

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

Project No.: SCE14-0308

Project Title:

Automated Scripts For Testing Verilog Designs

With iVerilog

A Final Year Project Report

Presented to Nanyang Technological University (NTU)
in Partial Fulfilment of the Requirements for the
Bachelor of Engineering Degree in Computer Engineering

Prepared By:

Name : Ng Jia Hao Gary

Matric Number : U1221594E

Supervisor : Assistant Professor Suhaib A Fahmy

Examiner : Dr Owen Noel Newton Fernando

Abstract

Writing a testbench to test Verilog designs is a tedious process. The user is required to first have a very clear understanding of the design specifications. After which, a test plan can then be devised to document the test bench architecture and scenarios in detail. Undergraduates often lack the required knowledge.

Icarus Verilog, better known as iVerilog, is an open source tool primarily used in the simulation and synthesis of Verilog hardware description language for digital electronics design. With the ability to create simulation models, actual behaviour of the designed device can be tested and used to verify the correctness of the source design.

In this project, the use of iVerilog as a potential tool for the decoding of Verilog hardware description language was explored. Python scripts were written to check for the coding bugs or errors of the source design through the usage of testbenches. The resulting output was then further used on the GTKWAVE Analyzer to check for the correctness of the verilog design implemented. To ensure the robustness of the scripts written, these scripts were also tested under different test scenarios to ensure that all its intended functionalities could be performed without any issues.

Acknowledgement

The author will like to take this opportunity to express his heartfelt gratitude to the following people who have provided him with all the guidance, support and advice during the term of his final year project:

- His FYP supervisor, Assistant Professor Suhaib A Fahmy, for his invaluable advices throughout the entire project. Despite his busy schedule, he was always there to render guidance whenever there were issues or problems encountered during the term of the project.
- His fellow course mates for their encouragement during the rectification of various technical bugs and programming errors.
- Laboratory staffs for their logistical support and kind assistance on technical issues.
- Lastly, special thanks to Dr Owen Noel Newton Fernando for taking his time to read and access this report.

Table of Contents

Abstract	2
Acknowledgement	3
Table of Contents	4
List of Tables	6
List of Figures	7
Chapter 1: Introduction	10
1.1 Purpose	10
1.2 Scope	10
1.3 Project Schedule	11
1.4 Resources Used	12
1.5 Report Organisation	12
Chapter 2: Background of the project	13
2.1 What is a testbench?	13
2.2 What is Icarus Verilog?	14
2.3 Installing Icarus Verilog	14
2.3.2 Icarus Verilog Installation Guide on Linux (Ubuntu)	16
2.4 Exploring the GTKWAVE on iVerilog	17
Chapter 3: Automated Testbench Creation Script	19
3.1 Overview	19
3.2 Design Process	19
3.3 How is the testbench created?	20
3.3.1 Determine the operating environment – Ubuntu or Windows	20
3.3.2 Search and List verilog files found in directory	21
3.3.3 Removal of unnecessary comments in the selected file	22
3.3.4 Information Extraction	23

3.3.5	Implementation of Test Vectors for the testbench	26
3.3.5.1	Random Testing.....	27
3.3.5.2	Ascending Exhaustive Testing.....	28
3.3.5.3	Descending Exhaustive Testing.....	31
3.4	Test Scenarios Conducted	35
3.4.1	Project Test Cases	35
3.4.2	Test Scenarios and Results.....	35
Chapter 4: Integration with iVerilog Tools		38
4.1	Design Process	38
4.2	How does the script works?.....	38
4.2.1	Determine the operating environment – Ubuntu or Windows	38
4.2.2	Search and List verilog files found in directory	39
4.2.3	Creating of the VCD dump file.....	40
4.2.4	Using the GTKWAVE Analyzer to open the VCD dump file	41
4.3	Test Scenarios Conducted	43
4.3.1	Project Test Cases	43
4.3.2	Test Scenarios and Results.....	44
Chapter 5: Conclusion		45
5.1	Summary	45
5.3	Recommendation & Further Work	46
List of References.....		48
Appendix A: GitHub Cheat Sheet		50
Appendix B: Ubuntu Installation Guide		52

List of Tables

Table 1: Table showing the list of resources used	12
Table 2: Table showing summary of the test scenarios conducted on the testbench creation script	37
Table 3: Table showing summary of the test scenarios conducted on the script to integrate with the iVerilog tool.....	44

List of Figures

Figure 1: Figure showing the breakdown of the project development.....	11
Figure 2: Figure showing a snippet of the testbench	13
Figure 3: Figure showing a snippet of the iVerilog commands	14
Figure 4: Figure showing the Environment Variables window	16
Figure 5: Figure showing a snippet of the \$dumpfile call	18
Figure 6: Figure showing the command to launch the GTKWAVE	18
Figure 7: Figure showing the GTKWAVE Analyzer in iVerilog.....	18
Figure 8: Figure showing the design flow for the testbench creation script.....	19
Figure 9: Figure showing snippets of the code checking for operating environment	20
Figure 10: Figure showing snippet of the code to scan and display the list of verilog files found within the specified directory.....	21
Figure 11: Figure showing a list of verilog files in the same directory	21
Figure 12: Figure showing list of verilog files found	22
Figure 13: Figure showing the code snippet of a full_add.v file	22
Figure 14: Figure showing the code snippet of removeComments function	23
Figure 15: Figure showing the code snippet of full_add.v file after removing all the comments in the file using the removeComments function.....	23
Figure 16: Figure showing snippet of code to extract information for the testbench	24
Figure 17: Figure showing types of module declarations understood by the script	24
Figure 18: Figure showing snippet of the code to obtain the declared ports.....	25

Figure 19: Figure showing the sorted ports after extraction.....	26
Figure 20: Figure showing the test scenarios available for the user to select.....	26
Figure 21: Figure showing the Random Testing scenario conducted	27
Figure 22: Figure showing the code for the Random Testing	27
Figure 23: Figure showing the created testbench file of simple3.v	28
Figure 24: Figure showing the Ascending Exhaustive Testing	28
Figure 25: Figure showing the Ascending Exhaustive Testing conducted	29
Figure 26: Figure showing the code for the Ascending Exhaustive Testing.....	30
Figure 27: Figure showing the created testbench file of simple3.v using Ascending Exhaustive Testing	31
Figure 28: Figure showing the Descending Exhaustive Testing	32
Figure 29: Figure showing a snippet of the Descending Exhaustive Testing conducted.....	32
Figure 30: Figure showing the code for the Descending Exhaustive Testing	33
Figure 31: Figure showing the created testbench file of simple3.v using Descending Exhaustive Testing	34
Figure 32: Figure showing the design flow for the script to integrate with the iVerilog tool.....	38
Figure 33: Figure showing snippets of the code checking for operating environment.....	39
Figure 34: Figure showing file names found in the directory tree	39
Figure 35: Figure showing the commands to invoke the iVerilog tool	40
Figure 36: Figure showing the snippet of a vvp file.....	41

Figure 37: Figure showing the \$dumpfile call in simple3_tb.v.....	41
Figure 38: Figure showing the command to launch the GTKWAVE	42
Figure 39: Figure showing the waveform output of simple3.v	42
Figure 40: Figure showing the execution process experienced by the user	43
Figure 41: Figure showing a snippet code of a sequential logic source design	47

Chapter 1: Introduction

1.1 Purpose

The purpose of this project is to make use of the open source Icarus Verilog tool, also known as iVerilog, to develop automated scripts for testing of Verilog hardware description language design. By doing so, it provides a basis for undergraduates to learn the Verilog hardware description language better, using the Verilog structures taught in CE1005 and CE2003 modules.

1.2 Scope

The entire project is divided into 3 different phases:

Phase 1: Preparation Phase

In this phase, various functionalities within the iVerilog tool were explored. This encompassed the writing and executing of simple Verilog modules and testbenches to generate a simple waveform output using the GTKWAVE Analyzer, a feature found in the iVerilog tool.

After which, the entire installation and configuration process of the iVerilog tool for both the Windows and Ubuntu OS platform was documented in a step-by-step guide. This would allow future users of the program to be familiar with the tool faster.

Phase 2: Design & Implementation Phase

After the familiarization phase, the following were performed:

- Integration of basic iVerilog tool and Python
- Implementation of script to run iVerilog and GTKWAVE Analyzer
- Extraction of variables of an input verilog file
- Generation of a testbench via algorithm
- Generation of test vectors via algorithm
- Creation of a testbench using the extracted information

In general, a script using Python scripting language would be developed to extract variables from a given verilog file. The extracted variables would then be used to create a testbench. Test vectors to be inserted to the testbench (Random Testing, Exhaustive Ascending Testing or Exhaustive Descending Testing) were determined, before being used to produce a waveform output using the GTKWAVE Analyzer together with the given input verilog file.

Phase 3: Testing Phase

In the last phase of the project, various test scenarios were conducted to ensure that the scripts implemented were robust and error-free. In addition, a cross platform test was conducted to ensure that the scripts would be able to execute and run perfectly on both the Windows and Ubuntu OS platform.

1.3 Project Schedule

The figure below provides a detailed breakdown of the project development. Each phase was further divided into sub category to highlight the actions taken and the length of time spent for each implementation.

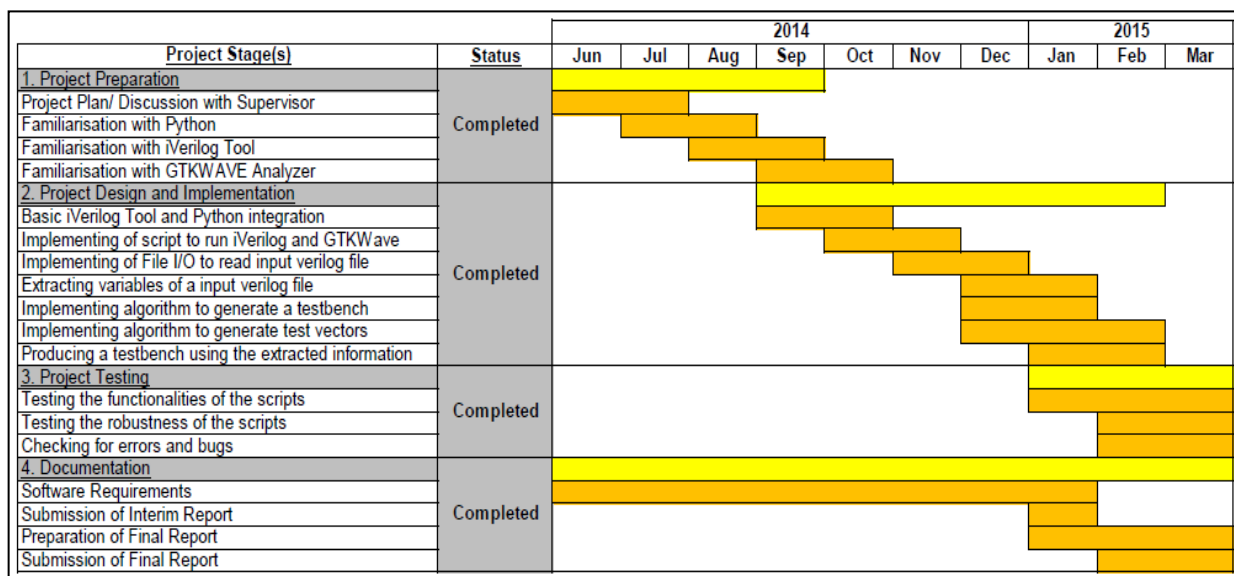


Figure 1: Figure showing the breakdown of the project development

1.4 Resources Used

The table below shows the list of resources used for the project.

<u>Name of Resources</u>	<u>Version</u>
Operating System(s)	
Windows 7	64-bits with Service Pack 1
Ubuntu OS	14.0.1
Software(s)	
Icarus Verilog (iVerilog)	0.9.7
GTKWAVE Analyzer	3.3.57
Python IDLE	3.4.1

Table 1: Table showing the list of resources used

1.5 Report Organisation

This report documented the thinking process, steps undertaken and results obtained in this project. There were 5 chapters in total, each explaining a different part of the process.

Chapter 1 began with an introduction of this project, outlining the purpose and scope. Resources used to complete the project were also listed.

Chapter 2 would provide the project background, together with iVerilog installation steps for both the OS platforms and described the various features that the iVerilog tools have – GTKWAVE Analyzer.

Chapter 3 would explain the design process of the testbench using various test scenarios on the python script.

Chapter 4 described how the script worked with the iVerilog tool.

Chapter 5 concluded with a list of recommendation for future work implementations.

Chapter 2: Background of the project

2.1 What is a testbench?

A testbench is a self-contained module with no inputs or outputs. It can be a file with simple data input such as clock pulses (positive or negative edge) or a complicated dataset that encompasses error checking and conditional testing [1].

It comprises of the following four components: (1) input of deliverables needed to interact with the unit under testing (UUT), (2) procedures to transform the input to output during the design simulation, (3) checking procedures to determine if the output meet the required standards and lastly (4) output which refers to the deliverables produced.

An example of a testbench is shown in the figure below.

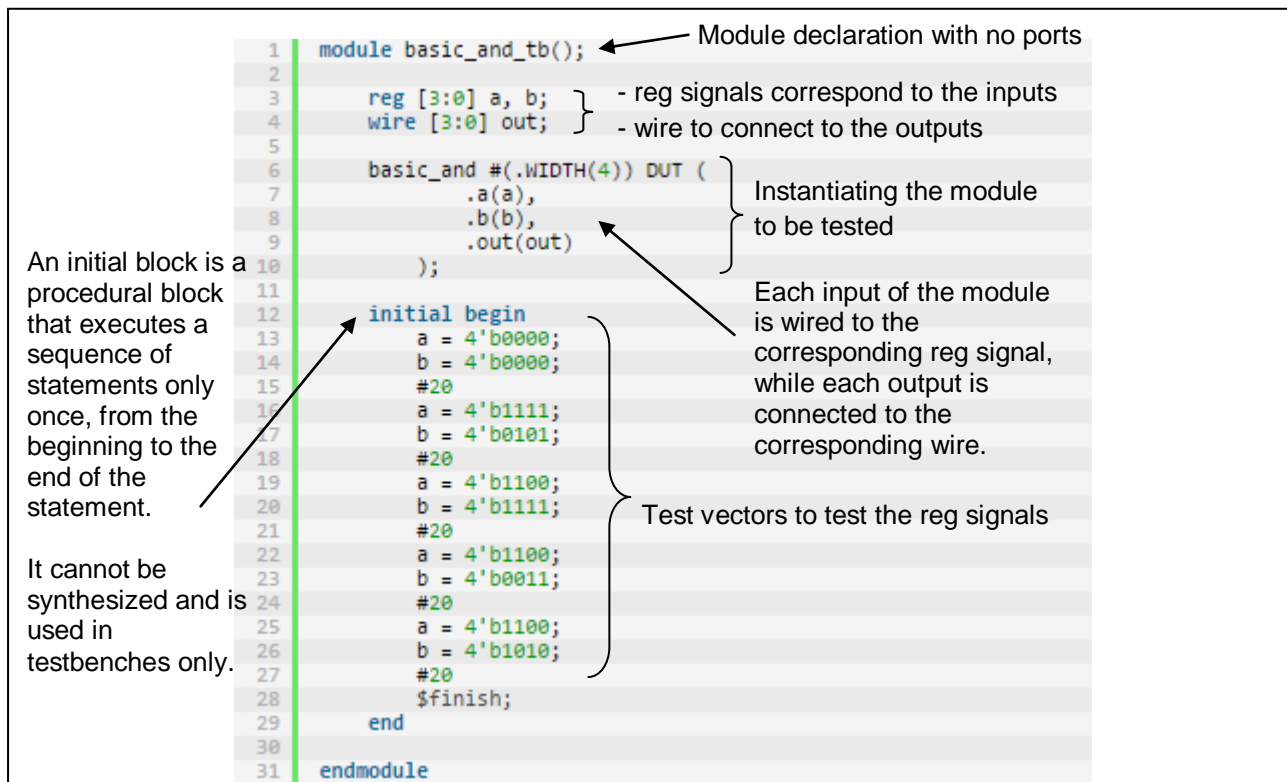


Figure 2: Figure showing a snippet of the testbench

2.2 What is Icarus Verilog?

Icarus Verilog, also known as iVerilog, is an open source tool used primarily for simulation and synthesis of Verilog hardware description language for digital electronics design [2]. It serves as a command-line tool that compiles the source design written in Verilog with a vvp simulation runtime engine. It then builds the simulation models which mimic the actual behaviour of the designed device and validate the correctness of the design implemented using testbenches [3].

The command line tool will take in the same arguments for the various supported operating environment, like Linux and Windows. An example of the arguments is shown in the figure below, where the generic prompt character "%" occurs whenever the prompt string is appropriate. Under Windows, the commands are invoked in a command window [4].

```
% iverilog -o my_design counter_tb.c counter.v  
% vvp my_design
```

Figure 3: Figure showing a snippet of the iVerilog commands

By invoking the commands as shown in Figure 3, the tool will first read and interpret the given input source files and then generate the compiled results into the output file specified by the user. The compiled results are then placed into the output file "my_design" as "-o" flag to indicate the location of the compiled results for the compiler. The vvp command then interprets the "my_design" file for the execution of the program. Both the "iverilog" and "vvp" commands are the most important commands used in the iVerilog tool. The "iverilog" command invokes the compiler, while the "vvp" command invokes the simulation runtime engine to run the simulation.

2.3 Installing Icarus Verilog

iVerilog tools can be installed to various supported operating environment such as Linux and Windows. The installation guide is compiled in Section 2.3.1 to aid future users in the setting up of the tool [5] [6].

2.3.1 Icarus Verilog Installation Guide on Windows

How to install Icarus Verilog on Windows (x64/ x86)

1. Download the latest version of iVerilog setup file (iverilog-0.9.7 setup.exe [10.5MB]) to your machine.
2. Login as an administrator and double-click on the iverilog-0.9.7 setup.exe file. It will install the iVerilog into the default path "c:\iVerilog"
3. Add the iVerilog executables "C:\iverilog\bin" to the Window Path manually. This will allow all the commands issued to be recognized by the command prompt (CMD).

How to set the path and environment variables in Windows (x64/ x86)

1. Right-click the Computer icon on the Desktop and select "Properties". If you don't have a Computer icon on your desktop, click the "Start" button, right-click the "Computer" option in the Start menu, and select "Properties".
2. Click the "Advanced System Settings" link in the left column.
3. Click on the "Advanced" tab In the System Properties window, then click the "Environment Variables" button near the bottom of that tab.
4. In the Environment Variables window (pictured below), highlight the Path variable in the "System variables" section and click the "Edit" button. Add or modify the path lines with the paths you want the computer to access. Each different directory is separated with a semicolon as shown below.

C:\Program Files;C:\Winnt;C:\Winnt\System32

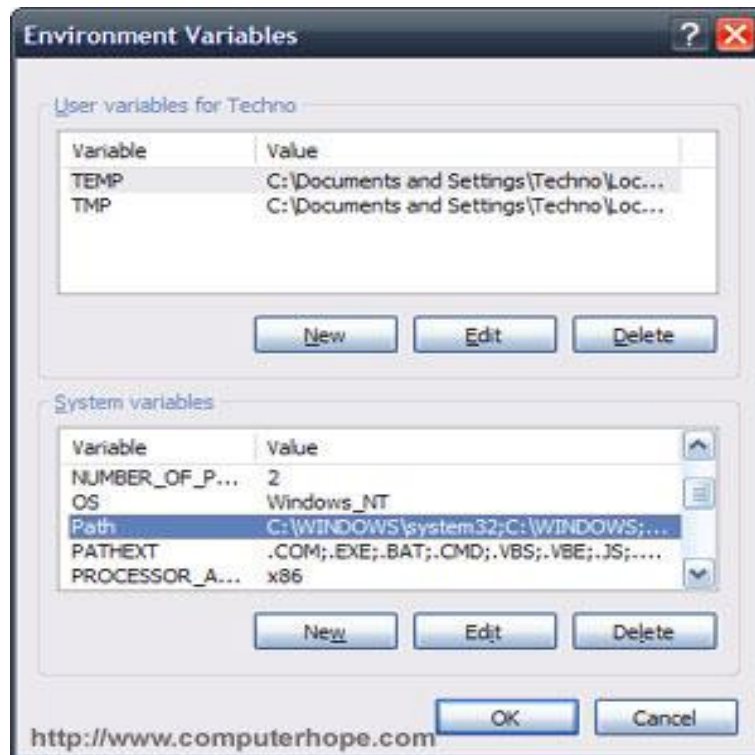


Figure 4: Figure showing the Environment Variables window

5. Add in "C:\iverilog\bin;" into the PATH and Click "OK"

How to use Icarus Verilog on Windows (x64/ x86)

```
iverilog simple.v simple_tb.v -o simple_tb.vvp
vvp simple_tb.vvp
gtkwave simple_tb.vcd
```

2.3.2 Icarus Verilog Installation Guide on Linux (Ubuntu)

How to install Icarus Verilog on Linux (Ubuntu) [7]

```
sudo su
apt-get purge verilog
apt-get install build-essential git-core autoconf gperf flex bison
git clone git://github.com/steveicarus/iverilog.git
cd iverilog
source autoconf.sh
```



```
./configure  
make  
make install  
cd ..  
rm -rf iverilog
```

How to install Icarus GTKWAVE on Linux (Ubuntu)

```
apt-get install gtkwave
```

How to use Icarus Verilog on Linux (Ubuntu) [8]

```
sudo su  
iverilog simple.v simple_tb.v -o simple_tb.vvp  
vvp simple_tb.vvp  
gtkwave simple_tb.vcd
```

2.4 Exploring the GTKWAVE on iVerilog

Besides simulating and synthesizing of the Verilog hardware design, the iVerilog tool also has a Value Change Dump (VCD) waveform viewer called GTKWAVE Analyzer. The GTKWAVE Analyzer is built upon on the GTK library, which is used as an analysis tool to perform debugging operation on the various verilog simulation models [9].

The GTKWAVE Analyzer allows the viewing of the simulation results for both the analog and digital data but it does not run interactively. Instead, it relies on the use of the dumpfiles generated by the vvp simulation runtime engine while executing the testbenches [10].

By default, the vvp simulation runtime engine will generate the VCD dump into the file specified by the \$dumpfile system task. If the \$dumpfile call is absent, the compiler will then choose the file name dump.vcd or dump.lxt, depending on

runtime flags specified. The figure below shows an example of how the \$dumpfile call is used to generate the VCD dump.

```
initial
begin
  $dumpfile("test.dump");
  $dumpvars(0, test);
end
...
```

vvp simulation runtime engine will generate the VCD dump into the "test.dump" specified by the user.

Figure 5: Figure showing a snippet of the \$dumpfile call

Once the VCD dump has been generated by the vvp simulation runtime engine, the GTKWAVE Analyzer can then be used to view the waveform output by issuing the following command:

```
gtkwave test.dump
```

Figure 6: Figure showing the command to launch the GTKWAVE

Upon the execution of the command as shown in Figure 6, the GTKWAVE Analyzer will be launched, displaying the corresponding waveform results produced by the simulation from the VCD dump file. An example of the waveform output produced is shown in the figure below.

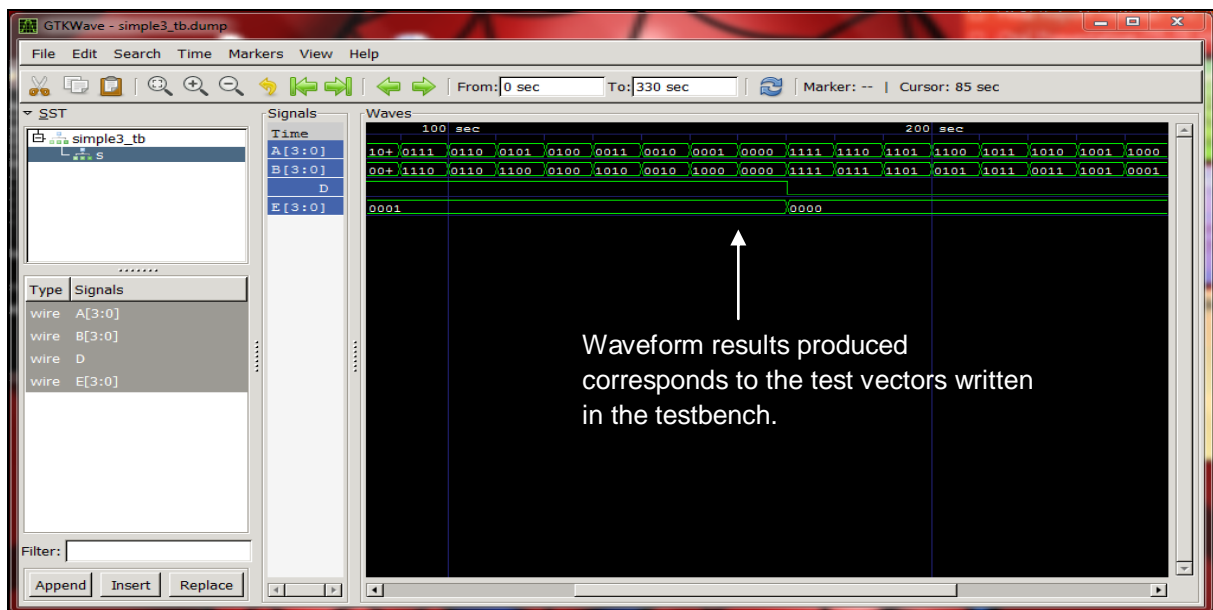


Figure 7: Figure showing the GTKWAVE Analyzer in iVerilog

Chapter 3: Automated Testbench Creation Script

3.1 Overview

Python is a high-level feature-rich programming language that has gained broad acceptance in a wide range of applications. Python emphasizes greatly on code readability and its syntax allows programmer to express concepts using fewer lines of coding than what is required in other language such as C++ or Java [11]. In addition, Python is also cross-platform compatible in which both Linux and Windows operating environment can be used. Therefore, python programming language will be used to create the testbench creation script.

3.2 Design Process

The figure below shows the process flow that was put in place for the design and implementation of the testbench creation script.

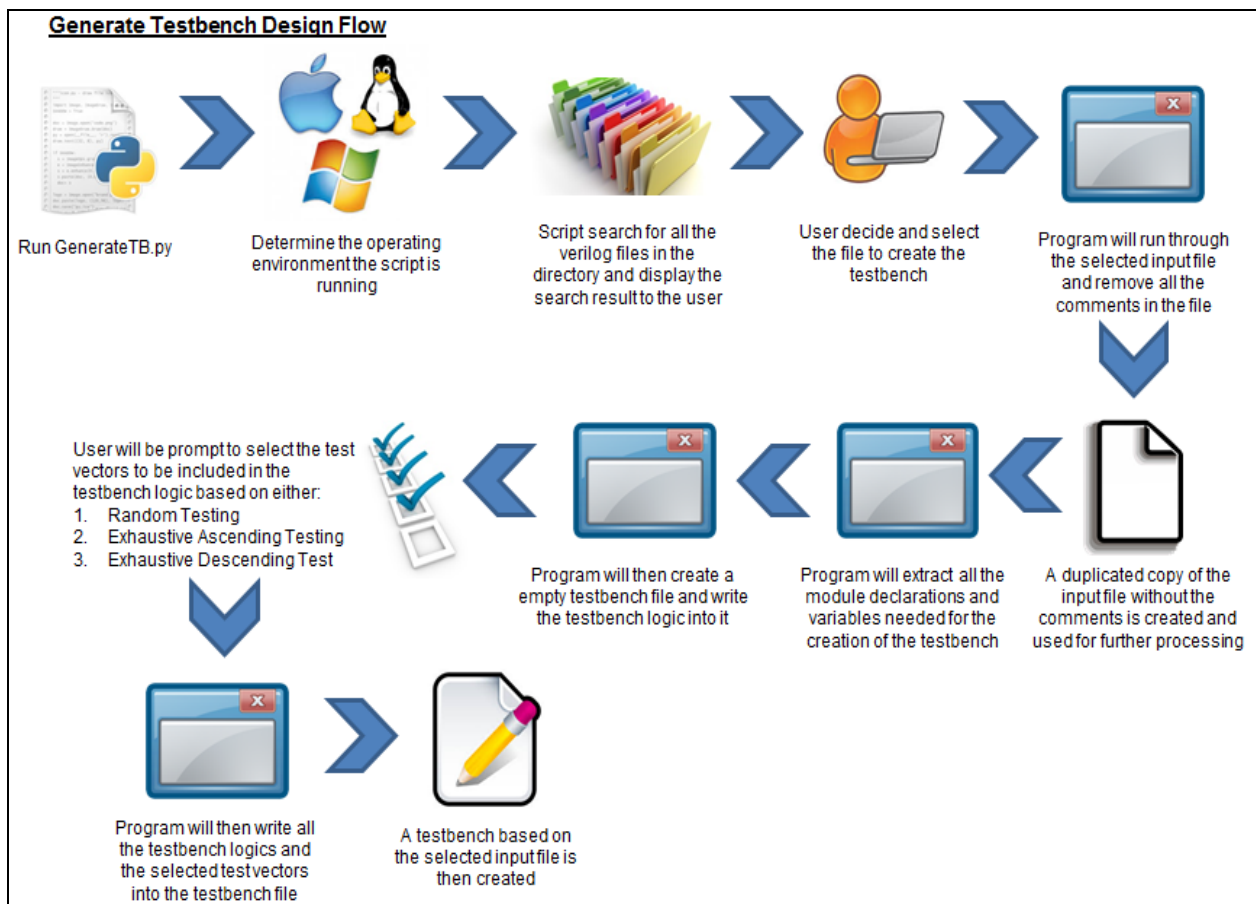


Figure 8: Figure showing the design flow for the testbench creation script

3.3 How is the testbench created?

3.3.1 Determine the operating environment – Ubuntu or Windows

Before the testbench creation script runs, it will first check and determine the environment in which the script will be executed in. For instance, if the script is operating in the Windows environment, the script will automatically execute the segment of the written program code for the execution in Windows. Conversely, the program code for execution in Ubuntu operating environment will run instead if the operation is performed in Ubuntu. Snippets of the code implemented to check for the operating environment that the script is executed in, are shown in the figure below.

```
# Determine the OS to decide which method to execute
# For Linux Platform - Run using os.system
if sys.platform.startswith('linux'):

    # Infinite loop
    while (1):

        PrintIntroduction()
        # Declaring variables to list files in directory
        filesFound = []
        listFile = []
        i = 1
        print("List of Verilog Files found to execute:")
        for root, dirs, files in os.walk('/home/'):

#-----#
# For Windows Platform - Run using .bat file
if sys.platform.startswith('win'):

    # Infinite loop
    while (1):

        PrintIntroduction()
        # Declaring variables to list files in directory
        filesFound = []
        listFile = []
        i = 1
        print("List of Verilog Files found to execute:")
        for root, dirs, files in os.walk('C://iverilog/modules/')
```

Figure 9: Figure showing snippets of the code checking for operating environment

By detecting the operating environment that the script will be executed in, appropriate segment of the program code can then be executed correctly. This checking process is essential, as the directory where all the verilog files will be

stored and used for execution is different for both operating environments. For example, in the Ubuntu operating environment, the verilog files will be stored and executed from the directory of '/home/' whereas in Windows, the directory will be in 'C:\iverilog\modules\'.

3.3.2 Search and List verilog files found in directory

After executing the segment of code based on the operating environment, the script will subsequently use the “os.walk()” function to recursively scan a directory for any verilog files found within the specified directory and display the scanned result to the user. An example is illustrated in the figures below.

```
PrintIntroduction()
# Declaring variables to list files in directory
filesFound = []
listFile = []
i = 1
print("List of Verilog Files found:")
for root, dirs, files in os.walk('C://iverilog/modules/'):
    for file in files:
        if file.endswith('.v') and not file.endswith('.tb.v'):
            listFile = file
            filesFound.append(file)
            print(str(i) + '. ', listFile)
            i += 1
print("\n")
```

Figure 10: Figure showing snippet of the code to scan and display the list of verilog files found within the specified directory

In this example, a total of 5 different verilog files are placed within the “C:\iverilog\modules” directory.

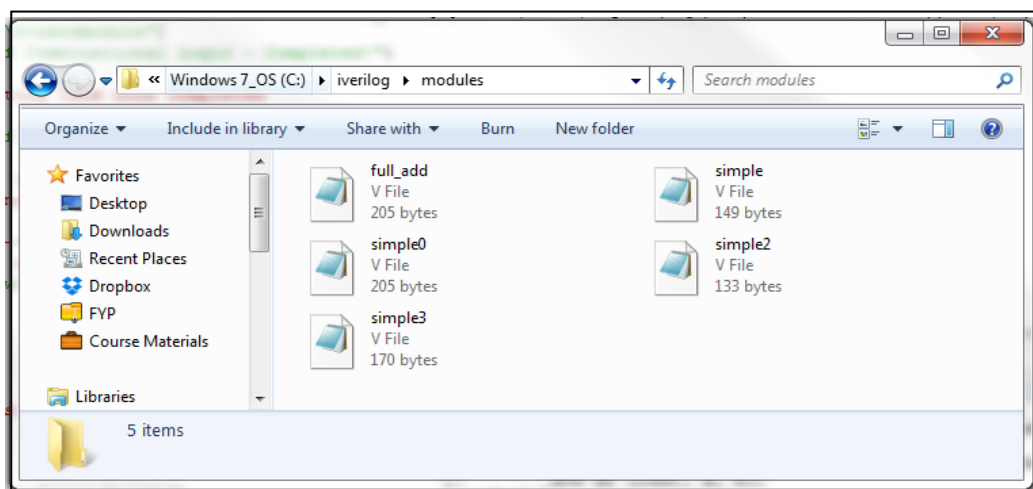


Figure 11: Figure showing a list of verilog files in the same directory

If the verilog files are found within the specified directory upon the execution of the script, it will be displayed to the user as shown in Figure 12 below.

```
iVerilog Testbench Generator
=====
This program generate a testbench file for execution purposes.
Files should be saved in C:\iverilog\modules (Win) or \home\ (Linux).
=====
List of Verilog Files found:
1. full_add.v
2. simple.v
3. simple0.v
4. simple2.v
5. simple3.v
} List of verilog files found corresponds
  to the example given in Figure 11.

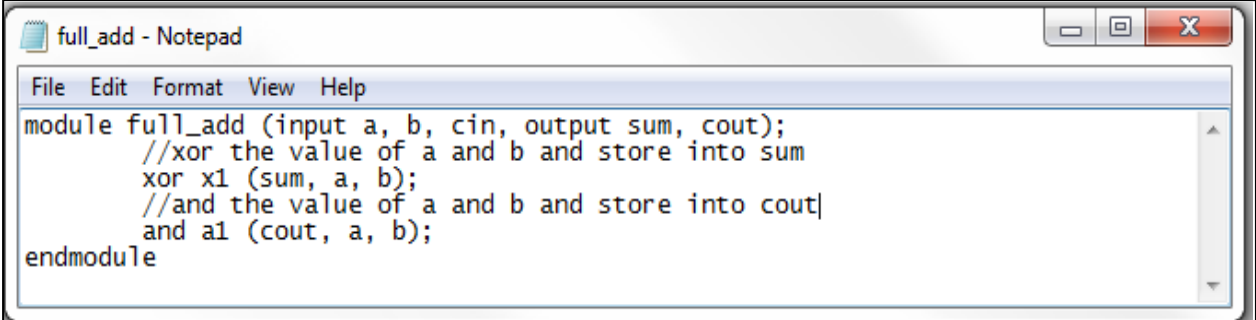
Please select a file: |
```

Figure 12: Figure showing list of verilog files found

The user can then select the file of interest to be used as input for the execution. The script will process the input file further to produce a testbench based on the source design of the selected file.

3.3.3 Removal of unnecessary comments in the selected file

Once the input file to be executed is selected, the script will first remove all the comments found in the file. This will prevent the script from processing wrong information and extracting incorrect ports declaration which are needed in the later phase of the implementation instead. This process is illustrated in Figures 13 to 15 below.



```
File Edit Format View Help
module full_add (input a, b, cin, output sum, cout);
    //xor the value of a and b and store into sum
    xor x1 (sum, a, b);
    //and the value of a and b and store into cout
    and a1 (cout, a, b);
endmodule
```

Figure 13: Figure showing the code snippet of a full_add.v file

```
# removing the comments in the modules of the verilog file
def removeComments(textInput):
    def reConstruct(match):
        i = match.group()
        if i.startswith('/'):
            return ""
        else:
            return i
    pattern = re.compile(r'//.*?$|/\*.*?\*/|\'(?:\\.|[^\']*)*\'|"(?:\\.|[^\"])*"'
                        , re.DOTALL | re.MULTILINE)
    return re.sub(pattern, reConstruct, textInput)
```

Figure 14: Figure showing the code snippet of removeComments function

As shown in the figure above, the removeComments function makes use of the regular expression module in the Python library, to find any comments written in the verilog file. The verilog comments usually begin with the character of '/'. The function in the regular expression module then checks if a particular string matches with the given expression, and returns that particular string by replacing the original string with the replacement string. If the pattern is not found, the string is returned unchanged. The figure below shows the full_add.v file after the execution.

```
Please select a file: 1
Opening file: full_add.v
Reading Contents of the file - Completed!
module full_add (input a, b, cin, output sum, cout);

    xor x1 (sum, a, b);

    and a1 (cout, a, b);
endmodule
Removing Comments in File - Completed!
Extracting Module Name - Completed!
```

Figure 15: Figure showing the code snippet of full_add.v file after removing all the comments in the file using the removeComments function

3.3.4 Information Extraction

Once the file is free from any comments, the script will then extract all the necessary information required to create a testbench. This information includes the module declarations, the module name, as well as the various ports declared. These input parameters will be converted to reg signals to create a testbench.

Likewise, the output variables will be converted to wires to connect the output of the testbench. The figure below shows snippet of the code used to obtain the various input parameters required for the parameterised testbench.

```
# extract module declaration as section between module keyword and semicolon
moduleDeclaration = output_noComments.split('module')[1].split(';')[0]

# extract module name
module_name = moduleDeclaration.split('(')[0].strip()

# creating output verilog file at the specified directory
os.chdir("C:\\iVerilog\\Output")
new_tbFileName = module_name + "_tb.v"
new_file = open(new_tbFileName, 'w') ## a will append, w will over-write

# writing the testbench block
new_file.write("module " + module_name + "_tb;\n")

# replacing the variables: input to reg, output to wire
inputVariables_dict = dict()
outputVariables_dict = dict()
listofVariables = []
```

Figure 16: Figure showing snippet of code to extract information for the testbench

To obtain the variables, also known as the ports declared in the file, the script will check if the ports are declared within the module declaration. The written python script can only understand two different types of module declarations (one with the directions and widths within the brackets, one with only the ports declared within the brackets). An example of the two different accepted module declarations is shown in the figure below.

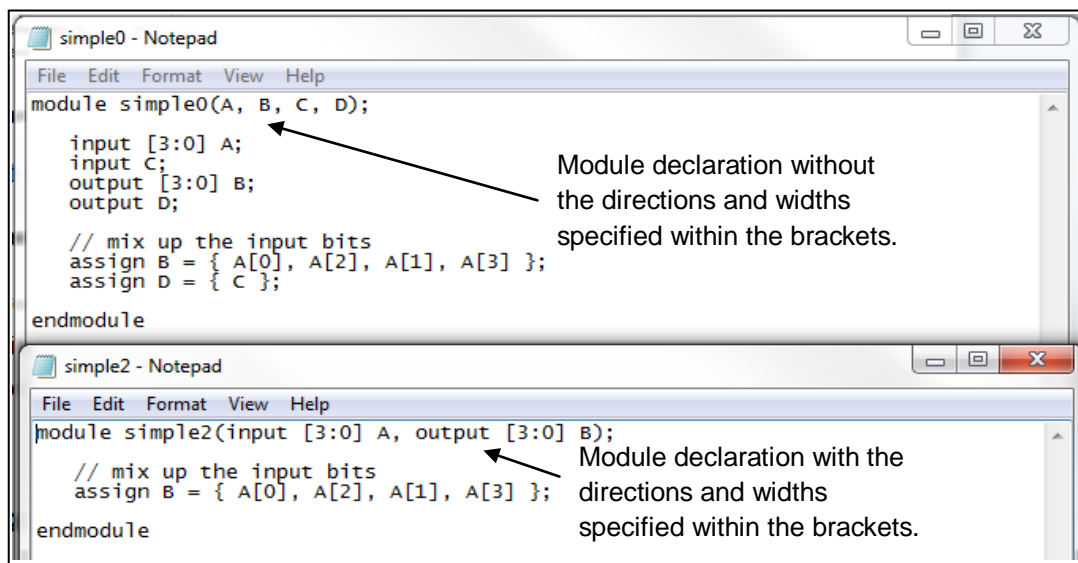


Figure 17: Figure showing types of module declarations understood by the script

The script will determine the type of module declaration for the file, before looping through the declaration to obtain the respective declared ports. This includes obtaining the direction of the port (either input or output), the bit-width of the port as well as the name of the port. The implementation code is shown in the Figure 18.

```
# Looping through the moduleDeclaration to extract the data
for x in range(len(newPortList)):
    if newPortList[x] == "input":
        typeOfVariable = newPortList[x]
        if(newPortList[x+1].startswith '[') == True:
            bitPort = newPortList[x+1]

            # Extract two numbers separated by colon surrounded by square brackets
            bits = [int(x) for x in newPortList[1].split('[')[1].split(')')[0].split(':')]
            bit_width = abs(bits[0]-bits[1])+1
        else:
            bit_width = 1
            bitPort = ''
    elif newPortList[x] == "output":
        typeOfVariable = newPortList[x]
        if(newPortList[x+1].startswith '[') == True:
            bitPort = newPortList[x+1]

            # Extract two numbers separated by colon surrounded by square brackets
            bits = [int(x) for x in newPortList[1].split('[')[1].split(')')[0].split(':')]
            bit_width = abs(bits[0]-bits[1])+1
        else:
            bit_width = 1
    elif (newPortList[x].startswith '[') == False:

        # Extract variables without the keywords or bitwidth
        value = newPortList[x]
        # writing the variables
        # renaming the variables; input to reg, output to wire
        if(typeOfVariable == "input"):
            new_file.write(str("\nreg " + str(bitPort) + " " + value + ";"))
            print(str("Input Variables: reg " + str(bitPort) + " " + value + ";"))
            listOfVariables.append(value)

        else:
            new_file.write(str("\nwire " + str(bitPort) + " " + value + ";"))
            print(str("Output Variables: wire " + str(bitPort) + " " + value + ";"))
            listOfVariables.append(value)

        # Sorting the variables respecting into input and output
        if (typeOfVariable == "input"):
            inputVariables_dict[value] = bit_width
        else:
            outputVariables_dict[value] = bit_width

# Sorting the variables based on their bits width
flipped = {}
for key, value in inputVariables_dict.items():
    if value not in flipped:
        flipped[value] = [key]
    else:
        flipped[value].append(key)
```

Figure 18: Figure showing snippet of the code to obtain the declared ports

As shown in the figure above, the ports extracted from the module declaration will be converted to either reg signals or wires respectively and sorted based on the direction of the port (either input or output). Once the sorting is done, it will then be stored into the respective dictionaries. The ports bit-width sizes are also obtained by taking the absolute value of the most significant bit (MSB) to the least significant bit (LSB). The ports are further sorted based on their bit-width size to be used for the test vectors implementation [12]. The figure below shows the sorted ports after extraction.

```

Opening file: simple3.v
Reading Contents of the file - Completed!
module simple3(input [3:0] A, input D, output [3:0] B, E);

    assign B = { A[0], A[2], A[1], A[3] };
    assign E = { D } ;

endmodule
Removing Comments in File - Completed!
Extracting Module Name - Completed!

Creating output file .....
Writing testbench blocks .....
Input Variables: reg [3:0] A;
Input Variables: reg D;
Output Variables: wire [3:0] B;
Output Variables: wire [3:0] E;
Input ports: {1: ['D'], 4: ['A']}
Writing of Combinational Logic .....
  
```

List of sorted ports after the extraction:
 1. Input ports changed to reg signals
 2. Output ports changed to wires

Figure 19: Figure showing the sorted ports after extraction

3.3.5 Implementation of Test Vectors for the testbench

Once the ports with their respective bit-width size are obtained, test vectors can be then be implemented to the test the circuit logic design of the input verilog file. The test scenarios available are shown in the figure below.

```

Test Scenarios Available:
=====
1. Random Testing
2. Ascending Exhaustive Testing
3. Descending Exhaustive Testing

Please select a test scenario:
  
```

Figure 20: Figure showing the test scenarios available for the user to select

3.3.5.1 Random Testing

In this test scenario, randomised binary values are generated to be assigned to respective reg signals. These values are obtained from the sum of all the bit-width size of the respective reg signals. For instance, a reg signal D with size of 1 bit-width and reg signal A with size of 4 bit-width, using the assigned test vector value of 5'b10010, the binary value of 1'b1 will be allocated to D while the value of 4'b0010 will be allocated to A respectively. The output of the Random Testing approach conducted is shown in the figure below.

```
Test Scenarios Available:
=====
1. Random Testing
2. Ascending Exhaustive Testing
3. Descending Exhaustive Testing

Please select a test scenario: 1
You have selected: Random Testing

Variable: D,A, Assigned Binary Value is: 5'b11110
Variable: D,A, Assigned Binary Value is: 5'b11011
Variable: D,A, Assigned Binary Value is: 5'b10010
Variable: D,A, Assigned Binary Value is: 5'b10010
Writing of Combinational Logic - Completed!
Writing of output file completed!
Output File Directory: C:\iVerilog\Output
```

Figure 21: Figure showing the Random Testing scenario conducted

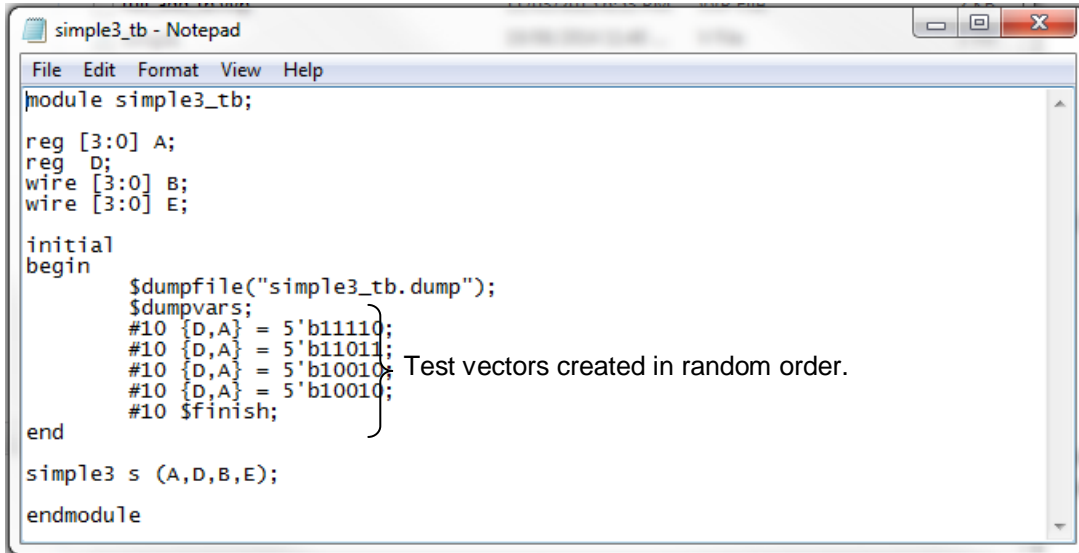
The figure below shows the code for implementing the Random Testing.

```
bit_width = 0
assignVariable = []
if inputValue == 1:
    print("You have selected: Random Testing\n")
    for x in flipped.keys():
        assignVariable.append(', '.join(map(str, flipped[x])))
        bit_width += x
    assignVariable = ', '.join(assignVariable)

    for x in range (startpoint,i+1):
        endpoint = (2**bit_width-1)
        inputVar = random.randint(startpoint,endpoint)
        charFormat = '#0' + str(bit_width+2) + 'b'
        binaryValue = format(inputVar, charFormat)
        print("Variable: " + assignVariable + ", Assigned Binary Value is: " + str(bit_width) + " " + str(binaryValue)[1:])
        new_file.write("\n\t#" + str(time) + " {" + assignVariable + "} = " + str(bit_width) + " " + str(binaryValue)[1:] + ";")
```

Figure 22: Figure showing the code for the Random Testing

Once the test vectors have been implemented, the script will then create the testbench using all the extracted information obtained from the earlier phases. The figure below shows the created testbench of simple3.v file using Random Testing approach to test the source design.



```

module simple3_tb;
reg [3:0] A;
reg D;
wire [3:0] B;
wire [3:0] E;

initial
begin
    $dumpfile("simple3_tb.dump");
    $dumpvars;
    #10 {D,A} = 5'b11110;
    #10 {D,A} = 5'b11011;
    #10 {D,A} = 5'b10010;
    #10 {D,A} = 5'b10010;
    #10 $finish;
end

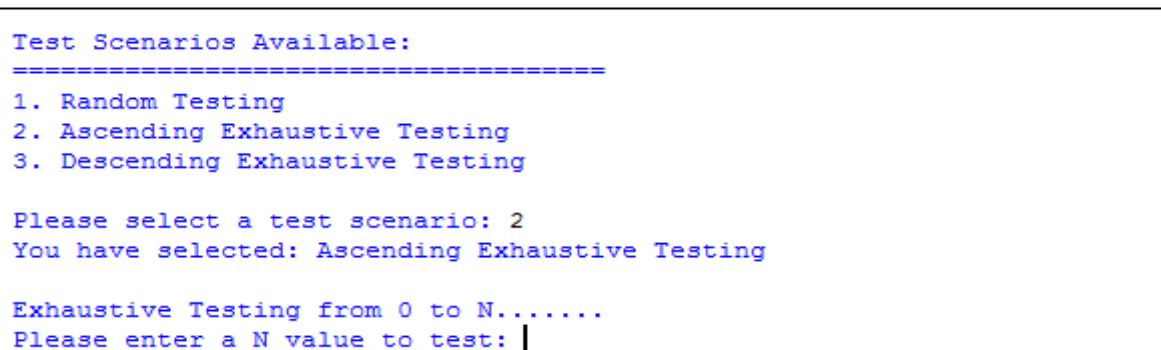
simple3 s (A,D,B,E);
endmodule
  
```

Test vectors created in random order.

Figure 23: Figure showing the created testbench file of simple3.v

3.3.5.2 Ascending Exhaustive Testing

In this test scenario, exhaustive testing will be conducted on the source design in an ascending manner. This means that all the possible data combinations will be used as the test vectors for the testing the source design. For instance, if the sum of all the bit-width size of the respective reg signals is 8-bit wide, then the total possible test data combination to be done will range from binary values of 8'b00000000 to 8'b11111111. The Ascending Exhaustive Testing approach executed is shown in the figure below.



```

Test Scenarios Available:
=====
1. Random Testing
2. Ascending Exhaustive Testing
3. Descending Exhaustive Testing

Please select a test scenario: 2
You have selected: Ascending Exhaustive Testing

Exhaustive Testing from 0 to N.....
Please enter a N value to test: |
  
```

Figure 24: Figure showing the Ascending Exhaustive Testing

The script will prompt the user for a value to be tested. If the input value exceeds the maximum possible value of the total bit-width size, then the maximum value of which the total bit-width size is will be used. Otherwise, the value input by the user will be used as the terminating value for the implementation of the test vectors. An example is shown in the figure below.

```
Exhaustive Testing from 0 to N.....
Please enter a N value to test: 100
Error! Input value of 100 exceed the possible test value!
Ascending Exhaustive Test from 0 to the max possible value of N will be done instead!
Variable: D,A, Assigned Binary Value is: 5'b000000
Variable: D,A, Assigned Binary Value is: 5'b000001
Variable: D,A, Assigned Binary Value is: 5'b000010
Variable: D,A, Assigned Binary Value is: 5'b000011
Variable: D,A, Assigned Binary Value is: 5'b000100
Variable: D,A, Assigned Binary Value is: 5'b000101
Variable: D,A, Assigned Binary Value is: 5'b000110
Variable: D,A, Assigned Binary Value is: 5'b000111
Variable: D,A, Assigned Binary Value is: 5'b001000
Variable: D,A, Assigned Binary Value is: 5'b001001
Variable: D,A, Assigned Binary Value is: 5'b001010
Variable: D,A, Assigned Binary Value is: 5'b001011
Variable: D,A, Assigned Binary Value is: 5'b001100
Variable: D,A, Assigned Binary Value is: 5'b001101
Variable: D,A, Assigned Binary Value is: 5'b001110
Variable: D,A, Assigned Binary Value is: 5'b001111
Variable: D,A, Assigned Binary Value is: 5'b010000
Variable: D,A, Assigned Binary Value is: 5'b010001
Variable: D,A, Assigned Binary Value is: 5'b010010
Variable: D,A, Assigned Binary Value is: 5'b010011
Variable: D,A, Assigned Binary Value is: 5'b010100
Variable: D,A, Assigned Binary Value is: 5'b010101
Variable: D,A, Assigned Binary Value is: 5'b010110
Variable: D,A, Assigned Binary Value is: 5'b010111
Variable: D,A, Assigned Binary Value is: 5'b011000
Variable: D,A, Assigned Binary Value is: 5'b011001
Variable: D,A, Assigned Binary Value is: 5'b011010
Variable: D,A, Assigned Binary Value is: 5'b011011
Variable: D,A, Assigned Binary Value is: 5'b011100
Variable: D,A, Assigned Binary Value is: 5'b011101
Variable: D,A, Assigned Binary Value is: 5'b011110
Variable: D,A, Assigned Binary Value is: 5'b011111
Writing of Combinational Logic - Completed!
Writing of output file completed!
Output File Directory: C:\iVerilog\Output
```

Figure 25: Figure showing the Ascending Exhaustive Testing conducted

As illustrated in Figure 25, the binary value of 1'b1 will be allocated to reg signal D with size of 1 bit-width while the value of 4'b0010 will be allocated to reg signal A with size of 4 bit-width respectively using the assigned test vector value of 5'b10010.

The figure below shows the code for implementing the Ascending Exhaustive Testing.

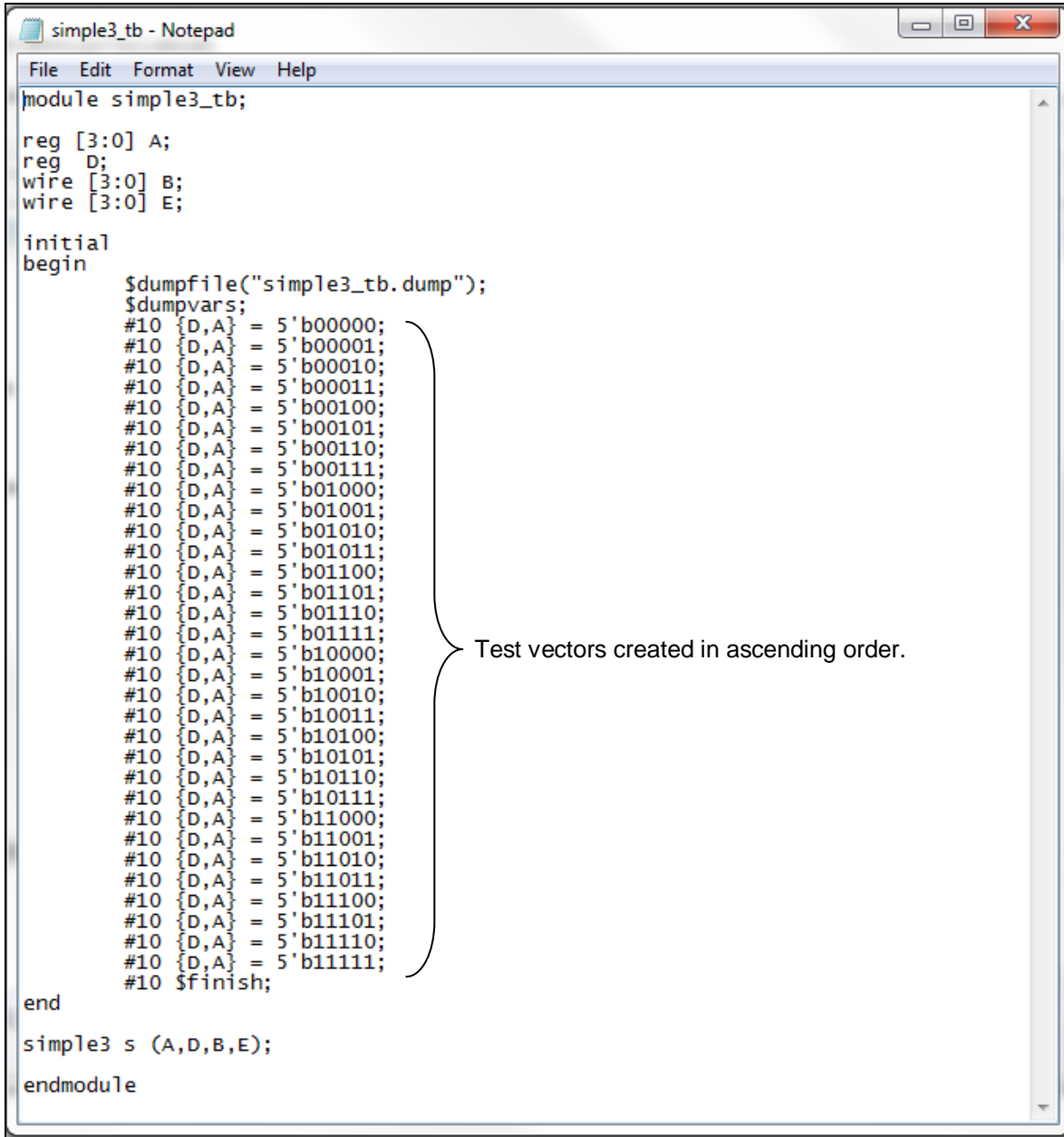
```
elif inputValue == 2:
    print("You have selected: Ascending Exhaustive Testing\n")
    print("Exhaustive Testing from 0 to N.....")
    for x in flipped.keys():
        assignVariable.append(', '.join(map(str, flipped[x])))
        bit_width += x
    assignVariable = ', '.join(assignVariable)
    possibleValue = 2**bit_width-1

    inputNvalue = int(input("Please enter a N value to test: "))
    # Check if the input N value will exceed the possible value to be tested
    while (inputNvalue > 2**bit_width or inputNvalue < 1):
        if(inputNvalue > 2**bit_width):
            print("Error! Input value of " + str(inputNvalue) + " exceed the possible test value!")
        else:
            print("Error! Input value of " + str(inputNvalue) + " is less than 1!")
        print("Ascending Exhaustive Test from 0 to the max possible value of N will be done instead!")

    endpoint = (2**bit_width)
    for x in range (startpoint, endpoint):
        charFormat = '#0' + str(bit_width+2) + 'b'
        binaryValue = format(x, charFormat)
        print("Variable: " + assignVariable + ", Assigned Binary Value is: " + str(bit_width) + "'" + str(binaryValue)[1:])
        new_file.write("\n\t#" + str(time) + " {" + assignVariable + "} = " + str(bit_width) + "'" + str(binaryValue)[1:] + ";")
        break
    else:
        endpoint = inputNvalue
        for x in range (startpoint, endpoint):
            charFormat = '#0' + str(bit_width+2) + 'b'
            binaryValue = format(x, charFormat)
            print("Variable: " + assignVariable + ", Assigned Binary Value is: " + str(bit_width) + "'" + str(binaryValue)[1:])
            new_file.write("\n\t#" + str(time) + " {" + assignVariable + "} = " + str(bit_width) + "'" + str(binaryValue)[1:] + ";")
```

Figure 26: Figure showing the code for the Ascending Exhaustive Testing

Once the test vectors are implemented, the script will then create the testbench using all the extracted information obtained from the earlier phases. The figure below shows the created testbench of simple3.v file using the Ascending Exhaustive Testing approach to test the source design.



```

module simple3_tb;
reg [3:0] A;
reg D;
wire [3:0] B;
wire [3:0] E;

initial
begin
    $dumpfile("simple3_tb.dump");
    $dumpvars;
    #10 {D,A} = 5'b00000;
    #10 {D,A} = 5'b00001;
    #10 {D,A} = 5'b00010;
    #10 {D,A} = 5'b00011;
    #10 {D,A} = 5'b00100;
    #10 {D,A} = 5'b00101;
    #10 {D,A} = 5'b00110;
    #10 {D,A} = 5'b00111;
    #10 {D,A} = 5'b01000;
    #10 {D,A} = 5'b01001;
    #10 {D,A} = 5'b01010;
    #10 {D,A} = 5'b01011;
    #10 {D,A} = 5'b01100;
    #10 {D,A} = 5'b01101;
    #10 {D,A} = 5'b01110;
    #10 {D,A} = 5'b01111;
    #10 {D,A} = 5'b10000;
    #10 {D,A} = 5'b10001;
    #10 {D,A} = 5'b10010;
    #10 {D,A} = 5'b10011;
    #10 {D,A} = 5'b10100;
    #10 {D,A} = 5'b10101;
    #10 {D,A} = 5'b10110;
    #10 {D,A} = 5'b10111;
    #10 {D,A} = 5'b11000;
    #10 {D,A} = 5'b11001;
    #10 {D,A} = 5'b11010;
    #10 {D,A} = 5'b11011;
    #10 {D,A} = 5'b11100;
    #10 {D,A} = 5'b11101;
    #10 {D,A} = 5'b11110;
    #10 {D,A} = 5'b11111;
    #10 $finish;
end

simple3 s (A,D,B,E);
endmodule
  
```

Test vectors created in ascending order.

Figure 27: Figure showing the created testbench file of simple3.v using Ascending Exhaustive Testing

3.3.5.3 Descending Exhaustive Testing

In this test scenario, exhaustive testing will be conducted on the source design in a descending manner. This means that all the possible data combinations will be used as the test vectors for the testing the source design. For instance, if the sum of all the bit-width size of the respective reg signals is 8-bit wide, then the total possible test data combination to be done will range from binary values of

8'b11111111 to 8'b00000000. The Descending Exhaustive Testing approach executed is shown in the figure below.

```
Test Scenarios Available:
=====
1. Random Testing
2. Ascending Exhaustive Testing
3. Descending Exhaustive Testing

Please select a test scenario: 3
You have selected: Descending Exhaustive Testing

Exhaustive Testing from 0 to N.....
Please enter a N value to test: |
```

Figure 28: Figure showing the Descending Exhaustive Testing

As shown in the figure above, the script will prompt the user for a value to be tested. If the input value exceeds the maximum possible value of the total bit-width size, then the maximum value of which the total bit-width size is will be used. Otherwise, the value input by the user will be used as the commencing value for the implementation of the test vectors. An example is shown in the figure below.

```
Exhaustive Testing from 0 to N.....
Please enter a N value to test: 100
Error! Input value of 100 exceed the possible test value!
Descending Exhaustive Test from the max possible value N to 0 will be done instead!
Variable: D,A, Assigned Binary Value is: 5'b11111
Variable: D,A, Assigned Binary Value is: 5'b11110
Variable: D,A, Assigned Binary Value is: 5'b11101
Variable: D,A, Assigned Binary Value is: 5'b11100
Variable: D,A, Assigned Binary Value is: 5'b11011
Variable: D,A, Assigned Binary Value is: 5'b11010
Variable: D,A, Assigned Binary Value is: 5'b11001
Variable: D,A, Assigned Binary Value is: 5'b11000
Variable: D,A, Assigned Binary Value is: 5'b10111
Variable: D,A, Assigned Binary Value is: 5'b10110
Variable: D,A, Assigned Binary Value is: 5'b10101
Variable: D,A, Assigned Binary Value is: 5'b10100
```

Figure 29: Figure showing a snippet of the Descending Exhaustive Testing conducted

As shown in the figure above, for instance if a reg signal D with size of 1 bit-width and reg signal A with size of 4 bit-width, using the assigned test vector value of 5'b10010, the binary value of 1'b1 will be allocated to D while the value of 4'b0010 will be allocated to A respectively.

The figure below shows the code for implementing the Descending Exhaustive Testing.

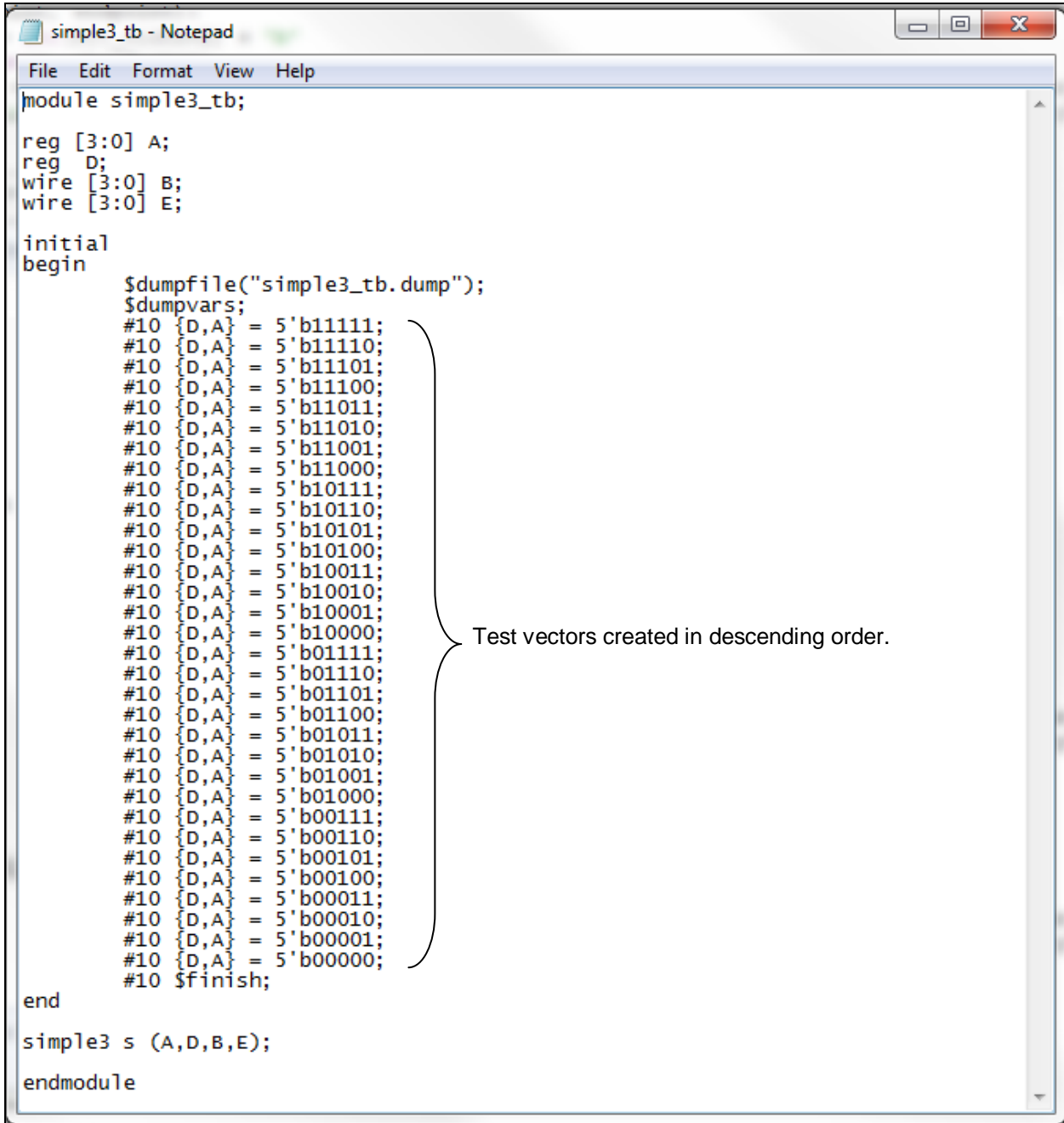
```
elif inputValue == 3:
    print("You have selected: Descending Exhaustive Testing\n")
    print("Exhaustive Testing from 0 to N.....")
    for x in flipped.keys():
        assignVariable.append(', '.join(map(str, flipped[x])))
        bit_width += x
    assignVariable = ', '.join(assignVariable)
    possibleValue = 2**bit_width-1

    inputValue = int(input("Please enter a N value to test: "))
    # Check if the input N value will exceed the possible value to be tested
    while (inputNvalue > 2**bit_width or inputNvalue < 1):
        if(inputNvalue > possibleValue):
            print("Error! Input value of " + str(inputNvalue) + " exceed the possible test value!")
        else:
            print("Error! Input value of " + str(inputNvalue) + " is less than 1!")
        print("Descending Exhaustive Test from the max possible value N to 0 will be done instead!")

    endpoint = possibleValue
    for x in range (endpoint, startpoint-1, -1):
        charFormat = '#0' + str(bit_width+2) + 'b'
        binaryValue = format(x, charFormat)
        print("Variable: " + assignVariable + ", Assigned Binary Value is: " + str(bit_width) + "'" + str(binaryValue)[1:])
        new_file.write("\n\t#" + str(time) + " {" + assignVariable + "} = " + str(bit_width) + "'" + str(binaryValue)[1:] + ";")
        break
    else:
        endpoint = inputNvalue
        for x in range (endpoint-1, startpoint-1, -1):
            charFormat = '#0' + str(bit_width+2) + 'b'
            binaryValue = format(x, charFormat)
            print("Variable: " + assignVariable + ", Assigned Binary Value is: " + str(bit_width) + "'" + str(binaryValue)[1:])
            new_file.write("\n\t#" + str(time) + " {" + assignVariable + "} = " + str(bit_width) + "'" + str(binaryValue)[1:] + ";")
```

Figure 30: Figure showing the code for the Descending Exhaustive Testing

Once the test vectors have been implemented, the script will then create the testbench using all the extracted information obtained from the earlier phases. The figure below shows the created testbench of simple3.v file using the Descending Exhaustive Testing approach to test the source design.



```

module simple3_tb;

reg [3:0] A;
reg D;
wire [3:0] B;
wire [3:0] E;

initial
begin
    $dumpfile("simple3_tb.dump");
    $dumpvars;
    #10 {D,A} = 5'b11111;
    #10 {D,A} = 5'b11110;
    #10 {D,A} = 5'b11101;
    #10 {D,A} = 5'b11100;
    #10 {D,A} = 5'b11011;
    #10 {D,A} = 5'b11010;
    #10 {D,A} = 5'b11001;
    #10 {D,A} = 5'b11000;
    #10 {D,A} = 5'b10111;
    #10 {D,A} = 5'b10110;
    #10 {D,A} = 5'b10101;
    #10 {D,A} = 5'b10100;
    #10 {D,A} = 5'b10011;
    #10 {D,A} = 5'b10010;
    #10 {D,A} = 5'b10001;
    #10 {D,A} = 5'b10000;
    #10 {D,A} = 5'b01111;
    #10 {D,A} = 5'b01110;
    #10 {D,A} = 5'b01101;
    #10 {D,A} = 5'b01100;
    #10 {D,A} = 5'b01011;
    #10 {D,A} = 5'b01010;
    #10 {D,A} = 5'b01001;
    #10 {D,A} = 5'b01000;
    #10 {D,A} = 5'b00111;
    #10 {D,A} = 5'b00110;
    #10 {D,A} = 5'b00101;
    #10 {D,A} = 5'b00100;
    #10 {D,A} = 5'b00011;
    #10 {D,A} = 5'b00010;
    #10 {D,A} = 5'b00001;
    #10 {D,A} = 5'b00000;
    #10 $finish;
end

simple3 s (A,D,B,E);

endmodule

```

Test vectors created in descending order.

Figure 31: Figure showing the created testbench file of simple3.v using Descending Exhaustive Testing

3.4 Test Scenarios Conducted

3.4.1 Project Test Cases

To ensure that the written script is able to perform the intended objectives, it will have to be challenged under different scenarios. The test scenarios comprise of the following factors:

1. Ability to detect the verilog files in a directory correctly
2. Ability to execute in both Ubuntu and Windows operating environment
3. Ability to extract the ports declaration, module declaration correctly
4. Ability to implement the test vectors correctly
5. Ability to generate the testbench successfully
6. Ability to aid undergraduates in learning the Verilog hardware language
 - a. Testbench created should be easily to understand
 - b. Testbench created should be similar to the syntax taught in lectures

3.4.2 Test Scenarios and Results

The summary test results based on the evaluation factors of the test scenarios is shown in the table below

<u>Evaluation Factors</u>	<u>Description</u>	<u>Outcome</u>
Cross-platform compatibility	The script implemented should be able to identify the operating environment the script is executing in and execute the segment of the written program code correctly.	✓
Listing of verilog files found	The script implemented should be able to correctly list out all the verilog files found within the specified directory.	✓

User input validation	If a user enters an index value which exceed the total index value of the listed files found, the script should be able to display an error message to the user and prompt the user to re-enter the index value again.	✓
Checking if the input verilog file is supported by the script	The script should be able to determine if the module declaration of the input verilog file is supported by the script. At present, the script is only able to understand two different types of module declarations (one with the directions and widths within the brackets, one with only the ports declared within the brackets).	✓
Removing of the comments in the verilog file	The script should be able to remove all the comments written in the verilog file to prepare for the information extraction to create the testbench file.	✓
Extracting of all the necessary information required to create a testbench	The script should be able to extract the following information from the given file: 1. Module declaration 2. Ports declaration 3. Module name	✓
Convert the ports obtained correctly	The script should be able to change all the input ports to reg signals and all the output ports to wires correctly.	✓

Obtaining the correct bit-width value of the port correctly	The script should be able to obtain the correct bit-width size values of the respective ports correctly based on the port MSB and LSB value.	✓
Creating test vectors for testing the source design	The script should be able to generate the respective test vectors correctly that will be written into the testbench file to verify the correctness of the source design implemented.	✓
Producing a testbench file at the end of the execution	The script should be able to successfully generate the testbench output file with all the necessary input parameters and test vectors.	✓
Syntax of testbench created	The syntax of the testbench created should be similar to the testbench taught in CE1005 and CE2003 lectures	✓
Readability of the testbench created	The testbench created should be easy to understand, well-structured and organised.	✓

Table 2: Table showing summary of the test scenarios conducted on the testbench creation script

From the test results shown in Table 2, the written python script achieved all the 12 intended objectives. The various functionalities of the script have been tested thoroughly and verified to be functioning correctly.

Chapter 4: Integration with iVerilog Tools

4.1 Design Process

The figure below shows the process flow that was put in place for the design and implementation of the script to integrate with iVerilog tool.

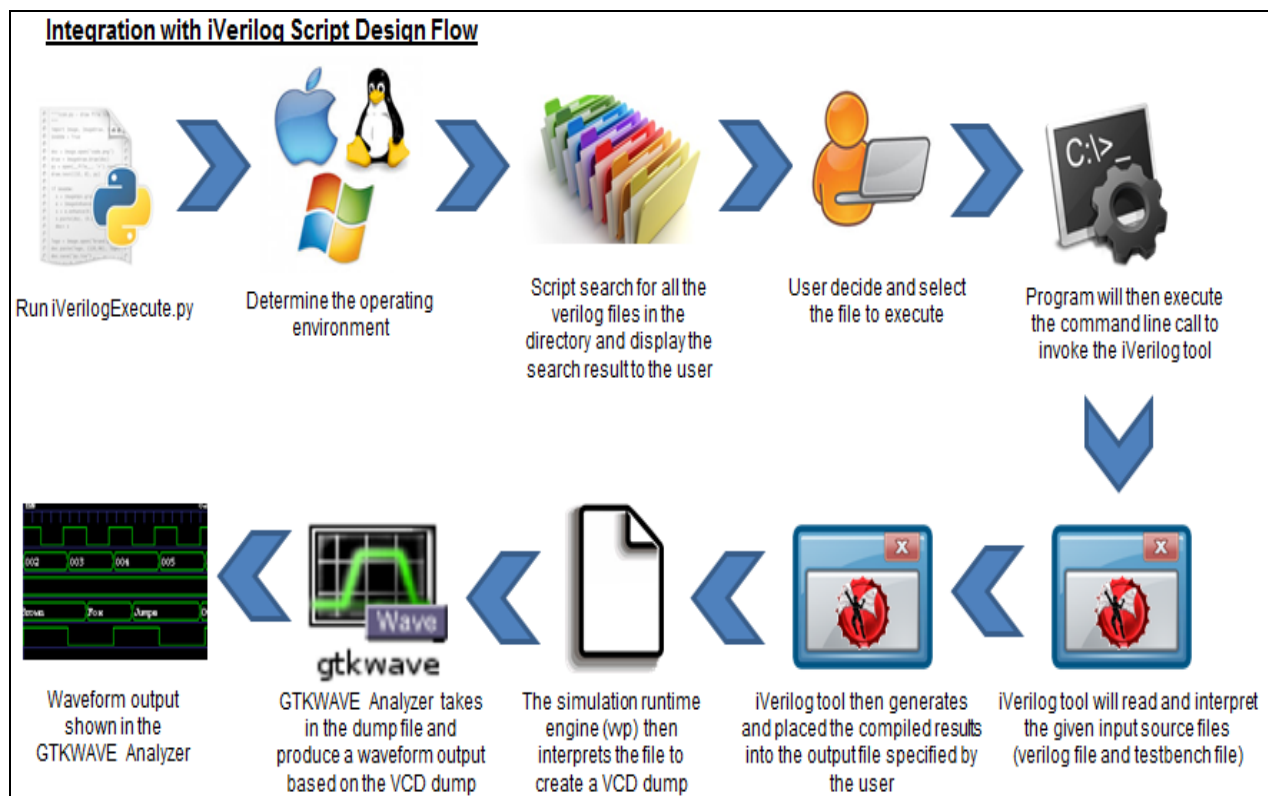


Figure 32: Figure showing the design flow for the script to integrate with the iVerilog tool

4.2 How does the script works?

4.2.1 Determine the operating environment – Ubuntu or Windows

Similar to the python script that was explained previously, the script that is used to integrate with the iVerilog tool will also first identify the operating environment that the script will be running in and execute the appropriate segment of the program code for either Windows or Ubuntu. An example is shown in the figure below, where snippets of the code are implemented to check for the operating environment that the script is executed in.

```
# Determine the OS to decide which method to execute
# For Linux Platform - Run using os.system
if sys.platform.startswith('linux'):

    # Declaring variables to list files in directory
    filesFound = []
    i = 1
    PrintIntroduction();
    print("List of Verilog Files found to execute:")

# For Windows Platform - Run using .bat file
if sys.platform.startswith('win'):

    PrintIntroduction();
    # Declaring variables to list files in directory
    filesFound = []
    i = 1
    print("List of Verilog Files found to execute:")
```

Figure 33: Figure showing snippets of the code checking for operating environment

Since the syntax declarations in the command line call are different for Windows and Ubuntu, this checking process allows the script to work in both operating systems. In Ubuntu, the command line call will be made using the “os.system()” module instead of using a batch file that contains a series of commands to be executed by the command line interpreter.

4.2.2 Search and List verilog files found in directory

For the convenience of the user, the script will use the “os.walk()” function to recursively scan a directory for any verilog files stored within the specified directory. The “os.walk()” function will then generate the file names found in the directory tree by walking the tree either in a top down or bottom-up approach. The file names will then be shown to the user. An example is illustrated in the figure below.

```
iVerilog Automated Script Program Execution
=====
This program takes in verilog files to generate a output waveform.
Files should be saved in C:\iverilog\modules (Win) or \home\ (Linux).
=====
List of Verilog Files found to execute:
1. full_add.v
2. simple.v
3. simple0.v
4. simple2.v
5. simple3.v

Please select a file: |
```

Figure 34: Figure showing file names found in the directory tree

Subsequently, the user can then select the file of interest that the script will be based on to execute the command line call and invoke the iVerilog tool. For example, if the user decides to run simple3.v file, the script will then use the command line call to invoke the iVerilog tool using the commands as shown below. The iVerilog tool will take in the same arguments for both the Ubuntu and Windows operating environments.

```
% iverilog -o simple3 simple3_tb.v simple3.v  
% vvp simple3
```

Figure 35: Figure showing the commands to invoke the iVerilog tool

4.2.3 Creating of the VCD dump file

Once the commands are invoked, the iVerilog tool will read and interpret the input source files given by the user. The compiled results are then written into “simple3” output file. If the compilation is successful, a vvp file will be generated and used to feed to the simulator at the later phase of the execution.

The output from the iVerilog tool is not executable on any platform by itself. Hence, the vvp command will invoke the simulation runtime engine to run the simulation which will then interpret “simple3” file, thereby allowing the program to execute. Both the “iverilog” and “vvp” commands are the most important and commonly used commands in the iVerilog tool.

A snippet of how a vvp file will look like is shown in the figure below. The vvp file will contain all the arguments and parameters required for the simulation.


```

#! c:/iverilog/bin/vvp
:ivl_version "0.10.0 (devel)" "(s20130827)";
:ivl_delay_selection "TYPICAL";
:vpi_time_precision + 0;
:vpi_module "system";
:vpi_module "vhdl_sys";
:vpi_module "v2005_math";
:vpi_module "va_math";|
S_006DBC20 .scope module, "simple3_tb" "simple3_tb" 2 1;
.timescale 0 0;
v0061C848_0 .var "A", 3 0;
v00642FC8_0 .net "B", 3 0, L_00643230; 1 drivers
v00643020_0 .var "D", 0 0;
v00643078_0 .net "E", 3 0, L_006432E0; 1 drivers
S_006DF858 .scope module, "s" "simple3" 2 47, 3 1 0, S_
006DBC20;
.timescale 0 0;
.port_info 0 /INPUT 4 "A"
.port_info 1 /INPUT 1 "D"
.port_info 2 /OUTPUT 4 "B"
.port_info 3 /OUTPUT 4 "E"
v00621F48_0 .net "A", 3 0, v0061C848_0; 1 drivers
v006DE128_0 .net "B", 3 0, L_00643230; alias, 1 drivers

```

Figure 36: Figure showing the snippet of a vvp file

4.2.4 Using the GTKWAVE Analyzer to open the VCD dump file

By default, the vvp simulation runtime engine will generate a VCD dump into the file specified by the \$dumpfile system task during compilation. If the \$dumpfile call is absent, the compiler will then choose the file name dump.vcd or dump.txt, depending on runtime flags specified.

The figure below shows an example of how the \$dumpfile call is used to generate the VCD dump.

```

initial
begin
    $dumpfile("simple3_tb.dump");
    $dumpvars();
end
...

```

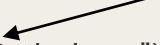

 vvp simulation runtime engine will generate the VCD dump into the "simple.dump" specified in the testbench.

Figure 37: Figure showing the \$dumpfile call in simple3_tb.v

Once the VCD dump is created, the GTKWAVE Analyzer can then be used to view the waveform output by issuing the following command:

```
gtkwave simple3_tb.dump
```

Figure 38: Figure showing the command to launch the GTKWAVE

Upon the execution of the command as shown above, the GTKWAVE Analyzer will be launched, displaying the corresponding waveform results produced by the simulation from the VCD dump file. An example of the waveform output produced is shown in the figure below.

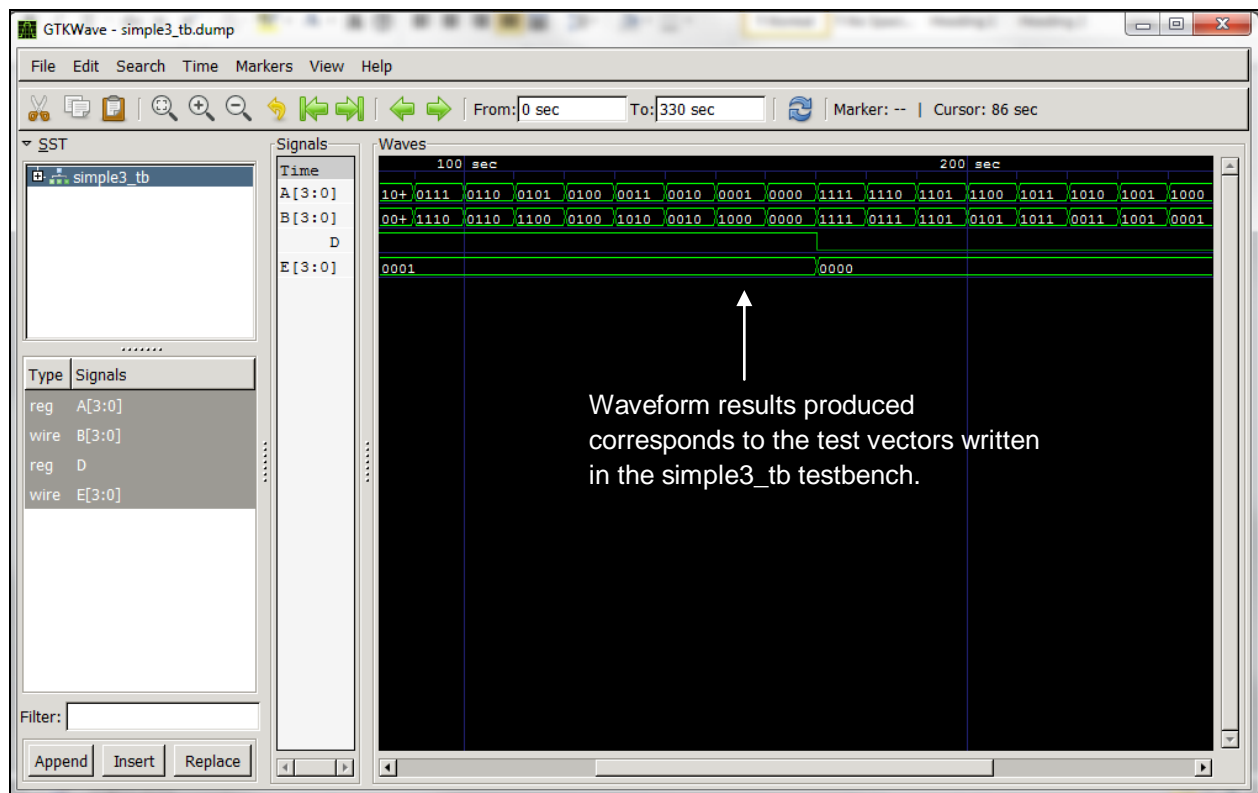


Figure 39: Figure showing the waveform output of simple3.v

By comparing the waveform output obtained with the testbench created based on the input verilog file, the user will be able to check the behaviour of the source design to validate if it has been implemented correctly.

Having to memorise all the execution commands in order to generate the waveform output is tedious. Hence, the script is redesigned such that the

command instructions are inbuilt. The user will just need to select the file of interest and the script will automatically managed from there to produce the waveform output file.

The waveform will be displayed on the GTKWAVE Analyzer for the user. No user intervention is required once the user selects the file of interest. The entire process will transit in the following format as shown in the figure below.

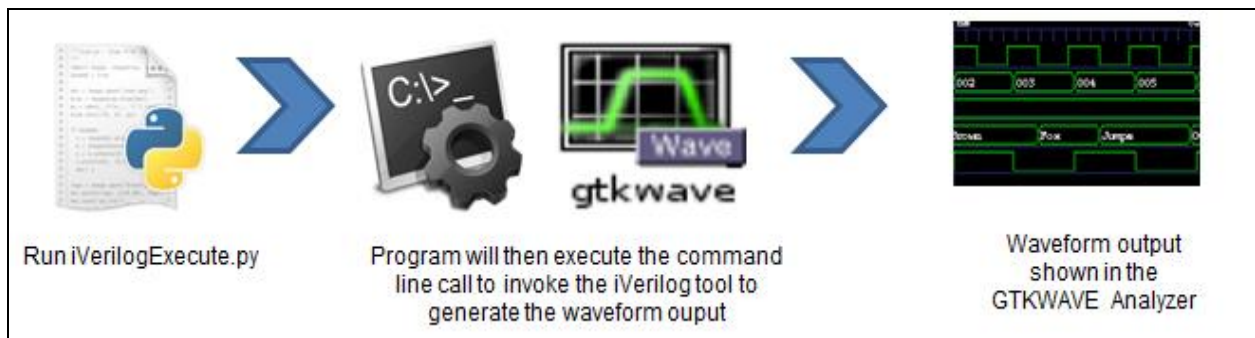


Figure 40: Figure showing the execution process experienced by the user

4.3 Test Scenarios Conducted

4.3.1 Project Test Cases

The script was tested and validated under different test scenarios, comprising of the following factors:

1. Ability to detect the verilog files in a directory correctly
2. Ability to execute in both Ubuntu and Windows operating environment
3. Ability to self-run with no user intervention once the file is selected
4. Ability to generate the waveform output successfully
 - a. Waveform output is displayed on the GTKWAVE Analyzer
 - b. Waveform display on the GTKWAVE Analyzer corresponds with the testbench test vectors created

4.3.2 Test Scenarios and Results

The summary test results based on the evaluation factors of the test scenarios are shown in the table below

<u>Evaluation Factors</u>	<u>Description</u>	<u>Outcome</u>
Cross-platform compatibility	The script implemented should be able to identify the operating environment the script is executing in and execute the segment of the written program code correctly.	✓
Listing of verilog files found	The script implemented should be able to correctly list out all the verilog files found within the specified directory.	✓
User input validation	If a user enter an index value which exceed the total index value of the listed found files, the script should be able to display an error message to the user and prompt the user to re-enter the index value again.	✓
Executing of the command line calls	The script should be able to generate the command line calls to invoke the iVerilog tool and simulation runtime engine so that the waveform output can be generated using the GTKWAVE Analyzer.	✓
User Intervention	No user intervention is required once the input file is selected.	✓

Table 3: Table showing summary of the test scenarios conducted on the script to integrate with the iVerilog tool

From the test results as shown in Table 3, the written python script is able to meet its intended objectives. The various functionalities of the script have also been tested thoroughly and verified to be functioning correctly.

Chapter 5: Conclusion

5.1 Summary

This project demonstrates how python scripts can be created and integrated together to automate the process of creating testbenches via the interface of the iVerilog tool. It involves the initial exploration phase of the iVerilog tool to understand its features and functionalities, to the design of the python scripts and finally the successful implementation and validation under different scenarios. The objectives and the goals of the project have been met, to generate a robust testbench from a given input verilog file.

5.2 Discussion

The project utilises recent IT technology to create testbenches of source design. This is beyond the scope of undergraduate academic context and requires independent learning of the tool.

As such, several problems were encountered during the process. One of which is due to the usage of Python to obtain the declared ports from a given verilog file. These ports can be of various bit-width and different directions like either input or output. Therefore, there is a need to design an effective algorithm to sort the ports appropriately.

In the initial phase of script development, only the extraction of the ports could be performed. The script could only copy out the declared ports and convert them to reg or wire respectively in the output testbench file. They were not sorted based on their respective bit-width size correctly. Since the sorting was necessary for the next phase of development to determine the correct test vectors size to be created, this shortfall hindered the entire development process. It took numerous research as well as several rounds of trial and error before the script was rewritten to resolve this issue.

Next, the scripts have to be cross-platform compatible and should work in both Ubuntu and Windows operating environments. However, both the operating

environments have different syntax of executing a command line call. For instance, in the Windows operating environment, the command line calls could be made using a batch file containing a series of commands to be executed by the command line interpreter. But this batch file would not be able to work in the Ubuntu operating environment. Instead, in Ubuntu, the command line call is made using the “os.system()” module. These syntax differences added the difficulty in the script writing.

Another problem encountered was the unfamiliarity with the usage of GitHub to store the code repository. Git is a version control software that manages changes made to a project without any overwriting. In this project, all the codes have to be uploaded to the GitHub file repository for access. With no prior experience in using GitHub, the author has to be familiarized with how it works.

To conclude, the requirements of this project have been met. The software setup configuration guides highlighted in this report could be leveraged by others with similar interests and serves as a foundation for future development works using the same iVerilog tool. This will also provide the developer with a better insight of the iVerilog tool.

5.3 Recommendation & Further Work

5.3.1 Sequential Logic Implementation

As the automated testbench creation script implemented now can only focus mainly on combinational logic designs, further improvement can be done to include sequential logic designs. This means that instead of working with files with mainly assign and always blocks, the script will then be working with edge sensitive elements in the sensitive list of an always block. An example of a sequential logic file is the flip-flop module as shown in the figure below.

```
module dff (input clk, reset, d, output q);
reg q;

always @ (posedge clk)
begin
    if (reset == 1) begin
        q <= 0;
    end else begin
        q <= d;
    end
end

endmodule
```

Edge sensitive elements in the sensitive list of always block.

Nonblocking assignments usually used for sequential logic circuit design.

Figure 41: Figure showing a snippet code of a sequential logic source design

5.3.2 Top Module Implementation

In addition, the implemented script can also be further modified to support top module implementation in circuit which allows more holistic and complex testbench designs to be created. For example, the script should be able to work with module that can be connected to another module in a hierarchical design model, that the lower level modules are instantiated within the higher level module.

5.3.3 Accepting of test vectors from user

Due to the time constraint, the author was not able to implement the additional add-on feature for the testbench creation script. This add-on feature will allow the user to be able to input their own test vectors from a given input file to test the source design apart from the given testing methods.

List of References

- [1] "Writing Test Benches," Embedded Mirco, 205. [Online]. Available: <https://embeddedmicro.com/tutorials/mojo/writing-test-benches>. [Accessed 2015].
- [2] "Icarus Verilog," Wikipedia, 17 January 2015. [Online]. Available: http://en.wikipedia.org/wiki/Icarus_Verilog. [Accessed 2015].
- [3] "Icraus Verilog," Wikia, [Online]. Available: <http://iverilog.wikia.com/wiki/Introduction>. [Accessed 2014].
- [4] S. Williams and M. Baxter, "Icarus Verilog: Open-Source Verilog More Than a Year Later," *LINUX JOURNAL*, p. 2, 2012.
- [5] S. Williams, "Icarus Verilog for Windows," Bleyer, [Online]. Available: <http://bleyer.org/icarus/>. [Accessed 2014].
- [6] "How to install Icarus Verilog on Ubuntu," GeorgeBlog, 14 Jun 2010. [Online]. Available: <http://georgeblog.nyarangi.com/2010/06/how-to-install-icarus-verilog-on-ubuntu.html>. [Accessed 2014].
- [7] "RootSudo," Ubuntu Documentation, 22 02 2015. [Online]. Available: <https://help.ubuntu.com/community/RootSudo>. [Accessed 2015].
- [8] "How to enable root login," askUbuntu, [Online]. Available: <http://askubuntu.com/questions/44418/how-to-enable-root-login>. [Accessed 2015].
- [9] "GTKWAVE," Wikia, [Online]. Available: <http://iverilog.wikia.com/wiki/GTKWAVE>. [Accessed 2014].
- [10] "GTKWave 3.3 Wave Analyzer User's Guide," [Online]. Available: gtkwave.sourceforge.net/gtkwave.pdf. [Accessed 2015].
- [11] "The Python Tutorial," 2015. [Online]. Available:

<https://docs.python.org/3/tutorial/>. [Accessed 2014].

- [12] "Split elements of a list in python," Stackoverflow, [Online]. Available: <http://stackoverflow.com/questions/6696027/split-elements-of-a-list-in-python>. [Accessed 2015].
- [13] "Using GitHub," Sparkfun, [Online]. Available: <https://learn.sparkfun.com/tutorials/using-github>. [Accessed 2014].
- [14] C. Buckey, "How to Use Git and GitHub," UDACITY, [Online]. Available: <https://www.udacity.com/course/ud775>. [Accessed 2014].
- [15] "Hello World," GitHub Guides, [Online]. Available: <https://guides.github.com/activities/hello-world/>. [Accessed 2014].
- [16] "Verilog Online," iVerilog.com, [Online]. Available: <http://iverilog.com/examples.php>. [Accessed 2014].
- [17] "Simulation," Wikia, [Online]. Available: <http://iverilog.wikia.com/wiki/Simulation>. [Accessed 2015].

Appendix A: GitHub Cheat Sheet

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT

GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows

<https://windows.github.com>

GitHub for Mac

<https://mac.github.com>

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms

<http://git-scm.com>

CONFIGURE TOOLING

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```

Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```

Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```

Enables helpful colorization of command line output

CREATE REPOSITORIES

Start a new repository or obtain one from an existing URL

```
$ git init [project-name]
```

Creates a new local repository with the specified name

```
$ git clone [url]
```

Downloads a project and its entire version history

MAKE CHANGES

Review edits and craft a commit transaction

```
$ git status
```

Lists all new or modified files to be committed

```
$ git diff
```

Shows file differences not yet staged

```
$ git add [file]
```

Snapshots the file in preparation for versioning

```
$ git diff --staged
```

Shows file differences between staging and the last file version

```
$ git reset [file]
```

Unstages the file, but preserve its contents

```
$ git commit -m "[descriptive message]"
```

Records file snapshots permanently in version history

GROUP CHANGES

Name a series of commits and combine completed efforts

```
$ git branch
```

Lists all local branches in the current repository

```
$ git branch [branch-name]
```

Creates a new branch

```
$ git checkout [branch-name]
```

Switches to the specified branch and updates the working directory

```
$ git merge [branch]
```

Combines the specified branch's history into the current branch

```
$ git branch -d [branch-name]
```

Deletes the specified branch

REFACTOR FILENAMES

Relocate and remove versioned files

```
$ git rm [file]
```

Deletes the file from the working directory and stages the deletion

```
$ git rm --cached [file]
```

Removes the file from version control but preserves the file locally

```
$ git mv [file-original] [file-renamed]
```

Changes the file name and prepares it for commit

SUPPRESS TRACKING

Exclude temporary files and paths

```
*.log  
build/  
temp-*
```

A text file named `.gitignore` suppresses accidental versioning of files and paths matching the specified patterns

```
$ git ls-files --other --ignored --exclude-standard
```

Lists all ignored files in this project

SAVE FRAGMENTS

Shelve and restore incomplete changes

```
$ git stash
```

Temporarily stores all modified tracked files

```
$ git stash pop
```

Restores the most recently stashed files

```
$ git stash list
```

Lists all stashed changesets

```
$ git stash drop
```

Discards the most recently stashed changeset

REVIEW HISTORY

Browse and inspect the evolution of project files

```
$ git log
```

Lists version history for the current branch

```
$ git log --follow [file]
```

Lists version history for a file, including renames

```
$ git diff [first-branch]...[second-branch]
```

Shows content differences between two branches

```
$ git show [commit]
```

Outputs metadata and content changes of the specified commit

REDO COMMITS

Erase mistakes and craft replacement history

```
$ git reset [commit]
```

Undoes all commits after `[commit]`, preserving changes locally

```
$ git reset --hard [commit]
```

Discards all history and changes back to the specified commit

SYNCHRONIZE CHANGES

Register a repository bookmark and exchange version history

```
$ git fetch [bookmark]
```

Downloads all history from the repository bookmark

```
$ git merge [bookmark]/[branch]
```

Combines bookmark's branch into current local branch

```
$ git push [alias] [branch]
```

Uploads all local branch commits to GitHub

```
$ git pull
```

Downloads bookmark history and incorporates changes

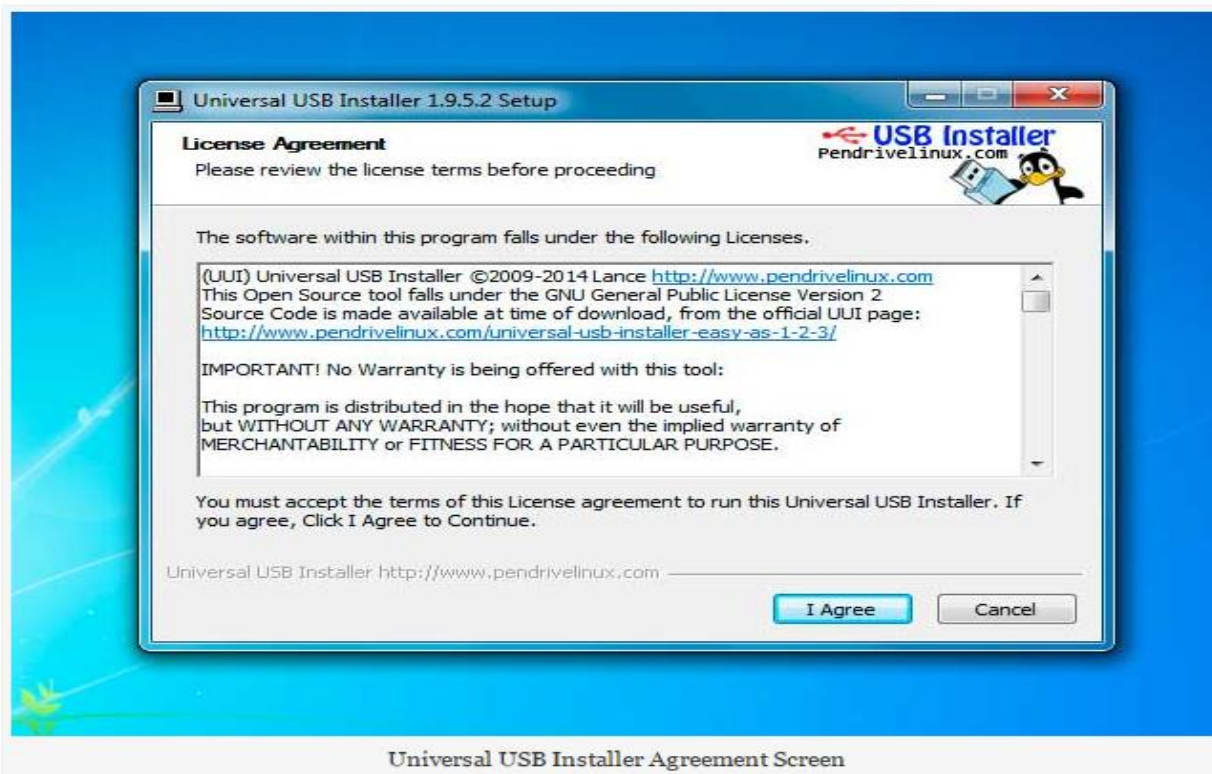
Appendix B: Ubuntu Installation Guide

Requirements before installation:

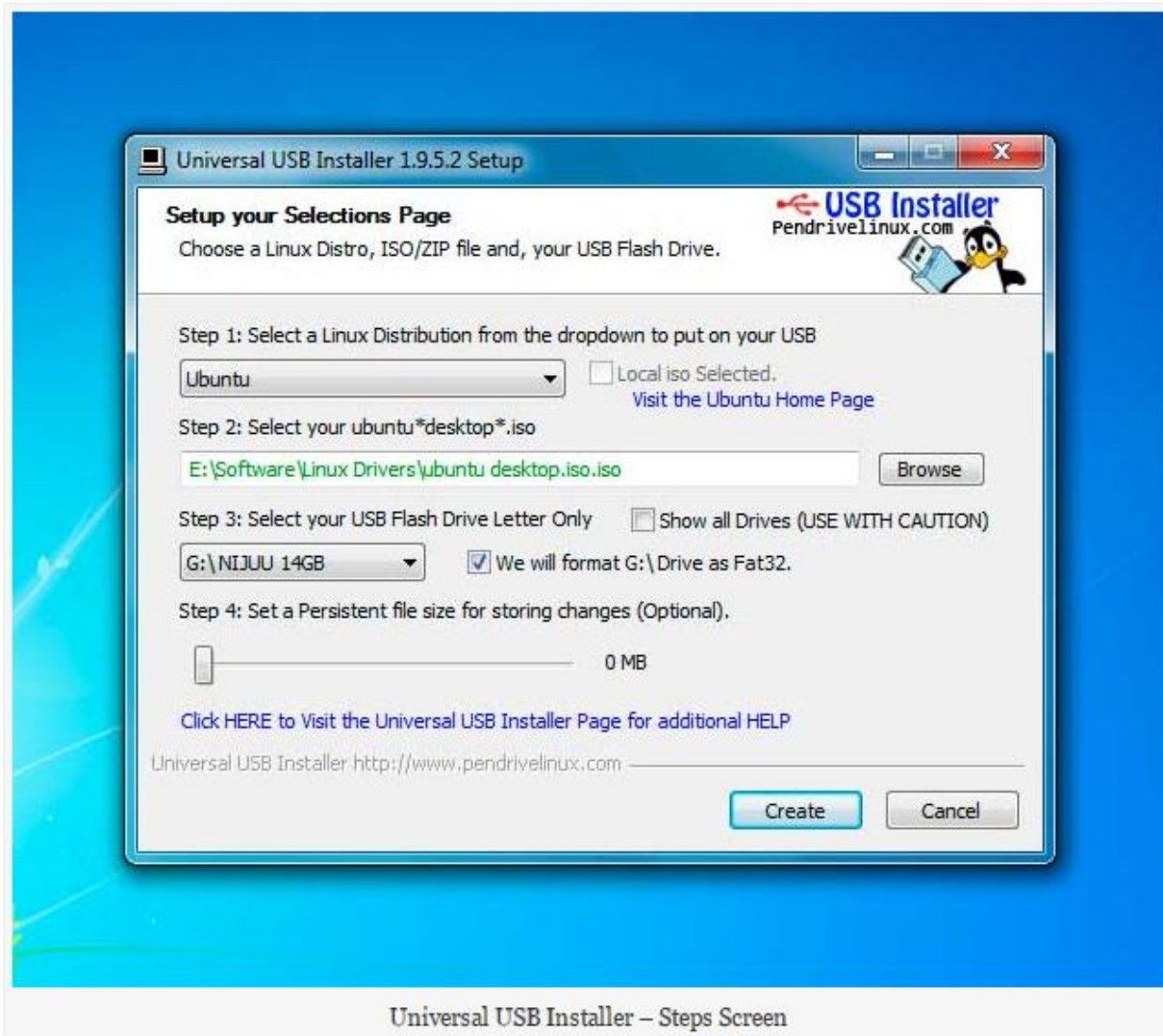
1. 4GB USB Pen drive or DVD (4.7GB).
2. .ISO file of Ubuntu 14.04 LTS – [Click here to download the ISO file](#).
3. Universal USB Installer – [Click here to download](#).

Installation Steps:

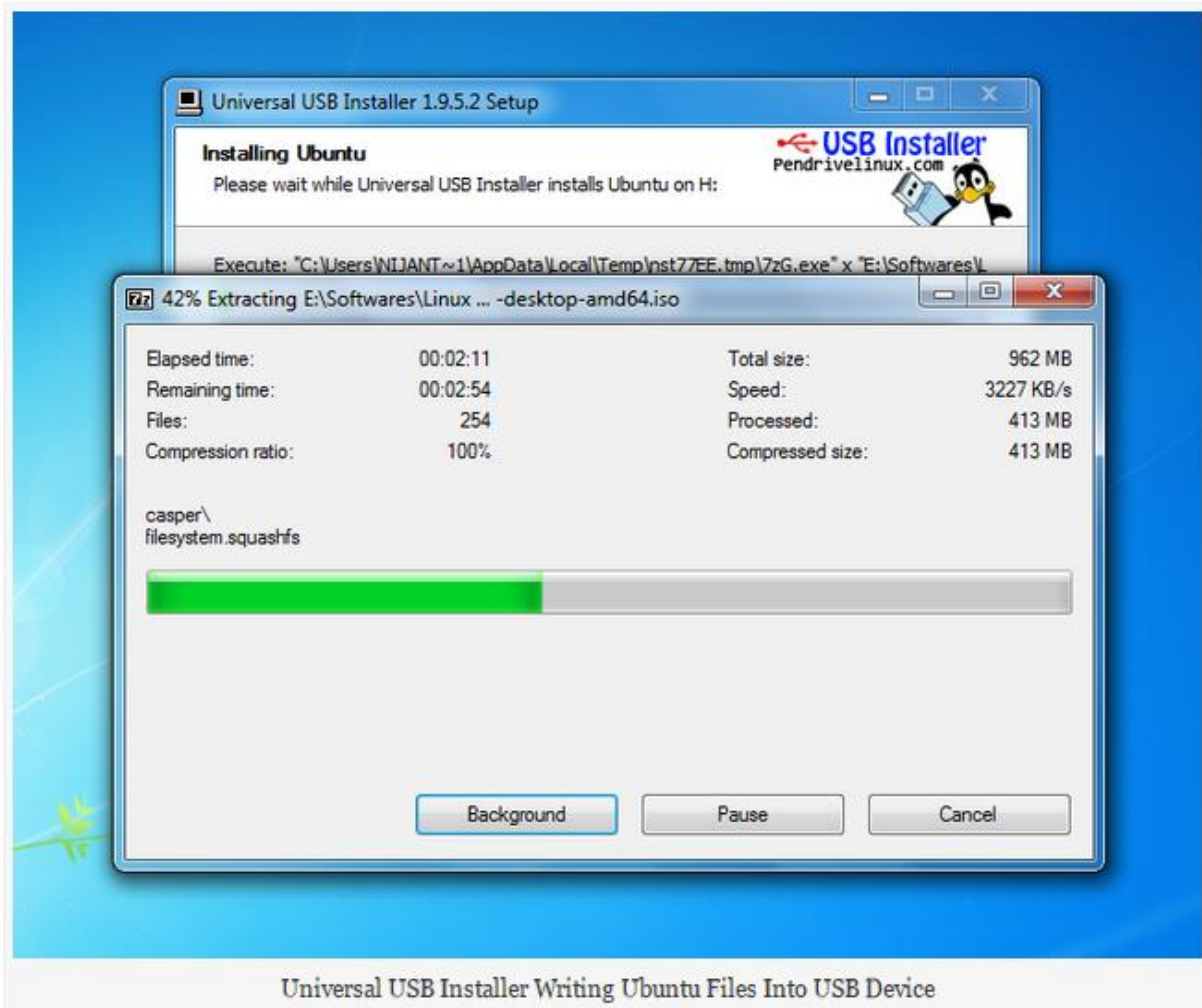
- Before installing Ubuntu 14.04 LTS, I would recommend you to have a backup of your important files and create a recovery media. So that, this might give you a hand if the installation fails.
- Prepare your USB Pen drive or DVD for installation. If you are using USB pen drive for installation, please move the files in that pen drive to another pen drive and format it.
- If you are installing Ubuntu using a CD or DVD then click here to read the instructions: [Ubuntu CD or DVD installation guide](#). I have discussed installing Ubuntu only through a USB device.
- After downloading the Universal USB Installer and .ISO file of Ubuntu 14.04 LTS, open the Universal USB Installer and accept the terms and conditions by clicking **I Agree** button.



- In the next screen you will be provided with some options. From the drop down list box provided select the Linux distribution as **Ubuntu** and locate the .ISO file of Ubuntu 14.04 LTS you have downloaded. Make sure that, the .ISO file is named as **ubuntu desktop.iso** or else the software won't recognize the .ISO file.
- Now in the next step, select the USB pen drive you need to burn the files into and check the **format** option too.



- Click **Create** button and wait for the application to perform its job. Once it is done, click **Close** and restart your computer.

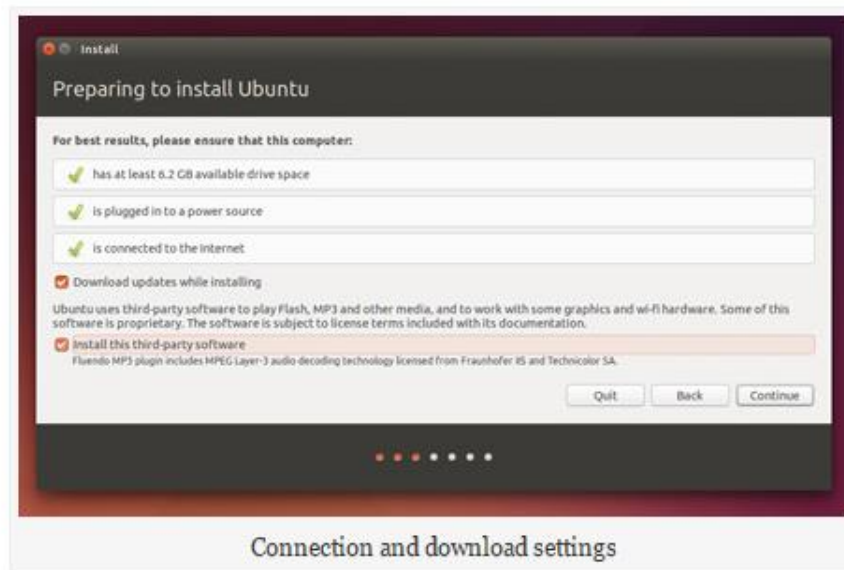


- Make sure that, you have set the *first boot device* to USB. And as soon as the Ubuntu installation screen loads quickly press any key to bring up the screen like the one in the screenshot below. (*I pressed **Up Arrow Key***).

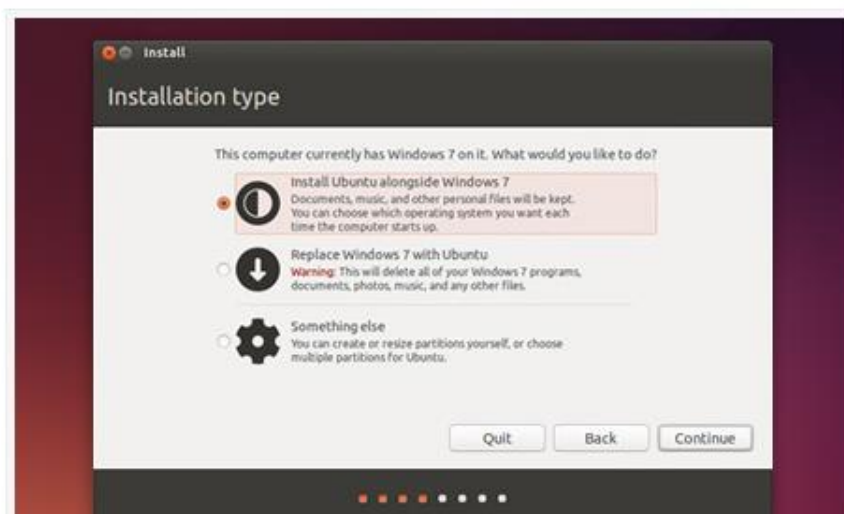


- Click **Install Ubuntu** and wait for the installation menu to appear. If you would like to install the updates during the installation process, make sure you have connected your computer to the internet and check **Download Updates While Installing**. If you have a good internet connection this is best as it saves your time after installation. Installing the *third-party software* is also optional but, this will download the codec plugins required for audio and video.

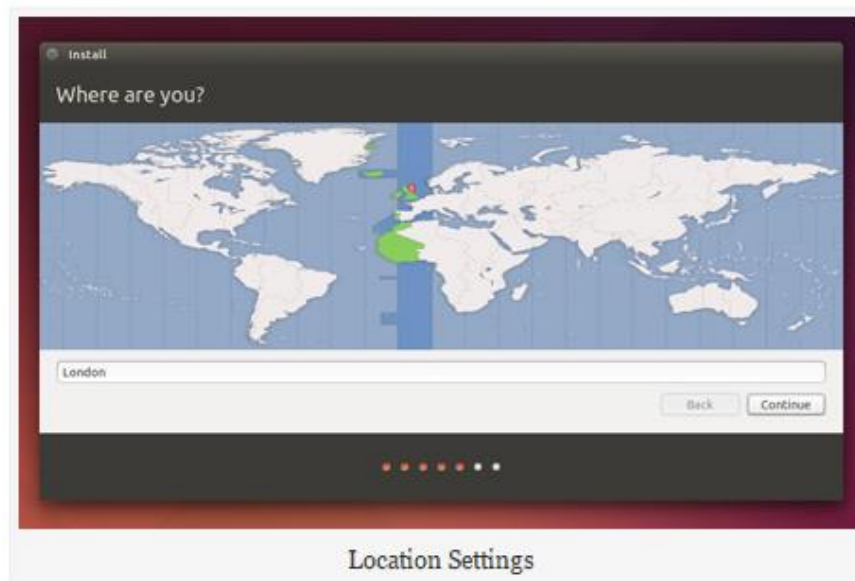
- Click **Install Ubuntu** and wait for the installation menu to appear. If you would like to install the updates during the installation process, make sure you have connected your computer to the internet and check **Download Updates While Installing**. If you have a good internet connection this is best as it saves your time after installation. Installing the *third-party software* is also optional but, this will download the codec plugins required for audio and video.



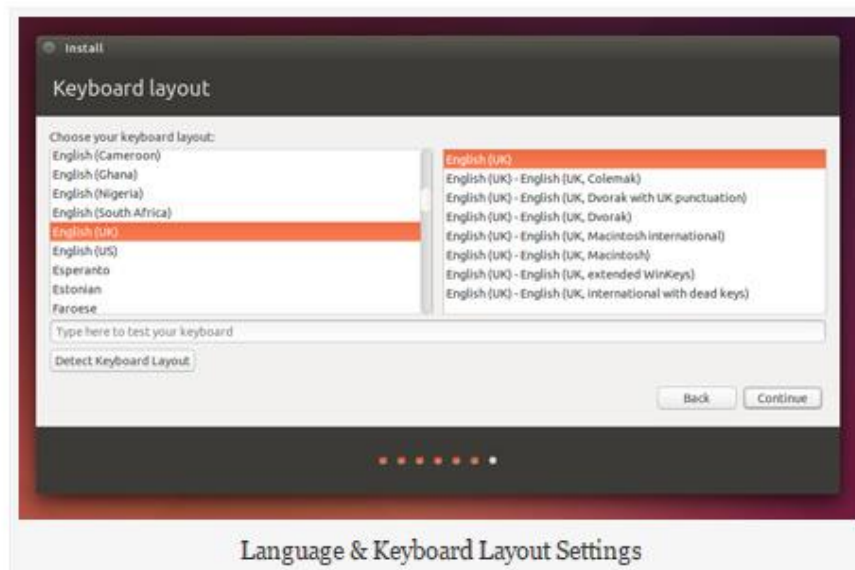
- In the next screen, you will be asked for the installation options. Check the **Installation Ubuntu alongside Windows 7** radio button and click **Continue** button and in the next screen if you are asked for the partition resizing options, decide the partition size. I would recommend you to provide at least 20GB of space to Ubuntu operating system.



- Choose your location for setting up the default time of your locale.



- Choose the language and the keyboard layout you are comfortable with.



- Enter your user name and the login credentials. For security reasons check the **Require my password to log in** radio button and click continue and wait till Ubuntu 14.04 LTS installation gets completed.



The screenshot shows the 'Who are you?' configuration screen in the Ubuntu 14.04 installer. It includes fields for 'Your name' (Lola Chang), 'Your computer's name' (lolachang-laptop), 'Pick a username' (lolachang), 'Choose a password' (masked with dots and marked 'Good password'), and 'Confirm your password' (also masked with dots). Below these fields are three radio button options: 'Log in automatically' (unselected), 'Require my password to log in' (selected), and 'Encrypt my home folder' (unselected). 'Back' and 'Continue' buttons are at the bottom right.

Who are you?

Your name: Lola Chang ✓

Your computer's name: lolachang-laptop ✓
The name it uses when it talks to other computers.

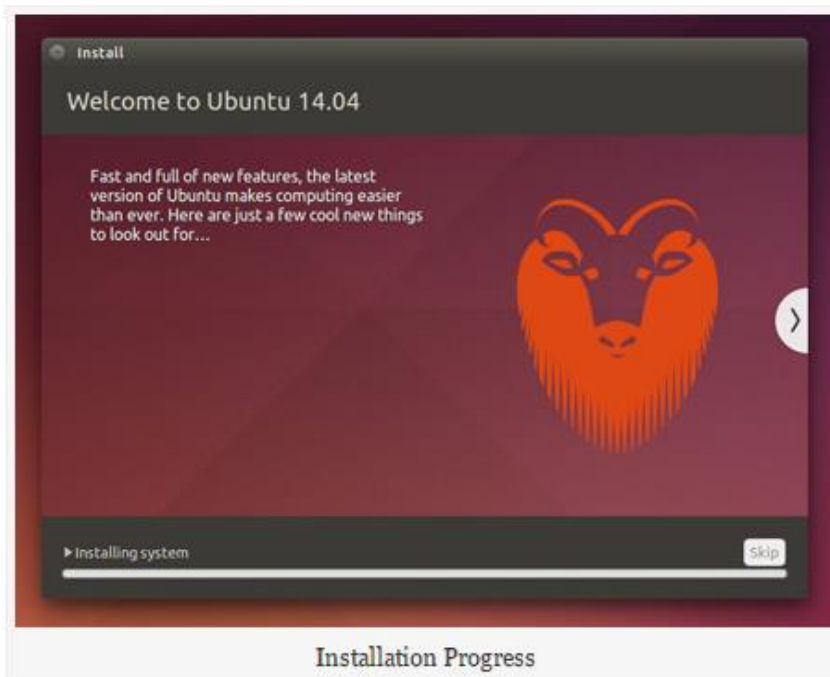
Pick a username: lolachang ✓

Choose a password: [masked] Good password

Confirm your password: [masked] ✓

☐ Log in automatically
☒ Require my password to log in
☐ Encrypt my home folder

Back Continue



- Once the installation is complete you will be prompted for restart. Click **Restart Now** to restart your computer.

