LAB # 0
Introduction to RISC-V, HDL and Visual Studio Code

Objective:

- To become familiar with RISC-V ISA
- To become familiar with Verilog HDL
- Download and Install the required extensions for Visual Studio Code

System Module (Hardware/Software)

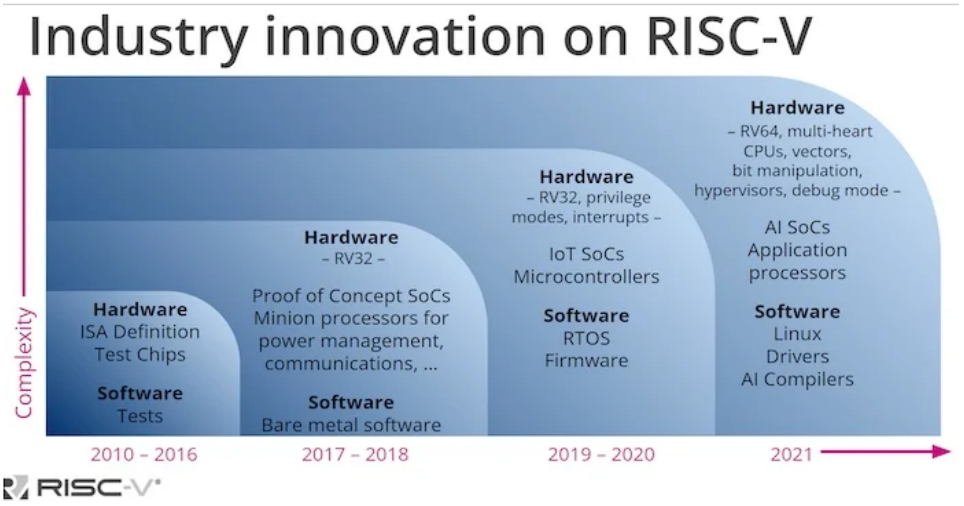
- Visual Studio Code

Theoretical Background:

RISC-V:
The RISC-V open instruction set architecture is a popular alternative to proprietary architectures available today, such as those by ARM. Since its birth, RISC-V has steadily gained academic and commercial popularity.

RISC-V is an open instruction set architecture (ISA), which means that you are free to implement a RISC-V CPU in a microprocessor or microcontroller without having to pay royalties to anyone for using this ISA.

RISC-V International is a global nonprofit organization that owns and maintains the RISC-V ISA intellectual property. One of its main goals is to keep the design of RISC-V based on simplicity and performance, as opposed to focusing on commercial interests. For this reason, RISC-V International relies on its members, who represent the microprocessor ecosystem population, ranging from individuals to organizations like Google, Intel, and Nvidia. Becoming a member has a host of benefits, including the possibility of contributing to the design of the ISA, and voting to approve proposed changes. Below in Figure 1, you can see a high-level timeline of the development of RISC-V over the years.



HDL:
In computer engineering, a hardware description language (HDL) is a specialized computer language used to describe the structure and behavior of electronic circuits, and most commonly, digital logic circuits.

The most popular hardware description languages are **Verilog**, **System-Verilog** and **VHDL**. They are widely used in conjunction with FPGAs, which are digital devices that are specifically designed to facilitate the creation of customized digital circuits.

1) **Verilog HDL:**
   Verilog is a Hardware Description Language (HDL) used to model digital systems. It provides the designer entry into the world of large, complex digital systems design. The Verilog language provides the digital system designer with a means of describing a digital 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. It also fulfills the need for verifying the design for functionality and timing constraints like propagation delay, setup and hold times.
   The module is the basic building block for modeling hardware with the Verilog HDL. The logic of a module can be described in any one (or a combination) of the following modeling styles:
   1. **Gate-level modeling** using instantiations of predefined and user-defined primitive gates.
   2. **Dataflow modeling** using continuous assignment statements with the keyword assign.
   3. **Behavioral modeling** using procedural assignment statements with the keyword “always”.

2) **System Verilog HDL:**
   SystemVerilog is the most transformative technology in EDA since the birth of logic synthesis. It is an extension of the popular Verilog language, bringing a higher level of abstraction to design and verification.

   SystemVerilog provides a complete verification environment, employing ***Constraint Random Generation***, ***Assertion Based Verification*** *and* ***Coverage Driven Verification***. These methods improve the verification process. SystemVerilog also provides enhanced hardware-modeling features, which improve the RTL design productivity and simplify the design process.

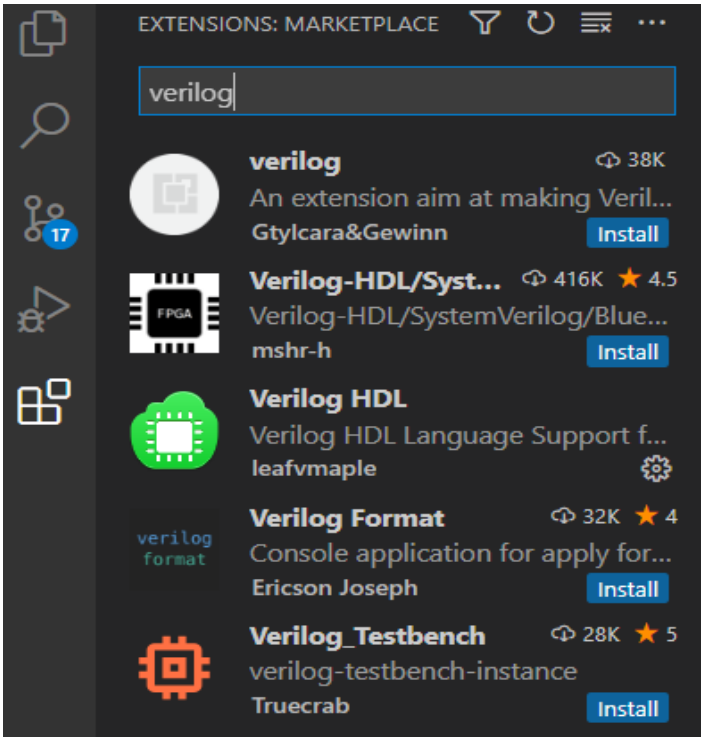
   SystemVerilog brings a higher level of abstraction to the Verilog designer. Constructs and commands like Interfaces, new Data types (logic, int), Enumerated types, Arrays, Hardware-specific always (always_ff, always_comb) and others allow modeling of RTL designs easily, and with less coding.

   SystemVerilog also extends the modeling aspects of Verilog by adding a Direct Programming Interface which allows C, C++, SystemC and Verilog code to work together without the overhead of the Verilog PLI.

Visual Studio Code:
Microsoft Visual studio code can be easily used to implement Verilog, VS code only provides an environment where Verilog designs could be written and edited, forming files with relevant extensions (.v).
In order to simulate the verilog designs "Icarus", which is an open source simulator, has to be used alongside VS code.

Procedure:

1. Download Microsoft VS code for windows.
2. Look for the Verilog extension available in marketplace of VS Code,



3. After downloading the relevant extension, from multiple available in the marketplace, you will be able to save files with (.v) extensions and work around with verilog files.
4. In order to be able to simulate the designs with Microsoft VS,Download "Icarus" using the provided link:/http://bleyer.org/icarus/
5. After successful installation write "iverilog" in command prompt, following message shall appear at this step.

```
C:\Users\HP> iverilog
iverilog: no source files.

Usage: iverilog [-EiSuvV] [-B base] [-c cmdfile|-f cmdfile]
                [-g1995|-g2001|-g2005|-g2005-sv|-g2009|-g2012] [-g<feature>]
                [-D macro[=defn]] [-I includedir] [-L moduledir]
                [-M [mode=]depfile] [-m module]
                [-N file] [-o filename] [-p flag=value]
                [-s topmodule] [-t target] [-T min|typ|max]
                [-W class] [-y dir] [-Y suf] [-l file] source_file(s)

See the man page for details.
```

6. In order to simulate the verilog designs write these two commands step by step in VS terminal,

   iverilog -o out.vvp Testbench_File_Name.v Design_File_Name.v

   vvp out.vvp

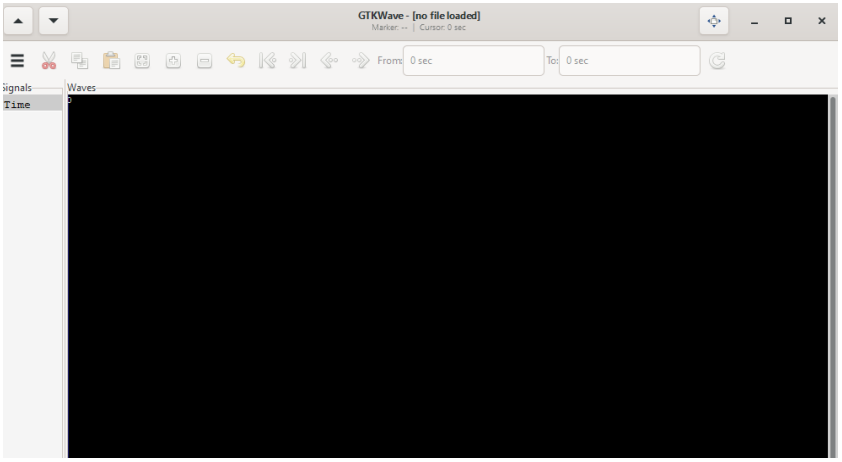
7. While running the commands make sure that the testbench and the design files are in the same folder, After running the first command the simulator would notify errors and warnings to you in design. following message would appear after the commands are successfully executed,

```
PS C:\Users\HP\Desktop\VS_Codes\CS_Course\Lab_1> iverilog -o out.vvp And_Gate_Test.v And_Gate.v
PS C:\Users\HP\Desktop\VS_Codes\CS_Course\Lab_1> vvp out.vvp
VCD info: dumpfile Dump.vcd opened for output.
And_Gate_Test.v:25: $finish called at 400 (1s)
PS C:\Users\HP\Desktop\VS_Codes\CS_Course\Lab_1>
```

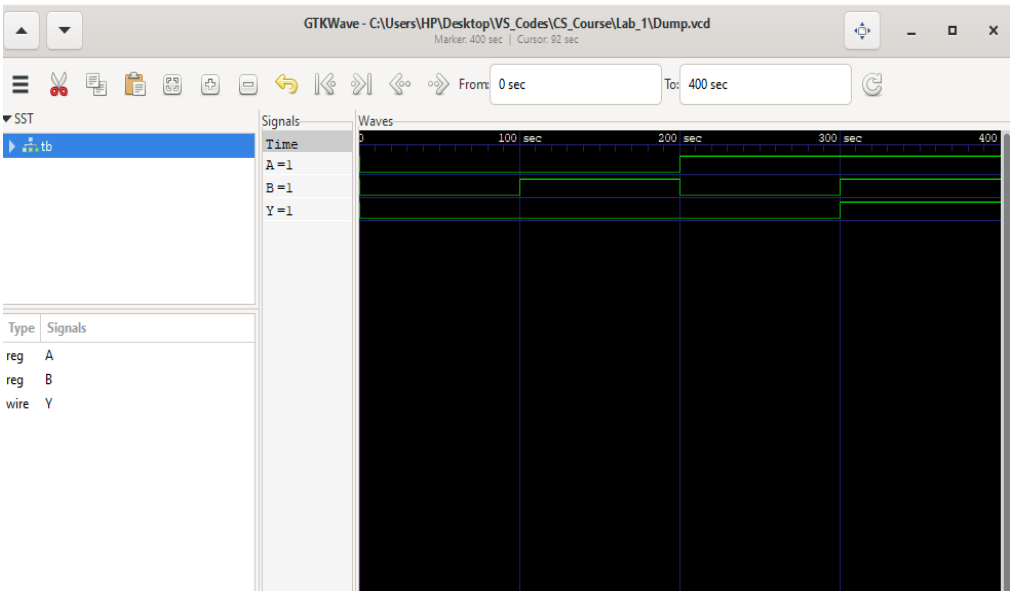
8. This message indicates that the VCD file , which is made through the testbench, is now ready to be opened for seeing the results.
9. Next up open another terminal window, cd to the directory where your design and testbench lies and enter the following command in to your terminal,

   gtkwave

10. This command will open a Gtkwave file,



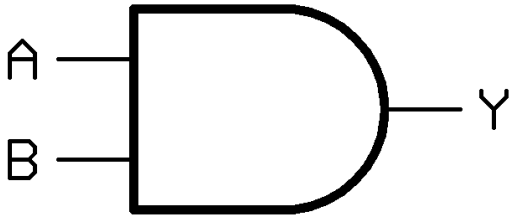
11. In the GTKwave file from the drop down menu go to Open New Tab option, select the dum.vcd file from the list of file options available in the specified directory.



12. The name of the testbench module will appear in the left menu, click on the link to make the signals appear in the bottom left menu, double click on signals to append them on the waveform screen, signals would appear on the screen as per the timing constraints provided to them in the testbench.

Implement the following design of And gate using verilog.

AND Gate:



Truth Table:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Code:

Verilog Code	Testbench

//Gate Level description of AND Gate
 	
module gate(A,B,Y);
 	
input A,B;
 	
output Y;
 	
and(Y,A,B);
 	
endmodule
	module tb();
	reg A; reg B; wire Y;
	gate dut (.A(A), .B(B), .Y(Y));
	
	initial
	begin
	$dumpfile("Dump.vcd");
	$dumpvars(0);
	end
	
	initial begin
	A <= 1'b0;
	B <= 1'b0;
	#100;
	A <= 1'b0;
	B <= 1'b1;
	#100;
	A <= 1'b1;
	B <= 1'b0;
	#100;
	A <= 1'b1;
	B <= 1'b1;

	#100; $finish; end endmodule

Conclusion:

What have you learnt from this lab?

Learning Outcomes:

Upon successful completion of the lab, students will be able to understand:

LO1: What is RISC-V ISA, HDL and Visual Studio Code..