioral;



15. OR Operation:



Truth Table

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

VHDL CODE:

```
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;
```

```
-- Uncomment the following library declaration if instantiating
```

-- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity OR is

Port (A : in STD_LOGIC_VECTOR (3 downto 0);

B : in STD_LOGIC_VECTOR (3 downto 0);

F : out STD_LOGIC_VECTOR (7 downto 0));

end OR;

architecture Behavioral of OR is

begin

F(7)<='0';

F(6)<='0';

F(5)<='0';

F(4)<='0';

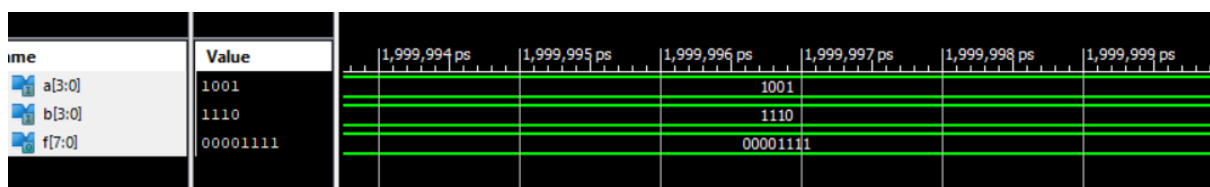
F(3)<=A(3) OR B(3);

F(2)<=A(2) OR B(2);

F(1)<=A(1) OR B(1);

F(0)<=A(0) OR B(0);

end Behavioral;



16. Set Operation:

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using

```

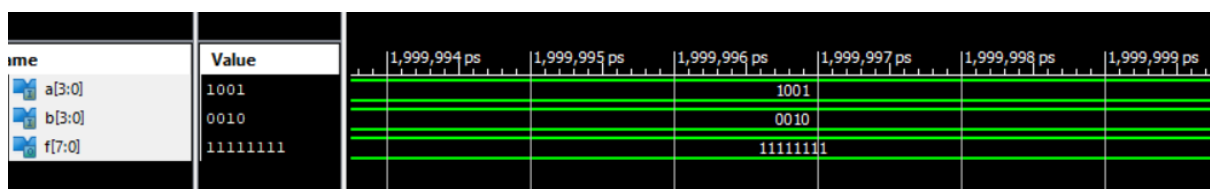
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

entity SET is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          B : in  STD_LOGIC_VECTOR (3 downto 0);
          F : out STD_LOGIC_VECTOR (7 downto 0));
end SET;

architecture Behavioral of SET is
begin
    F(7)<='1';
    F(6)<='1';
    F(5)<='1';
    F(4)<='1';
    F(3)<='1';
    F(2)<='1';
    F(1)<='1';
    F(0)<='1';
end Behavioral;

```



11.OVERALL IMPLIMENTATION:

Considering a Boolean Function:

$$F = AB*S(3)+A'B*S(2)+AB'*S(1)+A'B'*S(0)$$

Now S(3),S(2),S(1),S(0) has total 16 possible combinations (as from 4 bit 16 different combinations are possible). Now among these 16 possible combinations 10 are being used in Logical Section and other 6 combinations are 0000, 0010, 0100, 1011, 1101, 1111. If we choose 0000 then F=0 assign RESET condition. If we choose 0010 then F=AB', this is not useful in this ALU. So we are using MULTIPLICATION condition here. If we choose 0100 then F=A'B, this is also not useful in this ALU. So we are using ROTATION Operation here. If we choose 1011 means A>B condition, as here we have not used it, instead we have used ADDITION Operation. If we choose 1101 then B>A. Hence we have not used it so, we used it in SUBSTRUCTION Operation. Output is taken from a 16:1 multiplexer.

VHDL CODE:

```
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;
```

```
-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx primitives in this code.
```

```
--library UNISIM;
```

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

```
entity PROJECT is
```

```
Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
```

```
      B : in STD_LOGIC_VECTOR (3 downto 0);
```

```
      S : in STD_LOGIC_VECTOR (3 downto 0);
```

```
X : inout STD_LOGIC_VECTOR (3 downto 0);  
Z : out STD_LOGIC;  
OVERFLOW : out STD_LOGIC;  
C : inout STD_LOGIC_VECTOR (5 downto 0);  
F : inout STD_LOGIC_VECTOR (7 downto 0));  
end PROJECT;
```

architecture Behavioral of PROJECT is

```
begin  
PROCESS (S)  
begin  
IF(S="0000") THEN  
F(7) <='0';  
F(6) <='0';  
F(5) <='0';  
F(4) <='0';  
F(3) <='0';  
F(2) <='0';  
F(1) <='0';  
F(0) <='0';  
ELSIF(S="0001") THEN  
F(7) <='0';  
F(6) <='0';  
F(5) <='0';  
F(4) <='0';  
F(3) <=(A(3) NOR B(3));  
F(2) <=(A(2) NOR B(2));  
F(1) <=(A(1) NOR B(1));
```

```

F(0) <=(A(0) NOR B(0));

ELSIF(S="0010") THEN

C(0) <=(A(0) AND B(1)) AND (A(1) AND B(0));

C(1) <=((A(2) AND B(0)) AND (A(1) AND B(1))) OR ((A(2) AND B(0)) AND (A(0) AND B(2))) OR
((A(1) AND B(1)) AND (A(0) AND B(2))) OR ((A(2) AND B(0)) AND C(0)) OR ((A(1) AND B(1))
AND C(0)) OR ((A(0) AND B(2)) AND C(0));

C(2) <=((A(3) AND B(0)) AND (A(2) AND B(1))) OR ((A(3) AND B(0)) AND (A(1) AND B(2))) OR
((A(3) AND B(0)) AND (A(0) AND B(3))) OR ((A(3) AND B(0)) AND C(1)) OR ((A(2) AND B(1))
AND C(1)) OR ((A(1) AND B(2)) AND C(1)) OR ((A(0) AND B(3)) AND C(1)) OR ((A(2) AND B(1))
AND (A(1) AND B(2))) OR ((A(2) AND B(1)) AND (A(0) AND B(3))) OR ((A(1) AND B(2)) AND
(A(0) AND B(3)));

C(3) <=((A(3) AND B(1)) AND (A(2) AND B(2))) OR ((A(3) AND B(1)) AND (A(1) AND B(3))) OR
((A(2) AND B(2)) AND (A(1) AND B(3))) OR ((A(3) AND B(1)) AND C(2)) OR ((A(1) AND B(3))
AND C(2)) OR ((A(2) AND B(2)) AND C(2));

C(4) <=((A(3) AND B(2)) AND (A(2) AND B(3))) OR ((A(3) AND B(2)) AND C(3)) OR ((A(2) AND
B(3)) AND C(3));

C(5) <=((A(3) AND B(3)) AND C(4));

F(0) <=A(0) AND B(0);

F(1) <=(A(1) AND B(0)) XOR (A(0) AND B(1));

F(2) <=(A(2) AND B(0)) XOR (A(1) AND B(1)) XOR (A(0) AND B(2)) XOR C(0);

F(3) <=(A(3) AND B(0)) XOR (A(2) AND B(1)) XOR (A(1) AND B(2)) XOR (A(0) AND B(3)) XOR
C(1);

F(4) <=(A(3) AND B(1)) XOR (A(2) AND B(2)) XOR (A(1) AND B(3)) XOR C(2);

F(5) <=(A(3) AND B(2)) XOR (A(2) AND B(3)) XOR C(3);

F(6) <=(A(3) AND B(3)) XOR C(4);

F(7) <=C(5);

ELSIF(S="0011") THEN

F(7) <='0';

F(6) <='0';

F(5) <='0';

F(4) <='0';

F(3) <=(NOT B(3));

F(2) <=(NOT B(2));

```



```
F(1) <=(NOT B(1));  
F(0) <=(NOT B(0));  
ELSIF(S="0100") THEN  
F(0) <=A(3);  
F(1) <=A(2);  
F(2) <=A(1);  
F(3) <=A(0);  
F(4) <='0';  
F(5) <='0';  
F(6) <='0';  
F(7) <='0';  
ELSIF(S="0101") THEN  
F(7) <='0';  
F(6) <='0';  
F(5) <='0';  
F(4) <='0';  
F(3) <=(NOT A(3));  
F(2) <=(NOT A(2));  
F(1) <=(NOT A(1));  
F(0) <=(NOT A(0));  
ELSIF(S="0110") THEN  
F(7) <='0';  
F(6) <='0';  
F(5) <='0';  
F(4) <='0';  
F(3) <=(A(3) XOR B(3));  
F(2) <=(A(2) XOR B(2));  
F(1) <=(A(1) XOR B(1));  
F(0) <=(A(0) XOR B(0));
```

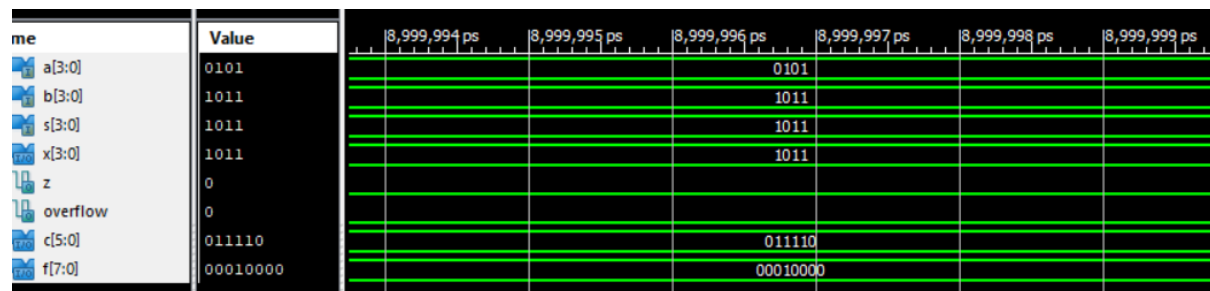
```
ELSIF(S="0111") THEN
F(7) <='0';
F(6) <='0';
F(5) <='0';
F(4) <='0';
F(3) <=(A(3) NAND B(3));
F(2) <=(A(2) NAND B(2));
F(1) <=(A(1) NAND B(1));
F(0) <=(A(0) NAND B(0));
ELSIF(S="1000") THEN
F(7) <='0';
F(6) <='0';
F(5) <='0';
F(4) <='0';
F(3) <=(A(3) AND B(3));
F(2) <=(A(2) AND B(2));
F(1) <=(A(1) AND B(1));
F(0) <=(A(0) AND B(0));
ELSIF(S="1001") THEN
F(7) <='0';
F(6) <='0';
F(5) <='0';
F(4) <='0';
F(3) <=(A(3) XNOR B(3));
F(2) <=(A(2) XNOR B(2));
F(1) <=(A(1) XNOR B(1));
F(0) <=(A(0) XNOR B(0));
ELSIF(S="1010") THEN
F(7) <='0';
```

```
F(6) <='0';
F(5) <='0';
F(4) <=A(3);
F(3) <=A(2);
F(2) <=A(1);
F(1) <=A(0);
F(0) <='0';
ELSIF(S="1011") THEN
X(0) <=B(0) XOR '0';
X(1) <=B(1) XOR '0';
X(2) <=B(2) XOR '0';
X(3) <=B(3) XOR '0';
C(0) <= '0';
C(1) <=(A(0) AND X(0)) OR (A(0) AND C(0)) OR (X(0) AND C(0));
C(2) <=(A(1) AND X(1)) OR (A(1) AND C(1)) OR (X(1) AND C(1));
C(3) <=(A(2) AND X(2)) OR (A(2) AND C(2)) OR (X(2) AND C(2));
C(4) <=(A(3) AND X(3)) OR (A(3) AND C(3)) OR (X(3) AND C(3));
C(5) <='0';
F(0) <=A(0) XOR X(0) XOR C(0);
F(1) <=A(1) XOR X(1) XOR C(1);
F(2) <=A(2) XOR X(2) XOR C(2);
F(3) <=A(3) XOR X(3) XOR C(3);
F(4) <=C(4);
F(5) <='0';
F(6) <='0';
F(7) <='0';
OVERFLOW <=(A(3) AND X(3) AND (NOT F(3))) OR ((NOT A(3)) AND (NOT X(3)) AND F(3));
IF(F="00000000") THEN
Z <='1';
```

```
ELSE
Z <='0';
END IF;
ELSIF(S="1100") THEN
F(7) <='0';
F(6) <='0';
F(5) <='0';
F(4) <=B(3);
F(3) <=B(2);
F(2) <=B(1);
F(1) <=B(0);
F(0) <='0';
ELSIF(S="1101") THEN
X(0) <=B(0) XOR '1';
X(1) <=B(1) XOR '1';
X(2) <=B(2) XOR '1';
X(3) <=B(3) XOR '1';
C(0) <= '1';
C(1) <=(A(0) AND X(0)) OR (A(0) AND C(0)) OR (X(0) AND C(0));
C(2) <=(A(1) AND X(1)) OR (A(1) AND C(1)) OR (X(1) AND C(1));
C(3) <=(A(2) AND X(2)) OR (A(2) AND C(2)) OR (X(2) AND C(2));
C(4) <=(A(3) AND X(3)) OR (A(3) AND C(3)) OR (X(3) AND C(3));
C(5) <='0';
F(0) <=A(0) XOR X(0) XOR C(0);
F(1) <=A(1) XOR X(1) XOR C(1);
F(2) <=A(2) XOR X(2) XOR C(2);
F(3) <=A(3) XOR X(3) XOR C(3);
F(4) <=NOT C(4);
F(5) <='0';
```

```
F(6) <='0';
F(7) <='0';
OVERFLOW <=(A(3) AND X(3) AND (NOT F(3))) OR ((NOT A(3)) AND (NOT X(3)) AND F(3));
IF(F="00000000") THEN
Z <='1';
ELSE
Z <='0';
END IF;
ELSIF(S="1110") THEN
F(7) <='0';
F(6) <='0';
F(5) <='0';
F(4) <='0';
F(3) <=(A(3) OR B(3));
F(2) <=(A(2) OR B(2));
F(1) <=(A(1) OR B(1));
F(0) <=(A(0) OR B(0));
ELSIF(S="1111") THEN
F(7) <='1';
F(6) <='1';
F(5) <='1';
F(4) <='1';
F(3) <='1';
F(2) <='1';
F(1) <='1';
F(0) <='1';
END IF;
END PROCESS;
end Behavioral;
```

12.RESULT:



If we take select line combination as 1011, then Addition Operation will be performed. Assuming A=0101 and B=1011 we should get 10000 as the output. From the output diagram we got 10000. In this way for different select line combinations different operations will be performed. We have taken the output as 8-bit. Because in a 4-bit number multiplication, the result will be of 8-bit. In other 13 operations output will be of 4-bit and in addition & subtraction, if overflow occurs then 5-bit result will be produced. Among these 13 logical operations we are getting only 4-bit output so, rest of bits to the MSB are taken as 0s. In shift operation when 1 bit left shift occurs then 5-bit output is produced. In this case 3 bits to the MSB are unused and are taken as 0. In addition & subtraction also 3 bits to the MSB are unused so, taken as 0. The 5th bit from LSB is connected to carry output of 4th bit.

13.CONCLUSION AND FUTURE SCOPE:

We are able to design and implement an **Arithmetic and Logical Unit**. Which can perform total 16 different operation including arithmetic, logical, multiplication and also rotational operations. We have also made our instruction sets. Our ALU can also perform left shifting, A's and B's complement along SET, RESET, and Multiplication which are very difficult to implement in the ALU. We have used all opcodes possible for 4 bits i.e 16 opcodes. The implementation of all codes were performed bit by bit. Thus we have successfully completed a 4-bit ALU as our project work.

In our project "Design and Implementation of a 4-bit ALU using VHDL" we have designed and implemented a 4-bit ALU. Arithmetic Logic Unit is the part of a computer that performs all arithmetic computations, such as addition and subtraction, increment, decrement, shifting and all sorts of basic logical operations. The ALU is one component of the CPU (Central Processing Unit). Here, using VHDL we have designed a 4-bit ALU which can perform the various arithmetic operations of Addition, Subtraction, Multiplication, Rotation, logical operations such as AND, OR, XOR, NOT and also the shift operation. All the above mentioned operations are then verified to see whether they match theoretically or not. The above given waveforms show that they match completely thereby verifying our results.

Looking at the future, there may be huge development in digital technology but the basic blocks like ALU, Microprocessors will always be studied in base logic levels. So this project will always be helpful irrespective of time and generation.

14.REFERENCES:

[1]. T. K. Ghosh and A. J. Pal, Computer Organization and Architecture, Tata McGraw-Hill Publishing Company Limited, New Delhi, 2009.

[2]. Douglas L. Perry, VHDL Programming by Example, 4th ed., Tata McGraw-Hill Publishing Company Limited, New Delhi, 2002.

[4]. VHDL Tutorial, Ardent Computech, PVT. LTD., 2011. [6]. <http://www.google.com>. [7]. <http://www.wikipedia.org>.