

DIGITAL DESIGN AND COMPUTER
ORGANISATION LABORATORY

IIIrd Semester B.Tech

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
ELECTRONIC CITY CAMPUS
PES UNIVERSITY

Sl.No	TABLE OF CONTENTS	Page No
SECTION 1	Introduction	
1.1	VERILOG Language Basics	
1.2	Installation of ICARUS VERILOG	
1.3	Installation and Viewing the Waveform using GTKWave	
1.4	Basic Gates using VERILOG	

SECTION 1

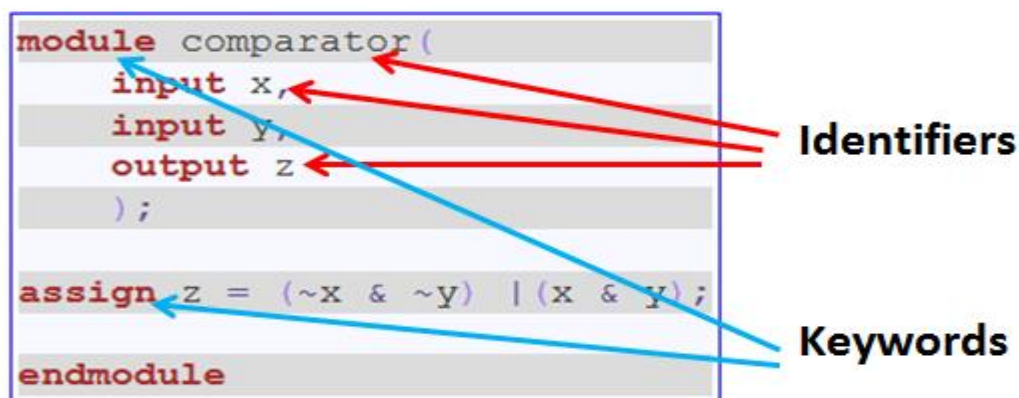
INTRODUCTION

1.1 Verilog Language Basics

We will briefly go through the terminologies of the Verilog language.

Identifier

When writing Verilog code, you will need to identify an object, such as an input port, a variable. An identifier is used for that purpose.



```
module comparator(  
    input x,  
    input y,  
    output z  
);  
  
assign z = (~x & ~y) | (x & y);  
  
endmodule
```

The diagram shows a Verilog code snippet with annotations. Red arrows point from the label 'Identifiers' to the words 'comparator', 'x', 'y', and 'z'. Blue arrows point from the label 'Keywords' to the words 'module', 'input', 'output', 'assign', and 'endmodule'.

In this example comparator is an identifier used to identify the module. x, y, z are used to denote an input or an output port.

An identifier must begin with an alphabetic character or an underscore.

Basically anything in the range (a-z, A-Z_).

An identifier may contain alphabetic or numeric characters, the underscore or the dollar sign. Basically anything in the range (a-z, A-Z, 0-9, _,\$)

Example of Valid identifiers

comparator_input

Compout\$p

_datax64

Example of Invalid identifiers

&comparator1

#compoutp

\$_datax64

Keywords

Keywords are special words used to describe the language construct.

As you can see in the above example, `module`, `assign` are keywords.

Comments

A Single line comment in Verilog starts with `//` Multiline comments start with `/*` and end with `*/`. Always use plenty of comments to make you code more readable.

The example below shows the proper use of the multiline and single line comments

```
/*  
Comparator Module for single bit comparison  
*/  
module comparator(  
    input x,  
    input y,  
    output z  
);  
    // z is a single bit comparator  
    assign z = (~x & ~y) | (x & y);  
endmodule
```

Whitespaces

Whitespaces are used to separate the different elements of Verilog code including keywords, identifiers and variables. White spaces include space, tab, new line. Proper use of white spaces makes the code more readable.

Optimal Use of White Spaces

```
module comparator(  
    input x,  
    input y,  
    output z  
);  
  
assign z = (~x & ~y) | (x & y);  
  
endmodule
```

Case Sensitivity

Verilog is case sensitive. There are couple of things you need to keep in mind

1. All the Verilog keywords are in lower case.
2. If you use an identifier that starts with upper case, it will be different from the one in lower case.

wire // A Verilog keyword

WIRE // This will not be a Verilog keyword

Verilog Modules

To allow the concept of hierarchical building blocks, Verilog provides the concept of modules. A real life design will have several modules. A Verilog provides the concept of module instantiation.

Ports

Modules are connected by Ports. A port can be an input, an output or an inout.

Summing up the modules and Ports

Typically a module and a port is declared as in the example below.

```
module comparator(  
    input x,  
    input y,  
    output z  
);  
  
assign z = (~x & ~y) |(x & y);  
  
endmodule
```

The module definition starts with the keyword module followed by the module name which is an identifier to identify the name of the module.

The module contains a list of the input and output port and enclosed by round bracket parenthesis and followed by ";".

The module definition is succeeded by a list of codes describing the behaviour of the module.

Finally the module definition ends with the keyword `endmodule`. Notice that the port directions could also be defined outside the module declaration as in the following example - a practice we will not follow.

It is being presented here to make you aware of the fact that such coding styles still exists and at times you may have to understand it.

```
module comparator(  
    x,  
    y,  
    z)  
    ;  
    input x,  
    input y,  
    output z  
    assign z = (~x & ~y) |(x & y);  
endmodule
```

Rules of Connecting Ports

There are certain rules that must be followed when connecting and input, output and the inout ports

Input Ports-Internally the input ports must always be of type net.

Externally-The inputs can connect to a variable of type reg or net.

Outputs -Internally output port can be net or reg.

Externally the outputs must connect to a variable of type net.

Inouts-The Inout port must always be of type net internally or externally.

Vector Data

Verilog provides the concept of Vectors.

Vectors are used to represent multi-bit buses.

A vector to represent a multi bit bus is declared as follows

```
reg [7:0] eightbitbus;    // 8-bit reg vector with MSB=7 LSB=0
```

The `reg [7:0]` means you start with 0 at the rightmost bit to begin the vector, then move to the left.

We could also declare the vector as

```
reg [0:7] eightbitbus;    // 8-bit reg vector with MSB=0, LSB=7
```

In which case the LSB will be represented by leftmost bit.

Let us rewrite our comparator example, so that it now uses two bit bus in place of one bit.

```
`timescale 1ns / 1ps
module comparator2bit(
    input [1:0] x,
    input [1:0] y,
    output z
);

assign z = (x[0] & y[0] & x[1] & y[1]) |
    (~x[0] & ~y[0] & x[1] & y[1]) |
    (~x[0] & ~y[0] & ~x[1] & ~y[1]) |
    (x[0] & y[0] & ~x[1] & ~y[1]);
endmodule
```

Verilog \$monitor

Verilog provides some system tasks and functions specifically for generating input and output to help verification. \$monitor is one such system task. These system tasks are not used (or ignored) by the synthesis tools.

Syntax of \$monitor statement

A monitor statement has the syntax of

\$monitor ("format_string", parameter1, parameter2 ...);

Where the "format_string", specifies how the parameters will be displayed.

\$monitor displays the values of its parameters EVERY time ANY of its parameter changes value.

We may have numerical, hexadecimal or binary outputs.

The parameter1, parameter2 ... etc are the list of the variables that need to be printed.

Some Examples

1. %d will print the variable in decimal
2. %4b will print the variable in binary - that has width of 4.
3. %h will print the variable in hexadecimal.

Gate Level Modelling

Although the circuit behaviour in Verilog is normally specified using assignment statements, in some cases modelling the circuit using primitive gates is done to make sure that the critical sections of circuit is most optimally laid out.

Verilog has built in primitives like gates, transmission gates, and switches to model gate level simulation.

To see how the gate level simulation is done we will write the Verilog code that that we used for comparator circuit using primitive gates.

```
module gate(  
    input x,  
    input y,  
    output z  
);  
    wire x_, y_, p, q;  
    not(x_, x);  
    not(y_, y);  
    and(p, x, y);  
    and(q, x_, y_);  
    or(z, p, q);  
  
endmodule
```

This example does the same function as the previous example, but we have used primitive gates in this example

The primitive not (x_, x);

creates a not gate with x as input and x_ as output. All primitives have at least two parameters. The first parameter is output and other parameters are input.

The statement

and (p, x, y);

creates an AND gate with two inputs and one output.

You can simulate the code with same stimulus that we did in the previous example, which is reproduced here.

```
initial begin
    // Initialize Inputs
    x = 0;
    y = 0;

    // Wait 100 ns for global reset to finish
    #100;
    #50 x = 1;
    #60 y = 1;
    #70 y = 1;
    #80 x = 0;
end
```

And it produces the same output

x=0, y=0, z=1

x=1, y=0, z=0

x=1, y=1, z=1

x=0, y=1, z=0

1.2 Installation of Icarus Verilog

Icarus Verilog for windows is a free compiler implementation for the IEEE-1364 Verilog hardware description language. Icarus is small and efficient compiler that is more than enough for learning the Verilog tutorials. Although Icarus is mainly tailored towards Linux, we have Windows installer available.

Icarus Verilog for windows is available at the following link:
<http://bleyer.org/icarus/>.

Download the latest version of Verilog setup file (iverilog-0.8.2 setup.exe [1.43MB]) to your machine.

Login as an administrator and “double click” on the iverilog-0.8.2 setup.exe file. It will install the iverilog with an interactive mode.

Double click the downloaded setup file and follow the instructions to install the Verilog on your windows.

The C: /iverilog/bin subdirectory contains the executable file verilog.exe that is used to run simulator.

Once installation is completed, type the installed path for example. c:\IcarusVerilog\bin\iverilog on your command prompt. This will give the iverilog compilation options.

How to Compile Verilog file on Windows

By a text editor edit your Verilog program and save it as “filename.v”.

Next, compile this program with following command.

iverilog -o filename filename.v

The results of this compile are placed into the file “filename”, as the “-o” flag tells the compiler where to place the compiled result.

Next, execute the compiled program using following command.

vvp filename

The vvp target generates code for the vvp runtime.

The output is complete program that simulates the design but must be run by the vvp command.

Other way of compilation is ***iverilog filename.v*** and this will generate the a.out executable file. For execution use the command. ***./a.out***.

1.3 Installation and Viewing the Waveform using GTKWave

The Icarus, also comes with a waveform viewing tool called GTKWave. You can see the binaries in the directory /iverilog/gtkwave/bin.

Before we wish to use the gtkwave, we may wish to add the pathname of the gtkwave in the list of the environment variables. This can be done in the same way as we did for iverilog.

Go to Start -> Computer -> Properties -> Advanced System Settings -> Environment Variable -> System Variable and click on the variable named path and then click edit.

Now add C:\iverilog\gtkwave\bin;

You need to add the following two lines of code in the stimulus file.

```
$dumpfile("test.vcd");  
$dumpvars(0,stimulus);
```

The complete code again with these two lines added.

comparator.v

```
module comparator(  
    input x,  
    input y,  
    output z  
);  
  
assign z = (~x & ~y) |(x & y);  
  
endmodule
```

stimulus.v

```
`timescale 1ns / 1ps  
module stimulus;  
    // Inputs  
    reg x;  
    reg y;  
    // Outputs  
    wire z;  
    // Instantiate the Unit Under Test (UUT)  
    comparator uut (  
        .x(x),
```

```

        .y(y),
        .z(z)
    );

    initial begin
        $dumpfile("test.vcd");
        $dumpvars(0,stimulus);
        // Initialize Inputs
        x = 0;
        y = 0;

        #20 x = 1;
        #20 y = 1;
        #20 y = 0;
        #20 x = 1;
        #40 ;
    end

    initial begin
        $monitor("t=%3d x=%d,y=%d,z=%d \n",$time,x,y,z, );
    end

endmodule

```

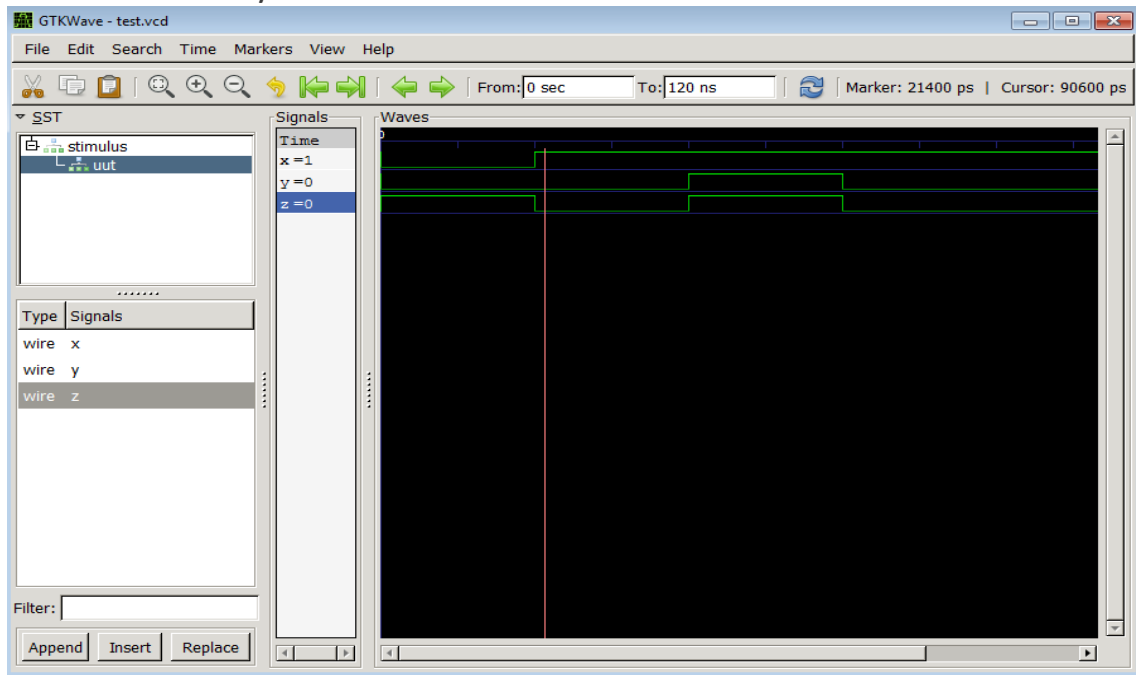
We will now compile the program with following commands

```

iverilog -o mydesign comparator.v stimulus.v
vvp mydesign
gtkwave test.vcd

```

This opens up a new window for displaying the graphs. Drag and drop the waveforms that you wish to see.



Verilog TUTORIAL - Running your first code

Create a new file called hello.v in the directory C: /iverilog/bin and edit it with notepad or any other text editor (Eg: notepad++).

Enter the following lines of code in hello.v

```
module main;  
  initial  
  begin  
    $display("Hello World");  
    $finish ;  
  end  
endmodule
```

- Go to your DOS prompt (**Start - > cmd**) and navigate to the directory

C: /iverilog/bin

C:\> cd \iverilog\bin

Run iverilog using the command

C:\>iverilog\bin > **iverilog hello.v**

If you have done everything right, it should generate a file called a.out which you can run using command

C:\>iverilog\bin > **vvp a.out**

At this point the code just gives out the output saying "Hello World". It just verifies that your set up is ready for running Verilog code.

Basic gates implementation:

1. NOT GATE (Single Input)

Verilog code for NOT gate

Gate Level Modelling

```
module not_gate(c, a);  
input a;  
output c;  
not (c, a);  
endmodule
```

Data Flow Modelling

```
module not_data(c, a);  
input a;  
output c;  
assign c=~a;  
endmodule
```

Behavioural Modelling

```
module not_beh(c, a);  
input a;  
output c;  
reg c;  
always@(a)  
begin  
if (a==0)  
c=1;  
else  
c=0;  
end  
endmodule
```

Test Bench

```
module not_test;
reg a;
wire c;
not_gate not_test(c, a);
initial
begin
#000 ;
a=0;
#100 ;
a=1;
end
initial
begin
$monitor($time , " a=%b, c=%b ", a, c);
end

initial
begin

$dumpfile ("not_test.vcd");
$dumppvars (0, not_test);
end
endmodule
```

Execution Steps

- iverilog -o test not.v not_test.v
- vvp test
- gtkwave not_test.vcd

Output

0 a=0, c=1
100 a=1, c=0

Output Waveform

