ARITHMETIC LOGIC UNIT USING VHDL

Gourav Singhal (071025)

Anshuman Singh (071054)

Shardul Singh (071089)

Supriya Rai (071046)

UNDER THE GUIDANCE OF:

Mr. Vipin Balyan
Sr.Lecturer
Dept. of Electronics & Comm.

विद्या तत्व ज्योतिसम

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

MAY – 2011

Submitted in partial fulfillment of the Degree of
BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT
SOLAN, HIMACHAL PRADESH
INDIA

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT

CERTIFICATE

This is to certify that the work titled , **"ARITHMETIC LOGIC UNIT USING VHDL"** submitted by **"Gourav Singhal(071025), Anshuman Singh(071054), Shardul Singh(071089), Supriya Rai(071046)"** in partial fulfillment for the award of degree of **B.TECH** of **Jaypee University of Information Technology**, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

(MR. VIPIN BALYAN)
Sr.Lecturer
Dept. of Electronics & Comm.
JUIT,Waknaghat ,173234
Date -

ACKNOWLEDGEMENT

We would like to thank my Project advisor , **Mr. Vipin Balyan** , Sr. Lecturer, Dept. of Electronics & Comm., Jaypee University of Information Technology , Waknaghat to provide me the opportunity to work on this Project , for his guidance ,encouragement, support and confidence in me through the course of my studies . Without his motivating discussions and unwavering desire for achieving high research standards , this work would not have been possible . His flawless & forthright suggestions blended with an innate intelligent application have crowned our task with success.

We are highly obliged to **Prof. Dr. Sunil Bhooshan** , H.O.D, Dept. of Electronics & Comm., Jaypee University of Information Technology , Waknaghat for allowing me to carry out my project work in this university . We would also like to offer my sincere thank to Mr. Rachit , all faculty , teaching & non-teaching staff , of E.C.E Dept. and staff of the Library Jaypee University of Information Technology , Waknaghat for their assistance.

Gourav Singhal (071025)

Asingh

Anshuman Singh(071054)

Shardul Singh (071089)

Supriya Rai

Supriya Rai (071046)

Table of Content

List of Figures

List of Tables

ABSTRACT

An ALU is the fundamental unit of any computing system. Understanding how an ALU is designed and how it works is essential to building any advanced logic circuits. Using this knowledge and experience, we can move on to designing more complex integrated circuits. We have designed a 32 bit ALU which performs eight functions through three select pins. According to the values of these pins different functions of the ALU are represented. After giving a specific combination of input selection pin the required output is produced. The functions performed by the ALU are and, or, nand, nor, addition, subtraction, left shift, right shift, left rotate, right rotate . All the results are testified by test bench waveforms .

CHAPTER 1

INTRODUCTION

The central processing unit (CPU) of a computer is the main unit that dictates the rest of the computer organization.

The CPU is made of three major parts

1. Register set: Stores intermediate data during the execution of instructions;
2. Arithmetic logic unit (ALU): Performs the required micro-operations for executing the instructions.
3. Control unit: supervises the transfer of information among the registers and instructs the ALU as to which operation to perform by generating control signals.

1.1 ALU

An arithmetic logic unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs. Mathematician John von Neumann proposed the ALU concept in 1945, when he wrote a report on the foundations for a new computer called the EDVAC. Research into ALUs remains an important part of computer science, falling under Arithmetic and logic structures in the ACM Computing Classification System.

Most of a processor's operations are performed by one or more ALUs. An ALU loads data from input registers, an external Control Unit then tells the ALU what operation to perform on that data, and then the ALU stores its result into an output register. Other mechanisms move data between these registers and memory.

A simple example arithmetic logic unit (2-bit ALU) that does AND, OR, XOR, and addition.

Most ALUs can perform the following operations:

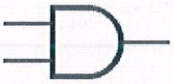
1. Integer arithmetic operations (addition, subtraction)
2. Bitwise logic operations (AND, NOT, OR, XOR)
3. Bit-shifting operations (shifting or rotating a word by a specified number of bits to the left or right.

ALU allows the computer to add, subtract, and to perform basic logical operations such as AND/OR. Since every computer needs to be able to do these simple functions, they are always included in a CPU. How a company designs their ALU has a significant impact on the overall performance of their CPU.ALU is a digital circuit that performs Arithmetic (Add, Sub . . .) and Logical (AND, OR, NOT) operations.

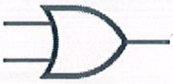
1.1.1 Logic Unit

Logic unit performs logical operations such as and, or, not etc.

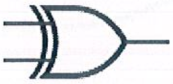
Figure 1.1 shows the basic logic gates.



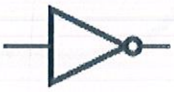
'AND' Gate



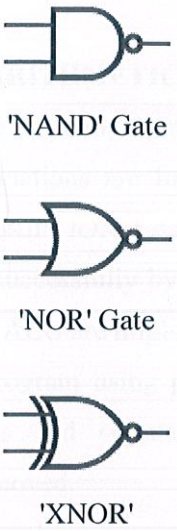
'OR' Gate



'XOR' Gate



'NOT' Gate



'NAND' Gate

'NOR' Gate

'XNOR'

Figure 1.1: Basic logic gates

These logic gates work by taking two inputs (one input for the 'NOT' gate) and producing an output. These logic functions are by themselves an important part of a CPU's functionality, but performing logic operations on two inputs is only so useful. By combining these gates together we can have devices with more inputs.

1.1.2 Arithmetic Unit

By combining these gates we can perform other useful functions, like addition, subtraction. Figure 1.2 shows a typical configuration referred to as a half-adder

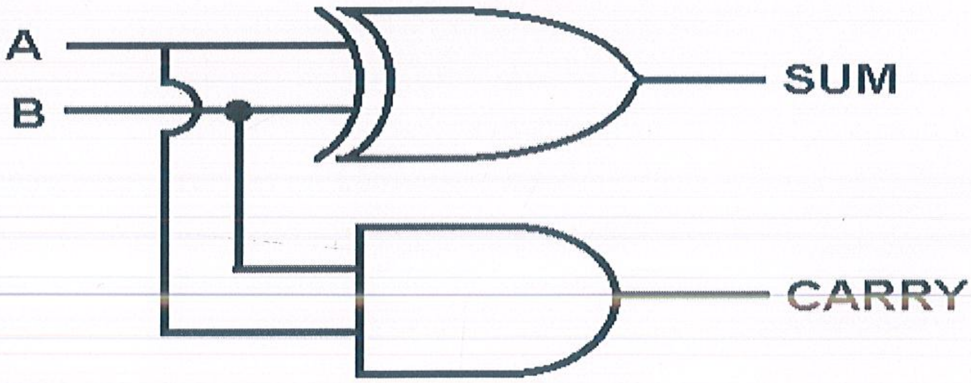


Figure 1.2 Half Adder

1.2 Project Overview

The motive of the project “**ARITHMETIC LOGIC UNIT USING VHDL**" is implementing fundamental operations like and, or, nor etc and some upper level functions like addition, rotation etc .These functions are implemented as well as simulated in VHDL(VHSIC HARDWARE DESCRIPTION LANGUAGE) using **XILINX 8.2**i. The implemented functions are testified successfully by test bench waveform simulation. In initial phase all eight functions of the ALU are implemented and simulated separately and later they are multiplexed in one program using port mapping. The faults and errors occurred in implemented functions and simulations were seriously taken in to consideration and were efficiently removed.

CHAPTER 2

INTRODUCTION TO VHDL

2.1 What Is VHDL?

VHDL is an acronym for VHSIC Hardware Description Language (VHSIC is an acronym for Very High Speed Integrated Circuits). It is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. The complexity of the digital system being modeled could vary from that of a simple gate to a complete digital electronic system, or anything in between. The digital system can also be described hierarchically. Timing can also be explicitly modeled in the same description.

2.2 History

The requirements for the language were first generated in 1981 under the VHSIC program. In this program, a number of U.S. companies were involved in designing VHSIC chips for the Department of Defense (DoD). At that time, most of the companies were using different hardware description languages to describe and develop their integrated circuits. As a result, different vendors could not effectively exchange designs with one another. Also, different vendors provided DoD with descriptions of their chips in different hardware description languages. Reprocurement and reuse was also a big issue. Thus, a need for a standardized hardware description language for design, documentation, and verification of digital systems was generated.

A team of three companies, IBM, Texas Instruments, and Intermetrics, were first awarded the contract by the DoD to develop a version of the language in 1983. Version 7.2 of VHDL was developed and released to the public in 1985. There was a strong industry participation throughout the VHDL language development process, especially from the companies that were developing VHSIC chips. After the release of version 7.2, there was an increasing need to make the language an industry-wide standard. Consequently, the language was transferred to the IEEE for standardization in 1986. After a substantial enhancement to the language, made by a team of industry, university, and DoD representatives, the language was standardized by the IEEE in December 1987; this version of the language is now known as the IEEE Std 1076-1987. The official language description appears in the IEEE Standard VHDL Language Reference Manual made available by the IEEE. The language described in this book is based on this standard. The language has since also been recognized as an American National Standards Institute (ANSI) standard.

The Department of Defense, since September 1988, requires all its digital Application-Specific Integrated Circuit (ASIC) suppliers to deliver VHDL descriptions of the ASICs and their subcomponents, at both the behavioral and structural levels. Test benches that are used to validate the ASIC chip at all levels in its hierarchy must also be delivered in VHDL. This set of government requirements is described in military standard 454.

2.3 Capabilities

- The language can be used as an exchange medium between chip vendors and CAD tool users. Different chip vendors can provide VHDL descriptions of their components to system designers. CAD tool users can use it to capture the behavior of the design at a high level of abstraction for functional simulation.
- The language can also be used as a communication medium between different CAD and CAE tools, for example, a schematic capture program may be used to generate a VHDL description for the design which can be used as an input to a simulation program.
- The language supports hierarchy, that is, a digital system can be modeled as a set of interconnected components; each component, in turn, can be modeled as a set of interconnected subcomponents.
- The language supports flexible design methodologies: top-down, bottom-up, or mixed.
- It is an IEEE and ANSI standard, and therefore, models described using this language is portable. The government also has a strong interest in maintaining this as a standard so that re-procurement and second-sourcing may become easier.

2.4 Hardware Abstraction

VHDL is used to describe a model for a digital hardware device. This model specifies the external view of the device and one or more internal views. The internal view of the device specifies the functionality or structure, while the external view specifies the interface of the device through which it communicates with the other models in its environment. . Figure 2.1 shows the hardware device and the corresponding software model.

In VHDL, each device model is treated as a distinct representation of a unique device, called an entity in this text. Figure 2.2 shows the VHDL view of a hardware device that has multiple device models, with each device model representing one entity. Even though entity I through N represents N different entities from the VHDL point of view, in reality they represent the same hardware device.

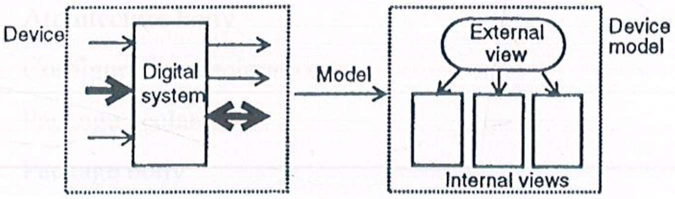


Figure 2.1 Device versus device model.

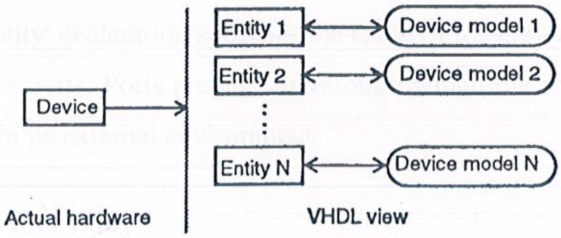


Figure 2.2 A VHDL view of a device.

CHAPTER 3

MODELING FEATURES OF VHDL

VHDL is a hardware description language that can be used to model a digital system. The digital system can be as simple as a logic gate or as complex as a complete electronic system. A hardware abstraction of this digital system is called an entity in this text. An entity X, when used in another entity Y, becomes a component for the entity Y. Therefore, a component is also an entity, depending on the level at which you are trying to model.

3.1 Entity

To describe an entity, VHDL provides five different types of primary constructs, called" design units . They are

1. Entity declaration
2. Architecture body
3. Configuration declaration
4. Package declaration
5. Package body

3.1.1 Entity Declaration

The entity' declaration specifies the name of the entity being modeled and lists the set of interface ports. Ports are signals through which the entity communicates with the other models in its external environment.

3.1.2 Architecture Body

The internal details of an entity are specified by an architecture body using any of the following modeling styles:

1. As a set of interconnected components (to represent structure).
2. As a set of concurrent assignment statements (to represent dataflow).
3. As a set of sequential assignment statements (to represent behavioral).
4. Any combination of the above three.

3.1.3 Configuration Declaration

A configuration declaration is used to select one of the possibly many architecture bodies that an entity may have, and to bind components, used to represent structure in that architecture body, to entities represented by an entity-architecture pair or by a configuration, that reside in a design library

3.1.4 Package Declaration

A package declaration is used to store a set of common declarations like components, types, procedures, and functions. These declarations can then be imported into other design units using a context clause.

3.1.5 Package body

A package body is primarily used to store the definitions of functions and procedures that were declared in the corresponding package declaration, and also the complete constant declarations for any deferred constants that appear in the package declaration. Therefore, a package body is always associated with a package declaration; furthermore, a package declaration can have at most one package body associated with it. Contrast this with an architecture body and an entity declaration where multiple architecture bodies may be associated with a single entity declaration.

3.2 Data Objects

A data object holds a value of a specified type. It is created by means of an object declaration. Every data object belongs to one of the following three classes:

3.2.1 Constant

An object of constant class can hold a single value of a given type. This value is assigned to the object before simulation starts and the value cannot be changed during the course of the simulation.

3.2.2. Variable

An object of variable class can also hold a single value of a given type. However in this case, different values can be assigned to the object at different times using a variable assignment statement.

3.2.3 Signal

An object belonging to the signal class has a past history of values, a current value, and a set of future values. Future values can be assigned to a signal object using a signal assignment statement.

3.3 Operators

The predefined operators in the language are classified into the following five categories:

1. Logical operators
2. Relational operators
3. Adding operators
4. Multiplying operators
5. Miscellaneous operators

3.3.1 Logical Operators

The six logical operators are

And, Or , Nand, Nor, Xor, Not

These operators are defined for the predefined types BIT and BOOLEAN. They are also defined for one-dimensional arrays of BIT and BOOLEAN. During evaluation, bit values '0' and 1' are treated as FALSE and TRUE values of the BOOLEAN type, respectively.

3.3.2 Relational Operators

These are

= /= < <= > >=

The result type for all relational operations is always BOOLEAN.

3.3.3 Adding Operators

These are

+ - &

The operands for the + (addition) and - (subtraction) operators must be of the same numeric type with the result being of the same numeric type The operands for the &

(concatenation) operator can be either a 1-dirnensional array type or an element type. The result is always an array type.

3.3.4 Multiplying Operators

These are

* / mod rem

The * (multiplication) and / (division) operators are predefined for both operands being of the same integer or floating point type. The result is also of the same type.

3.3.5 Miscellaneous Operators

The miscellaneous operators are

abs **

The abs (absolute) operator is defined for any numeric type.

The ** (exponentiation) operator is defined for the left operand to be of integer or floating point type and the right operand (i.e., the exponent) to be of integer type only.

CHAPTER 4

BEHAVIORAL MODELING

In this modeling style, the behavior of the entity is expressed using sequentially executed, procedural code. A process statement is the primary mechanism used to model the procedural type behavior of an entity. Irrespective of the modeling style used, every entity is represented using an entity declaration and at least one architecture body.

4.1 Entity Declaration

An entity declaration describes the external interface of the entity. It specifies the name of the entity, the names of interface ports, their mode (i.e., direction), and the type of ports. The syntax

```
entity entity-name is
        [port ( list-of-interface-port-names-and-their-types) ; ]
end [ entity-name ];
```

The entity-name is the name of the entity and the interface ports are the signals through which the entity passes information to and from its external environment. Each interface port can have one of the following modes:

1. in: the value of an input port can only be read within the entity model.
2. out: the value of an output port can only be updated within the entity model; it cannot be read.
3. inout: the value of a bidirectional port can be read and updated within the entity model.
4. buffer: the value of a buffer port can be read and updated within the entity model. However, it differs from the inout mode in that it cannot have more than one source and that the only kind of signal that can be connected to it can be another buffer port or a signal with at most one source.

4.2 Architecture Body

An architecture body describes the internal view of an entity. It describes the functionality or the structure of the entity. The syntax of an architecture body is

```
architecture architecture-name of entity-name is
```

```
          [ architecture-item-declarations ]
     begin
          concurrent-statements; these are —>
               process-statement
               block-statement
               concurrent-procedure-call
               concurrent-assertion-statement
               concurrent-signal-assignment-statement
               component-instantiation-statement
     end [ architecture-name ] ;
```

4.3 Process Statement

A process statement contains sequential statements that describe the functionality of a portion of an entity in sequential terms. The syntax of a process statement is

```
     process [ ( sensitivity-list ) ]
          [process-item-declarations]
     begin
          sequential-statements; these are ->
               variable-assignment-statement
               signal-assignment-statement
               if-statement
               case-statement
               loop-statement
     end process ;
```

4.4 Variable, Signal Assignment Statement

Variables can be declared and used inside a process statement. A variable is assigned a value using the variable assignment statement that typically has the form

```
     variable-object := expression;
```

The expression is evaluated when the statement is executed and the computed value is assigned to the variable object instantaneously, that is, at the current simulation time.

Signals are assigned values using a signal assignment statement The simplest form of a signal assignment statement is

```
signal-object <= expression ;
```

A signal assignment statement can appear within a process or outside of a process. If it occurs outside of a process, it is considered to be a concurrent signal assignment statement.

When a signal assignment statement appears within a process, it is considered to be a sequential signal assignment statement and is executed in sequence with respect to the other sequential statements that appear within that process.

4.5 If Statement

If statement selects a sequence of statements for execution based on the value of a condition. The condition can be any expression that evaluates to a boolean value. The general form of an if statement is

```
if boolean-expression then
        sequential-statements

[ elsif boolean-expression then
        sequential-statements ]
[ else
        sequential-statements ]
end if;
```

4.6 Case Statement

The format of a case statement is

```
case expression is
        when choices => sequential-statements
        when choices => sequential-statements
        -- Can have any number of branches.
        [ when others => sequential-statements ]
end case;
```

The case statement selects one of the branches for execution based on the value of the expression.

CHAPTER 5

DATAFLOW AND STRUCTURAL MODELING

5.1 Data Flow Modeling

A dataflow model specifies the functionality of the entity without explicitly specifying its structure. This functionality shows the flow of information through the entity, which is expressed primarily using concurrent signal assignment statements. The structure of the entity is not explicitly specified in this modeling style, but it can be implicitly deduced. In a signal assignment statement, the symbol <= implies an assignment of a value to a signal. The value of the expression on the right-hand-side of the statement is computed and is assigned to the signal on the left-hand-side, called the target signal. A concurrent signal assignment statement is executed only when any signal used in the expression on the right-hand-side has an event on it, that is, the value for the signal changes. Concurrent signal assignment statements are concurrent statements, and therefore, the ordering of these statements in an architecture body is not important.

5.2 Concurrent versus Sequential Signal Assignment

Signal assignment statements can also appear within the body of a process statement. Such statements are called sequential signal assignment statements, while signal assignment statements that appear outside of a process are called concurrent signal assignment statements. Concurrent signal assignment statements are event triggered, that is, they are executed whenever there is an event on a signal that appears in its expression, while sequential signal assignment statements are not event triggered and are executed in sequence in relation to the other sequential statements that appear within the process.

5.3 Structural Modeling

An entity is modeled as a set of components connected by signals. The behavior of the entity is not explicitly apparent from its model. The component instantiation statement is the primary mechanism used for describing such a model of an entity.

5.3.1 Component Declaration

A component instantiated in a structural description must first be declared using a component declaration. A component declaration declares the name and the interface of a component. The interface specifies the mode and the type of ports. The syntax of a simple form of component declaration is

```
component component-name
        port ( list-of-interface-ports ) ;
end component;
```

5.3.2 Component Instantiation

A component instantiation statement defines a subcomponent of the entity in which it appears. It associates the signals in the entity with the ports of that subcomponent.

```
component-label: component-name port map ( association-list) ;
```

5.3.3 Configuration

It is convenient to specify multiple views for a single entity and use any one of these for simulation. This can be easily done by specifying one architecture body for each view and using a configuration to bind the entity to the desired architecture body also it is desirable to associate a component with any one of a set of entities. The component declaration may have its name and the names, types, and number of ports different from those of its entities.

A configuration is, therefore, used to bind

1. An architecture body to its entity declaration,
2. A component with an entity.

The language provides two ways of performing this binding:

1. By using a configuration specification,
2. By using a configuration declaration.

5.3.3.1 Configuration Specification

A configuration specification is used to bind component instantiations to specific entities that are stored in design libraries. The specification appears in the declarations part of the architecture.

The syntax of a configuration specification is

for list-of-comp-labels: component-name binding-indication;

The binding-indication specifies the entity represented by the entity-architecture pair, and the generic and port bindings, its forms is

use entity entity-name [(architecture-name)] ;

5.3.3.2 Configuration Declaration

A configuration declaration is a separate design unit, therefore, it allows for late binding of components, that is, the binding can be performed after the architecture body has been written. It is also possible to have more than one configuration declaration for an entity, each of which defines a different set of bindings for components in a single architecture body, or possibly specifies a unique entity-architecture pair

Format of a configuration declaration is

configuration configuration-name **of** entity-name **is**
block-configuration
end [configuration-name];

CHAPTER 6

VHDL FUNCTIONS, PACKAGES AND LIBRARIES

6.1 Functions

Functions are used to describe frequently used sequential algorithms that return a single value. This value is returned to the calling program using a return statement. The general form of a function declaration is

```
function function-name (formal-parameter-list)
    return return-type is
    [declaration]
begin
    sequential statements
end function –name
```

6.2 Packages

A package provides a convenient mechanism to store and share declarations that are common across many design units. A package is represented by

1. A package declaration, and optionally,
2. A package body.

6.2.1 Package Declaration

A package declaration contains a set of declarations that may possibly be shared by many design units. It defines the interface to the package, that is, it defines items that can be made visible to other design units.

```
package package-name is
        package-item-declarations
end [ package-name ] ;
```

6.2.2 Package Body

A package body primarily contains the behavior of the subprograms and the values of the deferred constants declared in a package declaration. It may contain other declarations as well, as shown by the following syntax of a package body. The package name must be the same as the name of its corresponding package declaration.

```
package body package-name is
        package-body-item-declarations "
end [ package-name ];
```

6.3 Libraries

Each design unit – entity architecture, configuration, package declaration and package body is analyzed (compiled) and placed in design library. Libraries are generally implemented as directories and are referenced by logical names. In the implementation of VHDL environment, this logical name maps to a physical path to the corresponding directory and this mapping is maintained by the host implementation.

In VHDL, the libraries STD and WORK are implicitly declared therefore the user programs do not need to declare these libraries. The STD contains standard package provided with VHDL distributions. The WORK contains the working directory that can be set within the VHDL environment you are using. However if a program were to access functions in a design unit that was stored in a library with a logical name IEEE .Then this library must be declared at the start of the program.

.

CHAPTER 7

ARITHMETIC LOGIC UNIT

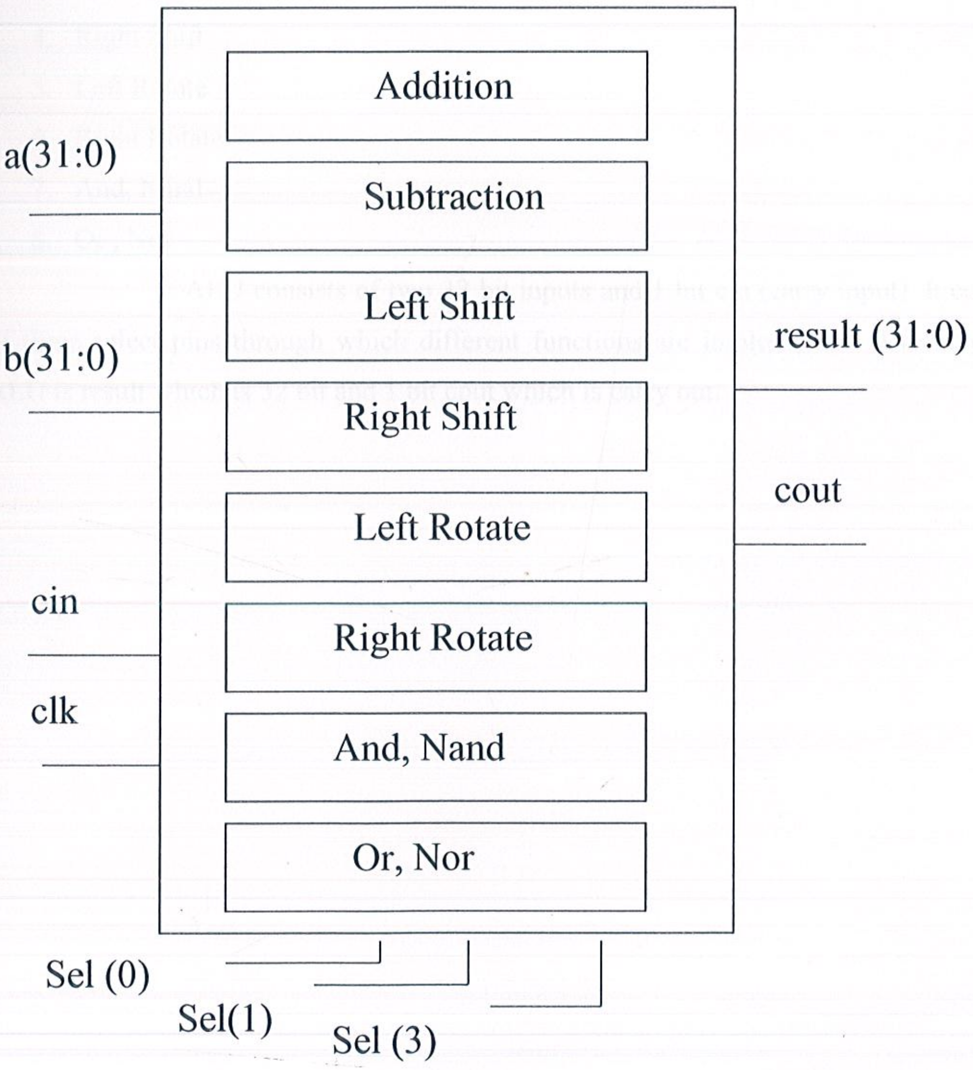


Figure 7.1 Block Diagram of ALU

ALU as shown in figure 7.1 consists of eight functions

1. Addition
2. Subtractions
3. Left Shift
4. Right Shift
5. Left Rotate
6. Right Rotate
7. And, Nand
8. Or , Nor

ALU consists of two 32 bit inputs and 1 bit cin (carry input). It consists of three select pins through which different functions are implemented. The output of ALU is result which is 32 bit and 1 bit cout which is carry out.

CHAPTER 8

ADDER

To understand how this adder works we have to think of the inputs not as true or false but as '1' or '0'. The output of this adder is the sum of the inputs with a carry bit. If the inputs are '1' and '1' we are adding 1 plus 1. The output labeled 'SUM' is just an 'XOR' of the inputs which will be '0'. The output labeled 'CARRY' is an AND gate which of course will be '1'. The addition answer therefore is 10 which is the binary addition of '1' and '1'. If the inputs are '1' and '0' the 'SUM' will be '1' and the 'CARRY' will be '0', giving

answer of 01 or just 1.

In order to add binary numbers greater than two bits we need the adder to be able to take in a carry bit along with the two input bits. This full-adder is shown in Figure 8.1. You can see that the full-adder is two half-adders with one additional 'OR' gate. To use a full-adder to add two binary numbers of arbitrary size you will begin with the right most bit, called the least significant bit (LSB) of each number with a carry in bit of '0'. You would then add the two bits, record the sum, and use the carry out bit as the carry in bit when adding the next two bits and moving towards the most significant bits (MSB). By repeating this process you can add two binary numbers of any arbitrary length. This process is known as a ripple carry.

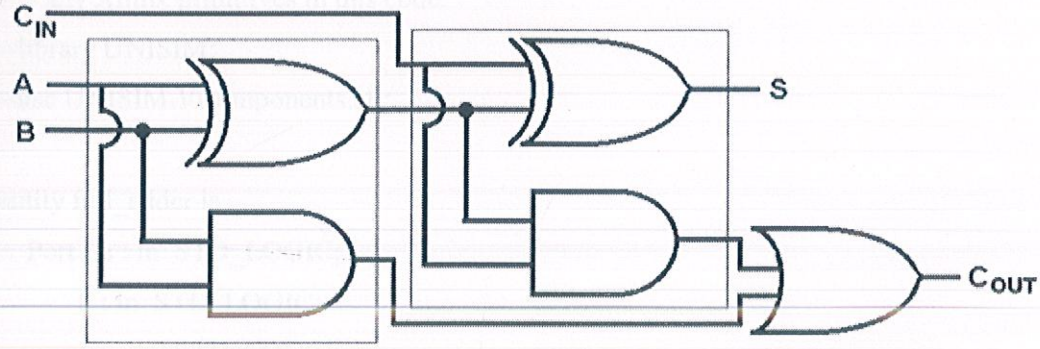


Figure 8.1: Full adder

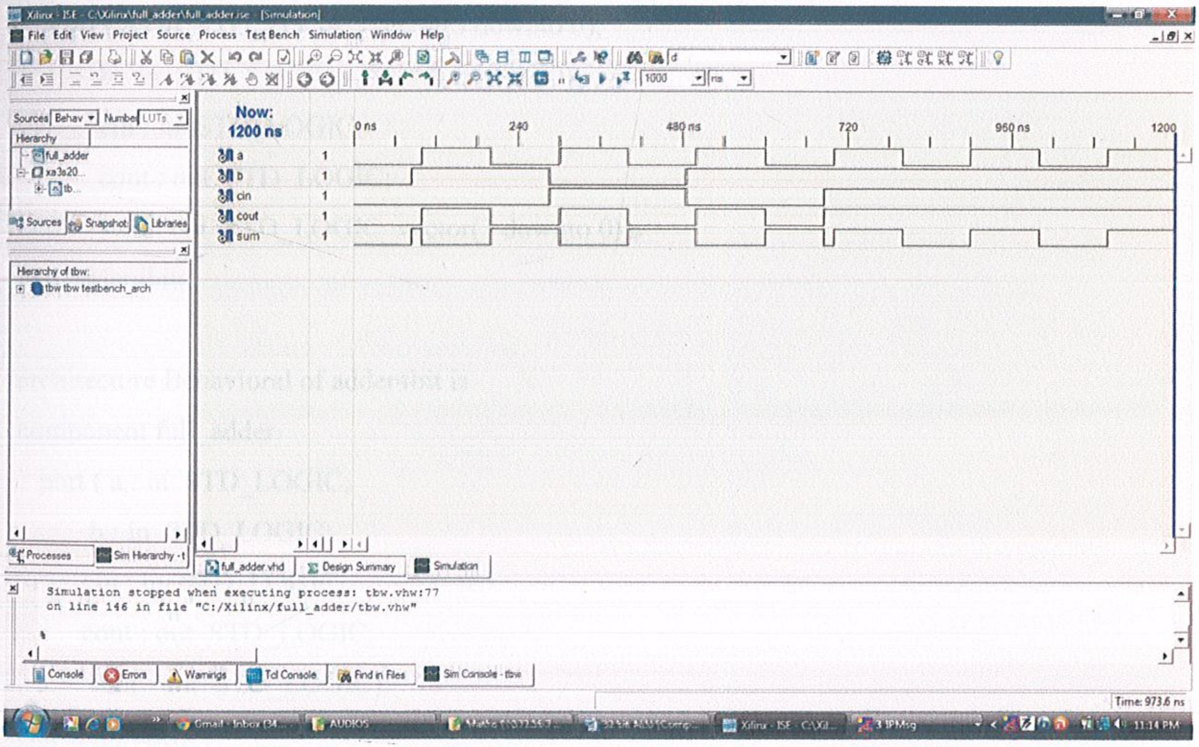
A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 8.1 Truth table of full adder

8.1 Full Adder

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

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

entity full_adder is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          cin : in  STD_LOGIC;
          cout : out  STD_LOGIC;
          sum : out  STD_LOGIC);
```

```
end full_adder;


architecture Behavioral of full_adder is


begin
process(a,b,cin)
                begin
                 sum <= a xor b xor cin;
                 cout <= (a and b) or (b and cin) or (a and cin);
end process;
end Behavioral;
```


Xilinx - ISE - C:\Xilinx\full_adder\full_adder.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Sources Behav Number LUTs
Hierarchy
full_adder
xa3s20...
tb...
Sources Snapshot Libraries
Hierarchy of tbw:
tbw tbw testbench_arch
Processes Sim Hierarchy
Now:
1200 ns
0 ns
240
480 ns
720
960 ns
1200
a 1
b 1
cin 1
cout 1
sum 1
full_adder.vhd Design Summary Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - tbw
Time: 973.6 ns
Gmail - Inbox (34... AUDIOS Xilinx - ISE - C:\Xil... 3 IPMsg
11:14 PM

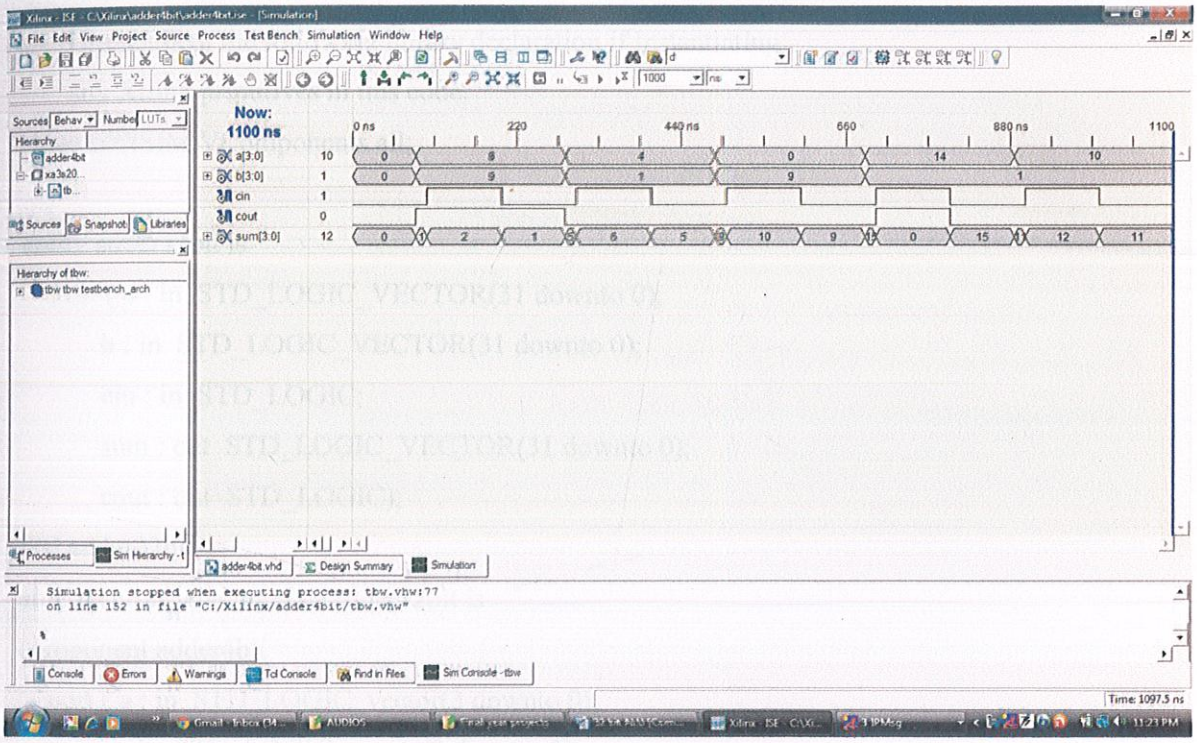
```
Simulation stopped when executing process: tbw.vhw:77
on line 146 in file "C:/Xilinx/full_adder/tbw.vhw"
```

Figure 8.2: Output window for Full adder

8.2 4 bit adder

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

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

entity adder4bit is
   Port ( a : in  STD_LOGIC_vector(3 downto 0);
          b : in  STD_LOGIC_vector(3 downto 0);
          cin : in  STD_LOGIC;
          cout : out  STD_LOGIC;
          sum : out  STD_LOGIC_vector(3 downto 0));
end adder4bit;

architecture Behavioral of adder4bit is
component full_adder
  port ( a : in  STD_LOGIC;
         b : in  STD_LOGIC;
        cin : in  STD_LOGIC;
        cout : out  STD_LOGIC;
         sum : out  STD_LOGIC);
end component ;
for u1,u2,u3,u4 : full_adder
 use entity work.full_adder(Behavioral);
signal t: STD_LOGIC_vector(2 downto 0);
begin
```

```
u1 : full_adder port map (a=>a(0),b=>b(0),cin=>cin,cout=>t(0),sum=>sum(0));
u2 : full_adder port map (a=>a(1),b=>b(1),cin=>t(0),cout=>t(1),sum=>sum(1));
u3 : full_adder port map (a=>a(2),b=>b(2),cin=>t(1),cout=>t(2),sum=>sum(2));
u4 : full_adder port map (a=>a(3),b=>b(3),cin=>t(2),cout=>cout,sum=>sum(3));

end Behavioral;
```



File Edit View Project Source Process Test Bench Simulation Window Help
Now: 1100 ns
0 ns
220
440 ns
660
880 ns
1100
a[3:0] 10
0 8 4 0 14 10
b[3:0] 1
0 9 1 9 1
cin 1
cout 0
sum[3:0] 12
0 1 2 1 5 6 5 9 10 9 1 0 15 1 12 11
Sources Behav Number LUTs
Hierarchy
adder4bit
xa3s20...
Sources Snapshot Libraries
Hierarchy of tbw:
tbw tbw testbench_arch
Processes Sim Hierarchy - t
adder4bit.vhd Design Summary Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - tbw
Time: 1097.5 ns

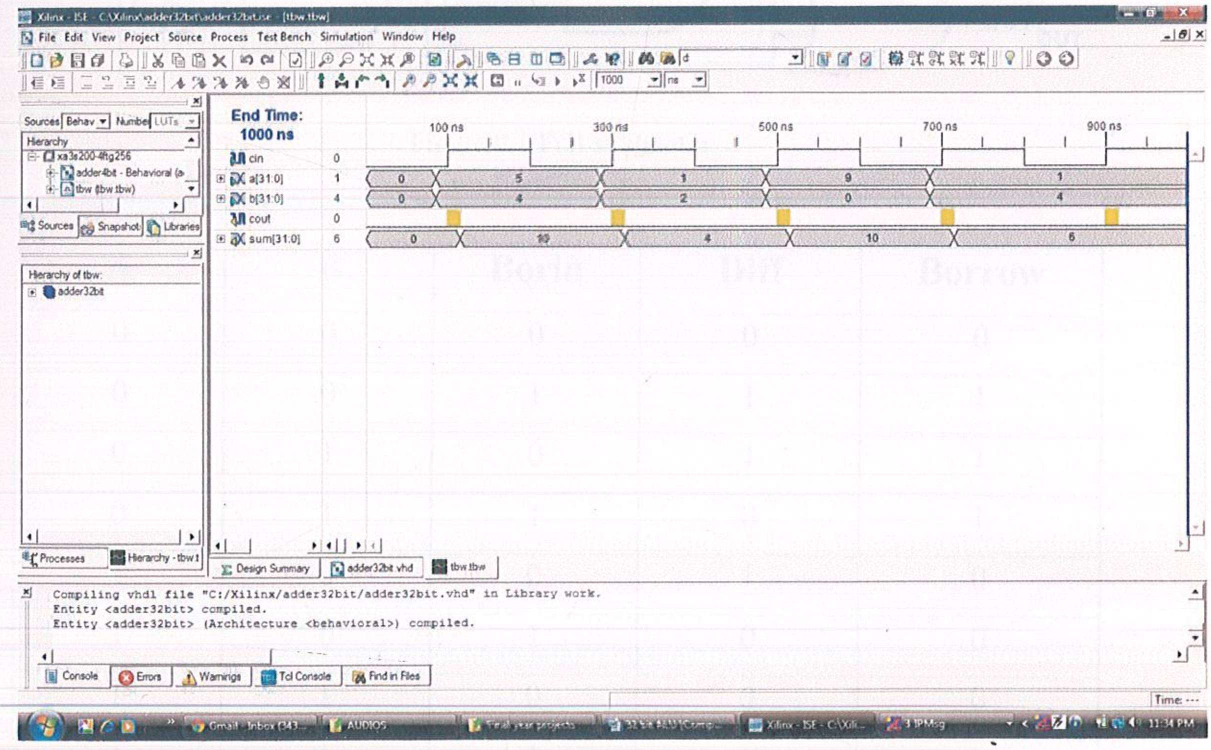
```
Simulation stopped when executing process: tbw.vhw:77
on line 152 in file "C:/Xilinx/adder4bit/tbw.vhw"
```

Figure 8.3: Output window for 4 bit adder

8.3 32 bit adder

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

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

entity adder32bit is
  Port ( a : in  STD_LOGIC_VECTOR(31 downto 0);
       b : in  STD_LOGIC_VECTOR(31 downto 0);
       cin : in  STD_LOGIC;
       sum : out  STD_LOGIC_VECTOR(31 downto 0);
       cout : out  STD_LOGIC);
end adder32bit;
architecture Behavioral of adder32bit is
component adder4bit
  port ( a : in  STD_LOGIC_vector(3 downto 0);
       b : in  STD_LOGIC_vector(3 downto 0);
       cin : in  STD_LOGIC;
       cout : out  STD_LOGIC;
       sum : out  STD_LOGIC_vector(3 downto 0));
end component;
for u1,u2,u3,u4,u5,u6,u7,u8 : adder4bit
 use entity work.adder4bit(Behavioral);
signal t: STD_LOGIC_vector(6 downto 0);
begin
u1 : adder4bit port map (a=>a(3 downto 0),b=>b(3 downto 0),
   cin=>cin,cout=>t(0),sum=>sum(3 downto 0));
```

```
u2 : adder4bit port map (a=>a(7 downto 4),b=>b(7 downto 4),
    cin=>t(0),cout=>t(1),sum=>sum(7 downto 4));
u3 : adder4bit port map (a=>a(11 downto 8),b=>b(11 downto 8),
    cin=>t(1),cout=>t(2),sum=>sum(11 downto 8));
u4 : adder4bit port map (a=>a(15 downto 12),b=>b(15 downto 12),
    cin=>t(2),cout=>t(3),sum=>sum(15 downto 12));
u5 : adder4bit port map (a=>a(19 downto 16),b=>b(19 downto 16),
    cin=>t(3),cout=>t(4),sum=>sum(19 downto 16));
u6 : adder4bit port map (a=>a(23 downto 20),b=>b(23 downto 20),
    cin=>t(4),cout=>t(5),sum=>sum(23 downto 20));
u7 : adder4bit port map (a=>a(27 downto 24),b=>b(27 downto 24),
    cin=>t(5),cout=>t(6),sum=>sum(27 downto 24));
u8 : adder4bit port map (a=>a(31 downto 28),b=>b(31 downto 28),
    cin=>t(6),cout=>cout,sum=>sum(31 downto 28));
end Behavioral;
```



```
Compiling vhdl file "C:/Xilinx/adder32bit/adder32bit.vhd" in Library work.
Entity <adder32bit> compiled.
Entity <adder32bit> (Architecture <behavioral>) compiled.
```

Figure 8.4: Output window for 32 bit adder

CHAPTER 9

SUBTRACTOR

Subtractor consists of two inputs and two outputs. The outputs in subtractor are difference bit and a borrow bit. The difference bit is the subtraction of the bits and borrow is what we have

borrowed from the previous bit.

Like the half-adder, the half-sub can be used to implement a full-sub, shown in Figure 9.1

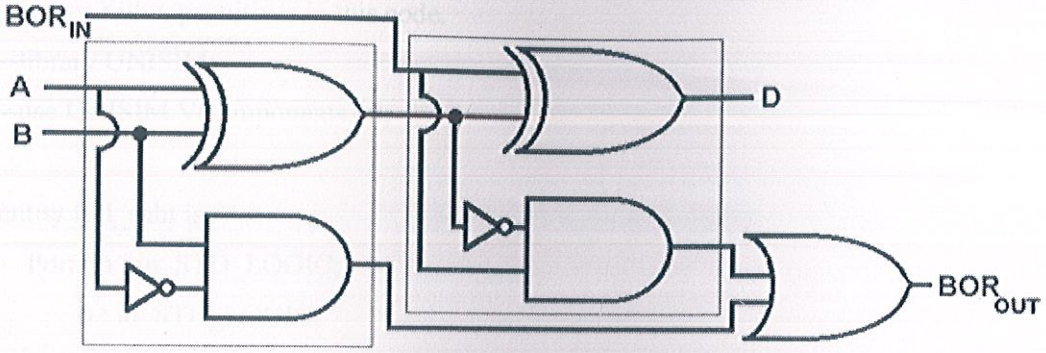


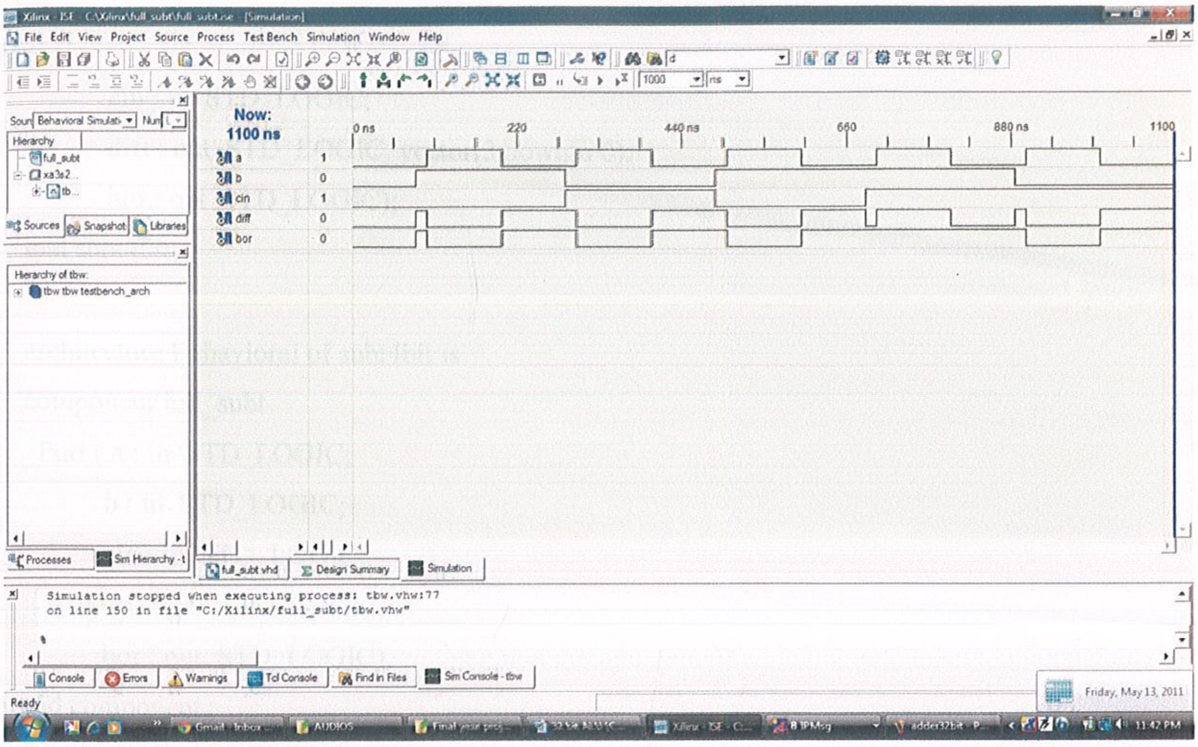
Figure 9.1 Full subtractor

A	B	Borin	Diff	Borrow
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

Table 9.1 Truth table of full subtractor

9.1 Full Subtractor

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

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

entity full_subt is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          cin : in  STD_LOGIC;
          diff : out  STD_LOGIC;
          bor : out  STD_LOGIC);
end full_subt;

architecture Behavioral of full_subt is

begin
process(a,b,cin)
                begin
                diff <= a xor b xor cin;
                bor <= ((not a) and b) or (b and cin) or ((not a) and cin);
end process;

end Behavioral;
```



```
Simulation stopped when executing process: tbw.vhw:77
on line 150 in file "C:/Xilinx/full_subt/tbw.vhw"
```

Figure 9.2: Output window for Full subtractor

9.2 4 Bit Subtractor

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating

---- any Xilinx primitives in this code.

--library UNISIM;

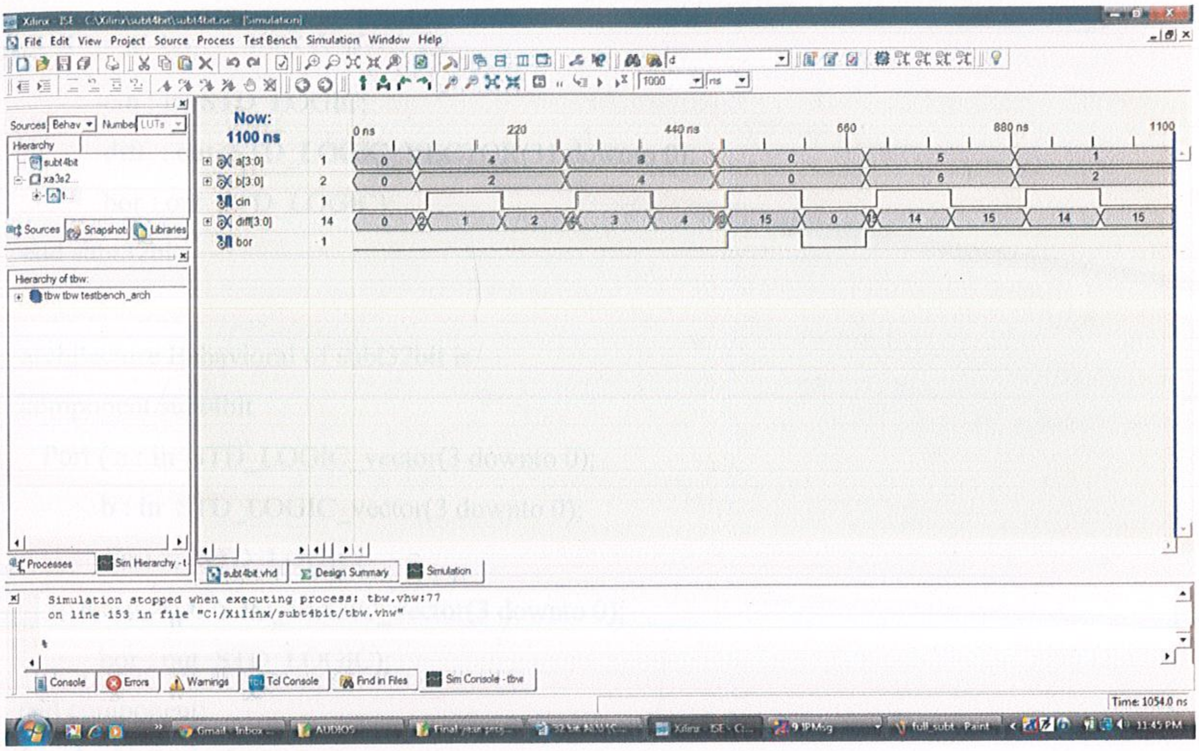
--use UNISIM.VComponents.all;

entity subt4bit is

  Port (a : in STD_LOGIC_vector(3 downto 0);

       b : in STD_LOGIC_vector(3 downto 0);

```
          cin : in  STD_LOGIC;
          diff : out  STD_LOGIC_vector(3 downto 0);
          bor : out  STD_LOGIC);
end subt4bit;

architecture Behavioral of subt4bit is
component full_subt
  Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          cin : in  STD_LOGIC;
          diff : out  STD_LOGIC;
          bor : out  STD_LOGIC);
end component ;
for u1,u2,u3,u4 : full_subt
 use entity work.full_subt(Behavioral);
signal t: STD_LOGIC_vector(2 downto 0);
begin
u1 : full_subt port map (a=>a(0),b=>b(0),cin=>cin,bor=>t(0),diff=>diff(0));
u2 : full_subt port map (a=>a(1),b=>b(1),cin=>t(0),bor=>t(1),diff=>diff(1));
u3 : full_subt port map (a=>a(2),b=>b(2),cin=>t(1),bor=>t(2),diff=>diff(2));
u4 : full_subt port map (a=>a(3),b=>b(3),cin=>t(2),bor=>bor,diff=>diff(3));
end Behavioral;
```


Xilinx - ISE - C:\Xilinx\subt4bit\subt4bit.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
1000 ns
Sources Behav Number LUTs
Hierarchy
subt4bit
xa3s2...
Sources Snapshot Libraries
Hierarchy of tbw:
tbw tbw testbench_arch
Processes Sim Hierarchy
Now:
1100 ns
0 ns 220 440 ns 660 880 ns 1100
a[3:0] 1 0 4 8 0 5 1
b[3:0] 2 0 2 4 0 6 2
cin 1
diff[3:0] 14 0 1 2 3 4 15 0 14 15 14 15
bor 1
subt4bit.vhd Design Summary Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - tbw
Time: 1054.0 ns
Gmail - Inbox ... AUDIOS Final year proj... 32 bit ALU ... Xilinx - ISE - C... 9 IPMsg full subt - Paint 11:45 PM

```
Simulation stopped when executing process: tbw.vhw:77
on line 153 in file "C:/Xilinx/subt4bit/tbw.vhw"
```

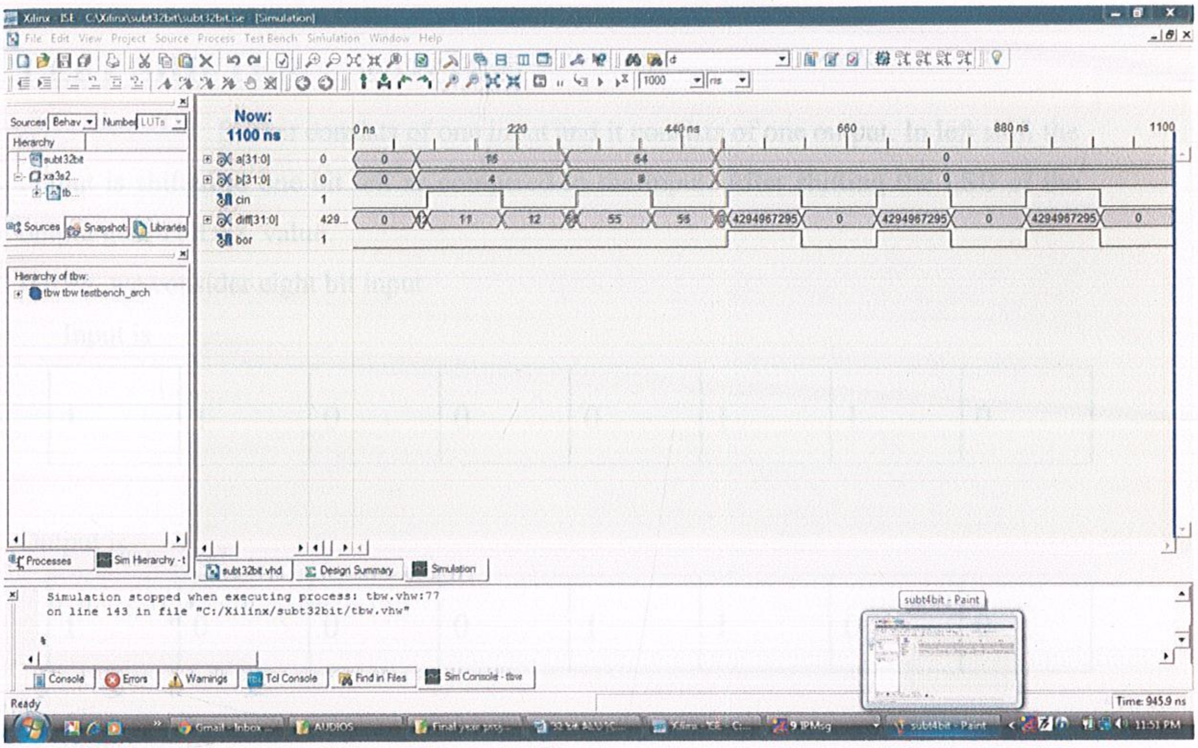
Figure 9.3: Output window for 4 bit subtractor

9.3 32 Bit Subtractor

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

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

entity subt32bit is
   Port ( a : in  STD_LOGIC_VECTOR(31 downto 0);
          b : in  STD_LOGIC_VECTOR(31 downto 0);
```

```
        cin : in  STD_LOGIC;
        diff  : out  STD_LOGIC_VECTOR(31 downto 0);
        bor : out  STD_LOGIC);
end subt32bit;

architecture Behavioral of subt32bit is
component subt4bit
  Port ( a : in  STD_LOGIC_vector(3 downto 0);
        b : in  STD_LOGIC_vector(3 downto 0);
        cin : in  STD_LOGIC;
        diff : out  STD_LOGIC_vector(3 downto 0);
        bor : out  STD_LOGIC);
end component;
for u1,u2,u3,u4,u5,u6,u7,u8 : subt4bit
 use entity work.subt4bit(Behavioral);
signal t: STD_LOGIC_vector(6 downto 0);
begin
u1 :subt4bit port map (a=>a(3 downto 0),b=>b(3 downto 0),
   cin=>cin,bor=>t(0),diff=>diff(3 downto 0));
u2 :subt4bit port map (a=>a(7 downto 4),b=>b(7 downto 4),
   cin=>t(0),bor=>t(1),diff=>diff(7 downto 4));
u3 :subt4bit port map (a=>a(11 downto 8),b=>b(11 downto 8),
   cin=>t(1),bor=>t(2),diff=>diff(11 downto 8));
u4 :subt4bit port map (a=>a(15 downto 12),b=>b(15 downto 12),
   cin=>t(2),bor=>t(3),diff=>diff(15 downto 12));
u5 :subt4bit port map (a=>a(19 downto 16),b=>b(19 downto 16),
   cin=>t(3),bor=>t(4),diff=>diff(19 downto 16));
u6 :subt4bit port map (a=>a(23 downto 20),b=>b(23 downto 20),
   cin=>t(4),bor=>t(5),diff=>diff(23 downto 20));
u7 :subt4bit port map (a=>a(27 downto 24),b=>b(27 downto 24),
   cin=>t(5),bor=>t(6),diff=>diff(27 downto 24));
```

```
u8 :subt4bit port map (a=>a(31 downto 28),b=>b(31 downto 28),
    cin=>t(6),bor=>bor,diff=>diff(31 downto 28));


end Behavioral;
```


Xilinx - ISE - C:\Xilinx\subt32bit\subt32bit.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Sources Behav Number LUTs
Hierarchy
subt32bit
xa3s2...
tb...
Sources Snapshot Libraries
Hierarchy of tbw:
tbw tbw testbench_arch
Processes Sim Hierarchy - t
Now: 1100 ns
0 ns
220
440 ns
660
880 ns
1100
a[31:0] 0
b[31:0] 0
cin 1
diff[31:0] 429...
bor 1
0
16
64
0
0
4
8
0
0
11
12
55
56
4294967295
0
4294967295
0
4294967295
0
subt32bit.vhd Design Summary Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - tbw
subt4bit - Paint
Ready
Time: 945.9 ns

```
Simulation stopped when executing process: tbw.vhw:77
on line 143 in file "C:/Xilinx/subt32bit/tbw.vhw"
```

Figure 9.4: Output window for 32 bit subtractor

CHAPTER 10

LEFT SHIFTER

Shifter consists of one input and it consists of one output. In left shift the output is shifted to one bit left as compared to the input. After shifting the LSB of the output gets a NULL value.

For eg. we consider eight bit input

Input is

1	1	0	0	0	1	1	0

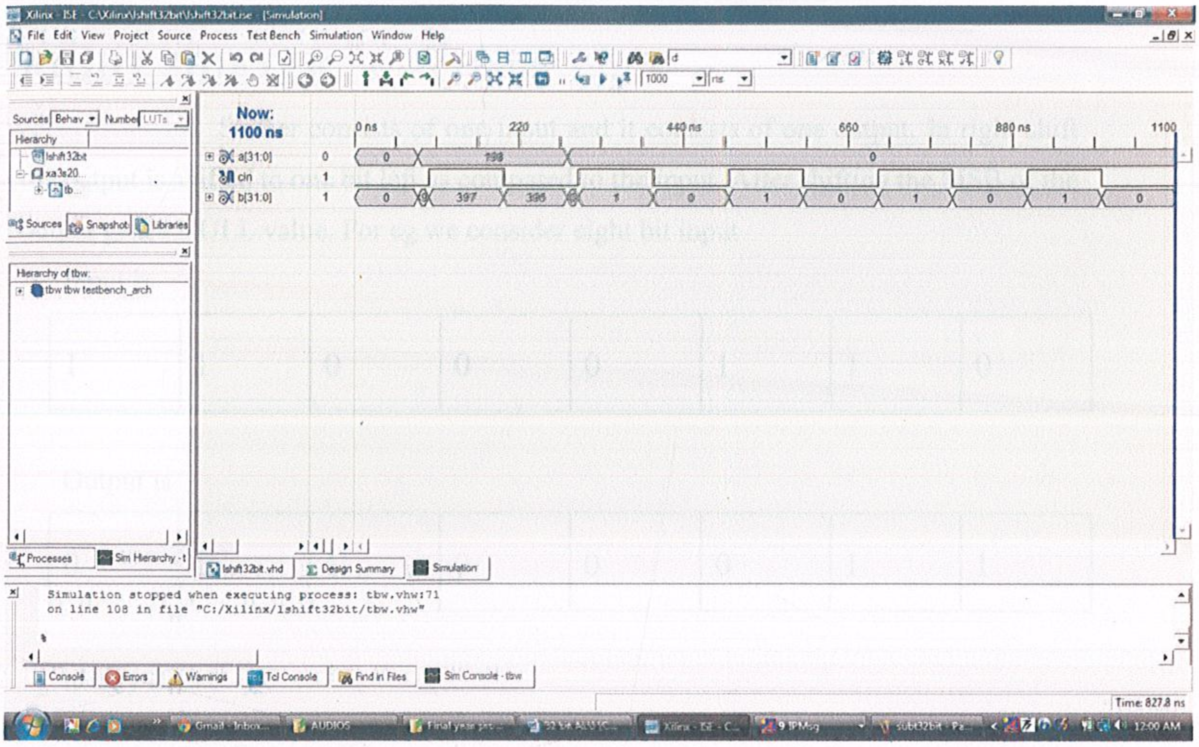
Output is

1	0	0	0	1	1	0	0

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

entity lshift32bit is
   Port ( a    : in    std_logic_vector ( 31 DOWNTO 0);
      cin    : in    std_logic;
      b    : out    std_logic_vector ( 31 DOWNTO 0));

end lshift32bit;
architecture Behavioral of lshift32bit is
begin
    process(a,cin)
```

```
begin
b(0) <= cin;
b(1) <= a(0);
b(2)<=a(1);
b(3)<=a(2);
b(4)<=a(3);
b(5)<=a(4);
b(6)<=a(5);
b(7)<=a(6);
b(8)<=a(7);
b(9)<=a(8);
b(10)<=a(9);
b(11)<=a(10);
b(12)<=a(11);
b(13)<=a(12);
b(14)<=a(13);
b(15)<=a(14);
b(16)<=a(15);
b(17)<=a(16);
b(18)<=a(17);
b(19)<=a(18);
b(20)<=a(19);
b(21)<=a(20);
b(22)<=a(21);
b(23)<=a(22);
b(24)<=a(23);
b(25)<=a(24);
b(26)<=a(25);
b(27)<=a(26);
b(28)<=a(27);
b(29)<=a(28);
```

```
        b(30)<=a(29);
        b(31)<=a(30);
    end process;
    end Behavioral;
```



Xilinx - ISE - C:\Xilinx\lshift32bit\lshift32bit.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Sources Behav
Number LUTs
Hierarchy
lshift32bit
xa3s20...
tb...
Sources Snapshot Libraries
Hierarchy of tbw:
tbw tbw testbench_arch
Processes Sim Hierarchy - t
Now:
1100 ns
0 ns
220
440 ns
660
880 ns
1100
a[31:0] 0
cin 1
b[31:0] 1
0
198
0
0
397
396
1
0
1
0
1
0
1
0
lshift32bit.vhd Design Summary Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - tbw
Time: 827.8 ns
Gmail - Inbox... AUDIOS Final year pro... 32 bit ALU IC... Xilinx - ISE - C... 9 IPMsg subt32bit - Pa...
12:00 AM

```
Simulation stopped when executing process: tbw.vhw:71
on line 108 in file "C:/Xilinx/lshift32bit/tbw.vhw"
```

Figure 10.1: Output window for Left Shifter

CHAPTER 11

RIGHT SHIFTER

Shifter consists of one input and it consists of one output. In right shift the output is shifted to one bit left as compared to the input. After shifting the MSB of the output gets a NULL value. For eg we consider eight bit input

Input is

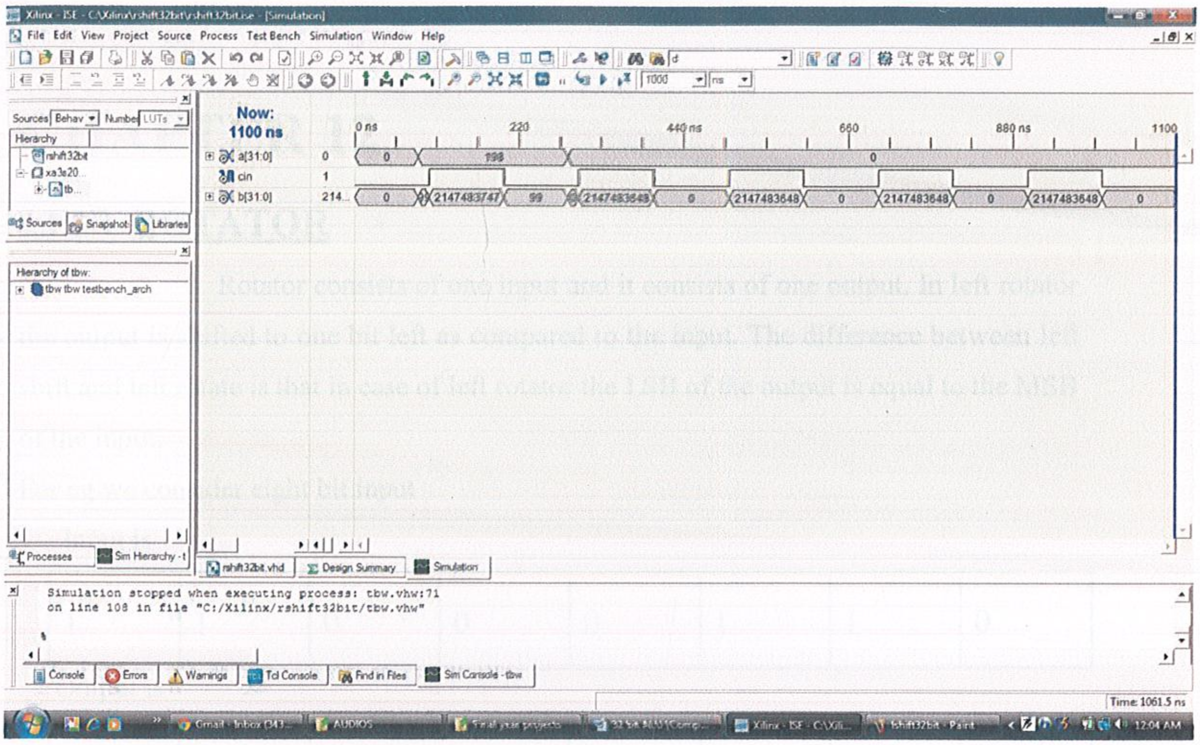
1	1	0	0	0	1	1	0

Output is

0	1	1	0	0	0	1	1

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


entity rshift32bit is
  Port ( a    : in    std_logic_vector ( 31 DOWNTO 0);
       cin  : in    std_logic ;
       b    : out  std_logic_vector ( 31 DOWNTO 0));
 end rshift32bit;
architecture Behavioral of rshift32bit is
begin
process(a,cin)
begin
 b(0) <= a(1);
 b(1)<=a(2);
 b(2)<=a(3);
 b(3)<=a(4);
 b(4)<=a(5);
 b(5)<=a(6);
 b(6)<=a(7);
 b(7)<=a(8);
 b(8)<=a(9);
 b(9)<=a(10);
 b(10)<=a(11);
 b(11)<=a(12);
 b(12)<=a(13);
 b(13)<=a(14);
 b(14)<=a(15);
 b(15)<=a(16);
 b(16)<=a(17);
 b(17)<=a(18);
 b(18)<=a(19);
 b(19)<=a(20);
 b(20)<=a(21);
 b(21)<=a(22);
 b(22)<=a(23);
 b(23)<=a(24);
 b(24)<=a(25);
 b(25)<=a(26);
 b(26)<=a(27);
 b(27)<=a(28);
 b(28)<=a(29);
 b(29)<=a(30);
 b(30)<=a(31);
 b(31)<=cin;
end process;
```

end Behavioral;



```
Simulation stopped when executing process: tbw.vhw:71
on line 108 in file "C:/Xilinx/rshift32bit/tbw.vhw"
```

Figure 11.1: Output window for Right Shifter

CHAPTER 12

LEFT ROTATOR

Rotator consists of one input and it consists of one output. In left rotator the output is shifted to one bit left as compared to the input. The difference between left shift and left rotate is that in case of left rotator the LSB of the output is equal to the MSB of the input.

For eg we consider eight bit input

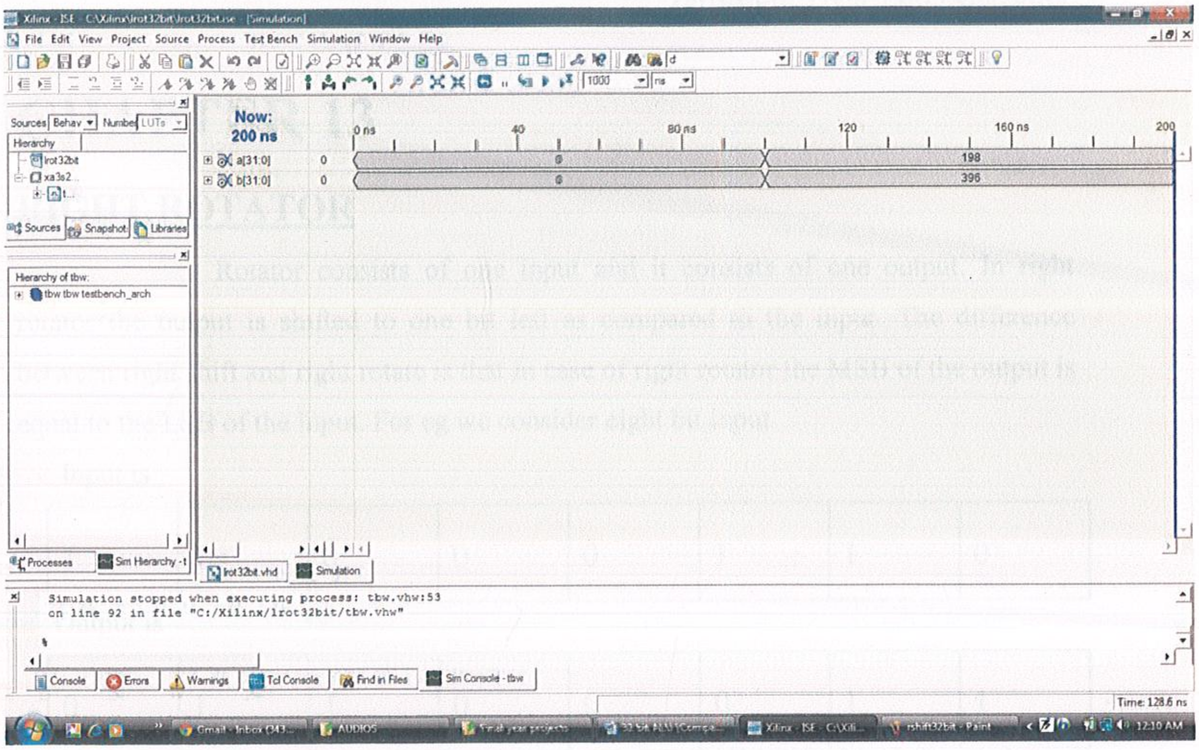
Input is

1	1	0	0	0	1	1	0

Output is

1	0	0	0	1	1	0	1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
    entity lrot32bit is
        Port ( a     : in     std_logic_vector ( 31 DOWNTO 0);
               b     : out    std_logic_vector ( 31 DOWNTO 0));
    end lrot32bit;
    architecture Behavioral of lrot32bit is
    begin
     process(a)
      begin
       b(0) <= a(31) ;
       b(1) <= a(0);
       b(2)<=a(1);
    b(3)<=a(2);
    b(4)<=a(3);
    b(5)<=a(4);
    b(6)<=a(5);
    b(7)<=a(6);
    b(8)<=a(7);
    b(9)<=a(8);
    b(10)<=a(9);
    b(11)<=a(10);
    b(12)<=a(11);
    b(13)<=a(12);
    b(14)<=a(13);
    b(15)<=a(14);
    b(16)<=a(15);
    b(17)<=a(16);
    b(18)<=a(17);
    b(19)<=a(18);
    b(20)<=a(19);
    b(21)<=a(20);
    b(22)<=a(21);
    b(23)<=a(22);
    b(24)<=a(23);
    b(25)<=a(24);
    b(26)<=a(25);
    b(27)<=a(26);
    b(28)<=a(27);
    b(29)<=a(28);
    b(30)<=a(29);
    b(31)<=a(30);
end process;
end Behavioral;
```



Xilinx - ISE - C:\Xilinx\lrot32bit\lrot32bit.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Sources Behav Number LUTs
Hierarchy
lrot32bit
xa3s2...
Sources Snapshot Libraries
Hierarchy of tbw:
tbw tbw testbench_arch
Processes Sim Hierarchy
Now: 200 ns
0 ns
40
80 ns
120
160 ns
200
a[31:0] 0
b[31:0] 0
0
198
0
396
lrot32bit.vhd Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - tbw
Time: 128.6 ns
12:10 AM

```
Simulation stopped when executing process: tbw.vhw:53
on line 92 in file "C:/Xilinx/lrot32bit/tbw.vhw"
```

Figure 12.1: Output window for Left Rotator

CHAPTER 13

RIGHT ROTATOR

Rotator consists of one input and it consists of one output. In right rotator the output is shifted to one bit left as compared to the input. The difference between right shift and right rotate is that in case of right rotator the MSB of the output is equal to the LSB of the input. For eg we consider eight bit input

Input is

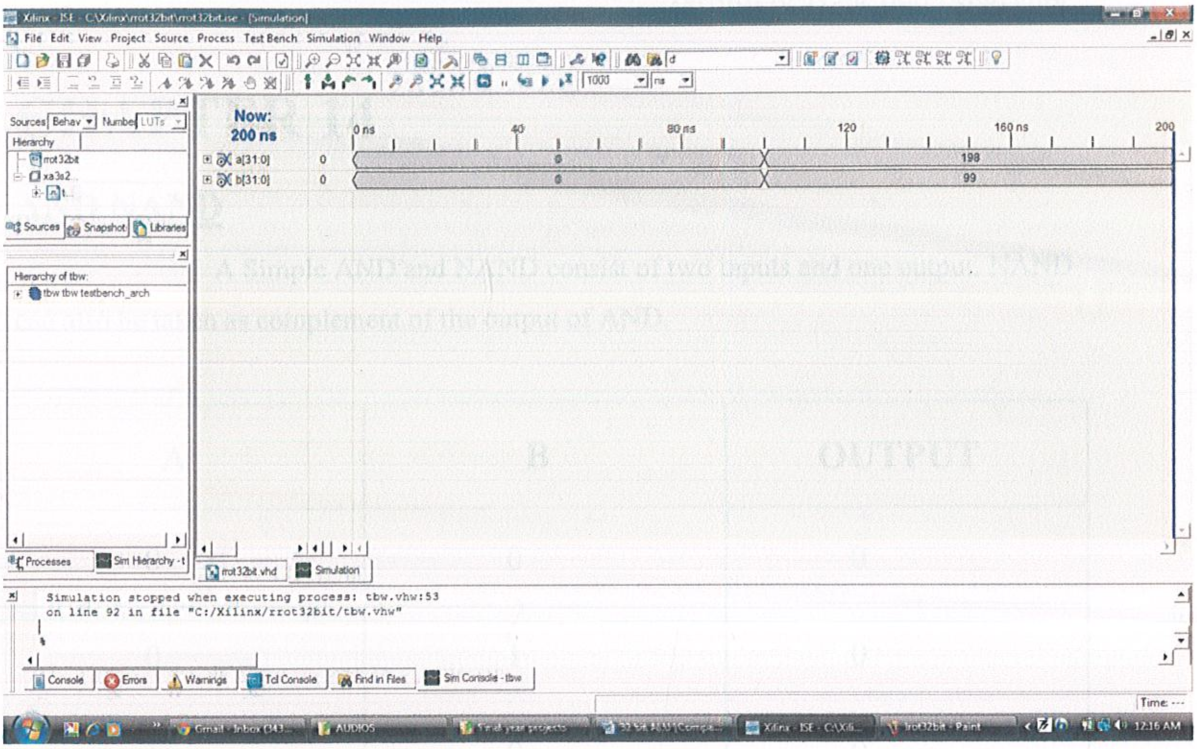
1	1	0	0	0	1	1	0

Output is

0	1	1	0	0	0	1	1

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

entity rrot32bit is
  Port ( a    : in    std_logic_vector ( 31 DOWNTO 0);
     b    : out   std_logic_vector ( 31 DOWNTO 0));

end rrot32bit;
architecture Behavioral of rrot32bit is
begin
process(a)
begin
    b(0) <= a(1);
    b(1)<=a(2);
     b(2)<=a(3);
```

```
        b(3)<=a(4);
        b(4)<=a(5);
        b(5)<=a(6);
        b(6)<=a(7);
        b(7)<=a(8);
        b(8)<=a(9);
        b(9)<=a(10);
        b(10)<=a(11);
        b(11)<=a(12);
        b(12)<=a(13);
        b(13)<=a(14);
        b(14)<=a(15);
        b(15)<=a(16);
        b(16)<=a(17);
        b(17)<=a(18);
        b(18)<=a(19);
        b(19)<=a(20);
        b(20)<=a(21);
        b(21)<=a(22);
        b(22)<=a(23);
        b(23)<=a(24);
        b(24)<=a(25);
        b(25)<=a(26);
        b(26)<=a(27);
        b(27)<=a(28);
        b(28)<=a(29);
        b(29)<=a(30);
        b(30)<=a(31);
        b(31)<=a(0);
end process;
end Behavioral;
```



```
Simulation stopped when executing process: tbw.vhw:53
on line 92 in file "C:/Xilinx/rrot32bit/tbw.vhw"
```

Figure 13.1: Output window for Right Rotator

CHAPTER 14

AND NAND

A Simple AND and NAND consist of two inputs and one output. NAND can also be taken as complement of the output of AND.

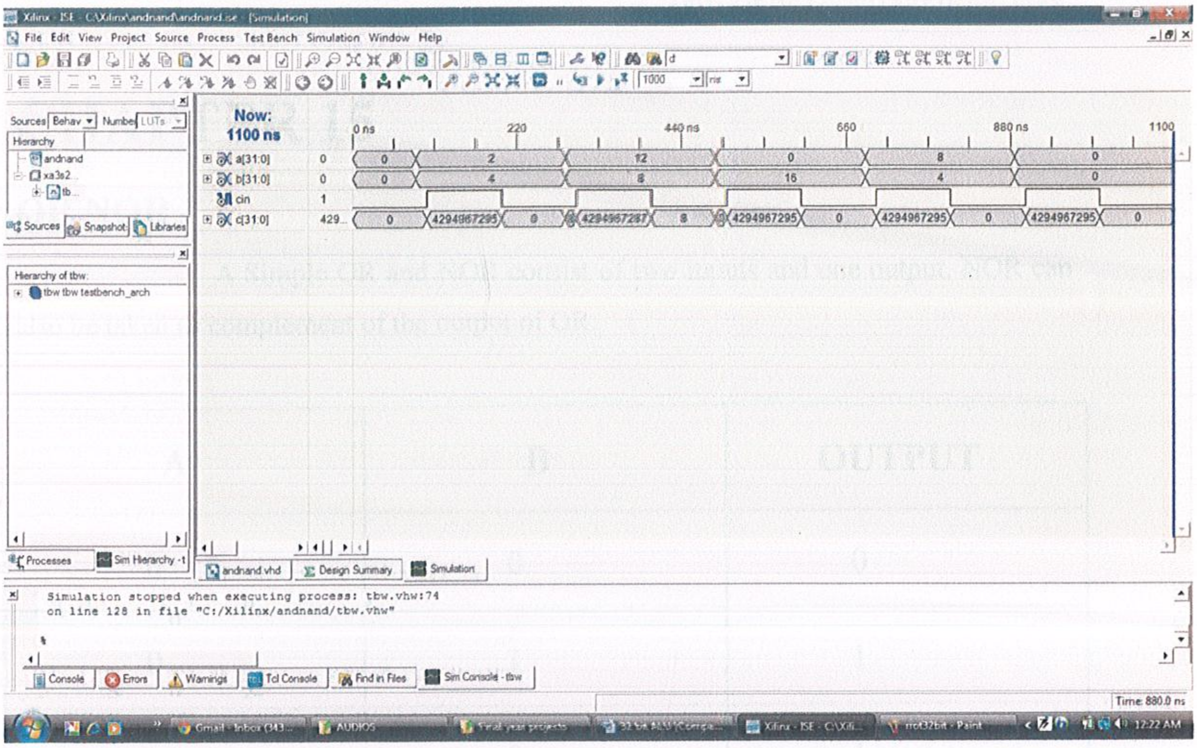
A	B	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

Table 14.1: Truth table of AND.

A	B	OUTPUT
0	0	1
0	1	1
1	0	1
1	1	0

Table 14.2: Truth table of NAND

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

entity andnand is
   Port ( a : in  STD_LOGIC_vector(31 downto 0);
          b : in  STD_LOGIC_vector(31 downto 0);
          cin : in  STD_LOGIC;
          c : out  STD_LOGIC_vector(31 downto 0));
end andnand;
architecture Behavioral of andnand is

begin
 process(a,b,cin)
  begin
  if (cin ='0')
   then c<= a and b;
 else
  c<= a nand b ;
  end if;
 end process;
end Behavioral;
```



```
Simulation stopped when executing process: tbw.vhw:74
on line 128 in file "C:/Xilinx/andnand/tbw.vhw"
```

Figure 14.1: Output window for And Nand

CHAPTER 15

OR NOR

A Simple OR and NOR consist of two inputs and one output. NOR can also be taken as complement of the output of OR.

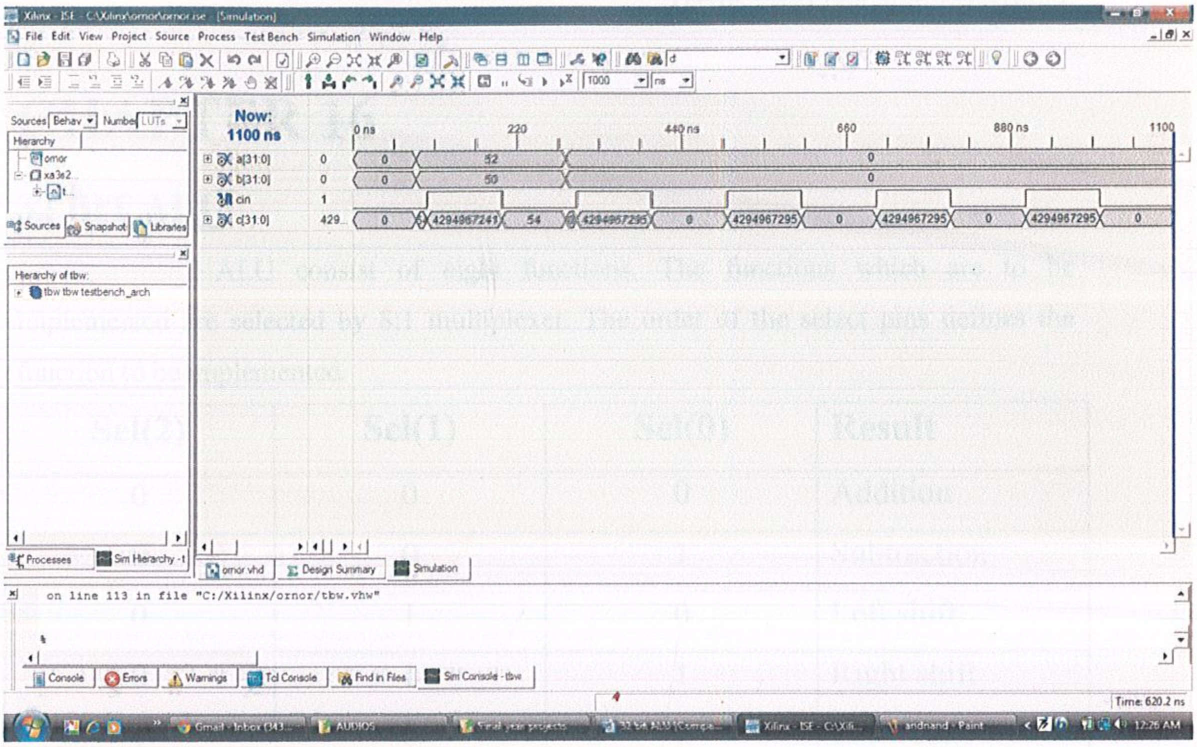
A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

Table 15.1: Truth table of OR

A	B	OUTPUT
0	0	1
0	1	0
1	0	0
1	1	0

Table 15.2: Truth table of NOR

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

entity ornor is
 Port ( a : in  STD_LOGIC_vector(31 downto 0);
        b : in  STD_LOGIC_vector(31 downto 0);
        cin : in  STD_LOGIC;
        c : out  STD_LOGIC_vector(31 downto 0));
end ornor;
architecture Behavioral of ornor is
begin
process(a,b,cin)
  begin
   if (cin ='0')
   then c<= a or b;
 else
  c<= a nor b ;
  end if;
 end process;
end Behavioral;
```



```
on line 113 in file "C:/Xilinx/ornor/tbw.vhw"
```

Figure 15.1: Output window for Or Nor

CHAPTER 16

32 BIT ALU

ALU consist of eight functions. The functions which are to be implemented are selected by 8:1 multiplexer. The order of the select pins defines the function to be implemented.

Sel(2)	Sel(1)	Sel(0)	Result
0	0	0	Addition
0	0	1	Subtraction
0	1	0	Left shift
0	1	1	Right shift
1	0	0	Left rotate
1	0	1	Right rotate
1	1	0	If cin=0 and If cin=1 nand
1	1	1	If cin =0 or If cin=1 nor

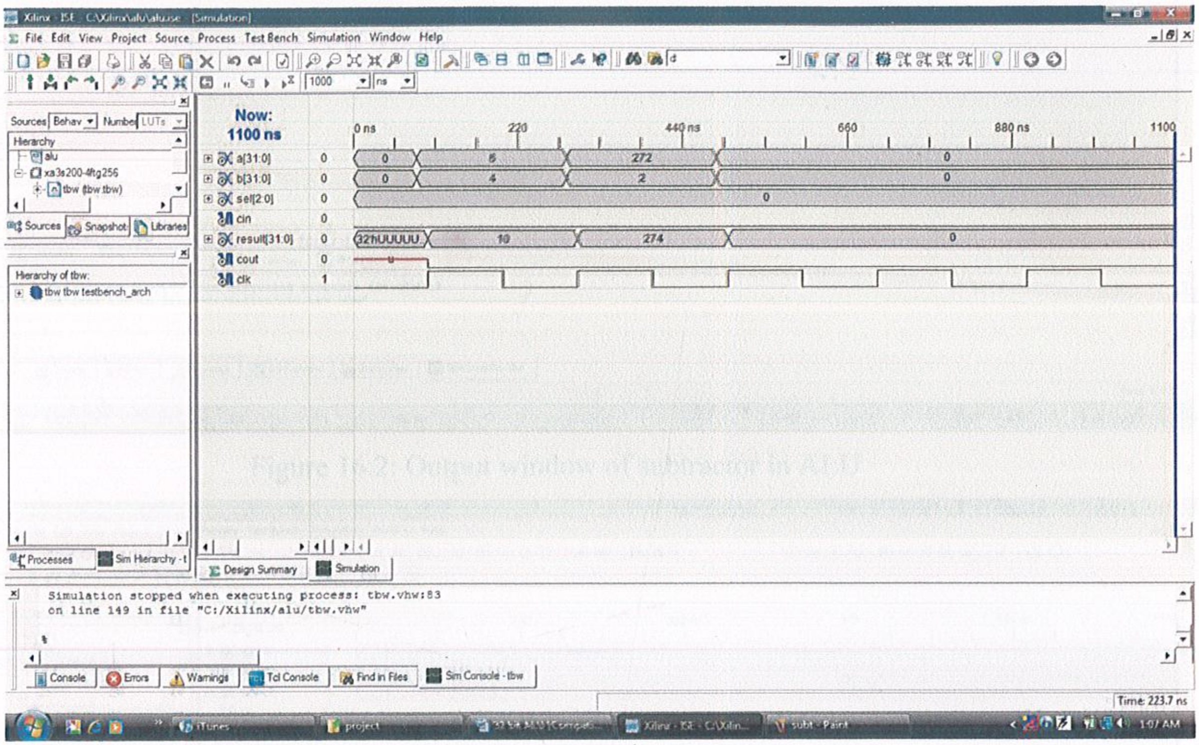
Table 16.1: Function table of 8:1 multiplexer.

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

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

entity alu is
  Port ( a  : in  STD_LOGIC_VECTOR(31 downto 0);
         b  : in  STD_LOGIC_VECTOR(31 downto 0);
        sel  : in std_logic_vector(2 downto 0);
        cin  :  in  STD_LOGIC;
        result : out  STD_LOGIC_VECTOR(31 downto 0);
        cout  : out  STD_LOGIC;
          clk : in STD_LOGIC);
end alu;

architecture Behavioral of alu is


component adder32bit
  Port ( a : in  STD_LOGIC_VECTOR(31 downto 0);
        b : in  STD_LOGIC_VECTOR(31 downto 0);
        cin : in  STD_LOGIC;
        sum : out  STD_LOGIC_VECTOR(31 downto 0);
        cout : out  STD_LOGIC);
end component;

for u1  : adder32bit
```

```
use entity work.adder32bit(Behavioral);

component subt32bit
   Port ( a : in  STD_LOGIC_VECTOR(31 downto 0);
          b : in  STD_LOGIC_VECTOR(31 downto 0);
          cin : in  STD_LOGIC;
         diff  : out  STD_LOGIC_VECTOR(31 downto 0);
          bor : out  STD_LOGIC);
end component;

for u2 : subt32bit
 use entity work.subt32bit(Behavioral);

component lshift32bit
  Port ( a     : in     std_logic_vector ( 31 DOWNTO 0);
         cin     : in     std_logic;
          b     : out    std_logic_vector ( 31 DOWNTO 0));
end component;

for u3 : lshift32bit
 use entity work.lshift32bit(Behavioral);

component rshift32bit
  Port ( a     : in     std_logic_vector ( 31 DOWNTO 0);
         cin     : in     std_logic;
          b     : out    std_logic_vector ( 31 DOWNTO 0));
end component;

for u4 : rshift32bit
 use entity work.rshift32bit(Behavioral);
```

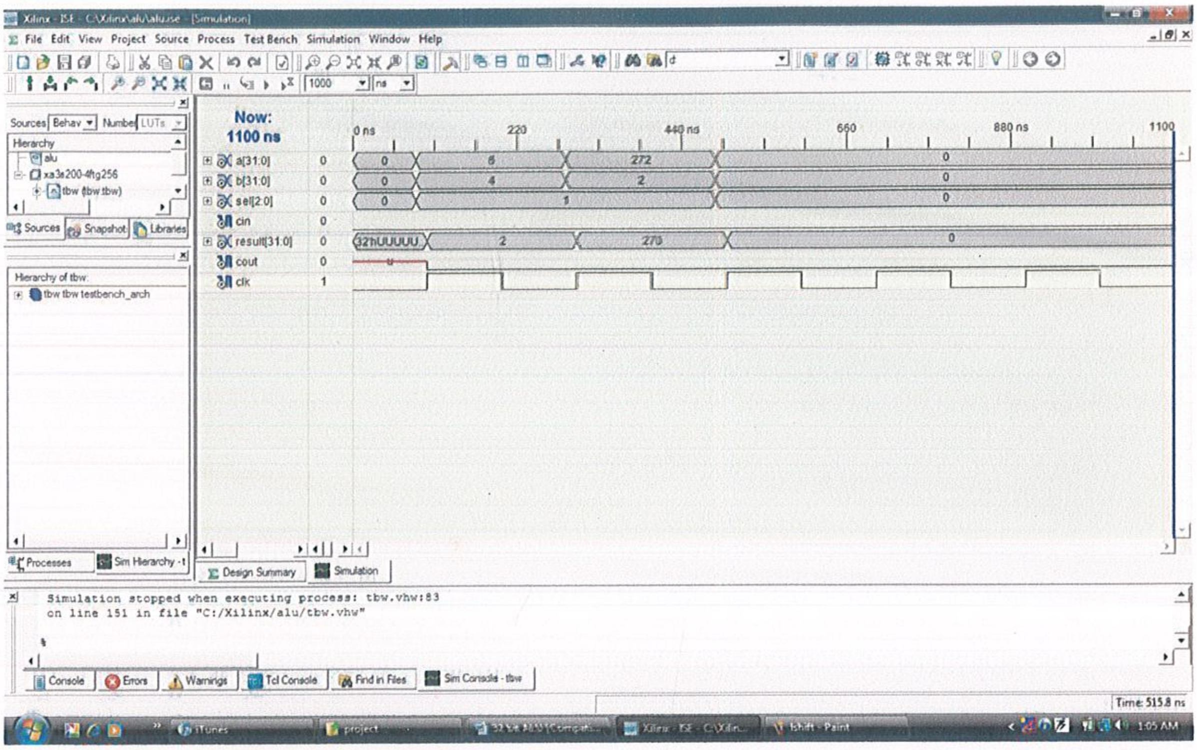
```
component lrot32bit
  Port ( a     : in     std_logic_vector ( 31 DOWNTO 0);
         b     : out    std_logic_vector ( 31 DOWNTO 0));
end component;

for u5 : lrot32bit
 use entity work.lrot32bit(Behavioral);

 component rrot32bit
  Port ( a     : in     std_logic_vector ( 31 DOWNTO 0);
         b     : out    std_logic_vector ( 31 DOWNTO 0));
end component;

for u6 : rrot32bit
 use entity work.rrot32bit(Behavioral);

component andnand
   Port ( a : in  STD_LOGIC_vector(31 downto 0);
         b : in  STD_LOGIC_vector(31 downto 0);
         cin : in  STD_LOGIC;
         c : out  STD_LOGIC_vector(31 downto 0));
end component;

for u7 : andnand
 use entity work.andnand(Behavioral);

component ornor
   Port ( a : in  STD_LOGIC_vector(31 downto 0);
         b : in  STD_LOGIC_vector(31 downto 0);
         cin : in  STD_LOGIC;
         c : out  STD_LOGIC_vector(31 downto 0));
```

```
end component;

for u8 : ornor
 use entity work.ornor(Behavioral);

 signal ta,tb : std_logic ;
 signal t1,t2,t3,t4,t5,t6,t7,t8: STD_LOGIC_vector(31 downto 0);
begin
  u1 : adder32bit port map (a=>a,b=>b,cin=>cin,cout=>ta,sum=>t1);
  u2 : subt32bit port map (a=>a,b=>b,cin=>cin,bor=>tb,diff=>t2);
  u3 : lshift32bit port map (a=>a,cin=>cin,b=>t3);
  u4 : rshift32bit port map (a=>a,cin=>cin,b=>t4);
  u5 : lrot32bit port map (a=>a,b=>t5);
  u6 : rrot32bit port map (a=>a,b=>t6);
  u7 : andnand port map (a=>a,b=>b,cin=>cin,c=>t7);
  u8 : ornor port map (a=>a,b=>b,cin=>cin,c=>t8);

 process(sel,clk)
   begin
                  if clk = '1' then
                   case sel is
                        when "000" =>result<=t1;
                                         cout<=ta;
                        when "001" =>result<=t2;
                                         cout<=tb;
                        when "010" =>result<=t3;
                        when "011" =>result<=t4;
                        when "100" =>result<=t5;
                        when "101" =>result<=t6;
                        when "110" =>result<=t7;
                        when "111" =>result<=t8;
                        when others
                                =>result<="00000000000000000000000000000000";
                        end case;
                end if;
        end process;
end Behavioral;
```



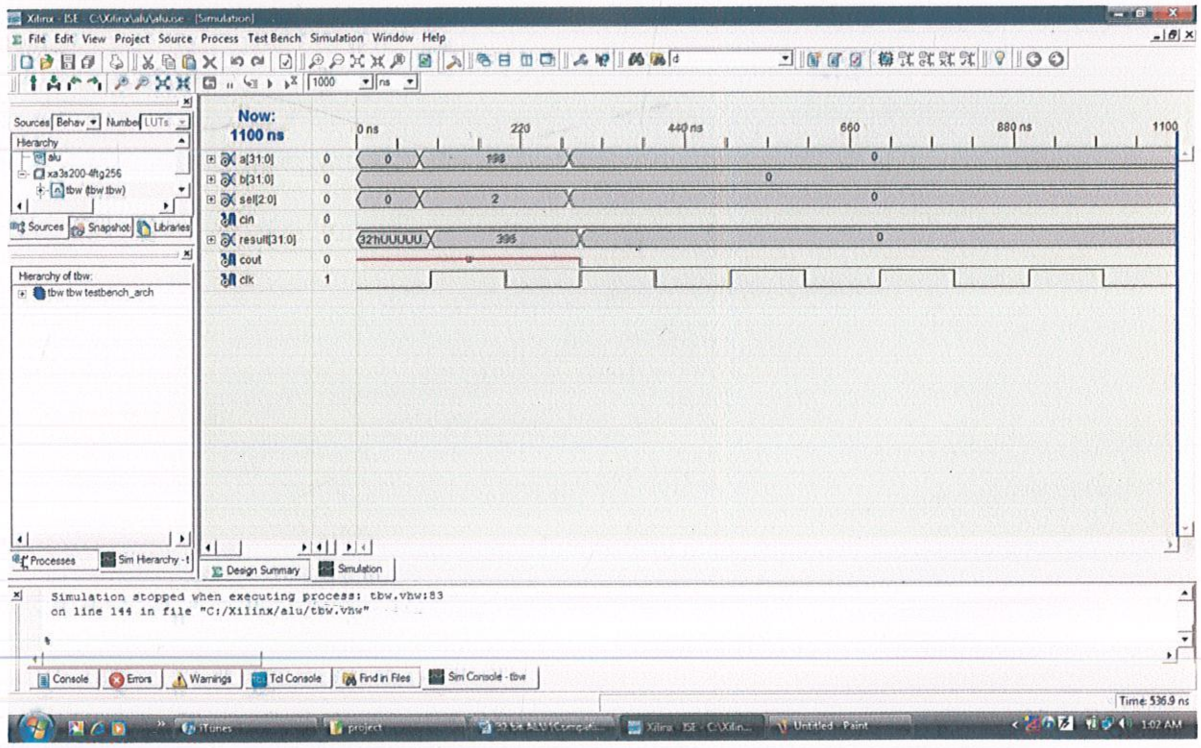
```
Simulation stopped when executing process: tbw.vhw:83
on line 149 in file "C:/Xilinx/alu/tbw.vhw"
```

Figure 16.1: Output window of adder in ALU



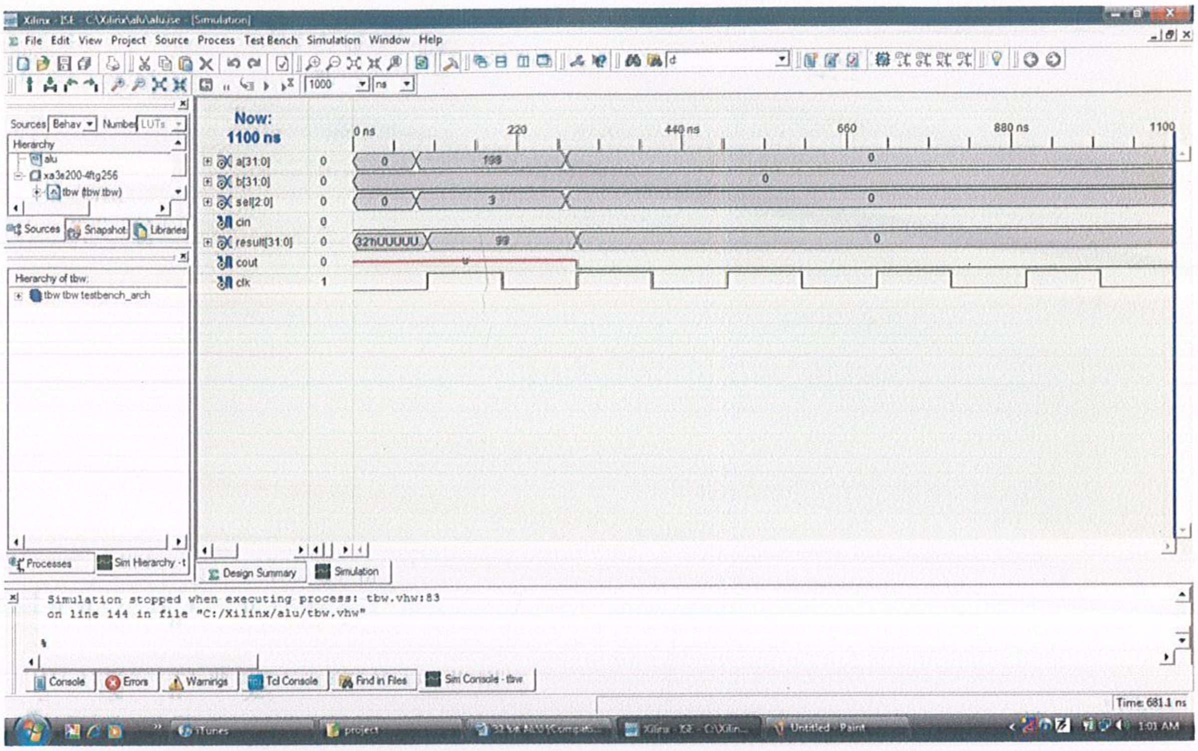
```
Simulation stopped when executing process: tbw.vhw:83
on line 151 in file "C:/Xilinx/alu/tbw.vhw"
```

Figure 16.2: Output window of subtractor in ALU



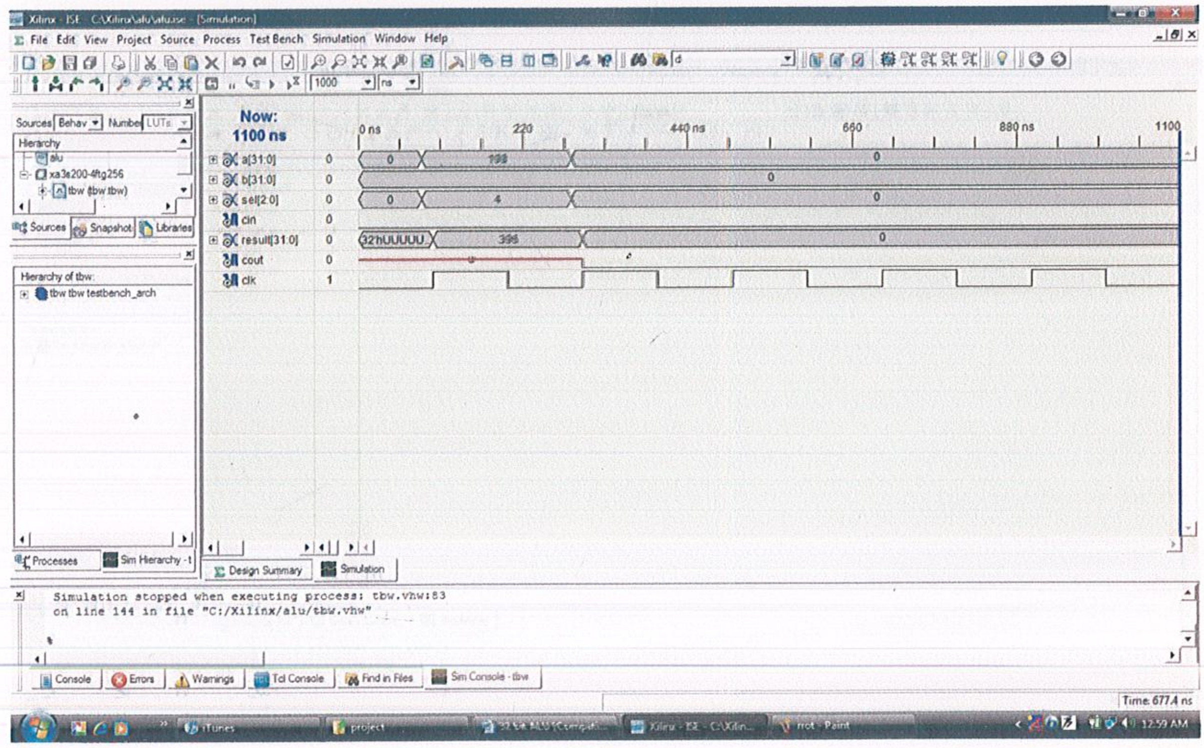
```
Simulation stopped when executing process: tbw.vhw:83
on line 144 in file "C:/Xilinx/alu/tbw.vhw"
```

Figure 16.3: Output window of Left shift in ALU


Xilinx - ISE - C:\Xilinx\alu\alu.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
1000 ns
Sources Behav Number LUTs
Hierarchy
alu
xa3s200-4ftg256
tbw (tbw.tbw)
Sources Snapshot Libraries
Hierarchy of tbw:
tbw tbw testbench_arch
Processes Sim Hierarchy - t
Now: 1100 ns
0 ns
220
440 ns
660
880 ns
1100
a[31:0] 0
b[31:0] 0
sel[2:0] 0
cin 0
result[31:0] 0
cout 0
clk 1
0
198
3
32'hUUUUU
99
U
Design Summary
Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - tbw
Time: 681.1 ns
iTunes project 32 bit ALU (Compat... Xilinx - ISE - C:\Xilin... Untitled - Paint
1:01 AM

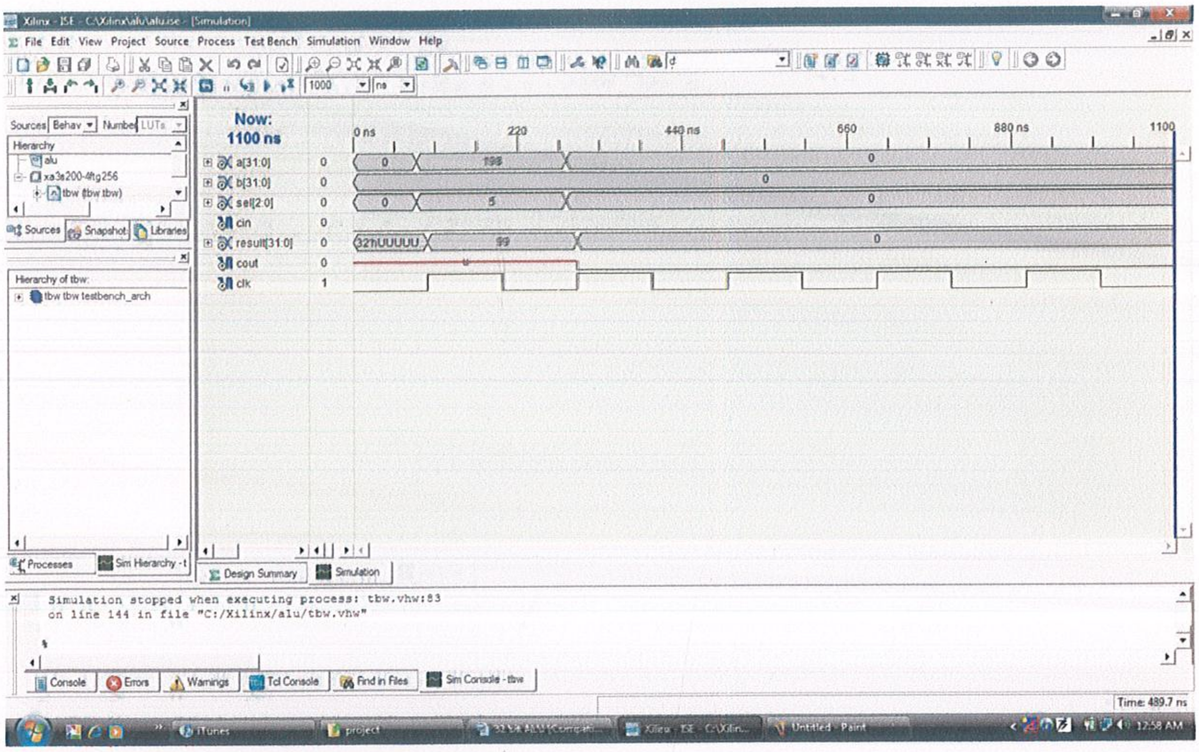
```
Simulation stopped when executing process: tbw.vhw:83
on line 144 in file "C:/Xilinx/alu/tbw.vhw"
```

Figure 16.4: Output window of Right shift in ALU


Xilinx - ISE - C:\Xilinx\alu\alu.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
1000 ns
Sources Behav Number LUTs
Hierarchy
alu
xa3s200-4ftg256
tbw (tbw.tbw)
Sources Snapshot Libraries
Hierarchy of tbw:
tbw tbw testbench_arch
Processes Sim Hierarchy - t
Now: 1100 ns
0 ns
220
440 ns
660
880 ns
1100
a[31:0] 0
b[31:0] 0
sel[2:0] 0
cin 0
result[31:0] 0
cout 0
clk 1
0
198
4
32'hUUUUU
396
U
Design Summary
Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - tbw
Time: 677.4 ns
iTunes project 32 bit ALU (Compat... Xilinx - ISE - C:\Xilin... rrot - Paint
12:59 AM

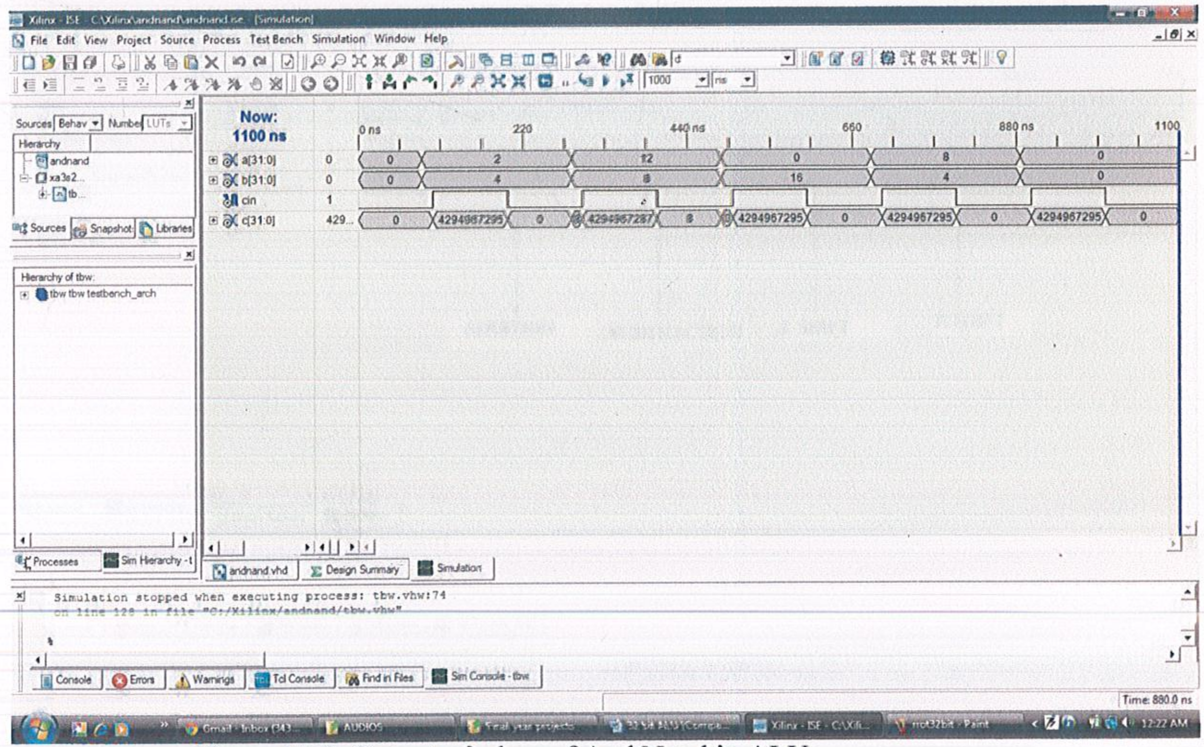
```
Simulation stopped when executing process: tbw.vhw:83
on line 144 in file "C:/Xilinx/alu/tbw.vhw"
```

Figure 16.5: Output window of Left rotate in ALU



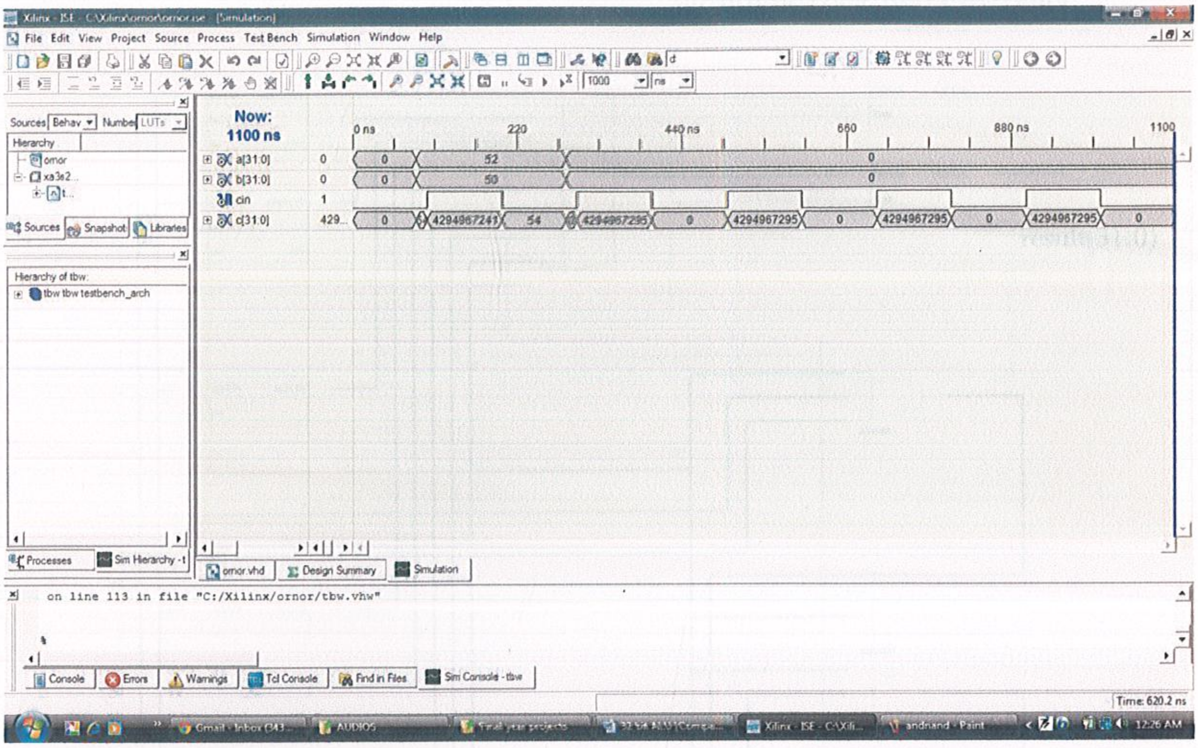
```
Simulation stopped when executing process: tbw.vhw:83
on line 144 in file "C:/Xilinx/alu/tbw.vhw"
```

Figure 16.6: Output window of Right rotate in ALU



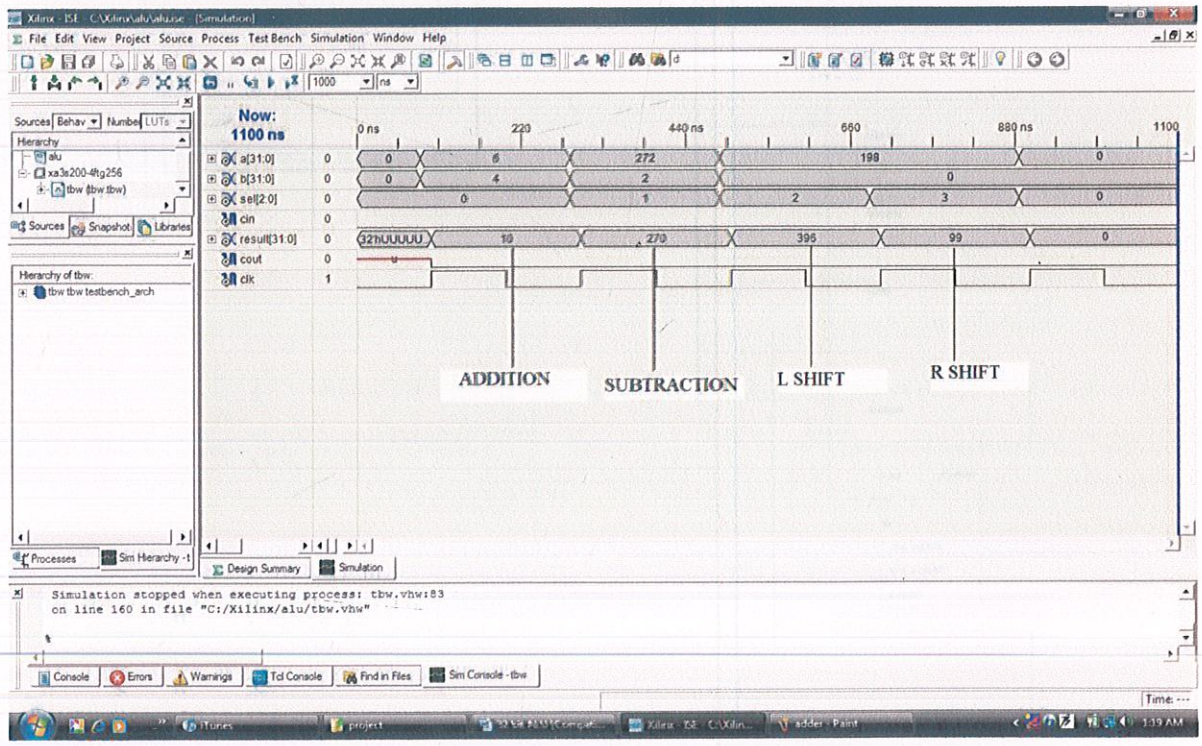
```
Simulation stopped when executing process: tbw.vhw:74
on line 128 in file "C:/Xilinx/andnand/tbw.vhw"
```

Figure 16.7: Output window of And Nand in ALU



```
on line 113 in file "C:/Xilinx/ornor/tbw.vhw"
```

Figure 16.8: Output window of Or Nor in ALU



```
Simulation stopped when executing process: tbw.vhw:83
on line 160 in file "C:/Xilinx/alu/tbw.vhw"
```

Figure 16.9: Output window of 4 Functions in ALU

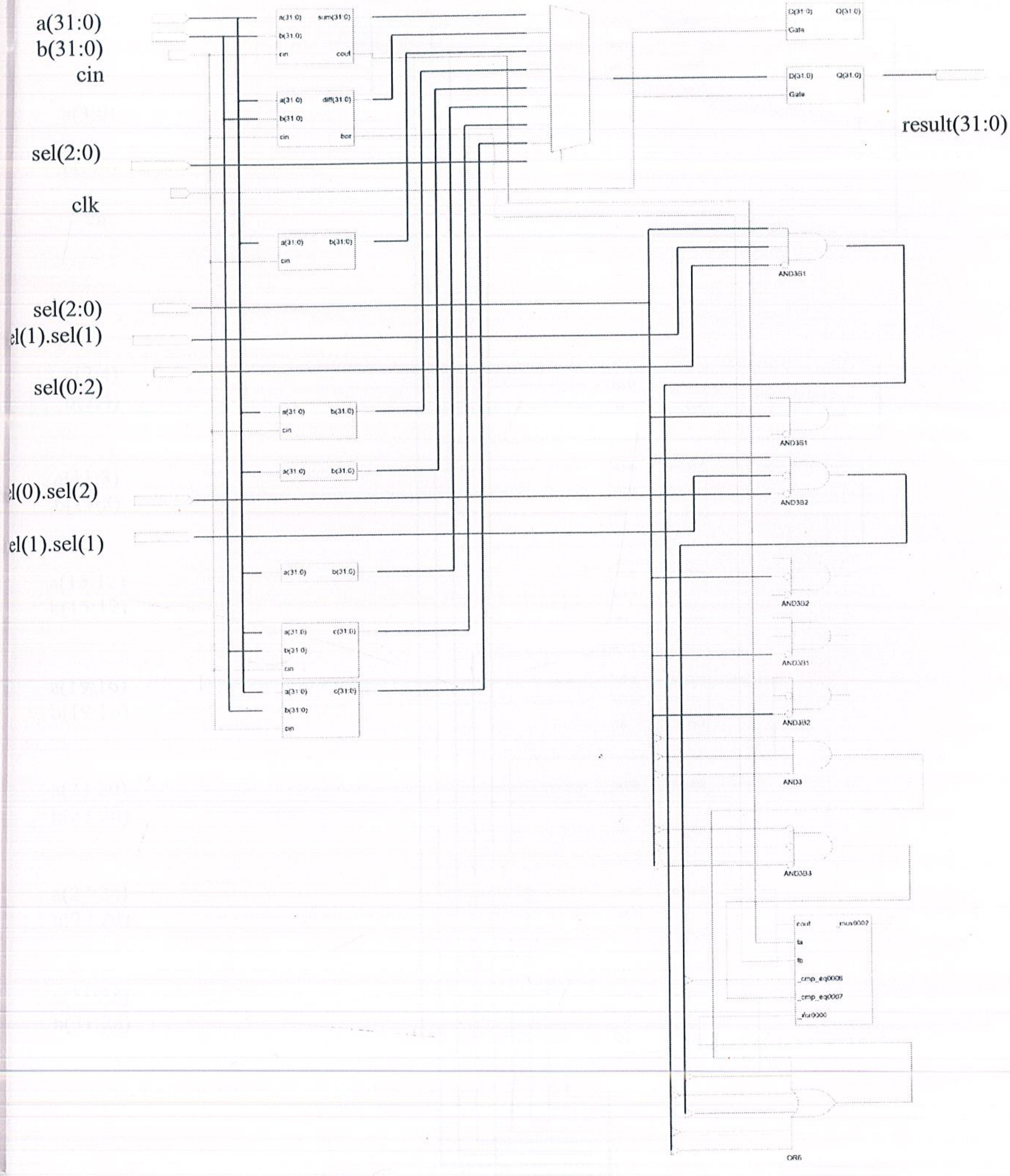


Figure 16.10: Expanded schematic view of ALU

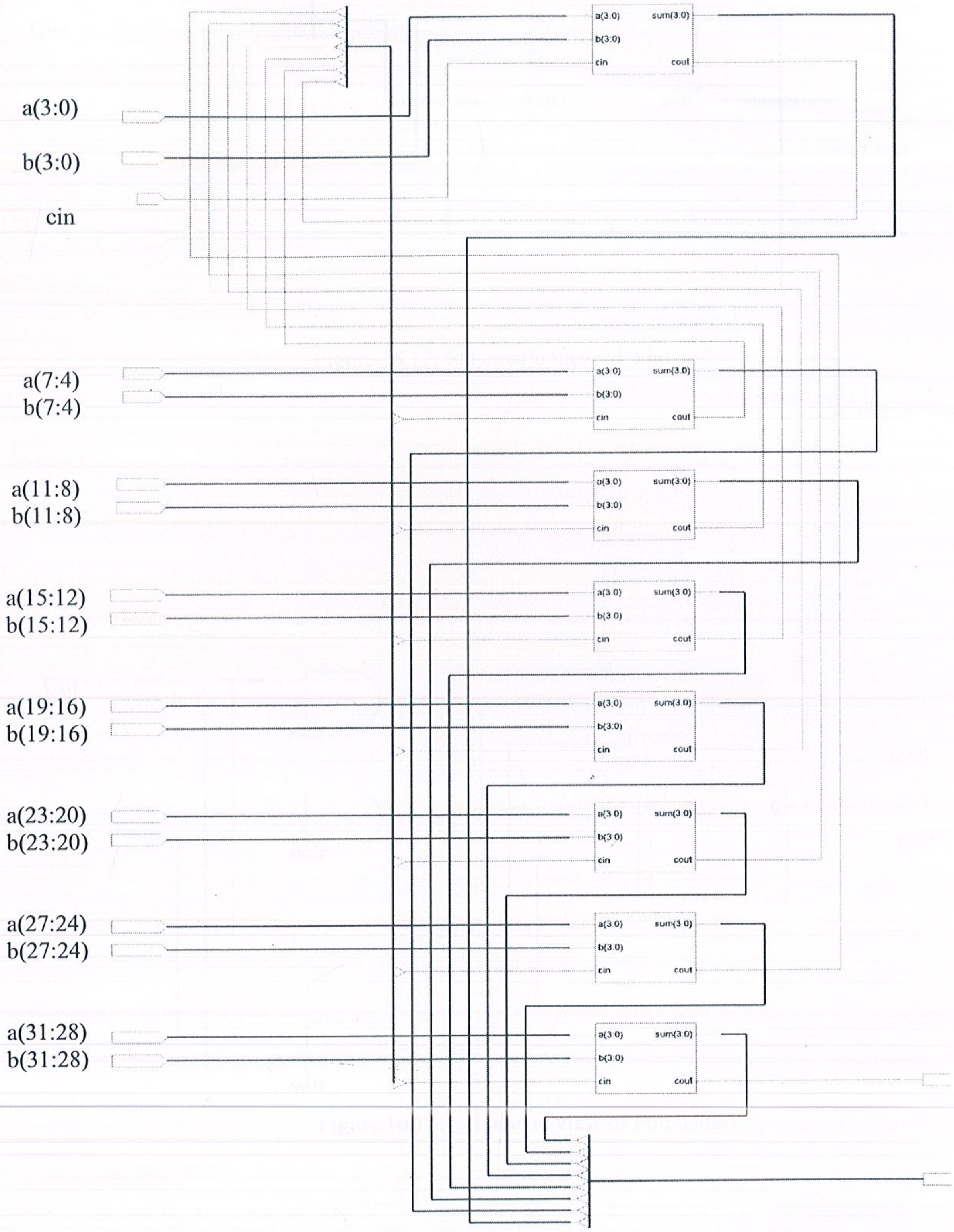


Figure 16.11: Schematic view of 32 bit adder.

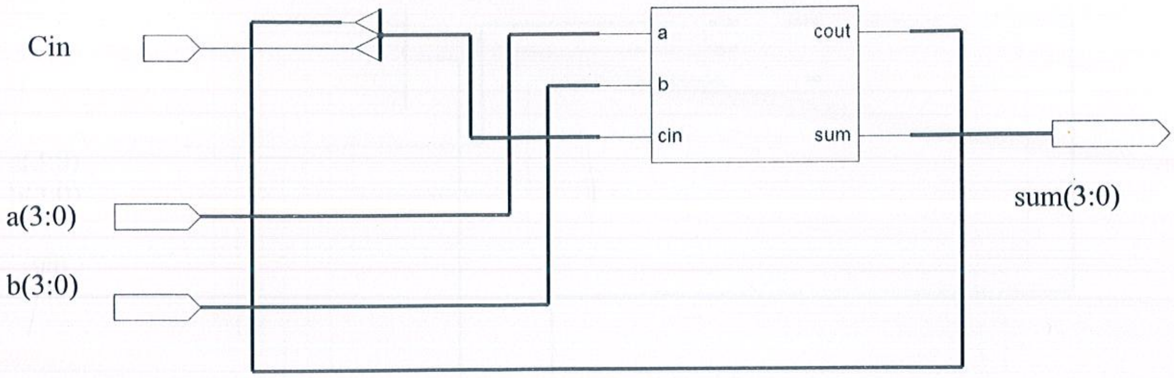


Figure 16.12: Schematic view of 4 bit adder.

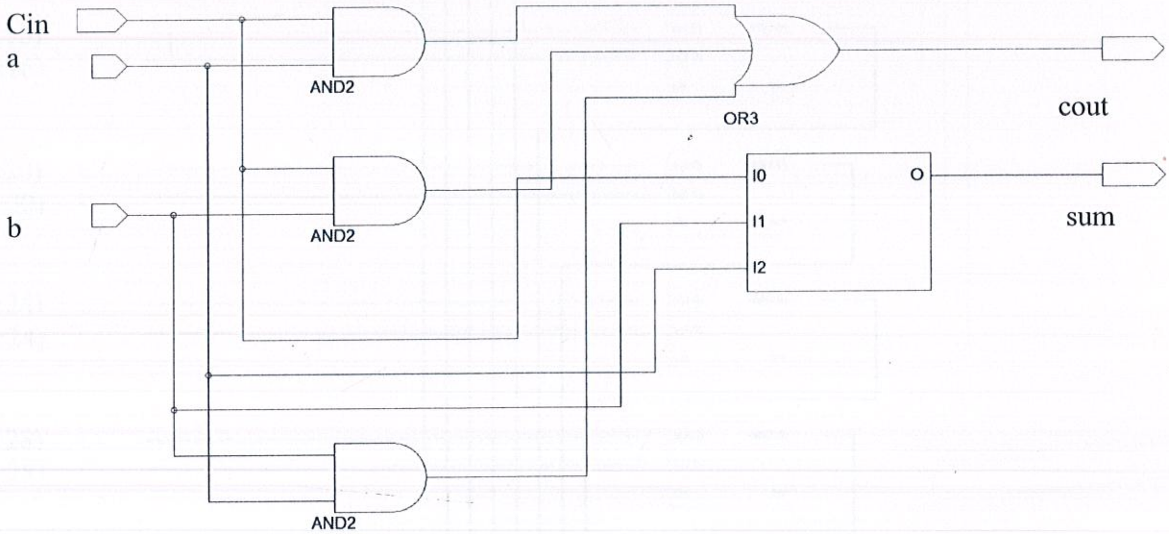


Figure 16.13: Schematic view of Full adder.

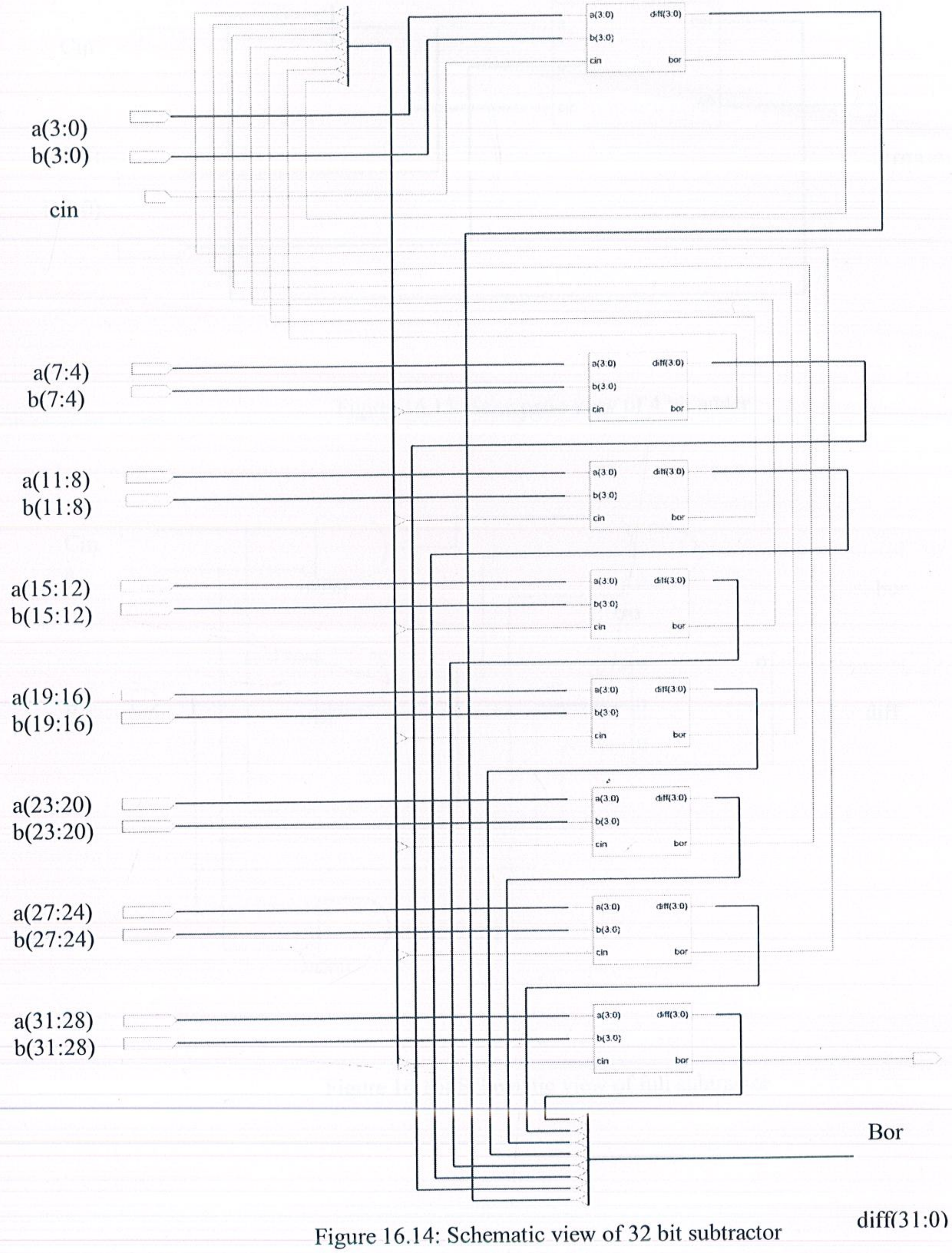


Figure 16.14: Schematic view of 32 bit subtractor

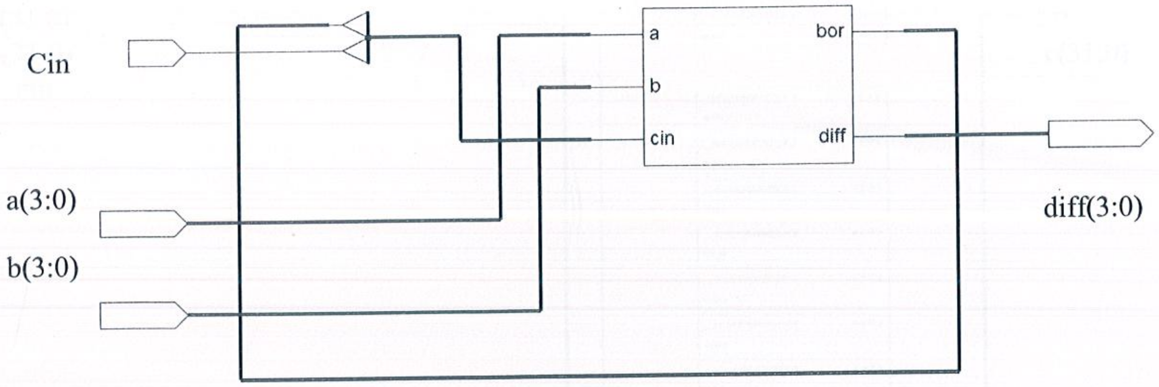


Figure 16.15: Schematic view of 4 bit adder

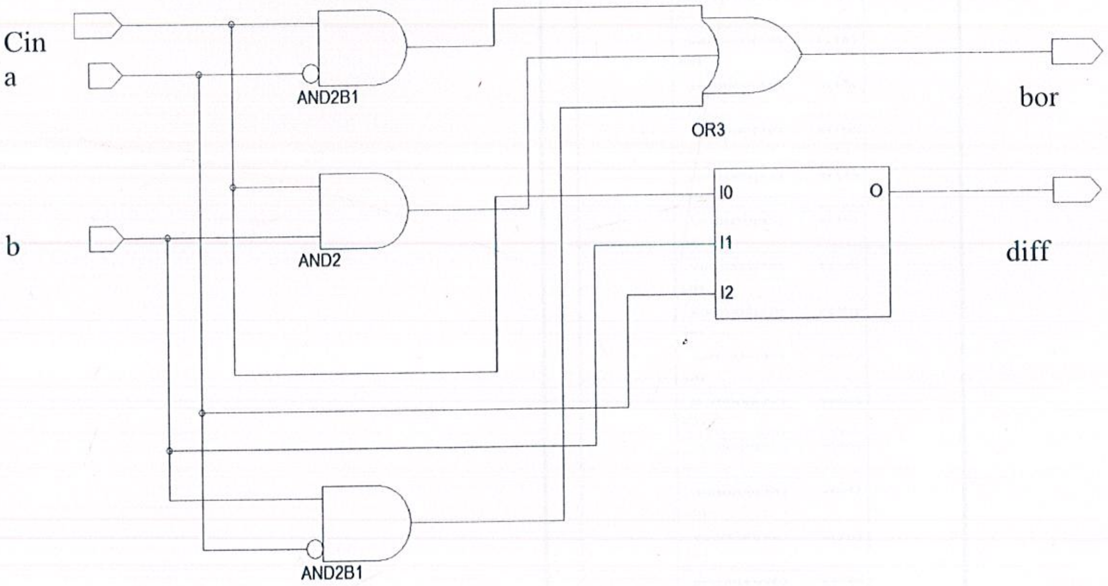


Figure 16.16: Schematic view of full subtractor

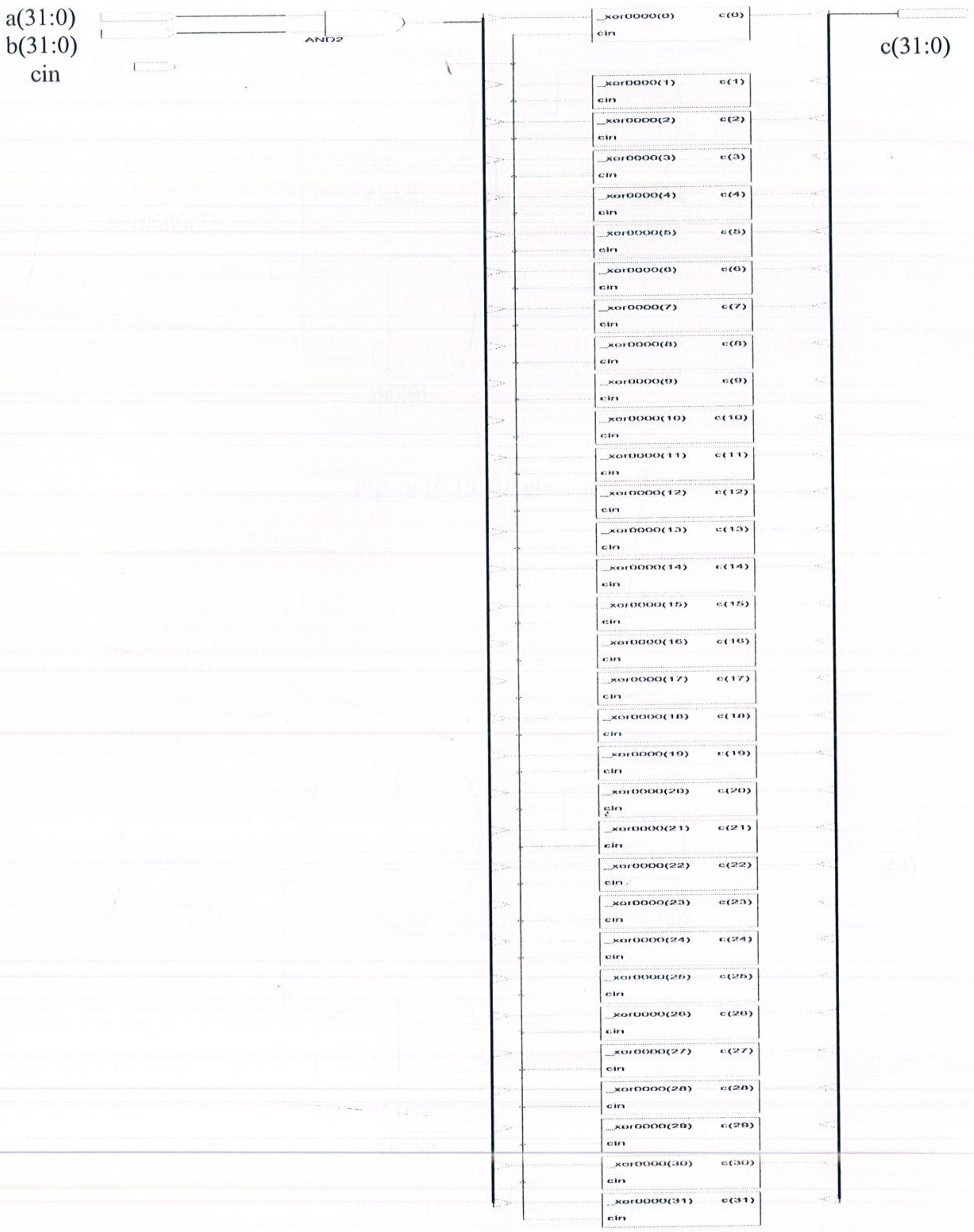


Figure 16.17: Schematic view of And Nand

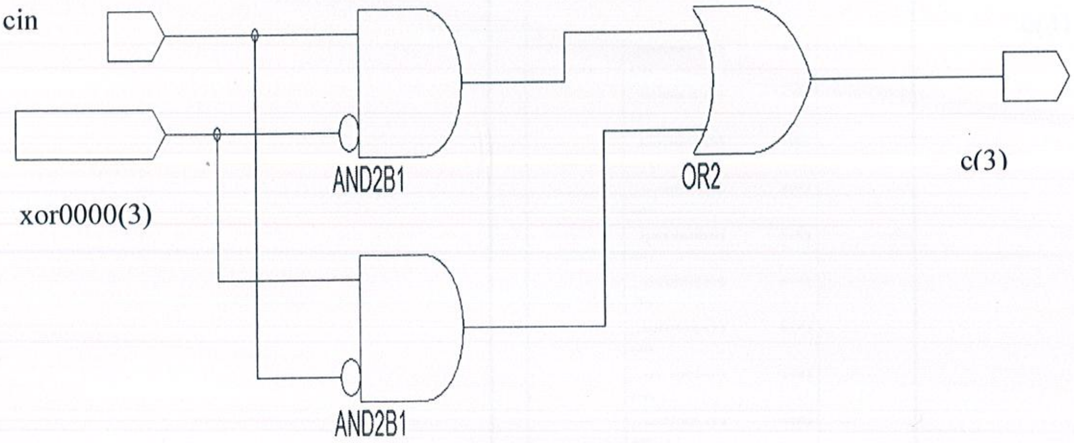


Figure 16.18: Single unit of And Nand.

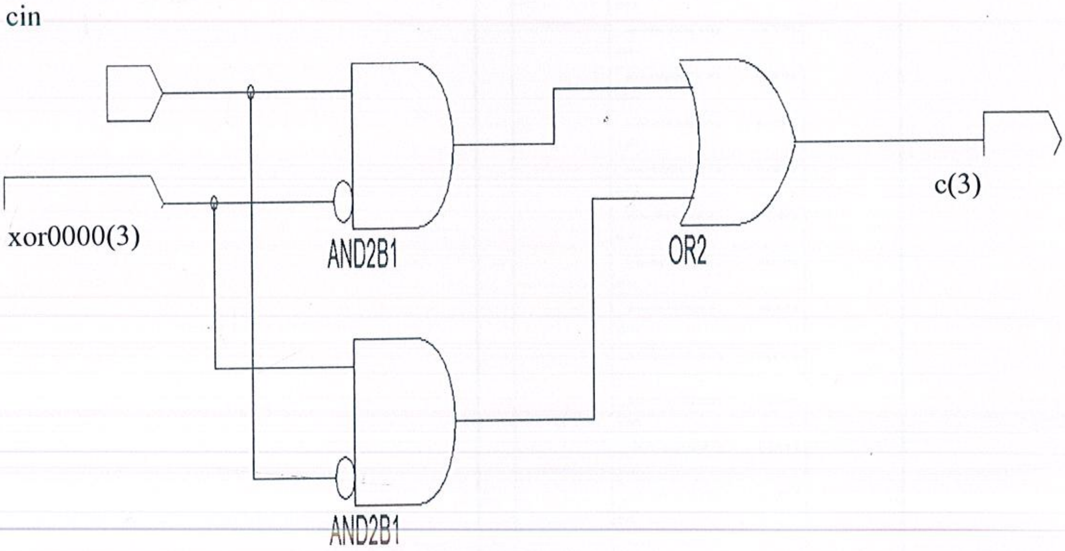


Figure 16.19: Schematic view of Or Nor

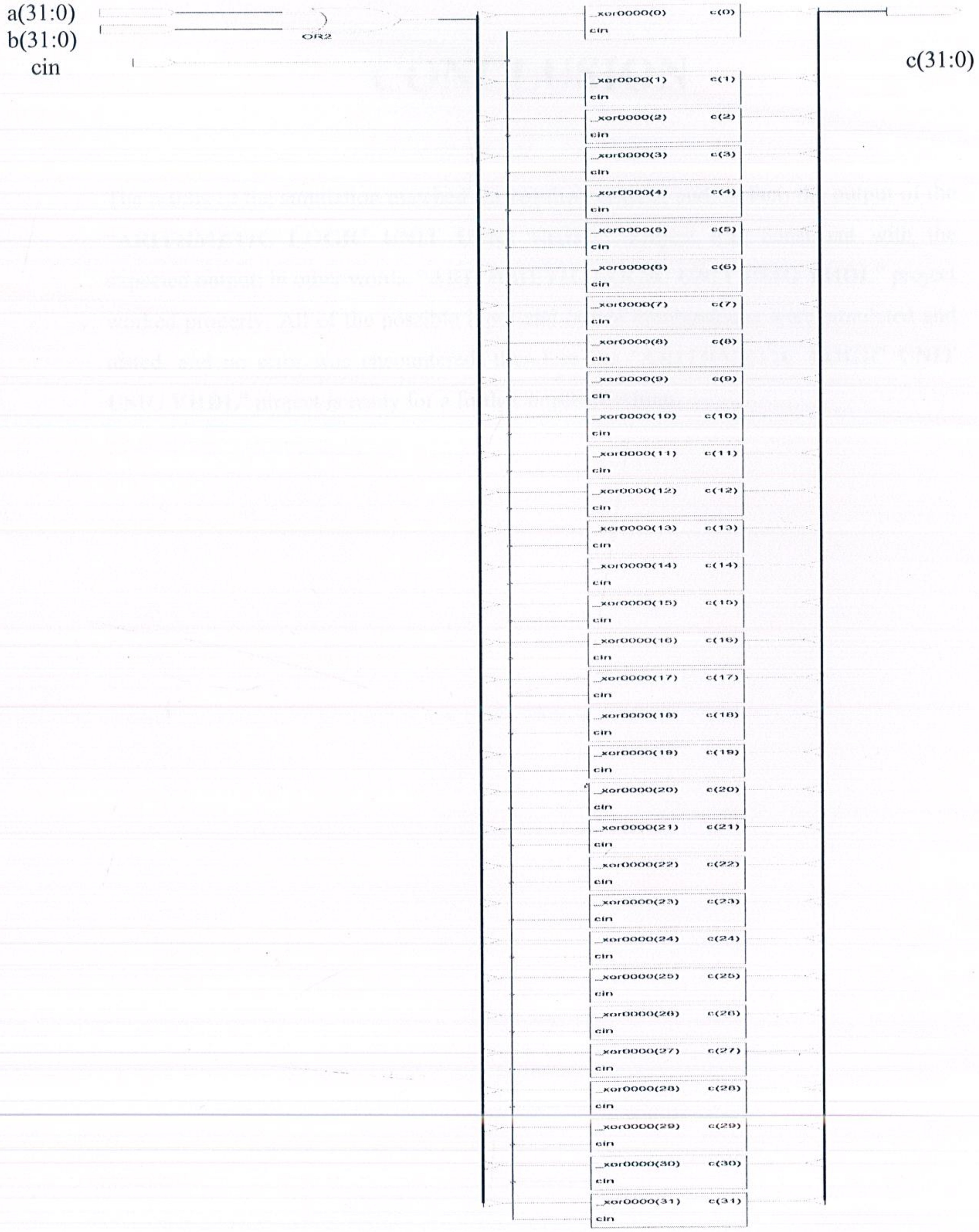


Figure 16.20: Schematic view of Or Nor

CONCLUSION

The results of the simulation matched the required criteria, and, in fact, the output of the "**ARITHMETIC LOGIC UNIT USIG VHDL**". Project was consistent with the expected output; in other words, "**ARITHMETIC LOGIC UNIT USIG VHDL**" project worked properly. All of the possible input and output combinations were simulated and tested, and no error was encountered; therefore the "**ARITHMETIC LOGIC UNIT USIG VHDL**" project is ready for a further implementation.

BIBLIOGRAPHY

- A VHDL Primer – J. Bhasker
- Digital Systems Design Using VHDL – Charles H Roth, Jr.
- Digital Logic and Computer Design – M. Morris Mano
- Digital Fundamentals – Floyd
- Principles of Digital Systems Design Using VHDL – Roth , John
- VHDL-2008 Just The New Stuff – Peter J. Ashenden , Jim Lewis
- VHDL Tutorial – William D. Bishop
- www.gogetpapers.com
- www.ieee.org
- www.wikipedia.com