# Amrita Vishwa Vidyapeetham
# Amrita School of Engineering, Bangalore
# Department of Computer Science and Engineering

## 19CSE211 Computer Organization and Architecture
## Lab Handout - 1

**Introduction to Verilog**

Verilog is one of the two-major Hardware Description Languages (HDL) used by hardware designers in industry and academia. VHDL (Very high speed integrated circuit Hardware Description Language) is the other one. Many feel that Verilog is easier to learn and use than VHDL. VHDL was made an IEEE Standard in 1987, and Verilog in 1995. Verilog is very similar to C-like.

Verilog allows a hardware designer to describe designs at a high level of abstraction such as at the architectural or behavioral level as well as the lower implementation levels (i. e., gate and switch levels) leading to Very Large Scale Integration (VLSI) Integrated Circuits (IC) layouts and chip fabrication. A primary use of HDLs is the simulation of designs before the designer must commit to fabrication.

The Verilog language provides the computer architect with a means of describing a computing system at a wide range of levels of abstraction, and, at the same time, provides access to computer-aided design tools to aid in the design process at these levels. Verilog allows hardware designers to express their design with behavioral constructs, deferring the details of implementation to a later stage of design. An abstract representation helps the designer explore architectural alternatives through simulations and to detect design bottlenecks before detailed design begins. Though the behavioral level of Verilog is a high level description of a digital system, it is still a precise notation. Computer-aided-design tools, i. e., programs, exist which will “compile” programs in the Verilog notation to the level of circuits consisting of logic gates and flip flops. One could then go to the lab and wire up the logical circuits and have a functioning system. And, other tools can “compile” programs in Verilog notation to a description of the integrated circuit masks for very large scale integration (VLSI). Therefore, with the proper automated tools, one can create a VLSI description of a design in Verilog and send the VLSI description via electronic mail to a silicon foundry and receive the integrated chip. Verilog also allows the designer to specify designs at the logical gate level using gate constructs and the transistor level using switch constructs.

Our goal in the course is not to create VLSI chips but to use Verilog to precisely describe the functionality of any digital system, for example, a computer. However, a VLSI chip designed using Verilog’s behavioral constructs will be rather slow and be wasteful of chip area. The lower levels in Verilog allow engineers to optimize the logical circuits and VLSI layouts to maximize speed and minimize area of the VLSI chip.

**Incarus-Verilog**

Icarus Verilog is a Verilog simulation and synthesis tool. It operates as a compiler, compiling source code written in Verilog (IEEE-1364) into some target format. For batch simulation, the compiler can generate an intermediate form called vvp assembly. This intermediate form is executed by the ``vvp'' command. For synthesis, the compiler generates netlists in the desired format. This tool can be used in Linux, Windows and MacOS.

Website: http://iverilog.icarus.com
Download: http://bleyer.org/icarus/
Installation Guide: https://iverilog.fandom.com/wiki/Installation_Guide

**Procedure to implement and execute a verilog code**

Step1 : Write the code in an text editor of your choice. Store it with ".v" extension

Step2: In command prompt type the following command to compile the code

**iverilog -o output_filename.vvp input_filename.v**

Step 3: If the compilation was successful, you will find a ".vvp" file. Run the below command to execute the code

**vvp output_filename.vvp**

**Exercise Problems**

**1. Write a verilog code and its test bench to synthesize a buffer**

```
module test(A,B);
input A;
output B;
assign B=A;
endmodule
```

```
module Test_tb;
reg A;
wire B;

test T1(A,B);
initial
begin
A=1'b1;
$monitor("Time=%0t A=%b, B=%b",
$time,A, B);
#5 A=1'b0;
#5 A=1'b1;
end
endmodule
```