The University of Nottingham

Faculty of Engineering

Department of Electrical and Electronic Engineering

UNITED KINGDOM · CHINA · MALAYSIA

HDL Test Bench Generation

AUTHORS	:	Kenneth Tan Kai Xian [18024351]
SUPERVISOR	:	Dr Nandha Kumaar Thulasiraman
MODERATOR	:	Dr Vimal Rau A/L Aparow
DATE	:	29th of April 2021

Fourth year project thesis submitted in partial fulfilment of the requirements of the degree of **Master of Engineering**

ABSTRACT

Design verification is one of the most time-consuming and labor-intensive process in semi-conductor industry as verification process takes up to 60-70% of overall design effort [1, 2]. With every growing complexity of electronics designs, verification process become more time consuming so is the time needed to market the product. Furthermore, commercially available automatic testbench tools are either too costly like Testbencher from Synapticad [3] or not being available like StateCAD from Xilinx [4]. Hence, automatic testbench generator was developed with intention to reduce the amount time and effort to generate testbench by reducing the amount of hard coding of testbench. This paper presents a method of developing automatic testbench tool that is able to use the VHDL design file and user input parameters to generate testbench successfully. Furthermore, with addition of GUI, the tool is simple and user friendly which could potentially help people with little to no prior knowledge about VHDL to learn about VHDL. It can be observed in results and discussion section that the automatic testbench generator displayed effectiveness in generating testbench.

Keyword: HDL Test Bench Generator, verification, VHDL

Acknowledgements

I would like to express my gratitude to my supervisor, Dr Nandha Kumaar Thulasiraman, who guided me throughout this project.

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

1.1. Background

Hardware Description Language (HDLs) is widely used among the electronics industry due to its convenient feature of enabling formal description of an electronic circuit be represented as behavioral model or structural model. HDL are needed to allow codesign of hardware and software. Complex design requires more time and effort for development and debugging. With HDL, each module is separated allow different team to work on each module. More customization of the design like power, latency and functionality are allow through HDL.

Very high-speed integrated circuit hardware description language (VHDL) is one of the two most well-known and widely used HDL for modelling the behavior of a circuit accurately in hierarchy fashion through varying levels of abstraction [5]. **Figure 1** demonstrate a schematic diagram of half adder circuit with XOR and AND gate. **Figure 2** demonstrate a simple example of structural modeling of half-adder circuit in VHDL.

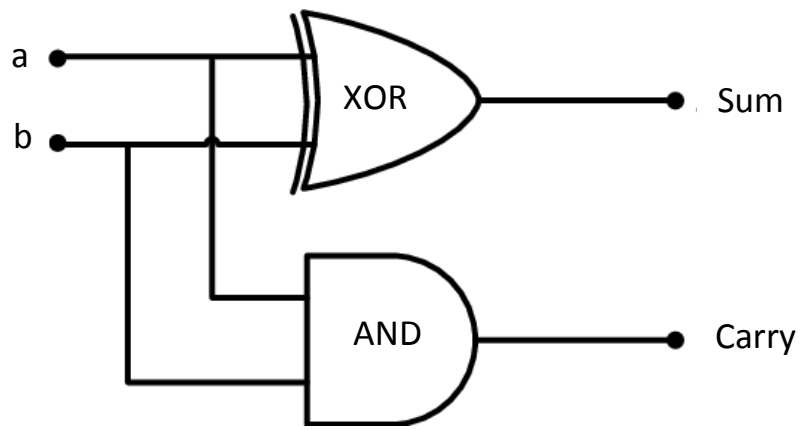


***Figure 1**: Schematic Diagram of Half Adder circuit*

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4
5   entity half_adder is
6       Port ( a : in  STD_LOGIC;
7              b : in  STD_LOGIC;
8              sum : out  STD_LOGIC;
9              carry : out  STD_LOGIC);
10  end half_adder;
11
12  architecture Behavioral of half_adder is
13
14  begin
15
16     sum <= a xor b;
17     carry <= a and b;
18
19  end Behavioral;
```

Entity: contain ports of the circuit

Input

Output

Architecture: Describes the functionality of the circuit

***Figure 2**: Half Adder circuit in VHDL*

The functionality of the circuit model is tested by developing test-bench for HDL. By generating input test vectors/stimulus to the unit under test (UUT), simulation results are generated and observed to verify the functionality of the design as shown in **Figure 3**.

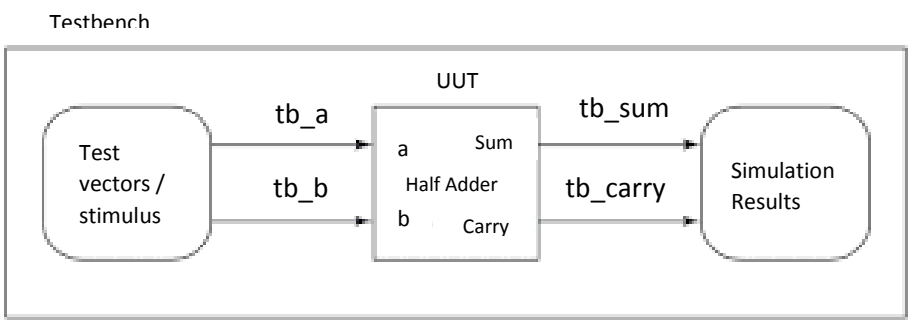


***Figure 3**: VHDL Testbench Architecture using Half Adder as example*

Using half adder as an example, test-bench generates input test vector/stimulus from input of half adder's truth table and verify the simulation results using output of half adder's truth table shown in **Table 1**.

Table 1: Truth table of half adder

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 4 demonstrate the test-bench for half adder and input of half adder's truth table is inputted in the red box. **Figure 5** demonstrate the simulation results for truth table and the simulation results are verify using half adder's truth table in **Table 1**.

```
entity half_adder_tb is
end half_adder_tb;

-- Begin module architecture/code.
architecture behave of half_adder_tb is

component half_adder
 port (
 a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           carry : out  STD_LOGIC);

end component;

-- Inputs
signal tb_a : std_logic;
signal tb_b : std_logic;

-- Outputs
signal tb_carry : std_logic;
signal tb_sum : std_logic;

-- *** Instantiate Constants ***
constant clk_PERIOD: time := 12 ns;
```

```
begin
-- Instantiate the UUT module.
uut : half_adder
port map (
   a  => tb_a,
   b  => tb_b,
   sum   => tb_sum,
   carry    => tb_carry);

-- Insert Processes and code here.
-- Stimulus process1
stim_proc1: process
begin
wait for 20 ns;
tb_a <= '0';
tb_b <= '0';
wait for 20 ns;
tb_a <= '0';
tb_b <= '1';
wait for 20 ns;
tb_a <= '1';
tb_b <= '0';
wait for 20 ns;
tb_a <= '1';
tb_b <= '1';
wait;
end process;

end behave; -- architecture
```

Input for A and B taken from the truth table

Figure 4: Testbench for half adder

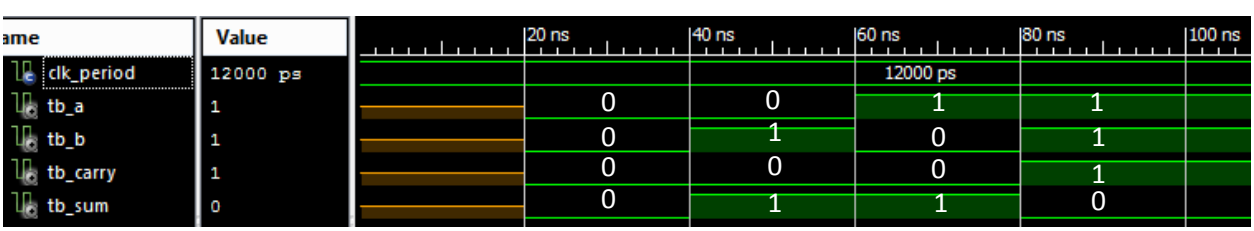


Figure 5: Simulation Result for half adder testbench

Several methodologies have been proposed regarding automatic HDL test bench generation that allow generation of testbench with each unique methodology. FPGA vendor such as Xilinx and simulator such as MATLAB and SynaptiCAD are example of commercially available tool in automatic HDL test bench generation. However, those tools are either not available for support currently, or being very costly to purchase, or being too complicated to use. The current state of the automatic HDL test bench generator will be explored further below in **Chapter 2.3** under related work.

1.2. Problem Statement

Generation of test bench using traditional gate level test generation methods are carried out by a human usually a validation engineer where test-bench is manually created for the given design. The test-bench generates input test stimuli to monitor the output of the design with given output reference. For very complex design, it requires large amount of test data to verify the functionality of the design. The validation and verification process could take up to 60% to 70% of overall design effort [1, 2] which is time consuming and labor-intensive. Furthermore, manually created testbench may not test all the operation of the circuit. Hence, automatic HDL testbench generator is extremely desirable as it assist the validation engineer of the time-consuming task of test-bench generation.

1.3. Project Aim and Objectives

The main aim of this final year project is to develop a tool that takes in the parameters for clock and I/O (Input/Output) of the UUT from users and generate the VHDL test bench design file with test coverage feature. The objectives of this project are listed as following:

Objective:

1. To develop a graphical user interface (GUI) to enable users to provide parameters for clock and I/O of the UUT.
2. Converting different graphical notations to the relevant HDL constructs.
3. Integrate all HDL constructs using Perl to develop an automatic HDL test bench generator.
4. Incorporate ModelSim Code Coverage to verify the progress of testing covered by the developed test bench and benchmark it against the commercial tool such as SynaptiCAD and VerTGen.
5. Validate the HDL test bench for different complex level of the HDL design.

1.4. Limitation of Project

Limitation of the project are as follow:

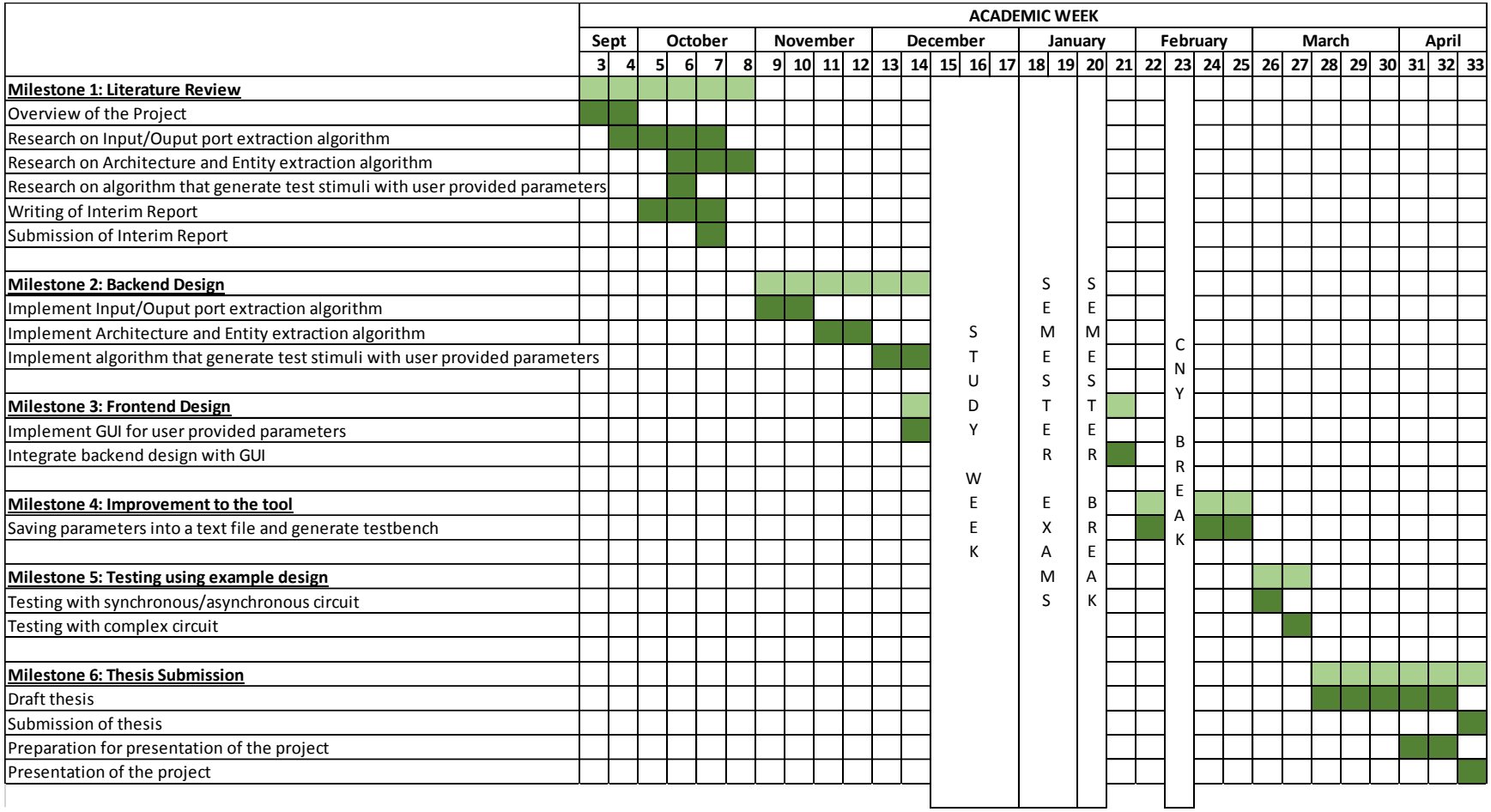
- The testing of the automatic testbench generation is only limited to the medium complexity of HDL design like serial communication protocol design. Using more complex design such as USB or microprocessor for testing is not applicable due to the limited time available and limited knowledge.

1.5. Deliverables

Deliverables:

1. An automatic HDL test bench generator for different medium complex level HDL designs using parameters for clock and I/O of the UUT inputted by the users.
2. Accuracy validation of developed HDL test bench.
3. Coverage tool to automatically produce the test coverages.
4. Limitations of developed test bench generator.

1.6. Project Timeline



1.7. Industrial Relevance

Design validation is a crucial process in designing electronic industry as it verifies the functionality of the design on its correctness and avoid any bug leaking before the circuit design enter fabrication stage. If bug leaking enters circuit design during fabrication stage, the circuit design would have to be redesigned which have tremendous impact on cost and delay in getting circuit design out on the market. The cost of fabricating the circuit design would drastically increase while the profit of the company is reduced [6]. Hence, validating the design is a crucial process in electronic industry.

As stated by Moore's law, the number of transistors in an integrated circuit would double approximately every 2 years [7]. Reducing transistor size contributes more complexity in electronic designs. With more complexity in electronic design, the time and effort for validating design increases drastically. Thus, HDL test bench generator has high relevance in the electronics industry especially for beginner/medium level HDL design engineers. As mentioned in **Chapter 1.2** under problem statement, designing test bench requires more time than designing the actual design itself [6]. With HDL test bench generator, it allows verification engineering to reduce the design validation time significantly.

HDL test bench generator could be used in universities where it provides assistant for students when learning HDL language. GUI are implemented in HDL test bench generator where drawn waveforms are used to generate test bench stimulus. With this tool, users can generate a test bench in a few minutes which could take several hours to test and code by hand for beginners. Students could refer to test bench code generated by tool as guideline when learning to code for HDL language. The tool allows easier teaching of HDL language with interactive simulation code where input signal can be changed using the GUI each time and new test bench and simulation results are updated on the spot.

1.8. Thesis Outline

The report has been divided in five chapters.

Chapter I Introduction: This chapter mainly deals with the background, objectives, and industrial relevance.

Chapter II Literature Review: This chapter is dedicated to illustrating the relevant literatures and the recent works related to the study.

Chapter III Methodology: This chapter explain the overall flow of the project work using a flow diagram and explain the tools planned to use.

Chapter IV Result and Discussion This chapter explain the complete design of the testbench code and scripting methods, results obtained etc.

Chapter V Conclusion and Future Improvements: The conclusion of study and future improvements are given in this chapter.

The code, photographs etc are given in appendix A to C

Chapter 2 - Literature Review

2.1 Overview

Functional verification is crucial process in validating that the electronic hardware is working as intended. As electronic hardware grew in complexity over the years following Moore's Law, functional test complexity grew exponentially. Functional verification is a major bottleneck in design stage as it takes up to 60-70% of development time [1] and 80% of HDL code [2] are made up of test-bench. Hence, automatic testbench generation is desirable as it save time and resources. In this chapter, functional verification background and related work is research where bottlenecks, functional verification challenges, current verification technology and current state of automatic test-bench are explored.

2.2 Background Research on functional verification

2.2.1 Bottlenecks

A. Design Bottleneck

The time to design electronic hardware scale with the complexity of the design. As complexity of design increases, so is the time needed to market the product as demonstrate in **Figure 6**. In 2007, 100 billion simulation vectors and 25 million lines of register-transfer level (RTL) code are written by 2000 engineers [8].

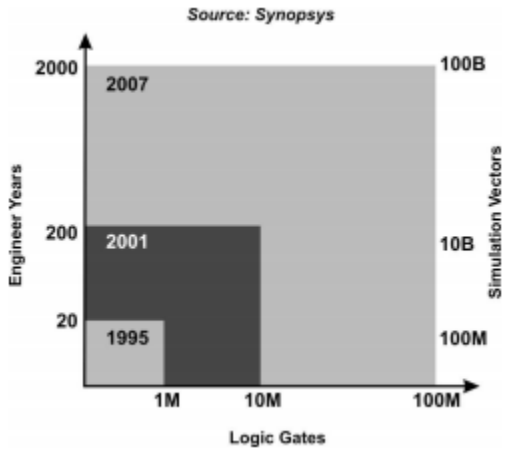


Figure 6: Demonstrate the complexity of system on chip (SOC) as logic gates exponentially increases over the years [8]

As number of transistors in design grew exponentially, there is only linear increase in programming time and not sufficient engineers to accommodate the required design time. To

remedy this problem, electronic design automation (EDA) industry introduces behavioral modelling on HDL such as Verilog and VHDL with highest level of design abstraction. Highest level of abstraction means that a module can be designed to desired specification without concern for hardware implementation details. SystemC and SystemVerilog are widely used example of EDA tools in current major EDA world. Hence, to some extent, design bottleneck has been remedied with the help of efficiency of the EDA tools.

B. Functional Verification Bottleneck on design

With the help of EDA tools, more complicated designs are built with ease. As design complexity rise, time and effort needed for functional verification almost doubled. Shown in **Figure 7**, design complexity growth is higher than design productivity growth and verification productivity growth. This is caused by verification time not design time as 60-70% [1, 2] of overall design effort came from functional verification. Hence, functional verification bottlenecks are caused.

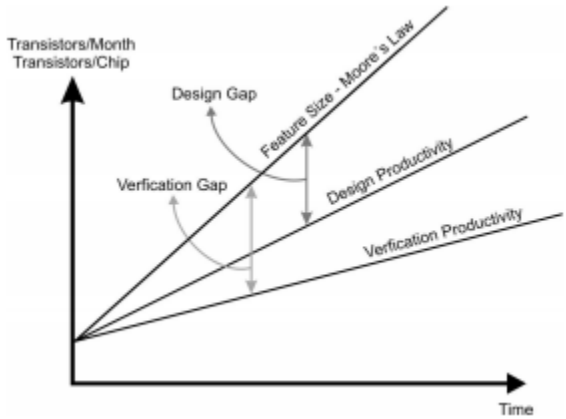


***Figure 7**: Design and Verification Gaps [9]*

The higher the level of design abstraction, the less detail of the design is demonstrated which may cause information loss and misinterpretation when transforming and mapping design to the end product. Hence, additional verification steps are needed to ensure the transformation is correct. For example, synthesis is a process that convert HDL design models of hardware to gate level. Functional verification is needed to ensure the transformation from HDL design models of hardware to gate level is correct and information is not loss.

As demand for electronic devices grows, functional complexity increases due to the diverse design of electronics we had nowadays. To achieve high system reliability, effort and time are put into verification to ensure the chip function according to the requirement in a system environment. The EDA industry reduce the verification bottlenecks by introducing the concept of abstraction similar to design bottleneck. High-level language constructs such as threading (fork join), tasks and control structures are embedded into Verilog and VHDL. This reduce verification

time by allowing more control to test all the functionality of the design. Even with the help of these constructs, functional verification process still requires plenty of effort and time.

2.2.2 Challenges in Functional Verification

There are four major challenges in functional verification:

A. Large Scale and complexity of the design
B. Detecting incorrect behavior
C. Insufficient golden reference model and comprehensive functional coverage metric

A. Large Scale and complexity of the design

Exhaustively testing is needed for complex and large-scale chip design to test that the chip is functionally correct. The verification engineers need to verify the possible state by applying a set of inputs and compare the outputs with the expected outcome. With complex design, it takes plenty of time and effort to verify all the possible states. Hence, instead of verifying the whole chip, sections of the design are verified separately which reduces verification time. Once each sections of the design are verified, each section of the design are stitches back together.

B. Detecting incorrect behavior

Verification engineers need to identify fault in the design and detect whether the design is performed as expected based on current state and inputs. As verifying all the possible state in a complex system takes up time and effort, the logic of the design is validated at a higher level of abstraction where inputs are organized into data sets and valid command. The behavior of the design is tested by the verification engineers based on input test vectors. However, the higher the level of abstraction used in validation; the lesser detail functional tests are performed on the design. Hence, the number of logic bugs have exponential growth of 300-400% [10] from each generation of products as shown in **Figure 8**. As design get more complicated, functional validation will be become exponentially difficult which will greatly impact the time needed for products to enter the market with extra risk of products with undetected bugs being shipped to consumers' hand.

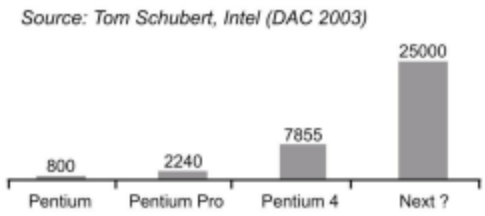


***Figure 8**: Pre-silicon bugs per generation found in the Intel IA32 family of microarchitectures [10]*

C. Insufficient golden reference model and comprehensive functional coverage metric

The cause of logical bugs is due to insufficient golden reference models and insufficient functional coverage metric. Due to having multiple specification models (verification model, timing model etc.), consistency across these models is a major concern. Hence, careless coding and miscommunication is ranked first and second respectively on the list of bug causes shown in **Figure 9** due to insufficient golden reference.

Verification problem is further enhanced by lack of functional coverage metric for chip design. There are two types of coverage metrics:

- Code Coverage - A metric that measure how many lines of code in the implementation of your design have been executed. Code coverage is automatically extracted from the design performed by the simulator tool (etc. VHDL and Verilog). Code coverage process is relatively easy to perform as the process is automatically done by the simulator tool.
- Functional Coverage - A metric that measure verification tests accuracy on the design. This process require time and effort from the verification engineers as lines of code are written to test whether the testbench covered the required functionality of the design.

Increasing the functional coverage should in theory increase the confidence in the design's functionality. However, design error is restrictive and complicated. High coverage numbers do not necessarily mean high quality testing. Coverage numbers are only important for verification engineers to know whether the amount of verification tests is enough. Design error could potentially still occur even if coverage numbers are high. Hence, lack of comprehensive functional coverage metric is considered one of the challenges in functional verification.

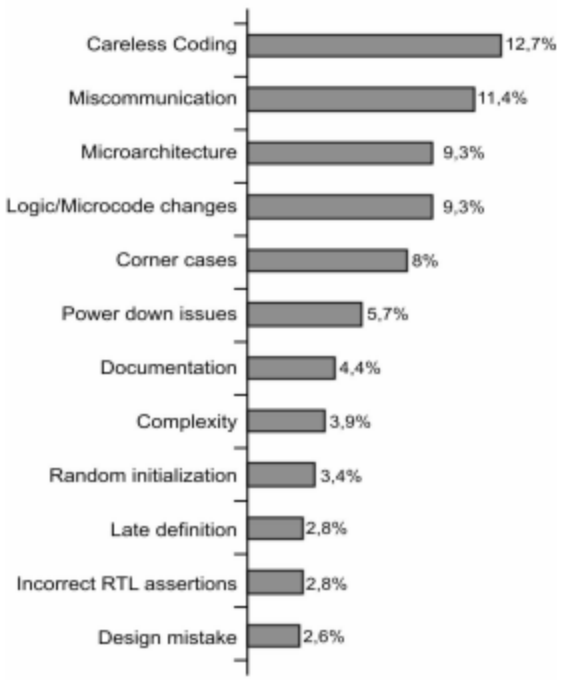


***Figure 9**: Causes of logical bugs. Statistical study of 7855 bugs found in Pentium 4 processor design pre-production. [10]*

Complexity is ranked 8th among the bug causes list shown in **Figure 9**. However, a lot of the top categories contributes to the complexity of the design. As complexity of design increases, more documentation and team members are needed, increasing chances of miscommunication between team members, and increasing the likelihood of corner cases.

2.2.3 Current Verification Technologies

When design gets more complex, verification time and effort increases. To reduce the verification time, random dynamic simulation is introduced by providing random test vectors to the design. As random testing for large and complex design has nearly infinite variable and outcome, EDA industry introduced higher level of abstraction for verification language such as SystemC Verification Library (SVL) [11]. New features such as random stimulus distribution, reactive testbenches and constrained-random stimulus are introduced to verification language. Hence, verification time is reduced as productivity is increased due to the new verification tools.

As higher level of abstraction is used, verification tools are needed to ensure the transformation is correct. Hence, “Equivalence check” is introduced to keep in check that the gate level representation is the same as the HDL implementation. “Equivalence check” generate data structure and output value of HDL implementation is compared to gate level representation. If the patterns are the same, the design pass the “Equivalence check” test.

2.3 Related Work

FPGA vendor such as Xilinx and simulator such as and SynaptiCAD are example of commercially available tool in automatic HDL test bench generation. Journal and conference papers of automatic testbench generation is also written. Below section demonstrate these proposed automatic HDL test bench generation tools in detail.

2.3.1 BugHunter Pro and VeriLogger Extreme simulator from SynaptiCAD

BugHunter Pro and VeriLogger Extreme simulator from SynaptiCAD [3] support automatic graphical test bench generation. SynaptiCAD offer 3 levels of test bench generation to meet the type and complexity of the design:

A. WaveFormer - producing stimulus-based test benches
B. VeriLogger - fast unit-level testing
C. TestBencher Pro – producing complex bug-functional models to represent complex, reactive interfaces

A. WaveFormer

For small test benches, the input required to generate test bench stimulus is using drawn waveform created by the users with the tool assist in extracting top-level module ports and transferring it to timing diagram of the simulator. The benefits of WaveFormer are it allows much faster and accurate generation of small test benches compare to manually coding of test bench by hand. Drawn waveform are easier to edit on compare to raw VHDL or Verilog code.

For large test benches, outside source such as simulator, spreadsheet or logic analyzer can import waveform data to WaveFormer. Waveform data are used as input stimulus which are used to generated timing diagram.

User can edit on the timing diagram to desired specification and test benches are created after timing diagram is done. This test bench model can then be instantiated in a user's project and compiled and simulated with the rest of the design. An example of a timing diagram is shown in **Figure 10**.

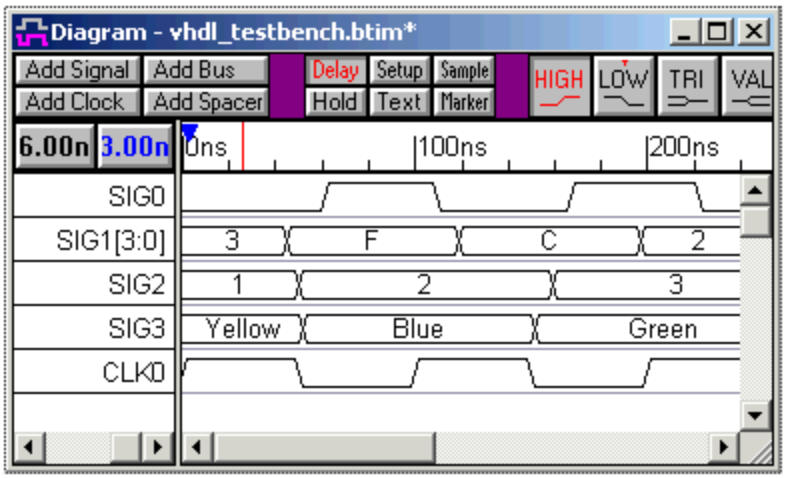


***Figure 10**: Waveform Editor and timing diagram on WaveFormer from SynaptiCAD [3]*

B. VeriLogger

Unique features of VeriLogger is that test bench generation features are integrated closely with the HDL simulator such as VHDL and Verilog which allow fast unit-level testing The fast testing speed of VeriLogger allow true bottom-up testing of every model in your design. This step is traditionally been very time consuming and is often skipped in verification process.

VeriLogger extract the ports from the design and automatically add them to the Diagram window. Input signals waveforms can be generated by equations, graphically drawn, or copied from existing signals. Once input signals are generated, VeriLogger will create automatically input stimulus for the testbench to drive the design.

Another unique features of VeriLogger is interactive simulation mode where new test bench is automatically generated and simulated every time an input signal is changed. This allow quick and simple tweaking of the design before the design is complete. It also allows to test ideas very quickly without putting time and effort to generate test benches.

In **Figure 11**, 4-bit Adder Verilog file is used as an example. VeriLogger extract ports from the design into the timing diagram window. User draw waveforms on the black input signals and test bench is automatically generated from the drawn input waveforms. Outputs of the simulation will be displayed in the same diagram as the input stimulus. In the interactive simulation mode, whenever the user changes the input vector, the design and test bench are simulated again allowing user to test a small change in the timing of an input signal with relative ease. Timing diagram file can be saved allowing the user to edit the test bench later.

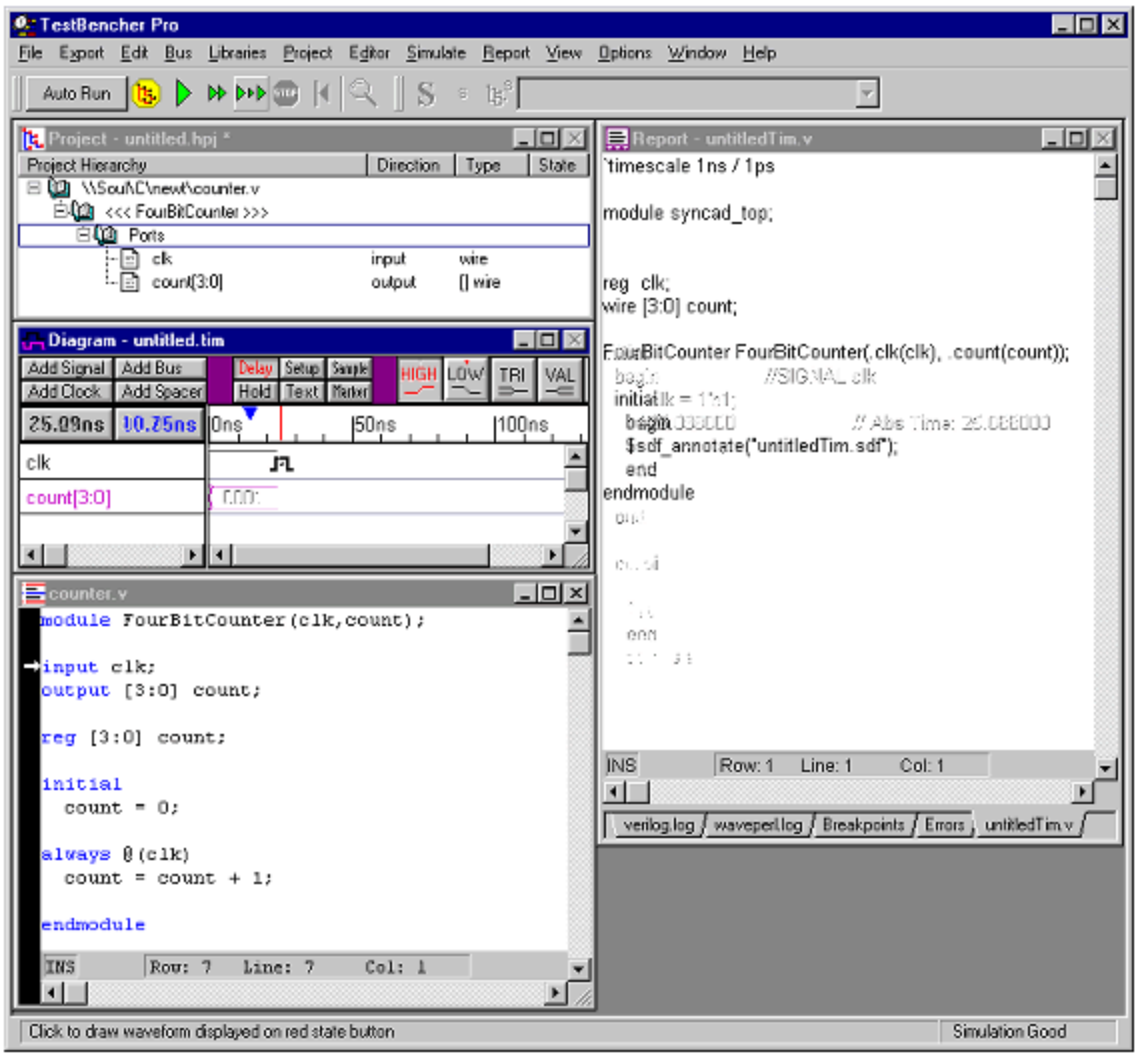


***Figure 11**: VeriLogger with add4.v as example [3]*

C. TestBencher Pro

The highest level of testbench generation is provided by TestBencher Pro, which allows a user to design bus functional models using multiple timing diagrams to define transactors and a sequencer process to apply the diagram transactions. TestBencher Pro is used to generate test benches for complex design like microprocessor or bus interface. Functionality of TestBencher Pro are listed below:

1. <u>Graphical Representation of Transactions</u> – uses timing diagram to represent timing transactions of the test bench which free verification engineers the tedious details of the underlying code

2. Automatic Tracking of Signal and Port Code – The signals and ports of the design is automatically maintained and track between the test benches and unit under test. When a change in signals and ports occur in the design, it will automatically be updated in the test benches. This prevent tedious work of rewriting signals and ports back and forth between design and test bench and avoid errors when ports and signals don't align.

3. Conceptual Modeling Constructs – 5 basic constructs are used to create a transaction for TestBencher Pro:

   i. Drawn Waveforms - describes stimulus and expected response
   ii. State Variables - parameterize state values
   iii. Delays - parameterize time delays between edge transitions
   iv. Samples - verify and react to output from model under test
   v. Markers - models looping constructs or to insert native HDL subroutine calls

   The functionality of these 5 basic constructs are easier to learn compare traditional manual written test benches.

4. Reactive Test Bench Option - Uses a single timing diagram to create testbench rather than the multi-diagram bus-functional models. Report about the performance of simulation is generated using Reactive test benches.

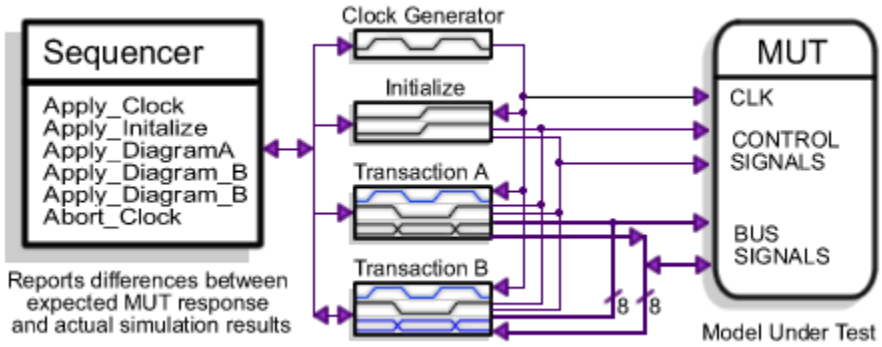


***Figure 12**: TestBencher Pro's features of generating a bus functional model from multiple diagrams [3]*

SynaptiCAD is great tool for automatic test bench creation for different type and complexity of design. However, it has disadvantage of costing a minimum of $2,500 for the VeriLogger Extreme simulator excluding maintenance fee. The high cost might disincentivize electronic design company and universities in investing SynaptiCAD simulator.

2.3.2 StateCAD by Xilinx

StateCAD [4] tool by Xilinx also does automatic test bench generation using state diagram of the design model as input stimulus. However, StateCAD has been deprecated in ISE 11.1 and will no longer be supporting bug fixes or enhancements to the tool [12].

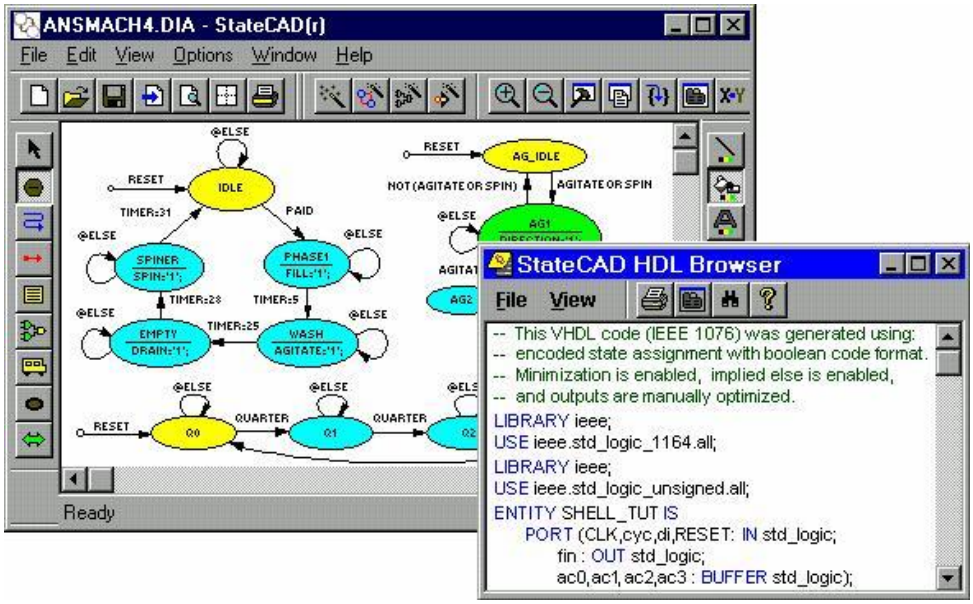


***Figure 13**: StateCAD software*

2.3.3 VerTGen

VerTGen [13] is an open source tool for automatic test bench generation created from National University of Sciences and Technology Islamabad, Pakistan. The tool uses random vector generation based on certain probability distribution such as binomial or exponential to generate test stimuli for the test bench. This allow faster validation of the circuit design and remove biased of the user on the test stimulus. However, it reduces the accuracy and coverage of the test bench. VerTGen only allows user to generate test bench of simple electronic design (e.x shift register). The overall flow of how VerTGen work is shown in **Figure 14**.

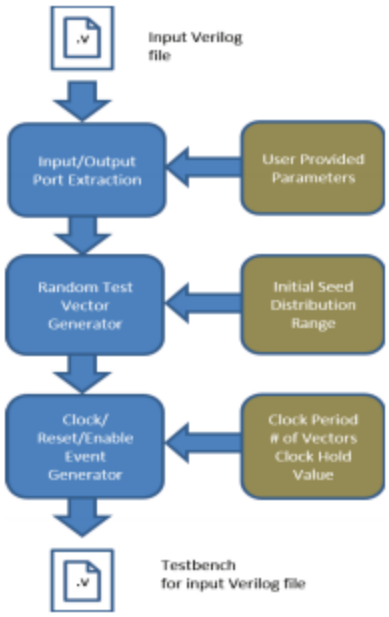


***Figure 14**: Block diagram of How VerTGen work. [13]*

The block diagram in **Figure 14** is composed of 3 main components:

A. File Parsing and input parameters

Verilog design file is inputted to VerTGen and ports are automatically extracted from design. Users are requested by VerTGen's Gui to input parameters:

1. Type of Input – Input can be either reset, clock, default, and enable.
2. Constraints – Maximum and Minimum value tested
3. Distribution of Input – Type of probability distribution included are Random, Erlang, Exponential, Binomial, Poisson, and Uniform. Seed value of randomization can be adjusted.
4. No. of vector generation – This parameter specifies number of clocks that an input signal needs to stay constate so that the effect of the test stimuli can be propagated to the output for proper verification of the sequential circuit.

B. Test Vector, Clock and Event Generation

For default type inputs, input stimulus generation can be randomized with Verilog language built in tools with different types of probability distribution. Randomization of input stimulus assist in automation and generalization of functional testing. Seed value is acquired from CPU clock so that true random input vectors are generated. If CPU's seed value is not used, random numbers

will only be generated from the initial seed value and thus, generating same stream of random input vectors every run. The user also has the option to input their own seed.

For clock inputs, clock cycle of 50% duty cycle are generate based on users input of clock period and numbers of clock cycle. For reset and enable input, events are used to trigger the sequence of their assertion and de-assertion.

C. GUI of VerTGen

Shown in **Figure 15**, C++ based GUI of VerTGen are demonstrated where user input Verilog design file and input parameters.

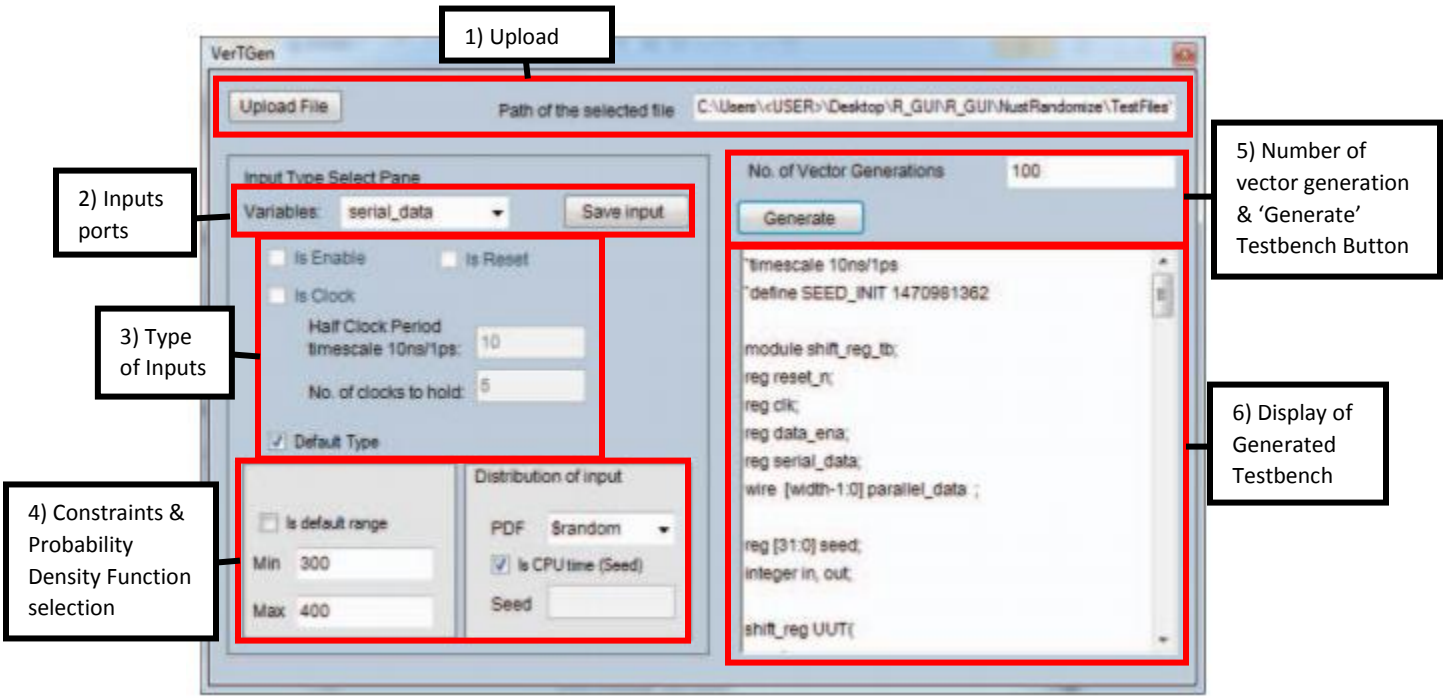


***Figure 15**: VerTGen Gui [13]*

1. <u>Upload file</u> – When 'upload file' button is pressed, directory of the system is shown, and Verilog design file can be selected. If testbench file is selected, error message will be shown.
2. <u>Input ports</u> – Variables menu list show the extracted input ports where user can select the input ports correspond to the parameters the user want to specify. When the parameters are filled, 'save input' button can be pressed to save the parameters correspond to the input port.
3. <u>Type of Inputs</u> – Enable, Reset, Clock and Default input type can be selected. If clock input type is selected, half clock period and number of clock cycle parameters are requested. If default input type is selected, constraints and type of probability density function parameters are requested.

4. Constraints & Probability Density Function selection - Maximum and Minimum value are inputted for testing. User can select type of probability distribution which include Random, Erlang, Exponential, Binomial, Poisson, and Uniform. Seed value can be specified by the user or chosen randomly using CPU's seed value.
5. Number of vector generation & 'Generate' test bench button – When generate 'button' is pressed, all the input ports, its parameters and number of vector generation are checked if they are specified. If the checks are successful, test bench will be generated.
6. Display of Generated Testbench – Generated test bench will be displayed on this window. Test bench file is simultaneously generated in the same directory as the design file.

2.3.4 Modeler's Assistant

The paper from [5] demonstrate automatic testbench using Modeler's Assistant [14]. Modeler's assistant generates Process Model Graph (PMG) which is a graphical representation of division of functionality within a VHDL design. Shown in **Figure 16**, large circle in PMG represent process of VHDL design while ports are presented by smaller circle placed on the circumference of larger circle. Black small circle represents a sensitive port. Sensitive ports are ports or signals in the sensitivity list of the given process. The VHDL source for the VHDL design is generated through Modeler's assistant.

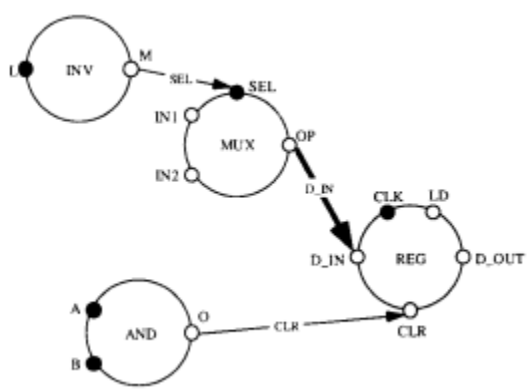


***Figure 16**: Process Model Graph (PMG) generated by Modeler's Assistant for an example circuit [5]*

Test stimulus are created using Process Test Generator (PTG) and VHDL source from Modeler's assistant and Control Flow Graph (CFG) is constructed which is a graphical representation of the flow of information for VHDL model. The test stimulus from PTG are used by Hierarchical Behavioral Test Generator (HBTG) which extracts the graphical interconnection and design

information from PMG database and functionality data from each process. Test sequence are generated hierarchically for the whole entity and it is converted to VHDL test bench by Test Bench Generator (TBG) program. This whole process is summarized in the block diagram shown in **Figure 17**.

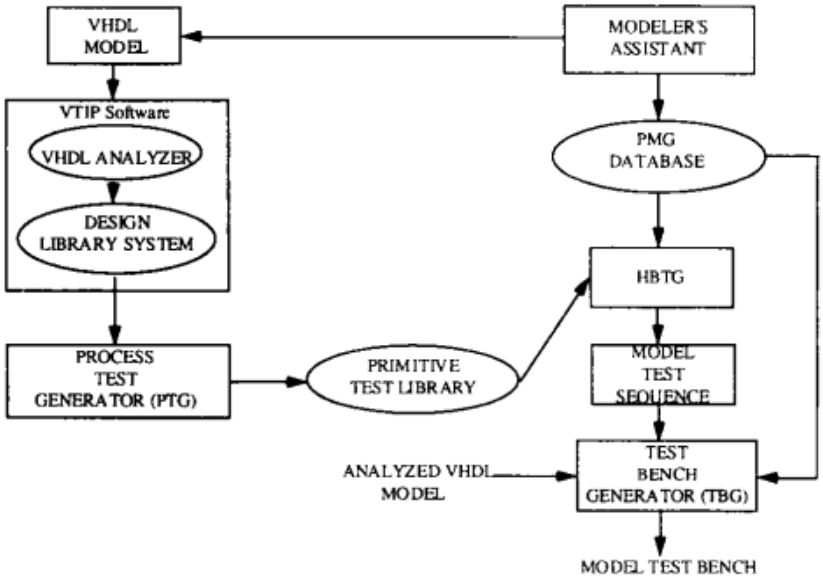


***Figure 17**: System block diagram [5]*

The paper [1] provides detail explanation on Process Test Generation algorithm, Hierarchical Behavioral Test Generation algorithm and Test Bench Generation algorithm. The paper also presents the method to evaluate the test stimulus generated by the testbench. The simulation result in the paper demonstrates excellent coverage using a complex VHDL code as an example. The simulation result in the paper demonstrates excellent coverage using a complex VHDL code as an example. This paper offers a very sophisticated automatic test bench generation system that uses hierarchy information of VHDL design to generate test bench. This assist the verification engineer task in creating test stimulus to test the functionality of design. However, the downside of this automatic testbench generation system is that Modeler's Assistant is not an open source software and it cost $30 to view the paper [14] regarding Modeler's Assistant.

2.3.5 Summary of current automatic testbench generator

BugHunter Pro and VeriLogger Extreme simulator [3] from SynaptiCAD use drawn waveform created by the users to generate input stimulus and transferring it to timing diagram of the simulator. However, the tool is very costly with minimum cost of $2,500. StateCAD [4] tool by Xilinx also does automatic test bench generation using state diagram of the design model as input

stimulus. However, StateCAD has been deprecated in ISE 11.1 and will no longer be supporting bug fixes or enhancements to the tool [12]. StateCAD tool by Xilinx also does automatic test bench generation using state diagram of the design model as input stimulus. The tool uses random vector generation to generate test stimuli for the test bench. This allow faster validation of the circuit design. However, it reduces the accuracy and coverage of the test bench.

The paper [5] offer a very sophisticated automatic test bench generation system that uses hierarchy information of VHDL design to generate test bench. The simulation result in the paper demonstrates excellent coverage using a complex VHDL code as an example. However, Modeler's Assistant is not an open source software and it cost $30 to view the paper [14] regarding Modeler's Assistant. The proposed automatic HDL test bench generation lack either ease of use, availability or being open source.

Chapter 3 – Methodology

3.1 System Design Overview

Perl (Practical Extraction and Reporting Language) is used over other object-oriented programming language such as Python is due to vast libraries CPAN module for Perl contain. As Perl is an older programming language compare to Python, backbone of signal extraction algorithm has been established which can be found here [15]. Furthermore, Perl is much faster at string manipulation and text processing due to the very powerful regular expression engine. Combined with Perl ability to accommodate wide range of tasks such as GUI development, Perl program is a perfect candidate for automatic testbench generator.

Figure 18 demonstrate the overall block diagram of automatic VHDL testbench generator. The Perl program will extract the input and output ports in the VHDL design and automatically add them to the Graphical User Interface (GUI). GUI will ask user for parameters of clock and I/O (Input/Output) ports of the UUT. After parameters for each of input ports are saved, testbench will be created based on the extracted inputs/outputs and user provided parameters. ModelSim, commercially available HDL simulator, is used to generate the simulation result and perform code coverage. Code coverage allow engineers to evaluate the performance and quality of the tests. The automatic VHDL testbench generator will then be tested using different level of complexity of VHDL code.

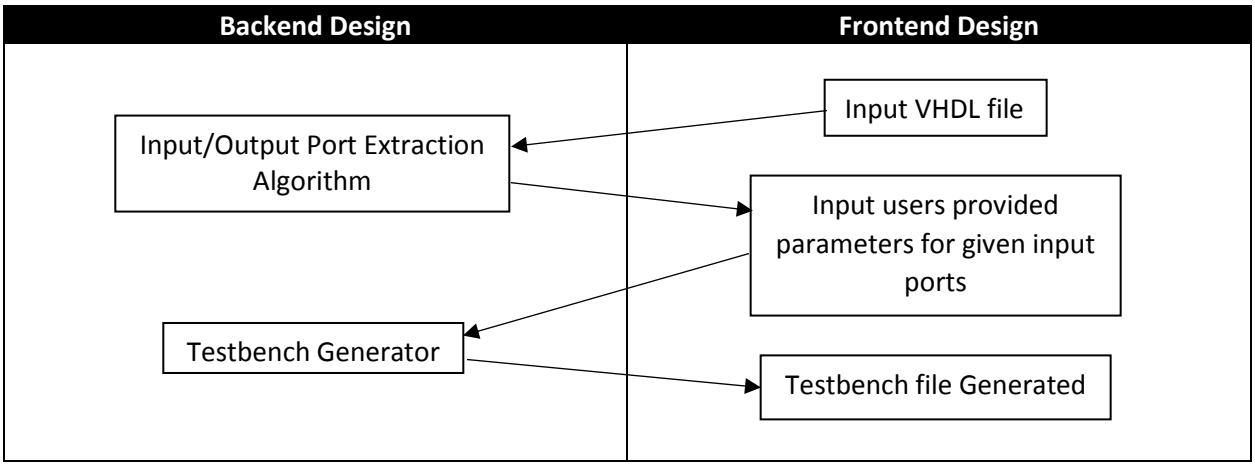


***Figure 18**: Block diagram of automatic VHDL testbench generator*

3.2 Backend Design

3.2.1 Signal Extraction Algorithm

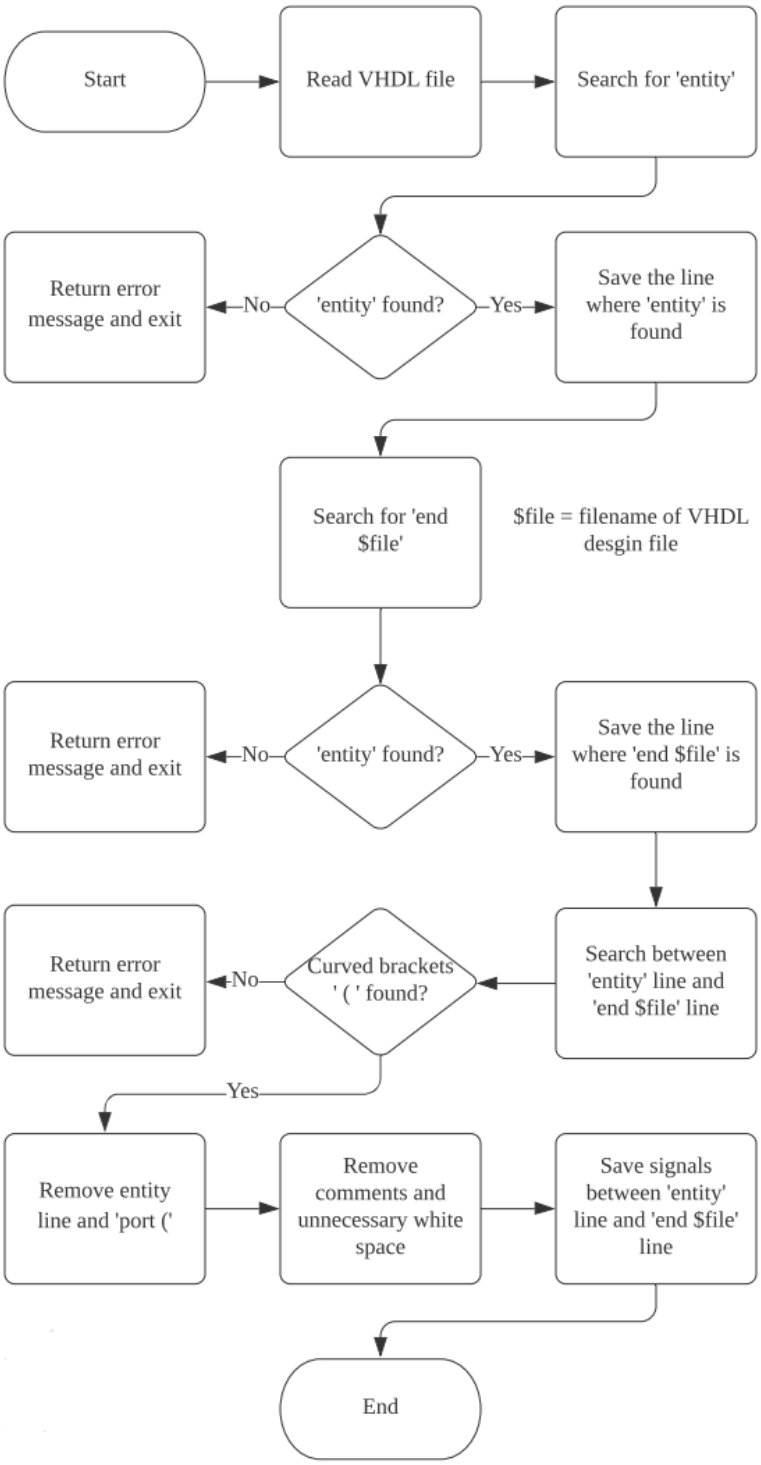


***Figure 19**: Signal extraction algorithm flow diagram*

Figure 19 demonstrate the overall flow chart of signal extraction algorithm. Signal extraction algorithm would extract input and output ports from the VHDL design file. Firstly, the VHDL design file is read. ‘entity’ and ‘end $file’ keywords are then search throughout the program and the line number of the keywords found are saved. ‘$file’ is filename of VHDL design. Using half_adder.vhd as an example, the text between ‘entity’ and ‘end $file’ will be shown in **Figure 20**.

```
entity half_adder is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           carry : out  STD_LOGIC);
end half_adder;
```

***Figure 20**: Text in between 'entity' and 'end $file'*

Next, curly bracket ‘ (‘ is searched in the text between the ‘entity’ and ‘end $file’. After curly bracket is found, entity line and ‘Port (‘ is removed along with comments and unnecessary white spaces. Signal along with input/output and vector information have been extracted from the VHDL file shown in **Figure 21**.

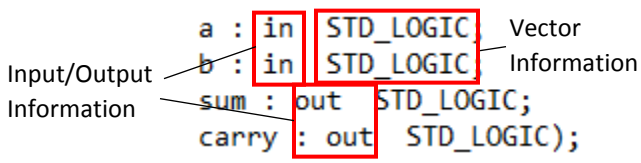


***Figure 21**: Text after removing entity line and 'Port ('*

The extracted text is used for component declaration for UUT in the testbench shown in **Figure 22**.

```
COMPONENT half_adder
 PORT(
a : in  STD_LOGIC;
        b : in  STD_LOGIC;
        sum : out  STD_LOGIC;
        carry : out  STD_LOGIC);

END COMPONENT;
```

***Figure 22**: Component declaration of UUT in testbench*

Every line in extracted text is concatenate with ‘signal tb_’ to declare the input and output signals. ‘in’ and ‘out’ string is removed from every line. At the last line, ‘); ‘ is removed and semicolon ‘;’

is added at the end of the string. Input and output signals declaration in testbench is shown in **Figure 23**.

```
-- Inputs & Outputs
signal tb_a : STD_LOGIC;
signal tb_out : STD_LOGIC;
signal tb_sum : STD_LOGIC;
signal tb_carry : STD_LOGIC;
```

***Figure 23**: Input and Output signals of testbench*

Further filtering is required to obtain name of every signal for other part of testbench.

3.2.2 Name and Input/Output information Extraction

Name of signal and input/output information is extracted by searching line by line for a given boundary and finding a certain character or string within the given boundary. The string on the left of found character is push to an array which stored the name of signals and input/output information correspond to the name of signals. This process is shown in **Figure 24** in the form of pseudo code.

```
For each line between 'entity' and 'end $file'
        Remove trailing comment
        Add spaces at the beginning of each line
        If colon ' : ' match the string of line
                Push the string on left of colon to an array in @ports
        If 'STD_LOGIC' match the string of line
                Push the string on left of 'STD_LOGIC' to an array in @inOut
```

***Figure 24**: Pseudo Code for Name extraction*

Each ports name is placed in the string '$port_name => tb_$port_name' for module instantiation as shown in the example in **Figure 25**.

```
-- Instantiate the UUT module.
uut : half_adder
port map (
        a       => tb_a,
        b       => tb_b,
        sum     => tb_sum,
        carry   => tb_carry);
```

***Figure 25**: Instantiation of half adder module in testbench*

3.2.3 Complication faced in signal extraction algorithm

Figure 26 demonstrate the complication faced in signal extraction algorithm.

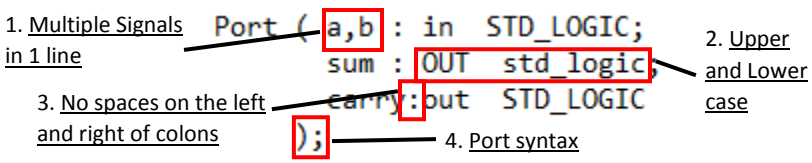


***Figure 26**: Complication faced in signal extraction algorithm*

1. Multiple Signals in 1 line: This is a problem because the way how the algorithm extract the name of signal is by searching for colon ' : ' and extract the string on the left of the colon after space. So, in the case shown in **Figure 26**, 'a,b' will be shown as input port instead of 2 separate port of 'a' and 'b'.
2. Upper and Lower case: This is a problem because the algorithm searches the string 'STD_LOGIC' in capital letter and extract the string on the left of 'STD_LOGIC' after space. So, in the case shown in **Figure 26**, 'std_logic' is in lower case and it will not match the string 'STD_LOGIC'. Hence, error message will be shown. Even if the string is in 'STD_LOGIC' upper case, 'OUT' is in upper case letter and error message will be shown as comparison between the string 'OUT' and 'out' is not the same. Hence, the input port 'sum' can't be verified as output. This problem can be easily solved by adding more condition in the appropriate 'If' loop.
3. No spaces on the left and right of colon: This is a problem as the algorithm need spaces to identify the name of the port on the left of colon ' ; '. Without the spaces, the name of port can't be identified, and error message will be shown.
4. Port syntax: The ');' syntax should be in the same line as last port declaration in the entity list like 'carry : out STD_LOGIC);'. This is done to prevent the algorithm for misplacing the ');' syntax in various part of the testbench.

Using 'STD_LOGIC_VECTOR' shown in **Figure 27** will not pose problem for algorithm as 'STD_LOGIC' is in the string which allow the algorithm to find the input/output information.

```
DataIn : in  STD_LOGIC_VECTOR (7 downto 0);
```

***Figure 27**: 'STD_LOGIC_VECTOR' in port*

3.2.4 User provided parameters (Perl's command line version)

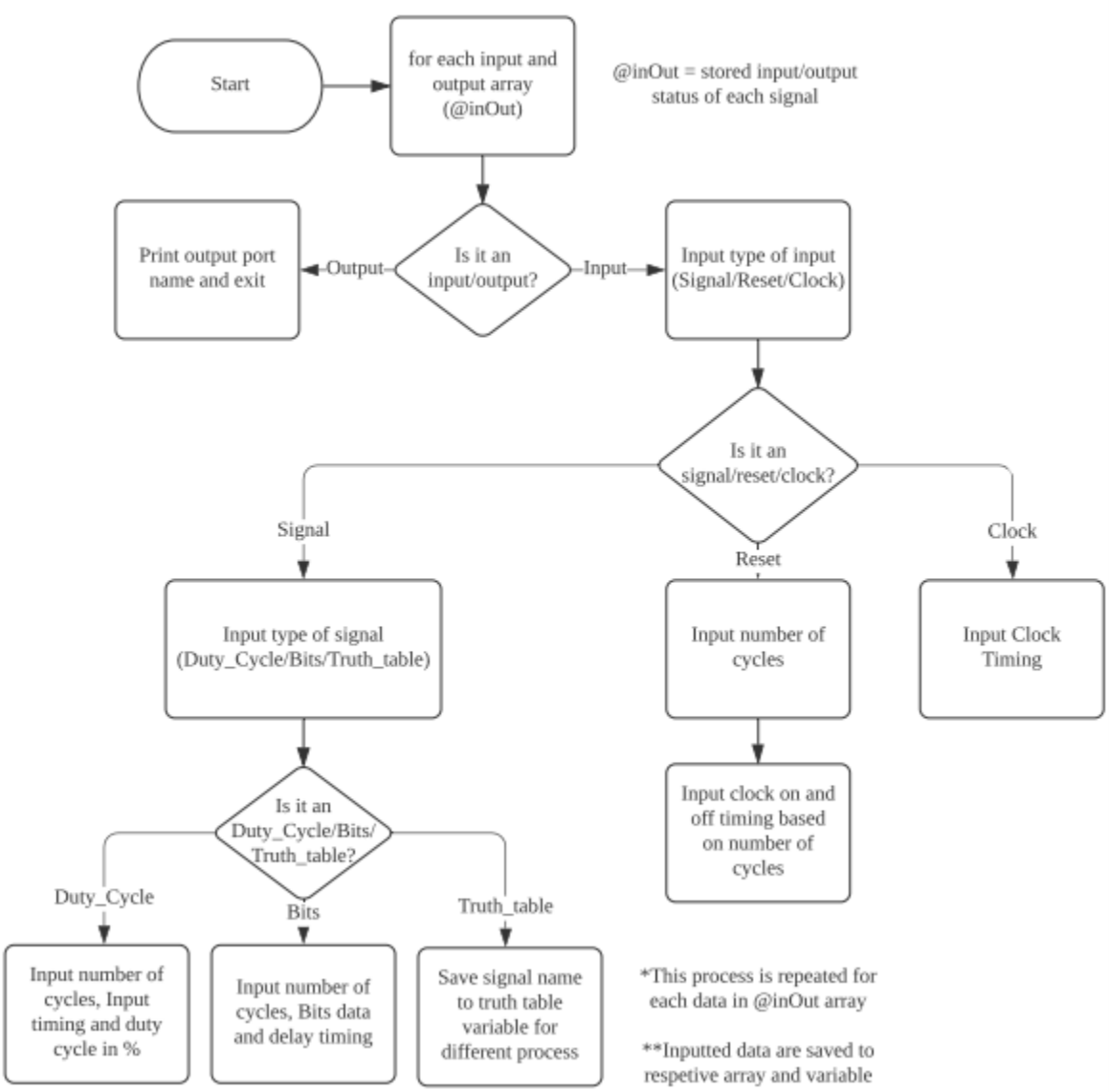


***Figure 28**: Flow chart of user provided parameters process (Perl's command line version)*

Perl's command line will be used for user to provide parameter for the given input ports. This provide backbone and rough idea for user provided parameter in GUI version. **Figure 28** demonstrate the flowchart of how Perl's command line request user for parameters for input signals. For each data in array of @inOut, data is checked whether is it an input or output. If the data is an input, Perl's command line will request for type of input based on given signals with 'Signal', 'Reset' and 'Clock' as option. The type of input for given signals will be saved in an array. If 'Signal' is chosen, Perl's command line will request for type of input signal with 'Duty_Cycle', 'Bits' and 'Truth_table'. After type of input signal is chosen, parameters based on respective input signal type will be requested by Perl's command line and the parameters will be saved in respective variable and array. Same can be applied to 'Reset' and 'Clock' input type. **Table 2** demonstrate the parameters requested by Perl's command line for given type of input.

For 'Truth_table' signal input type, the signal name will be saved to truth table variable and following process shown in **Figure 29** will be executed to request for truth table input.

```
If number of truth table variable is more than 0
        Input number of cycles
        For each cycle
                If not first cycle
                        Input delay timing
                For each value in number of truth table variable
                        Input truth table input (1/0)
```

***Figure 29**: Pseudo code for truth table input parameters*

***Table 2**: Parameters requested by Perl's command line for different type of input and signal*

Signal			Reset	Clock
Duty Cycle	**Bits**	**Truth Table**		
Number of Cycle	Number of Cycle	Number of Cycle	Number of Cycle	-
Input Timing (ns)	Delay Timing (ns) based on number of cycles	Delay Timing (ns) based on number of cycles	Delay Timing (ns) based on number of cycles	Input Timing (ns) with 50% duty cycle
Duty Cycle (out of 100)	Bits data input based on number of cycles	Truth table input (1/0) based on number of	Input Clock on and Off timing based on number of cycles	-

		truth table variable		
e.g. No. Cycle = 2 Timing (ns) = 20 Duty Cycle (%) = 50	e.g. No. Cycle = 3 Bits = 0000 Timing (ns) = 20 Bits = 1111 Timing (ns) = 10 Bits = 1010	e.g. Example for truth table input will be shown below	e.g. No. Cycle = 2 Clock On (ns) = 10 Clock Off (ns) = 20 Clock On (ns) = 30 Clock Off (ns) = 40	e.g. Clock Timing (ns) = 50
Output: tb_a <= '1'; wait for 10 ns; tb_a <= '0'; wait for 10 ns; tb_a <= '1'; wait for 10 ns; tb_a <= '0'; wait for 10 ns;	Output: tb_a <= '0000'; wait for 20 ns; tb_a <= '1111'; wait for 10 ns; tb_a <= '1010';	Output: Example for truth table input will be shown below	Output: tb_a <= '1'; wait for 10 ns; tb_a <= '0'; wait for 20 ns; tb_a <= '1'; wait for 30 ns; tb_a <= '0'; wait for 40 ns;	Output: constant clk_PERIOD1: time := 50 ns; tb_a <= '1'; wait for clk_PERIOD1 / 2; tb_a <= '0'; wait for clk_PERIOD1 / 2;

Figure 30 demonstrate the Perl's command line output for half_adder.vhd. 'Truth table' type of input is chosen for input 'tb_a' and 'tb_b'. Cycle chosen for truth table input is 4. Truth table input of 1 or 0 and delay timing is manually inputted. Automatic generation of truth table input is not enabled in Perl's command line due to the fact the input can't be edited or changed in Perl's command line. In the GUI, automatic generation of truth table inputs will be enabled.

```
 Signals input 1: tb_a
Type of Input (Signal/Reset/Clock): Signal

 Signal input type (Truth_table/Duty_Cycle/Bits): Truth_table

 Signals input 2: tb_b
Type of Input (Signal/Reset/Clock): Signal

 Signal input type (Truth_table/Duty_Cycle/Bits): Truth_table

 Signals output 1: tb_sum
 Signals output 2: tb_carry

How many cycle for Truth Table: 4

tb_a input (1/0)? :0

tb_b input (1/0)? :0

 Delay Timing: 20

tb_a input (1/0)? :0

tb_b input (1/0)? :1

 Delay Timing: 20

tb_a input (1/0)? :1

tb_b input (1/0)? :0

 Delay Timing: 20

tb_a input (1/0)? :1

tb_b input (1/0)? :1

The script has finished successfully! You can now use the file
```

User provided parameters for Truth table input

***Figure 30**: Perl's command line output for half_adder.vhd*

Figure 31 demonstrate the stimulus process of testbench generated based on the parameters inputted in **Figure 30**.

```
-- Insert Processes and code here.
-- Stimulus process
stim_proc: process
begin
tb_a <= '0';
tb_b <= '0';
wait for 20 ns;

tb_a <= '0';
tb_b <= '1';
wait for 20 ns;

tb_a <= '1';
tb_b <= '0';
wait for 20 ns;

tb_a <= '1';
tb_b <= '1';
wait;
end process;
```

***Figure 31**: Stimulus process of testbench generated for half_adder.vhd*

3.2.5 Testbench Generator

New file is generated for testbench and testbench syntax of library, entity, architecture, component declaration, inputs/outputs declaration, clock period declaration and instantiation of module is written into the new file as shown in **Figure 32** and **Figure 33**.

```
# Open new vhdl file for testbench
open(my $inF, ">", $new_file_vhd);

# Library
printf($inF "Library IEEE;\n");
printf($inF "use IEEE.STD_LOGIC_1164.all;\n");
printf($inF "use IEEE.std_logic_unsigned.all;\n");
printf($inF "use IEEE.std_logic_arith.all;\n");
printf($inF "use IEEE.Numeric_STD.all;\n");
#printf($inF "\n");
#printf($inF "library work;\n");
#my $new_text = join "_", $file, "pkgs.all";
#printf($inF "use work.$new_text;\n");
printf($inF "\n");
printf($inF "\n");

# Entity
printf($inF "-- Declare module entity. Declare module inputs, inouts, and outputs.\n");
printf($inF "entity $new_file is\n");
printf($inF "end $new_file;\n");
printf($inF "\n");

# Architecture
printf($inF "-- Begin module architecture/code.\n");
printf($inF "ARCHITECTURE behavior OF $new_file IS\n");
printf($inF "\n");

# Component
print ($inF "COMPONENT $file\n");       #print first line
print ($inF " PORT(\n");                #print second line
my $out= join "\n\t", @ports;
print ($inF "$out\t\n\t\nEND COMPONENT;\n"); #print ports and last couple of lines
print ($inF "\n");
```

***Figure 32**: Testbench syntax of Library, Entity, Architecture, Component*

```
# Input and Outputs
printf($inF "-- Inputs & Outputs\n");
printf($inF "$out3\n");

# Clock Constant
printf($inF "-- *** Instantiate Constants ***\n");
if ($no_clock != 0)
{
        for (my $i=1 ; $i <= $no_clock ; $i++)
        {
                printf($inF "constant clk_PERIOD$i: time := $clock[$i-1] ns;\n");
                printf($inF "\n");
        }
}

# Instantiate UUT
printf($inF "BEGIN\n");
printf($inF "\n");
printf($inF "-- Instantiate the UUT module.\n");
printf($inF "uut : $file\nport map (");       #print first line
printf($inF "\n\t$out2);\n\n");
printf($inF "\n");
```

***Figure 33**: Testbench syntax of inputs/output declaration, clock constant and instantiation of UUT*

Using the saved parameter of name of input signal, signal input type, number of cycles, number of data, timing information, data information in their respective array and variable, these parameters are written into the new file. **Figure 34**, **Figure 35, Figure 36**, and **Figure 37** demonstrate the code snippets of generation of testbench syntax for clock, reset, signal and truth table process respectively.

A. Clock Process

```
# Generate Clock
if ($no_clock != 0)
{
printf($inF "-- Generate necessary clocks.\n");

        for (my $i=1 ; $i <= $no_clock ; $i++)
        {
                printf($inF "Clk_process$i: process\n");
                printf($inF "begin\n");
                printf($inF "\ttb_$clock_port[$i-1] <= '1';\n");
                printf($inF "\twait for clk_PERIOD$i / 2;\n");
                printf($inF "\ttb_$clock_port[$i-1] <= '0';\n");
                printf($inF "\twait for clk_PERIOD$i / 2;\n");
                printf($inF "end process;\n");
                printf($inF "\n");
        }
}
```

Number of Clock Variable

Name of Clock variable

Clock Period

***Figure 34**: Clock process generation*

B. Reset Process

```
# Reset
if ($no_reset != 0)
{
printf($inF "-- Toggle the resets.\n");

        for (my $i=1 ; $i <= $no_reset ; $i++)
        {
                printf($inF "reset$i: process\n");
                printf($inF "begin\n");

                for (my $j=0 ; $j <= $reset_cycle[$i-1] - 1 ; $j++)
                {
                        printf($inF "\ttb_$reset_port[$i-1] <= '1';\n");
                        printf($inF "\twait for $reset_on[$i-1][$j] ns;\n");
                        printf($inF "\ttb_$reset_port[$i-1] <= '0';\n");
                        printf($inF "\twait for $reset_off[$i-1][$j] ns;\n");
                }
                printf($inF "\ttb_$reset_port[$i-1] <= '1';\n");

                printf($inF "\twait;\n");
                printf($inF "end process;\n");
                printf($inF "\n");
        }
}
```

Number of Reset Variable

Number of Cycle for given input

Name of signal

Clock on and off timing detail

***Figure 35**: Reset process generation*

C. Signal Process

```
# Stimulus process
if ($no_signal_input != 0)
{
printf($inF "-- Insert Processes and code here:\n");

        for (my $i=1 ; $i <= $no_signal_input ; $i++)
        {
                my $signal_match = '0';
                for (my $k=0 ; $k <= $no_truth_table - 1 ; $k++)
                {
                        if ($signal_input[$i-1] =~ $signal_truthTable[$k])
                        {
                                $signal_match = '1';
                        }
                }

                if ($signal_match =~ '0')
                {
                        printf($inF "-- Stimulus process$i\n");
                        printf($inF "$signal_input[$i-1]: process\n");
                        printf($inF "begin\n");

                        if ($signal_input_type[$i-1] =~ "Duty_Cycle")
                        {
                                for (my $j=0 ; $j <= $signal_cycle_DC[$i-1] - 1 ; $j++)
                                {
                                        printf($inF "tb_$signal_input[$i-1] <= '1';\n");
                                        my $delay_on = ($signal_input_duty_cycle[$i-1]/100)*$signal_input_timing[$i-1];
                                        printf($inF "wait for $delay_on ns;\n");

                                        printf($inF "tb_$signal_input[$i-1] <= '0';\n");
                                        my $delay_off = ((100 - $signal_input_duty_cycle[$i-1])/100)*$signal_input_timing[$i-1];
                                        printf($inF "wait for $delay_off ns;\n");
                                }
                        }

                    elsif ($signal_input_type[$i-1] =~ "Bits")
                    {
                            for (my $j=0 ; $j <= $signal_cycle_bits[$i-1] - 1 ; $j++)
                            {
                                    if ($j > 0)
                                    {
                                    printf($inF "wait for $signal_input_delay[$i-1][$j] ns;\n");
                                    }

                                    printf($inF "tb_$signal_input[$i-1] <= '$signal_input_bits[$i-1][$j]';\n");
                            }
                    }

                    printf($inF "wait;\n");
                    printf($inF "end process;\n");
                    printf($inF "\n");
            }
    }
```

Number of Signal input Variable

If signal input match truth table input variables, it will jump to truth table process in **Figure 37**.

Number of Duty Cycle Signal input

Name of Duty Cycle Input

Duty Cycle (%)

Duty Cycle input timing

Number of Duty Cycle Signal input

Delay timing for Bits

Name of signals

Bits data

***Figure 36**: Signal process generation*

```
        if ($no_truth_table > 0)
        {
                printf($inF "-- Stimulus process\n");
                printf($inF "stim_proc: process\n");
                printf($inF "begin\n");

                for (my $i=0 ; $i <= $cycle_truthTable - 1 ; $i++)
                {
                        if ($i > 0)
                        {
                                printf($inF "wait for $input_delay[$i] ns;\n\n");
                        }

                        for (my $j=0 ; $j <= $no_truth_table - 1 ; $j++)
                        {
                                printf($inF "tb_$signal_truthTable[$j] <= '$input_truthTable[$i][$j]';\n");
                        }
                }
                printf($inF "wait;\n");
                printf($inF "end process;\n");
                printf($inF "\n");
        }
}
```

Number of truth table

Number of cycles for truth table inputs

Input delay

in

Name of signals for truth table

Truth table data

***Figure 37**: Truth table process generation*

3.3 Frontend Design

3.3.1 Graphical User Interface (GUI)

Development of GUI is crucial for automatic VHDL testbench generator as it allow the program to be user-friendly. Using command line as input reduce the flexibility for the user to edit the inputted data. Furthermore, use of command line might be more prone to error as mistyped data can't be edited. Due to GUI being user-friendly, use of GUI has advantage of consuming lesser time to input data. Truth table input can be automatically generated and edited with the help of GUI making the process more intuitive. Tk library is used for Perl GUI development. **Figure 38** demonstrate the main page GUI of automatic testbench generator and **Table 3** demonstrate the list of controls in the main page.

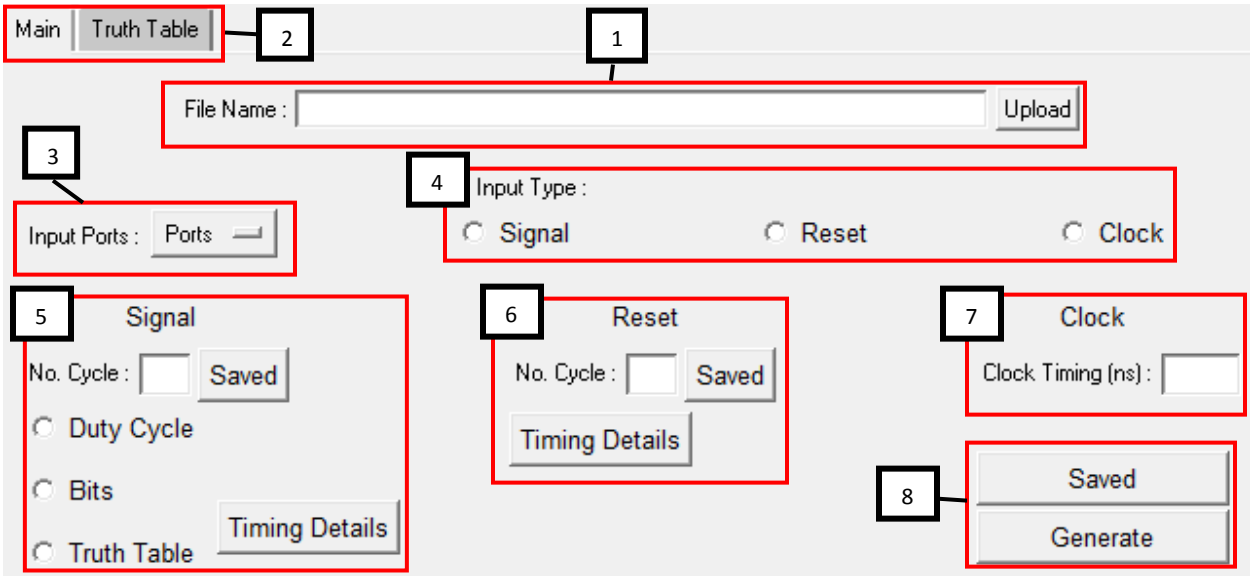


Figure 38: GUI main page

Table 3: List of controls in the main page

Controls	Function
1 – Upload file	When 'upload file' button is pressed, directory of the system is shown, and VHDL design file can be selected. If testbench file is selected, error message will be shown.
2 – Toggle page	Toggle between main page and truth table page
3 – Menu list for Input ports	Variables menu list show the extracted input ports where user can select the input ports
4 – Type of Input	Signal, Reset, and Clock input type can be selected.

5 – Signal Timing Input	When signal type is selected, user can input the number of cycle and saved it. Duty cycle, Bits and Truth table signal input type can be selected, and 'timing details' button will request for input parameters correspond to the signal input type and number of cycles.
6 – Reset Timing Input	When reset type is selected, user can input the number of cycle and saved it. 'timing details' button will request input parameters based on number of cycles.
7 – Clock Timing Input	When clock type is selected, user can input clock timing in nanoseconds with duty cycle set to 50%.
8 – Saved and Generate	After all the parameters are inputted for given input port, 'saved' button is pressed to save the parameters. After all the all the input ports have been saved with parameters, 'generate' button is pressed to generate testbench. Testbench will be automatically added in the directory of where VHDL design file is uploaded.

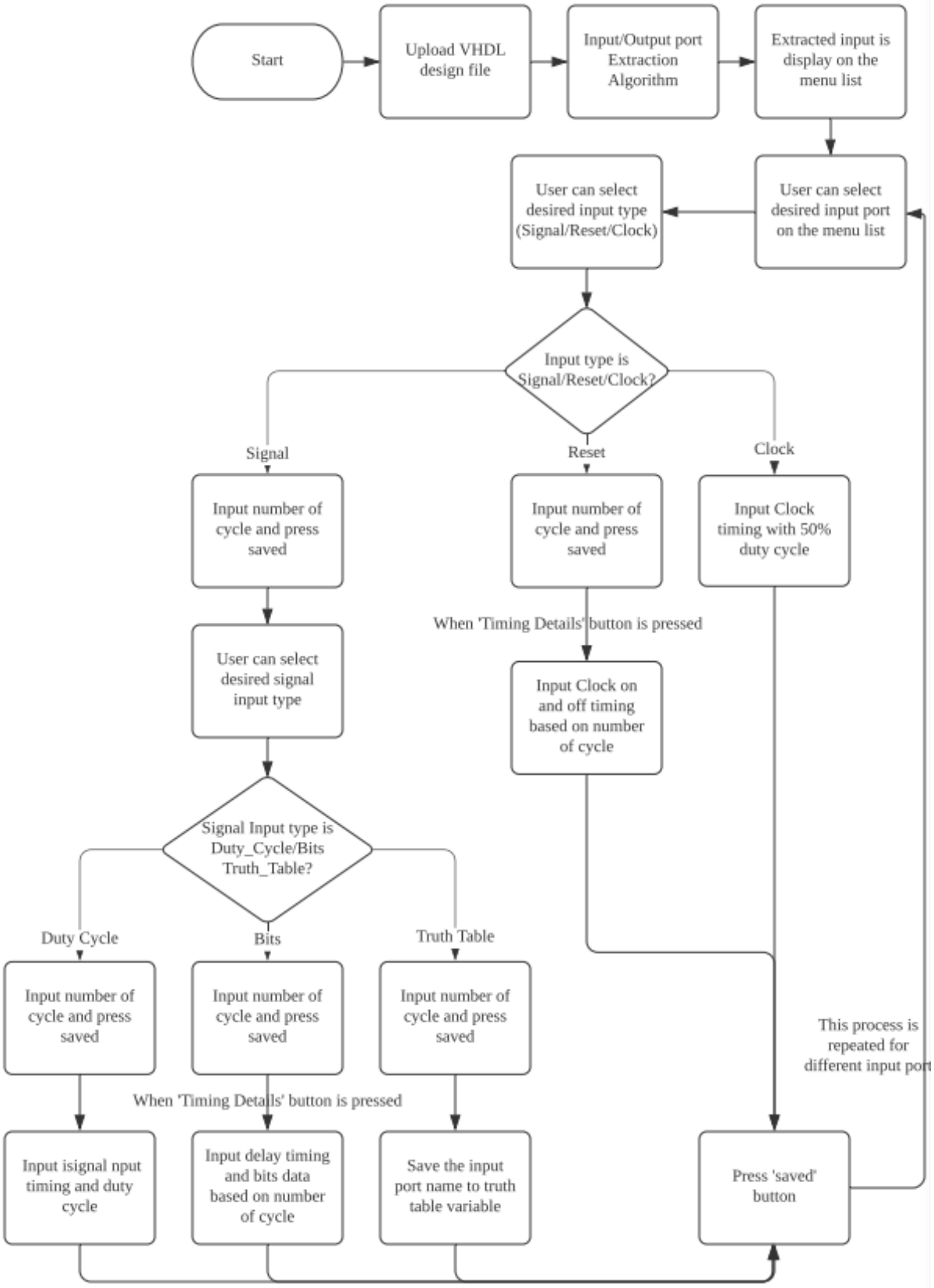


***Figure 39**: Flowchart of GUI process*

Figure 39 demonstrate the flow chart of GUI process. VHDL file is uploaded to the GUI and input/output port extraction algorithm is used. Extracted ports will be displayed in the menu list. User can select their desired input type and input the parameters correspond to the input type as discussed in detail before under **Chapter 3.2.4** named 'User Provided Parameters (Perl's command line version)'. Once parameters are filled, 'saved' button can be pressed to save the parameters correspond to the input port. This process is repeated for every input ports.

Once all the input ports are filled with desired parameters, 'generate' button is pressed and new file will be created. All the relevant testbench syntax and parameters will be written to the new file as discussed before under **Chapter 3.2.5** named 'Testbench Generator'. The variables and array that stored all the parameters are reset to 0 when 'generate' button is pressed.

3.3.2 User Provided Parameters (GUI version)

When 'Timing Details' button is pressed, a window will pop up and request user for parameters correspond to their respective input type and number of cycles as shown below.

A. Duty Cycle Signal input in GUI

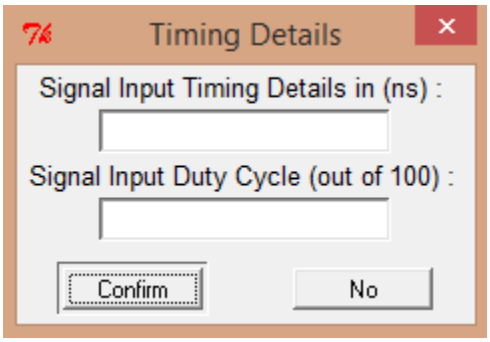


***Figure 40**: Duty Cycle Signal input in GUI*

B. Bits Signal input in GUI

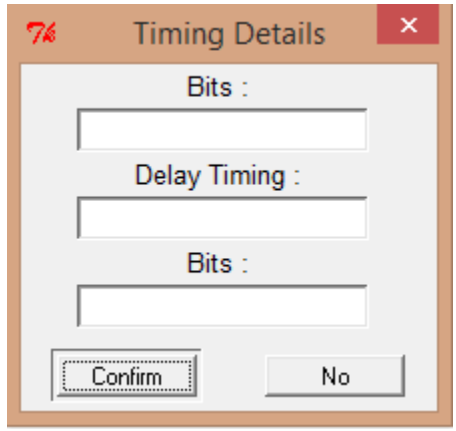


***Figure 41**: Bits Signal input in GUI with 2 cycle*

C. Truth Table input in GUI

For truth table input, name of input port is saved to truth table variable. After all the desired truth table variable is saved, 'Truth Table' page can be toggled. 'update' button is pressed to update all the desired truth table variable into truth table. The input of truth table is automatically filled with all the possible combination for a truth table based on number of truth table variable. The input of truth table can be changed to 0 or 1 which allow flexibility in the design. Delay timing can be inputted at the bottom of GUI and truth table input can be saved. 'Generate' button in the main page can be pressed to generate testbench with truth table input. **Figure 42** and **Figure 43** demonstrate truth table input in GUI with 2 and 3 truth table variables respectively.

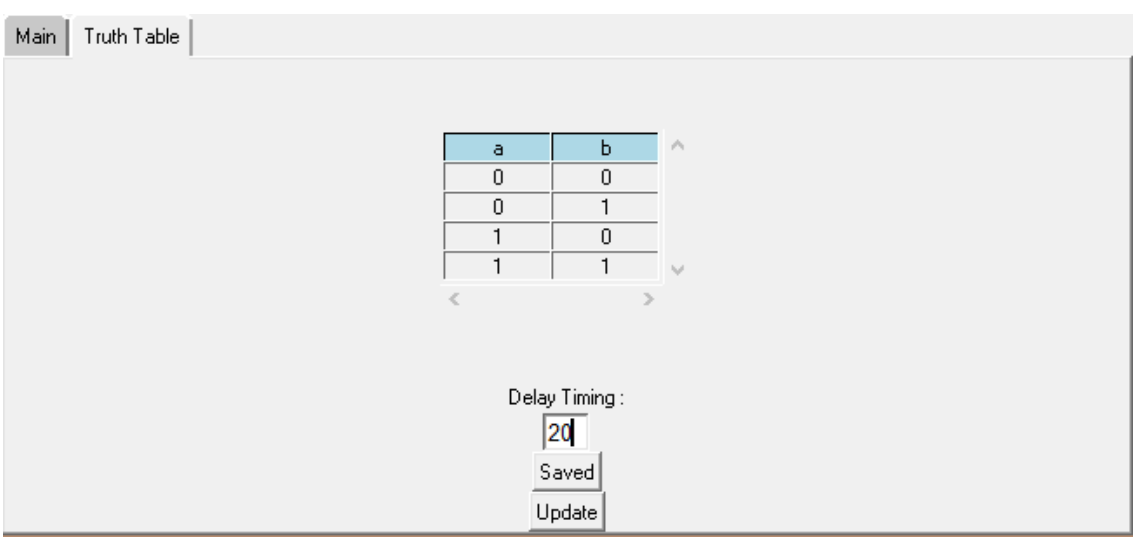


***Figure 42**: Truth Table input in GUI with 2 truth table variables*

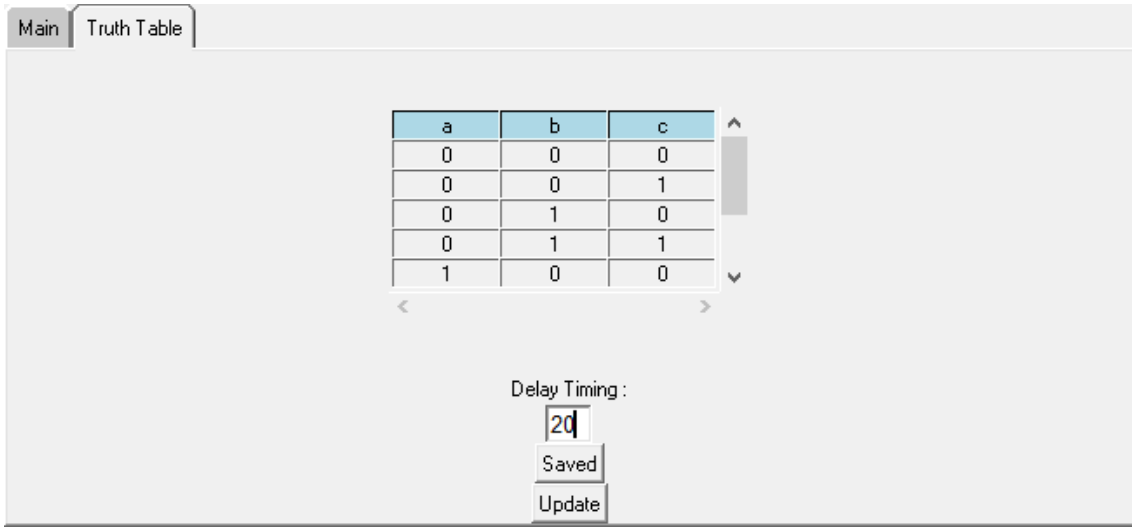


***Figure 43**: Truth Table input in GUI with 3 truth table variables*

D. Reset Input in GUI

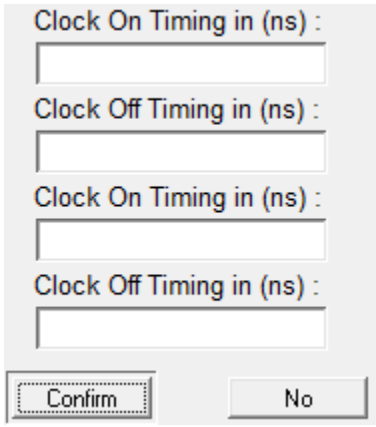


***Figure 44**: Reset input in GUI with 2 cycle*

E. Clock Input in GUI

***Figure 45**: Clock Input in GUI*

3.4 Improvement to automatic testbench generation tool

The ability to save user provided parameters to a text file is very convenient feature for user. This remove the hassle of re-entering the parameters and allow the user to edit the parameters of the design at a later date.

VHDL design file is read and text file with port name, input/output information and vector information are generated. Template for selecting the parameters based on type of input is generated in the testbench. User can input their desired parameters by editing the text file. Testbench is generated based on the information on the text file. The block diagram of this feature is shown in **Figure 46**.

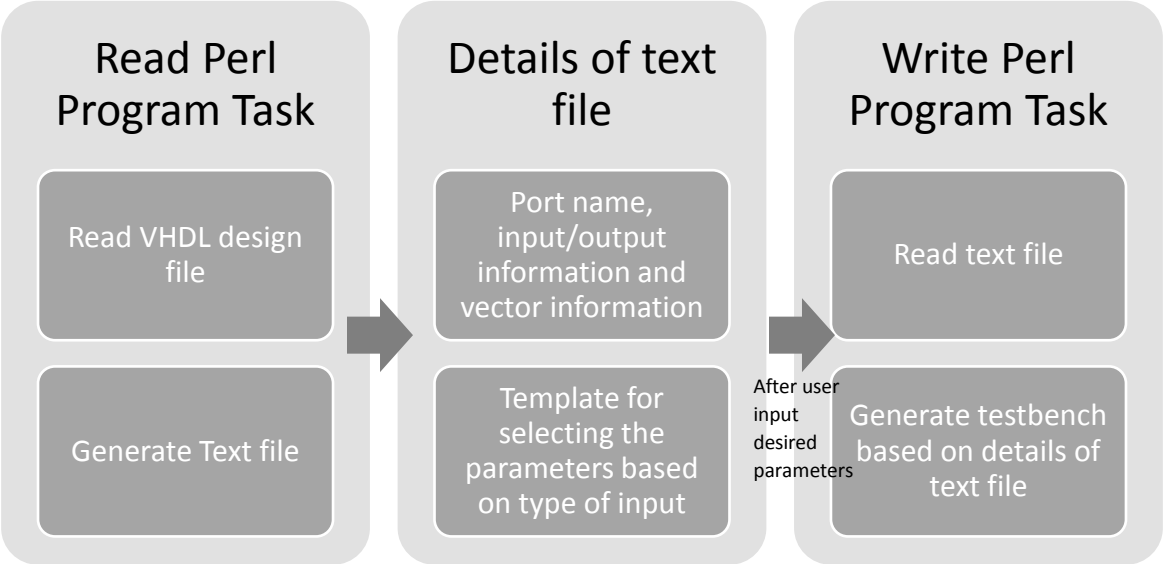


***Figure 46**: Block diagram of saving parameters feature*

A. Read Perl Program

VHDL design file is read, and signal extraction algorithm is used to extract port name, input/output information and vector information. Perl program will select the type of input for input ports by searching input ports name for particular keyword. The pseudocode shown in **Figure 47** will demonstrate this process.

```
For each input/output information in @inOut array
        If it is input
                If port name match 'Rst' or 'rst' or 'Reset' or 'reset'
                        Template for reset input type parameters is written to text file
                Elseif port name match 'Clk' or 'clk' or 'Clock' or 'clock'
                        Template for clock input type parameters is written to text file
                Else
                        Template of signal input type parameters is written to text file
        Else it is output
                Write signal output information to text file
        Else
                Return error
```

Figure 47: *Pseudocode for auto selection input type in Perl program*

To generate Truth table input, Perl program take the input ports that have been categorized to signal input type and generate all combination of truth table input for the given input ports. The combination of truth table input is written to text file at the bottom of the page.

The extracted information is written to text file along with the template for selecting the parameters based on the type of input. The text file is named using concatenation of filename and '_stimulus' (E.g. half_adder_stimulus.txt).

B. Text file

Figure 48, 49, 50 and 51 demonstrate the text file with template for selecting parameters based on the type of input. For signal input type, duty cycle, bits or truth table signal input type can be chosen and parameters on the chosen signal input type can be filled. The truth table input can be edited at the bottom of the text file by deleting or adding text according to the user desired parameters.

```
Signal input 1 =: Reset : in STD_LOGIC;

++++++++++++++++++++++++++++++++++++++++++++

-- Reset Input Type

------------------

How many Cycle : 2
Clock On Timing in (ns) : 20
Clock Off Timing in (ns) : 10

******************
******************************************

Signal input 2 =: Clock16x : in STD_LOGIC;

++++++++++++++++++++++++++++++++++++++++++++

-- Clock Input Type

------------------------------------------

Clock Timing in (ns) : 20
```

Port Name, Input/Output Information, and vector information

Template for Reset input type parameters

Template for Clock input type parameters

```
Signal input 3 =: Rxd1 : in STD_LOGIC;

+++++++++++++++++++++++++++++++++++++++++++++++++++

-- Signal Input Type --

---------------------------

Input Type (Duty_Cycle/Bits/Truth_Table) : Truth_Table

-----------------------------

Duty Cycle Input Type
How many Cycle :
Signals input Period in (ns) :
Signals input 3 duty cycle (out of 100) :

---------------------------------------------------

-- Bits Input Type --
Bits1 :
Delay Timing in (ns) :
Bits2 :
Delay Timing in (ns) :
Bits3 :
```

Select type of signal inputs

Template for Duty Cycle Signal input type parameters

Template for Bits Signal input type parameters

***Figure 48**: Text file using RS232 design as example*

```
-- Truth Table Input Type --

+++++++++++++++++++++++++++++++++++++++++++++++++++

Do You Want Truth Table Inputs (Yes/No) : Yes

Rxd1 test1 test2
0    0     0
0    0     1
0    1     0
0    1     1
1    0     0
1    0     1
1    1     0
1    1     1
Time : 20
```

Selection of type of signal inputs

3 input ports

All combination of truth table input and delay timing

***Figure 49**: Text file with truth table input type*

```
Signal output 1 == DataOut1 : out STD_LOGIC_VECTOR (7 downto 0);
Signal output 2 == DataOut2 : out STD_LOGIC);
```

***Figure 50**: Text file with output signal*

There are some criteria that needed to be meet in order for the algorithm to function correctly. The criteria are as follows:

1. The parameters for non-chosen signal input type must be left blank to not confused the algorithm. **Figure 51** demonstrate what should not be done in the text file.

```
Signal input 3 =: Rxd1 : in STD_LOGIC;

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

-- Signal Input Type --

-----------------------------------

Input Type (Duty_Cycle/Bits/Truth_Table) : Truth_Table

-----------------------------------------------------------

Duty Cycle Input Type
How many Cycle : 2
Signals input Period in (ns) : 100
Signals input 3 duty cycle (out of 100) : 50

-----------------------------------------------------------

-- Bits Input Type --
Bits1 : 1111
Delay Timing in (ns) : 20
Bits2 : 1010
Delay Timing in (ns) : 30
Bits3 :
```

Truth table input type selected

Duty Cycle parameters are filled

Bits parameters are filled

***Figure 51**: Parameters for non-chosen signal input type are filled.*

2. There must have space between the colon ':' when filling up the parameters unlike the example shown in **Figure 52**.

```
Input Type (Duty_Cycle/Bits/Truth_Table) :Duty_Cycle

-----------------------------------------------------

Duty Cycle Input Type
How many Cycle :2
Signals input Period in (ns) :100
Signals input 3 duty cycle (out of 100) :50
```

***Figure 52**: No spaces after colon ':'*

C. Write Perl Program

Perl program read the text file and certain string are search throughout the program to extract all the relevant information. The extracted information is then sorted to respective array and testbench is generated. **Figure 53, 54, 55 and 57** demonstrate the pseudocode for extracting relevant information in the text file and sorting the extracted information.

```
For each line in the text file
    If 'Signal input' is found in the line
        Save the line number to an array
        If equal and colon symbols '=:' is found
            Save input signal to an array (e.g. Reset : in STD_LOGIC;)
            If colon ':' is found
                Save input port name on the left of colon (e.g. Reset)
    Else if 'Signal output' is found in the line
        Save the (line number – 1) to an array
        If 2 equal symbols '==' is found
            Save output signal to an array (e.g. DataOut1 : out STD_LOGIC;)
            If colon ':' is found
                Save input port name on the left of colon (e.g. DataOut1)
```

Figure 53*: Extraction of signal input port from text file*

```
For each number of input data
    If 'Rst' or 'rst' or 'Reset' or 'reset' is found
        Set type as Reset
        Save port name as into reset array
    Else if 'Clk' or 'clk' or 'Clock' or 'clock' is found
        Set type as Clock
        Save port name as into clock array
    Else
        Set type as signal
        Save port name as into signal array
    For each line between signal input section (e.g. line between Signal input 1 and 2)
        If type is Reset
            If colon ':' is found, on the right of colon
                Save Reset Parameter to an array
        If type is Clock
            If colon ':' is found, on the right of colon
                Save Clock Parameter to an array
        If type is Signal
            If colon ':' is found, on the right of colon
                Save Signal Parameter to an array
                Save the number of Signal parameter found
    If number of signal parameter is not 0
        Save the number of Signal parameter found to an array
```

Figure 54*: Extraction of input parameters from text file*

```
For each data in reset array
    For each 3 runs
        Separate 3 parameters (number of cycles, clock on and off timing) to their
        respective reset input port


For each number of signal input
    If signal input is 'Duty_Cycle'
        For each number of Signal parameter found
            Separate 3 parameters (number of cycles, input timing and duty
            cycle) to their respective signal input port
    Else if signal input is 'Bits'
        For each number of Signal parameter found
            Separate parameters (Bits and delay timing) to their respective
            signal input port
```

***Figure 55**: Sorting of input variable*

For 'Bits' input type, the number of input parameters are variable shown in **Figure 56**. Unlike 'Bits' input type, 'reset' and 'Duty Cycle' input type has fixed input parameters of 3. 'Clock' input type has fixed input parameters of 1. Thus, number of Signal parameter found is saved to keep track of number of input parameters.

```
-- Bits Input Type --
Bits1 : 0000
Delay Timing in (ns) : 20
Bits2 : 0001
Delay Timing in (ns) :
Bits3 :
```

```
-- Bits Input Type --
Bits1 : 0000
Delay Timing in (ns) : 20
Bits2 : 0001
Delay Timing in (ns) : 30
Bits3 : 1111
```

***Figure 56**: 3 input variables (left) vs 5 input variables (right)*

```
For each line in truth table section
        If line has 'Time'
                If colon ':' is found, on the right of colon
                        Save delay timing input to an array
        Else
                Split each line to different column and saved it to an array
```

***Figure 57**: Extraction and sorting of truth table input*

After all the parameters have extracted and sorted, the saved arrays and variable are used to generate testbench as discussed in **Chapter 3.2.5** name 'Testbench Generator'. This feature of saving parameters of signals is not implemented with GUI due to time constraint.

Chapter 4 – Results and Discussion

4.1 Overview

In order to demonstrate the working of this automatic testbench generation tool, complete testbench generation flow is shown using example of half-adder design (Asynchronous Design), clock divider design (Synchronous Design) and transmitter of RS232 (Finite-State Machine implementation design). The testbench generated for these designs are simulated using Xilinx's ISE tool and the simulation results are verified.

To demonstrate the diversity of design the testbench generation tool can handle, asynchronous design, synchronous design and state machine implementation design are used. An asynchronous design is a design where state of device can be change at any time in response to changing inputs. It does not rely on the main clock signal. Synchronous design on the other hand is a design where the state of the device changes only at discrete times in response to a clock signal. Finite-state machine is behavior modelling of design of hardware digital systems. It is used to represent and control execution flow using finite number of states.

4.2 Asynchronous Design (Half-adder circuit)

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4
5   entity half_adder is
6       Port ( a : in  STD_LOGIC;
7              b : in  STD_LOGIC;
8              sum : out  STD_LOGIC;
9              carry : out  STD_LOGIC);
10  end half_adder;
11
12  architecture Behavioral of half_adder is
13
14  begin
15
16     sum <= a xor b;
17     carry <= a and b;
18
19  end Behavioral;
```

***Figure 58**: Half adder VHDL design file (half_adder.vhd)*

As explained earlier, the user only needs to input VHDL design file, input type and parameters correspond to the input type like clock timing, duty cycle, bits data etc. The following are simple steps for user to generate VHDL testbench of half adder design using this tool.

1) User need to upload the half_adder.vhd file shown in **Figure 58** from any local directory by pressing the 'upload' button in the GUI. The director of half_adder.vhd file will be shown in the text box.

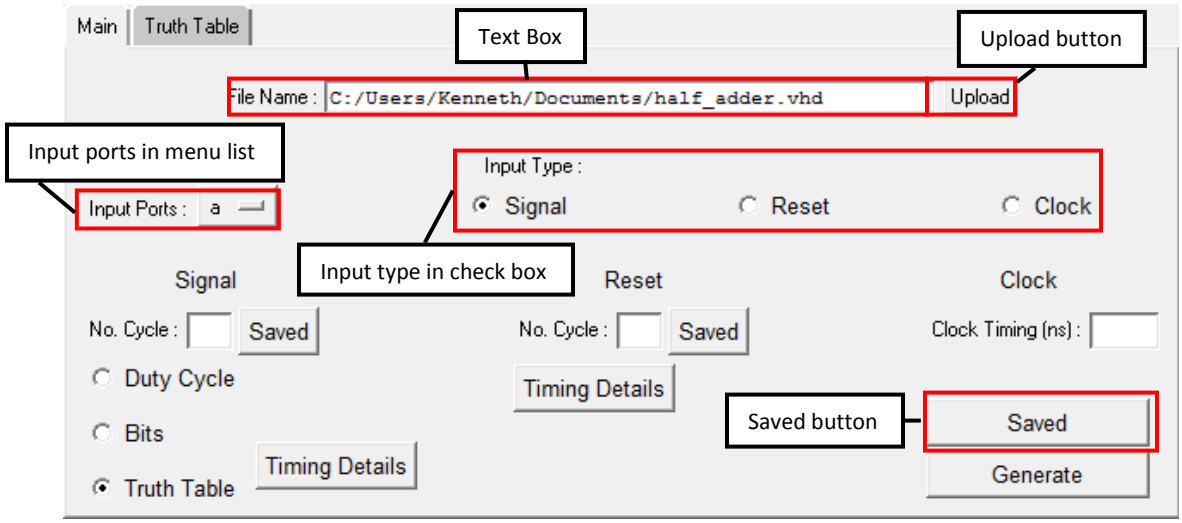


***Figure 59**: GUI main page for half adder design*

2) User can select input ports via the menu list and select the input type via the check box. Parameters correspond to input type can be filled. In the case of half adder, inputs would include *a* and *b* with both having same types *signal* and *truth table*. Parameters are saved by pressing the 'Saved' button.
3) After all the input and their corresponding parameters are saved, the next step is to toggle to truth table page since truth table input type is chosen and pressed the 'update' button. Truth table input with all the possible combination is generated for input *a* and *b*. Delay timing of 20ns is inputted. 'saved' button is pressed to save truth table input and delay timing.

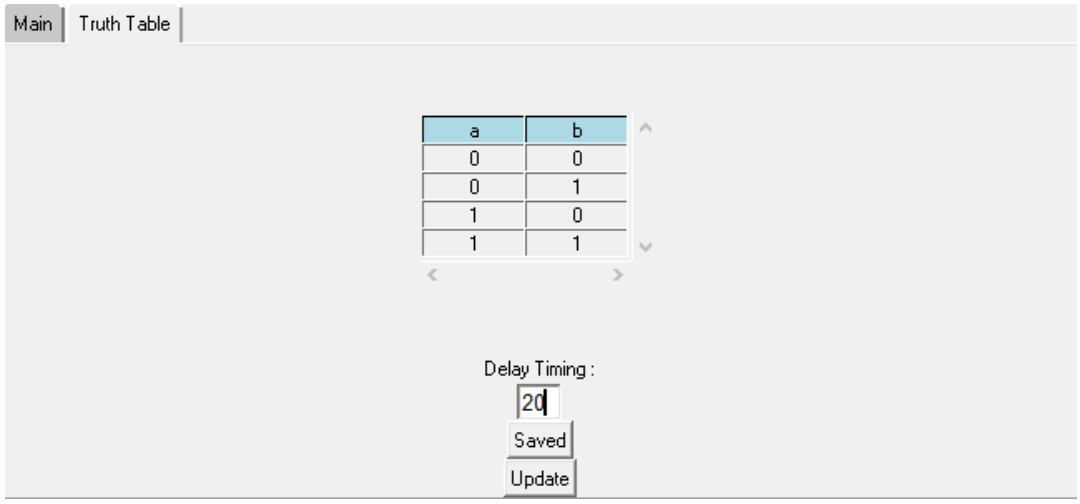


***Figure 60**: Truth table page for half adder design*

4) The final step is to press the 'generate' button back at the main page. The half adder testbench will be generated in the same directory the user inputted the half_adder.vhd file. 'half_adder_tb.vhd' testbench file is shown in **Figure 61**.

```
1  Library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4
5  -- Declare module entity. Declare module i
6  entity half_adder_tb is
7  end half_adder_tb;
8
9  -- Begin module architecture/code.
10 ARCHITECTURE behavior OF half_adder_tb IS
11
12 COMPONENT half_adder
13  PORT(
14 a : in  STD_LOGIC;
15    b : in  STD_LOGIC;
16    sum : out  STD_LOGIC;
17    carry : out  STD_LOGIC);
18
19 END COMPONENT;
20
21 -- Inputs & Outputs
22    signal tb_a :  STD_LOGIC;
23    signal tb_b :  STD_LOGIC;
24    signal tb_sum :  STD_LOGIC;
25    signal tb_carry :  STD_LOGIC;
26
27 -- *** Instantiate Constants ***
28 BEGIN
29
30 -- Instantiate the UUT module.
31 uut : half_adder
32 port map (
33    a  => tb_a,
34    b  => tb_b,
35    sum   => tb_sum,
36    carry    => tb_carry);
```

```
39 -- Insert Processes and code here.
40 -- Stimulus process
41 stim_proc: process
42 begin
43 wait for 20 ns;
44
45 tb_a <= '0';
46 tb_b <= '0';
47 wait for 20 ns;
48
49 tb_a <= '0';
50 tb_b <= '1';
51 wait for 20 ns;
52
53 tb_a <= '1';
54 tb_b <= '0';
55 wait for 20 ns;
56
57 tb_a <= '1';
58 tb_b <= '1';
59 wait;
60 end process;
61
62 END behavior; -- architecture
```

***Figure 61**: Testbench generated using automatic testbench tool*

The text file generated for saving half adder parameters is shown in **Figure 62** and **63**. This feature is not implemented with the GUI where the inputted parameter will not be automatically saved to text file. User have to retype the parameters in the text file. Text file is named 'half_adder_stimulus'. The testbench generated using text file are similar to the testbench in **Figure 61**.

```
Signal input 1 =: a : in  STD_LOGIC;

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

-- Signal Input Type --

--------------------------------------------------------------

Input Type (Duty_Cycle/Bits/Truth_Table) : Truth_Table

--------------------------------------------------------------

Duty Cycle Input Type
How many Cycle :
Signals input Period in (ns) :
Signals input 1 duty cycle (out of 100) :

--------------------------------------------------------------

-- Bits Input Type --
Bits1 :
Delay Timing in (ns) :
Bits2 :
Delay Timing in (ns) :
Bits3 :
```

User input

```
Signal input 2 =: b : in  STD_LOGIC;

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

-- Signal Input Type --

--------------------------------------------------------------

Input Type (Duty_Cycle/Bits/Truth_Table) : Truth_Table

--------------------------------------------------------------

Duty Cycle Input Type
How many Cycle :
Signals input Period in (ns) :
Signals input 2 duty cycle (out of 100) :

--------------------------------------------------------------

-- Bits Input Type --
Bits1 :
Delay Timing in (ns) :
Bits2 :
Delay Timing in (ns) :
Bits3 :
```

***Figure 62**: Text file for saving half adder parameters (part 1)*

```
Signal output 1 == sum : out  STD_LOGIC;
Signal output 2 == carry : out  STD_LOGIC);
```

```
**********************************************
**********************************************

-- Truth Table Input Type --

++++++++++++++++++++++++++++++++++++++++++++++

Do You Want Truth Table Inputs (Yes/No) : Yes

a    b
0    0
0    1
1    0
1    1
Time : 20
```

***Figure 63**: Text file for saving half adder parameters (part 2)*

As shown in simulation results in **Figure 64**, timing detail is 20ns as inputted in the design. The functionality of the half adder design is verified using truth table of half adder shown in **Table 4** as explained **Chapter 1.1**. For example, between 40ns to 60ns in the simulation, *tb_a* and *tb_b* input is 0 and 1 respectively. *tb_sum* and *tb_carry* output is 1 and 0 respectively which is verified to be the correct output using truth table of half adder.

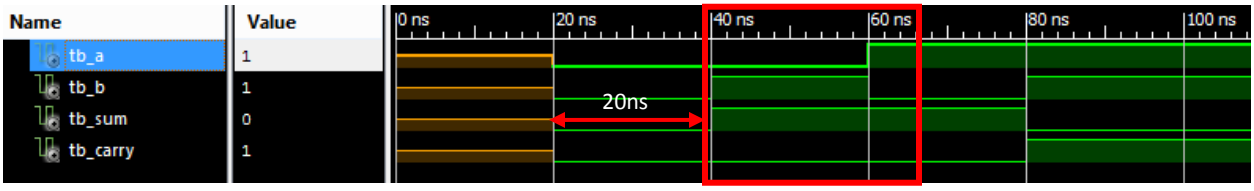


***Figure 64**: Simulation Results for half adder design*

***Table 4**: Truth Table of half adder*

Inputs		Outputs	
A	**B**	**Sum**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

4.3 Synchronous Design (Clock divider Design)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;

entity Counter is
    Port ( Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           clk_divider2 : out STD_LOGIC;
           clk_divider4 : out STD_LOGIC;
           countout : out  STD_LOGIC_VECTOR (3 downto 0));
end Counter;

architecture Behavioral of Counter is
Signal int_count: std_logic_vector(3 downto 0):= (others => '0');
begin
   Process (Clk, reset)
   begin
         if (reset = '0') then
            int_count <= (others => '0');
         elsif rising_edge (clk) then
            --if (int_count = "1001") then
               --int_count <= (others => '0');
            --else
               int_count <= int_count + '1';
            --end if;
         end if;

   end Process;
   countout <= int_count;

   clk_divider2 <= int_count(0);

   clk_divider4 <= int_count(1);

end Behavioral;
```

***Figure 65**: Clock divider VHDL design file (Counter.vhd)*

Figure 65 demonstrate code for clock divider VHDL design and Counter.vhd is uploaded to the automatic testbench generator. Parameters are inputted for *Clk* and *Reset* input ports are as follow:

<u>*Clk* input port</u>

Input type: Clock

Clock Timing in (ns): 20

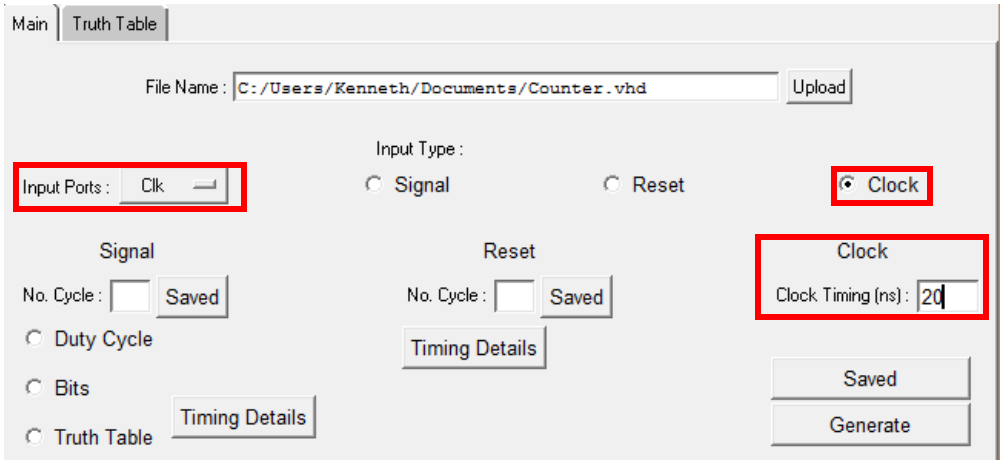


***Figure 66**: Parameters for Clk input port*

<u>*Reset* input port</u>

Input type: Reset

No. Cycle: 2

Timing Details:

Clock On Timing in (ns): 5

Clock Off Timing in (ns): 25

Clock On Timing in (ns): 100

Clock Off Timing in (ns): 5

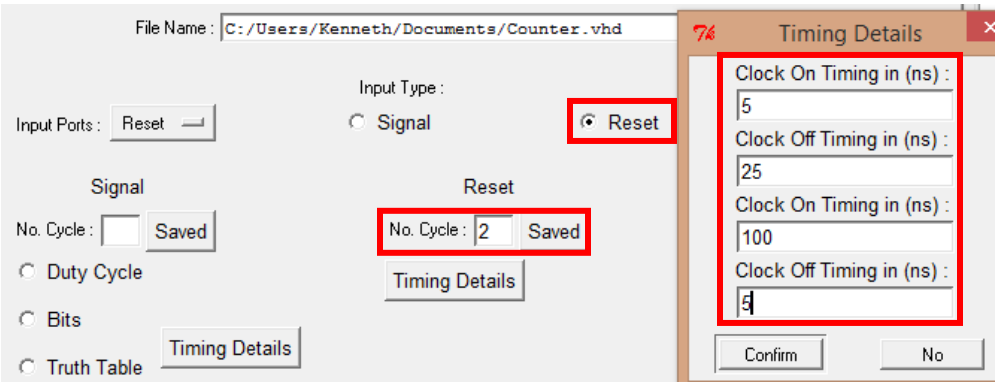


***Figure 67**: Parameters for Reset input port*

```vhdl
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- Declare module entity. Declare module inputs, inouts,
entity Counter_tb is
end Counter_tb;

-- Begin module architecture/code.
ARCHITECTURE behavior OF Counter_tb IS

COMPONENT Counter
 PORT(
Clk : in  STD_LOGIC;
   Reset : in  STD_LOGIC;
   clk_divider2 : out STD_LOGIC;
   clk_divider4 : out STD_LOGIC;
   countout : out  STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;

-- Inputs & Outputs
   signal tb_Clk :  STD_LOGIC;
   signal tb_Reset :  STD_LOGIC;
   signal tb_clk_divider2 : STD_LOGIC;
   signal tb_clk_divider4 : STD_LOGIC;
   signal tb_countout :  STD_LOGIC_VECTOR (3 downto 0);

-- *** Instantiate Constants ***
constant clk_PERIOD1: time := 20 ns;
```

```vhdl
-- Instantiate the UUT module.
uut : Counter
port map (
   Clk   => tb_Clk,
   Reset    => tb_Reset,
   clk_divider2   => tb_clk_divider2,
   clk_divider4   => tb_clk_divider4,
   countout    => tb_countout);


-- Generate necessary clocks.
Clk_process1: process
begin
   tb_Clk <= '1';
   wait for clk_PERIOD1 / 2;
   tb_Clk <= '0';
   wait for clk_PERIOD1 / 2;
end process;

-- Toggle the resets.
reset1: process
begin
   tb_Reset <= '1';
   wait for 5 ns;
   tb_Reset <= '0';
   wait for 25 ns;
   tb_Reset <= '1';
   wait for 100 ns;
   tb_Reset <= '0';
   wait for 5 ns;
   wait;
end process;

END behavior; -- architecture
```

***Figure 68**: Testbench generated for clock divider design*

```
Signal input 1 =: Clk : in  STD_LOGIC;

++++++++++++++++++++++++++++++++++++++++

-- Clock Input Type --

---------------------------------------

Clock Timing in (ns) : 20




****************************************
****************************************

Signal input 2 =: Reset : in  STD_LOGIC;

++++++++++++++++++++++++++++++++++++++++

-- Reset Input Type --

---------------------------------------

How many Cycle : 2
Clock On Timing in (ns) : 5
Clock Off Timing in (ns) : 25
Clock On Timing in (ns) : 100
Clock Off Timing in (ns) : 5
```

***Figure 69**: Text file for clock divider design*

The way how this clock divider work is that counter will count clock cycle at every rising edge of the clock when *reset* input port is 1. The counter has 4 bit which can count from “0000” to “1111” in binary. The counter will reset back to “0000” when *reset* input port is 0 or counter reach “1111” and re-start the counter. At every change in LSB (Least Significant Bit) for *countout(3 downto 0)* output port, clock will divide by 2 shown in **Figure 70**. At every change in second LSB for *countout(3 downto 0)* output port, clock will divide by 4 as shown **Figure 70**. This behavior is shown in simulation results in **Figure 71**. Furthermore, the reset timing is the same as the specified parameters highlighted in blue line in **Figure 71**.

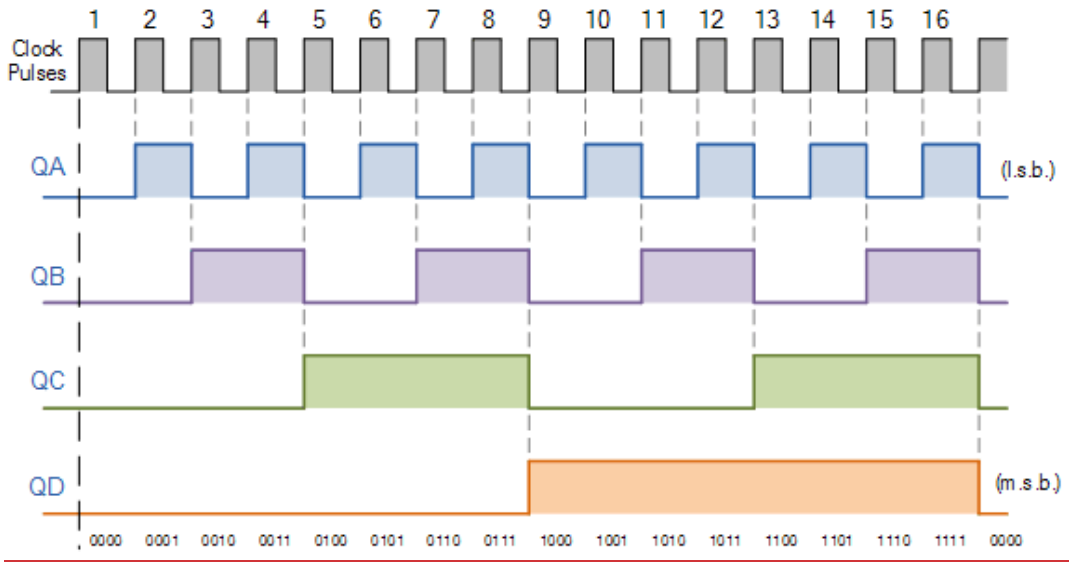


***Figure 70**: Clock divider expected outcome*

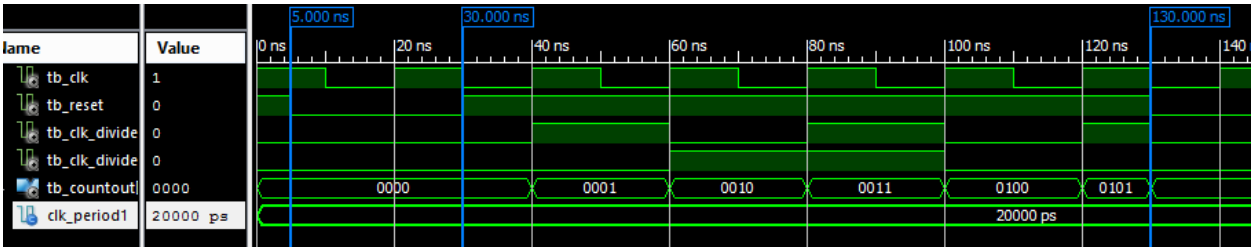


***Figure 71**: Simulation result for clock divider*

4.4 Finite-State Machine Implementation Design (Transmitter of RS232 design)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity Rs232Txd is
   Port ( Send : in  STD_LOGIC;
          DataIn : in  STD_LOGIC_VECTOR (7 downto 0);
          Reset : in  STD_LOGIC;
          Clock16x : in  STD_LOGIC;
          Txd : out  STD_LOGIC);
end Rs232Txd;

architecture Rs232Txd_Arch of Rs232Txd is

attribute enum_encoding: string;
-- state definitions
type stateType is (stIdle, stData, stStop, stTxdCompleted);
attribute enum_encoding of stateType: type is "00 01 11 10";
signal presState: stateType;
signal nextState: stateType;
signal iTxd, iSend1, iSend2, iReset, iClock1xEnable, iEnableTxdBuffer, iEnableShift: std_logic ;
signal iTxdBuffer: std_logic_vector (8 downto 0);
signal iClockDiv: std_logic_vector (3 downto 0);
signal iClock1x: std_logic;
signal iNoBitsSent: std_logic_vector (3 downto 0);

 begin
    process (Clock16x)
       begin
       if Clock16x'event and Clock16x = '1' then
          if Reset = '1' or iReset = '1' then
             iSend1 <= '0';
             iSend2 <= '0';
             iClock1xEnable <= '0';
             iEnableTxdBuffer <= '0';
             iClockDiv <= (others=>'0');

          else
             iSend1 <= Send;
             iSend2 <= iSend1;

             if iClock1xEnable = '1' then
                iClockDiv <= iClockDiv + '1';
             elsif iSend1 = '0' and iSend2 = '1' then
                iClock1xEnable <= '1';
                iEnableTxdBuffer <= '1';
             end if;
          end if;
       end if;
    end process;
 iClock1x <= iClockDiv(3);
```

***Figure 72**: VHDL code of Transmitter of RS232 design (RS232Txd.vhd) (part 1)*

```
   process (iClock1xEnable, iClock1x)
      begin
         if iClock1xEnable = '0' then
            iNoBitsSent <= (others=>'0');
            iTxdBuffer <= (others=>'0');
            iTxd <= '1';
            presState <= stIdle;

         elsif iClock1x'event and iClock1x = '1' then
            iNoBitsSent <= iNoBitsSent + '1';
            presState <= nextState;

            if iEnableTxdBuffer = '1' then
               iTxdBuffer <= DataIn & '0';

               if iEnableShift = '1' then
                  iTxd <= iTxdBuffer(0);
                  iTxdBuffer <=  '1' & iTxdBuffer(8 downto 1);
                  end if;
            end if;
         end if;
      end process;
Txd <= iTxd;


      process (presState, iClock1xEnable, iNoBitsSent)
      begin
      -- signal defaults
      iReset <= '0';
      iEnableShift <= '0';

      case presState is
         when stIdle =>
            if iClock1xEnable = '1' then
               nextState <= stData;
            else
               nextState <= stIdle;
            end if;

         when stData =>
            if iNoBitsSent = "1010" then
               iEnableShift <= '0';
               nextState <= stStop;
            else
               iEnableShift <= '1';
               nextState <= stData;
            end if;
         when stStop =>
            nextState <= stTxdCompleted;
         when stTxdCompleted =>
            iReset <= '1';
            nextState <= stIdle;
      end case;
      end process;

end Rs232Txd_Arch;
```

***Figure 73**: VHDL code of Transmitter of RS232 design (RS232Txd.vhd) (part 2)*

Figure 72 and **73** demonstrate code for transmitter of Rs232 design and RS232Txd.vhd is uploaded to the automatic testbench generator. Parameters are inputted for *Send, DataIn, Reset* and *Clock16x* input ports are as follow:

<u>*Send* input port</u>

Input type: Reset (Note: *Send* input port is an enable signal. Thus. Reset input type is used)

No. Cycle: 2

Timing Details:

Clock On Timing in (ns): 10

Clock Off Timing in (ns): 25

Clock On Timing in (ns): 100

Clock Off Timing in (ns): 1

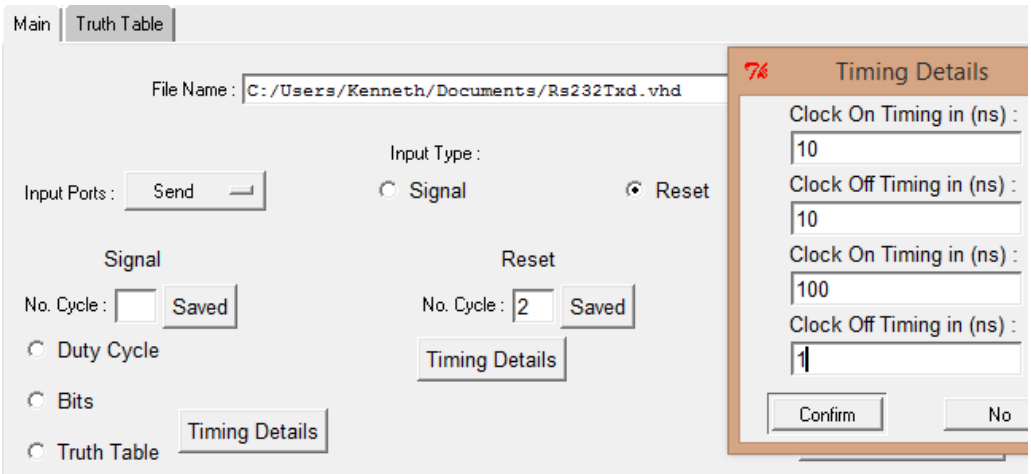


***Figure 74**: Parameters for enable signal and Rs232Txd.vhd*

<u>*DataIn* input port</u>

Input type: Signal

No. Cycle: 1

Signal type: Bits

Timing Details:

Bits: 10101010

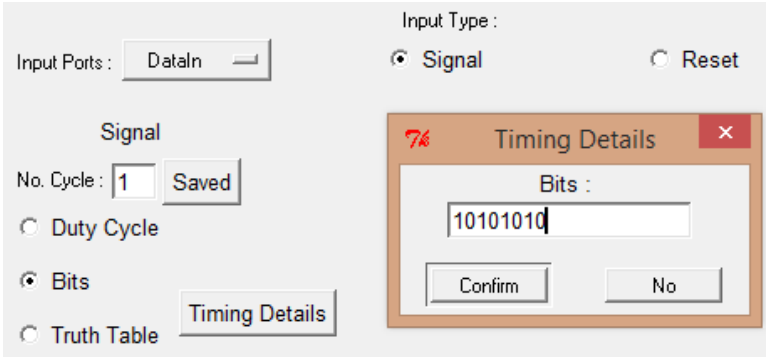


***Figure 75**: Parameter for bit signal input type and Rs232Txd.vhd*

<u>*Reset* input port</u>

Input type: Reset

No. Cycle: 1

Timing Details:

Clock On Timing in (ns): 5

Clock Off Timing in (ns): 10

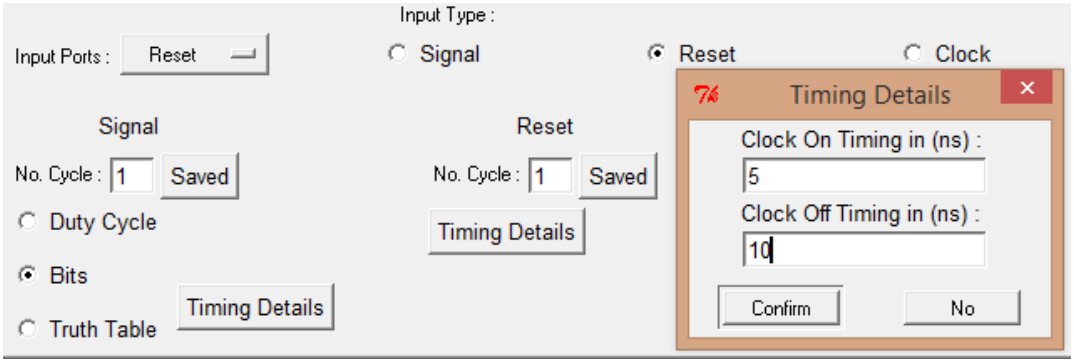


***Figure 76**: Parameter for reset input type and Rs232Txd.vhd*

<u>*Clock16x* input port</u>

Input type: Clock

Clock Timing (ns): 10

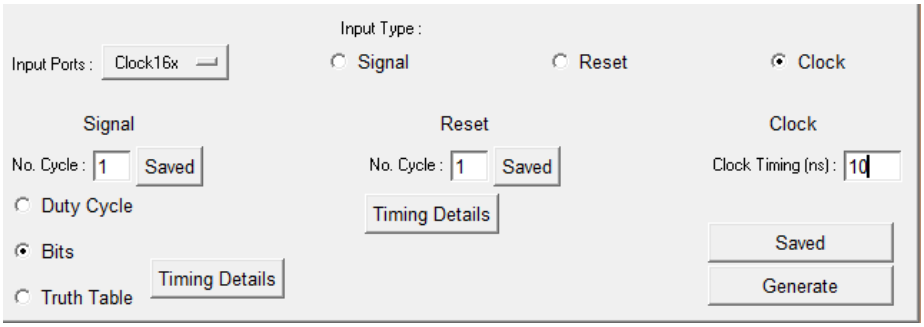


***Figure 77**: Parameter for clock input type and Rs232Txd.vhd*

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;


-- Declare module entity. Declare module inputs, inou
entity Rs232Txd_tb is
end Rs232Txd_tb;

-- Begin module architecture/code.
ARCHITECTURE behavior OF Rs232Txd_tb IS

COMPONENT Rs232Txd
 PORT(
Send : in  STD_LOGIC;
   DataIn : in  STD_LOGIC_VECTOR (7 downto 0);
   Reset : in  STD_LOGIC;
   Clock16x : in  STD_LOGIC;
   Txd : out  STD_LOGIC);

END COMPONENT;

-- Inputs & Outputs
   signal tb_Send :  STD_LOGIC;
   signal tb_DataIn :  STD_LOGIC_VECTOR (7 downto 0);
   signal tb_Reset :  STD_LOGIC;
   signal tb_Clock16x :  STD_LOGIC;
   signal tb_Txd :  STD_LOGIC;

-- *** Instantiate Constants ***
constant clk_PERIOD1: time := 10 ns;

BEGIN

-- Instantiate the UUT module.
uut : Rs232Txd
port map (
   Send  => tb_Send,
   DataIn   => tb_DataIn,
   Reset    => tb_Reset,
   Clock16x    => tb_Clock16x,
   Txd   => tb_Txd);

-- Generate necessary clocks.
Clk_process1: process
begin
   tb_Clock16x <= '1';
   wait for clk_PERIOD1 / 2;
   tb_Clock16x <= '0';
   wait for clk_PERIOD1 / 2;
end process;

-- Toggle the resets.
reset1: process
begin
   tb_Reset <= '1';
   wait for 5 ns;
   tb_Reset <= '0';
   wait for 10 ns;
   wait;
end process;

reset2: process
begin
   tb_Send <= '1';
   wait for 10 ns;
   tb_Send <= '0';
   wait for 10 ns;
   tb_Send <= '1';
   wait for 100 ns;
   tb_Send <= '0';
   wait for 1 ns;
   wait;
end process;

-- Insert Processes and code here.
-- Stimulus process1
DataIn: process
begin
tb_DataIn <= "10101010";
wait;
end process;

END behavior; -- architecture
```

***Figure 78**: Testbench generated for Rs232Txd.vhd*

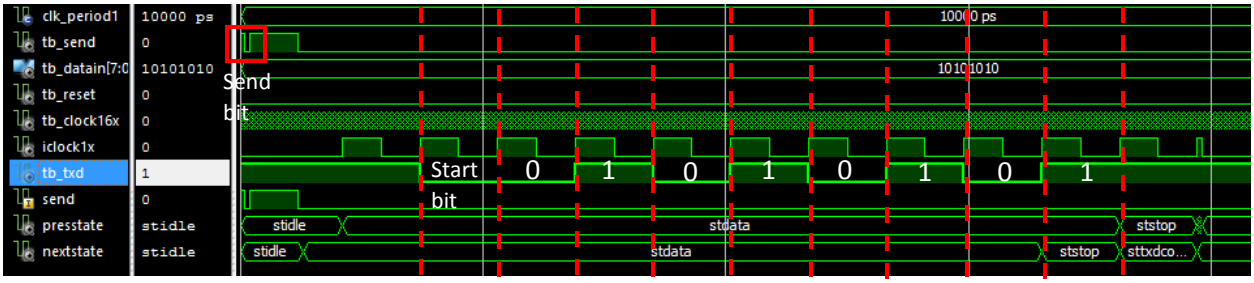


Figure 79: Simulation Results for transmitter of RS232 design

Data input of "10101010" is inputted to transmitter design. The design will store this data in a buffer and send it serially bit by bit after *send* signal transition from high to low. As shown in **Figure 79**, send bit is shown along with output of the design *Txd*. The design will output start bit and data from LSB to MSB. Hence, in simulation result in **Figure 79**, output of "001010101" is shown. This verified the testbench generated are indeed correct.

4.5 Limitation of automatic testbench generator

This testbench generator only allow generation of testbench for medium complexity design such as RS232. However, as design get more complicated, there will more input ports and more parameters are needed to fill. This might cause confusion for the users as the GUI are not intuitive enough to handle so many parameters. For example, if the number of cycles increase beyond 16, the GUI is not intuitive enough to show all the parameters on screen.

Furthermore, this testbench generator is not integrated with HDL compiler. Hence, it will not simulate the results automatically after the testbench is generated. This might be tedious for the user as they need to launch as HDL compiler, copy the design and the generated testbench to the HDL compiler. Other quality of life feature could be added to this testbench to further enhance the user experience such as:

- Button for changing timing input to another metric such as millisecond, microsecond instead of just nanosecond timing.
- Clock on and off timing could be toggle using the clock timing. This allow easy input on when to set the signal high or low.

Code coverage is not analyzed in the result session because Modelsim student edition is unavailable for student anymore due to some changes in US law [16]. Other tools are either unavailable for student or it is required to buy the software.

Chapter 5 - Conclusion

5.1 Summary

Design verification process is one of the most time-consuming processes in semi-conductor industry. As complexity of electronic design continue to increase, verification process become more time consuming. Sacrificing verification time to market the product as early as possible might lead to more bugs found in consumers' product. Furthermore, commercially available automatic testbench tools are either too costly like Testbencher from Synapticad or not available like StateCAD from Xilinx. Hence, automatic testbench generation was developed with intention to reduce the amount time and effort to generate testbench by reducing the amount of hard coding of testbench.

It can be observed in **chapter 4** that the results displayed effectiveness in generating testbench up to medium complexity. The automatic testbench tool is able to use the design file and user input parameters to generate testbench successfully. Furthermore, with addition of GUI, the tool is simple and user friendly which could potentially help people with little to no prior knowledge about VHDL to learn about VHDL. However, there is limitation to this tool mainly it can only generate testbench with medium complexity of HDL design like serial communication protocol design. For more complex testbench generation tool, Testbencher from Synapticad is recommended due to its wide range features to accommodate more complex design.

5.2 Future Improvement

There are plenty room for improvements in current version of automatic testbench generation tool. The following list some of suggestion could be implemented for improvement:

1) <u>Implement saving parameters on text file with GUI</u>: After inputting parameters in the GUI, text file is automatically generated with inputted parameters. GUI is able to read the text file and fill in the saved parameters to the GUI. This allow user to edit the parameters and generate testbench easily and conveniently. Current tool does not allow automatic filling of parameters in text file and GUI is not able to read the parameter file.
2) <u>Waveform Editor</u>: Editing the testbench using simulated waveform allow easy and fast generation of testbench. Drawn waveform are easier to edit on small testbenches compare to raw VHDL or Verilog code. This feature is available in WaveFormer from Synapticad.

3) Hierarchical Behavioral Test Generator (HBTG) algorithm: Current tool only allow generation of testbench for medium complexity design. With Hierarchical Behavioral Test Generator (HBTG) algorithm, testbench of sophisticated design can be generated using hierarchy information of VHDL design to generate test bench. This assist the verification engineer task in creating test stimulus to test the functionality of design. This algorithm is developed by Modeler's Assistant and implementation of HBTG is shown in this paper [10].

References

[1] F. Ferrari, "System-on-a-chip verification~methodology and techniques," *IEEE Circuits and Devices Magazine,* vol. 18, pp. 39-39, Nov 2002.

[2] C. Pixley, A. Chittor, F. Meyer, S. McMaster and D. Benua, "System-on-a-chip verification: methodology and techniques," in *Springer Science $ Business Media*, 2007.

[3] Syncad.com, "VHDL and Verilog Test Bench Synthesis," 2020. [Online]. Available: http://www.syncad.com/syn_tb_diff.htm#reactive.

[4] Xilinx, "StateBench and StateCAD webpage," 2016. [Online]. Available: https://www.xilinx.com/support/answers/32805.html.

[5] S. Kapoor, J. R. Armstrong and S. R. Rao, "An automatic test bench generation system," *Proceesings of VHDL International Users Forum,* pp. 8-17, 1994.

[6] P. Mishra and N. Dutt, ""Functional validation of programmable architectures,," *Digital System Design,* pp. 12-19, 2004.

[7] M. Law, "Moore's Law Document," 2005. [Online].

[8] M. A and C. O., "FUNCTIONAL VERIFICATION: APPROACHES AND CHALLENGES," *Latin American Applied Research,* pp. 65-65, 2007.

[9] Warren and H. A, "“Introduction: Special Issue on Microprocessor Verifications," *J. Formal Methods in,* pp. 135-137, 2002.

[10] S. Fine and Z. A, "“Coverage directed test generation," in *Proc. 40th Design Automation Conf.*, 2003.

[11] "SystemC Website," 2006. [Online]. Available: http://www.systemc.org.

[12] "11.1 ISE - I Do Not See The State Diagram Editor," Xilinx. 2020, [Online]. Available: https://www.xilinx.com/support/answers/32805.html.

[13] S. A. Murtza, O. Hasan and K. Saghar, ""VerTGen: An automatic verilog testbench," in *2016 International Conference on Emerging*, Islamabad, 2016.

[14] B. Singh, J. Wicks, P. Wright and J. R. Armstrong, ""The Modeler's Assistant: A CAD Tool for Behavioral Model Development," in *Proc. of CHDL '93*, 1993, pp. 347-354.

[15] jwwebbopen, "VHDLTools Github," 2017. [Online]. Available: https://github.com/jwwebbopen/VHDLTools.

[16] "ModelSim Student Edition Unavailable," Siemens, [Online]. Available: https://eda.sw.siemens.com/en-US/modelsim-student-edition-unavailable.

Appendix

Project Title		**HDL Test Bench Generator**
Project Code		NKT-MEng-20-02
Supervisor		Dr.T.Nandha Kumar
Moderator		Dr.Vimal
Project Description		Developing test-bench for HDL design is generally a time-consuming process and particularly test bench with good coverage is a challenging task. For any beginner/medium level HDL design engineers, availability of an automatic test bench generating tool is a boon particularly in reducing the design validation time significantly. But unfortunately, such tool is not available/ not provided by any tool vendor. So, this project aims in developing a tool that takes in the graphical representation of the inputs and provide the HDL test bench design file with the test coverage feature.
Project Objectives		1. To Understand different input signals required by a complex HDL design and to develop a graphical method to represent it. 2. Converting different graphical notations to the relevant HDL constructs. 3. Integrate all HDL constructs to develop an automatic HDL test bench generator. 4. Incorporate the test coverage method to the developed test bench and benchmark it against the commercial tool. 5. Validate the HDL test bench for different complex level of the HDL design.
Project Deliverables		1. An automatic HDL test bench generator for different complex level HDL designs using graphical input notations. 2. Accuracy validation of developed HDL test bench. 3. Coverage tool to automatically produce the test coverages. 4. Limitations of developed test bench generator.
Ratio of HW/SW/Research		**HW** / **SW** / **Research/Investigation/Design** 70 / 30

Lab Space Requirements		☐ Fixed Space ☒ Ad-hoc Space ☐ Others. Please state.
Software (This is software usually supported in the Software Lab)		Modelsim and high level programming language such as perl.

Perl Program for automatic testbench generator (GUI: Perl's command line)

```
use warnings;
use strict;
use File::Basename;
#my $file = $ARGV[0];
my $filename = 'C:\Users\Kenneth\Documents\Rs232Txd.vhd';

# check to see if the user entered a file name.
#die "syntax: [perl] vhdl_tb.pl existing_file.vhd\n" if ($file eq "");

# Read in the target file into an array of lines
open(inF, $filename) or dienice ("file open failed");
my @data = <inF>;
close(inF);

my ($file) = fileparse $filename;


# Make Date int MM/DD/YYYY
my $year      = 0;
my $month     = 0;
my $day       = 0;
($day, $month, $year) = (localtime)[3,4,5];


# Grab username from PC:
my $author= "$^O user";
if ($^O =~ /mswin/i)
{
  $author= $ENV{USERNAME} if defined $ENV{USERNAME};
}
else
{
  $author = getlogin();
}
```

```
#     Strip newlines
foreach my $i (@data) {
      chomp($i);
      $i =~ s/--.*//;        #strip any trailing -- comments
}

###################### Signal Extraction Algorithm
##########################
#     initialize counters
my $lines = scalar(@data);        #number of lines in file
my $line = 0;
my $entfound = -1;

#     find 'entity' left justified in file
for ($line = 0; $line < $lines; $line++) {
      if ($data[$line] =~ m/^entity/) {
            $entfound = $line;
            $line = $lines;  #break out of loop
      }
}

# find 'end $file', so that when we're searching for ports we don't
include local signals.
my $entendfound = 0;
$file =~ s/\.vhd$//;
for ($line = 0; $line < $lines; $line++) {
      if ($data[$line] =~ m/^end $file/) {
            $entendfound = $line;
            $line = $lines;  #break out of loop
      }
}


#     if we didn't find 'entity' then quit
if ($entfound == -1) {
      print("Unable to instantiate-no occurance of 'entity' left justified
in file.\n");
      exit;
}

#find opening paren for port list
$entendfound = $entendfound + 1;
my $pfound = -1;

# Remove entity line and port (
for ($line = $entfound; $line < $entendfound; $line++) { #start looking
from where we found module
      $data[$line] =~ s/--.*//;         #strip any trailing --comment

        if ($data[$line] =~ m/\(/) {          # 0x28 is '('
            $pfound = $line;
                $data[$line] =~ s/.*\x28//;   # remove "port ("
            print "$data[$line]\n";
```

```
            $line = $entendfound;  # break out of loop
      }
}

#     if couldn't find '(', exit
if ($pfound == -1) {
      print("Unable to instantiate-no occurance of '(' after module
keyword.\n");
      exit;
}

#collect port names
my @ports;

# print a(or b) : in  STD_LOGIC; sum(or carry) : out  STD_LOGIC;
for ($line = $pfound; $line < $entendfound; $line++) {
      $data[$line] =~ s/--.*//;         #strip any trailing --comment
      next if not $data[$line] =~ /:.*/;
      $data[$line] =~ s/^\s+|\s+$//; # trim right and left space
        push @ports , $data[$line];
      #print "$data[$line]";
}

my @portlines1;
my @portsInOut = ();
@portsInOut = @ports;
my $count_ports = 0;
foreach my $i (@portsInOut) {
      $i =~ s/ in //;
      $i =~ s/ out //;

      if( $count_ports == $#portsInOut ) {
            chop($i);
            chop($i);
            $i =~ s|/?$|;|;
      }

      push @portlines1, "\tsignal tb_$i";
      $count_ports++;
}

my $out3 = join "\n", @portlines1;

#print out instantiation
print ("component $file\n"); #print first line
print (" port (\n");         #print second line
my $out= join "\n", @ports;
print ("$out\t\n\t\nend component;\n"); #print ports and last couple of
lines

# Create the module instantiation.  A future enhancement would be to call
the script vhdl_inst.pl instead.
my @ports2;
my @inOut;
```

```
for ($line = $pfound; $line < $entendfound; $line++) {
      $data[$line] =~ s/--.*//;    # strip any trailing --comment

      #   next if not $data[$line] =~ /:.*;/;
      $data[$line] =~ s/^/ /mg;    # add space at the beginning of line

      if ($data[$line] =~ /\s+(\w+)\s+:/)
      {
            push @ports2, $1;
            print "\n$1";
      }

      if ($data[$line] =~ /\s+(\w+)\s+STD_LOGIC/)
      {
            push @inOut, $1;
            print "\n$1";
      }
}


my @portlines2;
foreach my $i (@ports2) {
  push @portlines2, "$i \t=> tb_$i";
}

my $out2= join ",\n\t", @portlines2;


# check to make sure that the file doesn't exist.
my $new_file = join "_", $file, "tb";
my $new_file_vhd = join ".",$new_file,"vhd";
#die "Oops! A file called '$new_file.vhd' already exists.\n" if -e
$new_file_vhd;


############## User provided parameters (Non-GUI version)
####################
############## Perl's command line version
##################################
my $in = 1;
my $out = 1;
my $count = 0;
my @input_type;

my $no_signal_input = 0;
my $no_truth_table = 0;
my @signal_input;
my @signal_input_timing;
my @signal_input_duty_cycle;
my @signal_cycle_DC;
my @signal_cycle_bits;
my @signal_input_type;
my @signal_input_bits;
my @signal_input_delay;
```

```
my @signal_truthTable;

my $no_reset = 0;
my @reset_on;
my @reset_off;
my @reset_port;
my @reset_cycle;

my $no_clock = 0;
my @clock;
my @clock_port;

foreach my $i (@inOut) {
      if ($i =~ "in")
      {
            print "\n Signals input $in: tb_$ports2[$count]";

            print "\nType of Input (Signal/Reset/Clock): ";
            $input_type[$in - 1] = <STDIN>;
            chomp $input_type[$in - 1];

            if ($input_type[$in - 1] =~ "Signal")
            {
                  print "\n Signal input type
(Truth_table/Duty_Cycle/Bits): ";
                  $signal_input_type[$no_signal_input] = <STDIN>;
                  chomp $signal_input_type[$no_signal_input];

                  $signal_input[$no_signal_input] = $ports2[$count];

                  if ($signal_input_type[$no_signal_input] =~ "Duty_Cycle")
                  {
                        print "\nHow many cycle: ";
                        $signal_cycle_DC[$no_signal_input] = <STDIN>;
                        chomp $signal_cycle_DC[$no_signal_input];

                        print "\n Signals input $in timing in (ns): ";
                        $signal_input_timing[$no_signal_input] = <STDIN>;
                        chomp $signal_input_timing[$no_signal_input];

                        print "\n Signals input $in duty cycle (out of
100): ";
                        $signal_input_duty_cycle[$no_signal_input] =
<STDIN>;
                        chomp $signal_input_duty_cycle[$no_signal_input];
                  }
                  elsif ($signal_input_type[$no_signal_input] =~ "Bits")
                  {
                        print "\nHow many cycle: ";
                        $signal_cycle_bits[$no_signal_input] = <STDIN>;
                        chomp $signal_cycle_bits[$no_signal_input];

                        for (my $j = 0; $j <=
$signal_cycle_bits[$no_signal_input] - 1; $j++)
```

```
                        {
                              if ($j > 0)
                              {
                                    print "\n Delay Timing: ";

      $signal_input_delay[$no_signal_input][$j] = <STDIN>;
                                    chomp
$signal_input_delay[$no_signal_input][$j];
                              }

                              print "\n Bits: ";
                              $signal_input_bits[$no_signal_input][$j] =
<STDIN>;
                              chomp
$signal_input_bits[$no_signal_input][$j];

                        }
                  }
                  elsif ($signal_input_type[$no_signal_input] =~
"Truth_table")
                  {
                        $signal_truthTable[$no_truth_table] =
$signal_input[$no_signal_input];

                        $no_truth_table = $no_truth_table + 1;
                  }
                  $no_signal_input = $no_signal_input + 1;
            }

            elsif ($input_type[$in - 1] =~ "Reset")
            {
                  print "\n Reset input type";
                  $reset_port[$no_reset] = $ports2[$count];

                  print "\nHow many cycle: ";
                  $reset_cycle[$no_reset] = <STDIN>;
                  chomp $reset_cycle[$no_reset];

                  for (my $j = 0; $j <= $reset_cycle[$no_reset] - 1; $j++)
                  {
                        print "\n Clock On Timing in (ns): ";
                        $reset_on[$no_reset][$j] = <STDIN>;
                        chomp $reset_on[$no_reset][$j];

                        print "\n Clock Off Timing in (ns): ";
                        $reset_off[$no_reset][$j] = <STDIN>;
                        chomp $reset_off[$no_reset][$j];
                  }

                  $no_reset = $no_reset + 1;
            }
            elsif ($input_type[$in - 1] =~ "Clock")
            {
                  print "\n Clock input type";
```

```
                $clock_port[$no_clock] = $ports2[$count];

                print "\n Clock Timing in (ns): ";
                $clock[$no_clock] = <STDIN>;
                chomp $clock[$no_clock];

                $no_clock = $no_clock + 1;
            }

            $in = $in + 1;
        }
        elsif ($i =~ "out")
        {
            print "\n Signals output $out: tb_$ports2[$count]";
            $out = $out + 1;
        }
        else
        {
            print "Error in file";
        }
$count = $count + 1;
}


my @input_truthTable;
my @input_delay;
my $cycle_truthTable;

if ($no_truth_table > 0)
{
        print "\n\nHow many cycle for Truth Table: ";
        $cycle_truthTable = <STDIN>;
        chomp $cycle_truthTable;

        for (my $i=0 ; $i <= $cycle_truthTable - 1 ; $i++)
        {
            if ($i > 0)
            {
                print "\n Delay Timing: ";
                $input_delay[$i] = <STDIN>;
                chomp $input_delay[$i];
            }

            for (my $j=0 ; $j <= $no_truth_table - 1 ; $j++)
            {
                print "\ntb_$signal_truthTable[$j] input (1/0)? :";
                $input_truthTable[$i][$j] = <STDIN>;
                chomp $input_truthTable[$i][$j];
            }
        }
}
```

```
############################ Testbench Generator
#############################################################
open(my $inF, ">", $new_file_vhd);

# Print title
printf($inF "-------------------------------------------------------------
------------------\n");
printf($inF "--
Revision: 1.1 \n");
printf($inF "--
Date: %02d/%02d/%04d \n", $month+1, $day, $year+1900);
printf($inF "-------------------------------------------------------------
------------------\n");
#printf($inF "--\t\t\t\t My Company Confidential Copyright © %04d My
Company, Inc.\n", $year+1900);
printf($inF "--\n");
printf($inF "--   File name :  $file.vhd\n");
printf($inF "--   Title     :  Automatic HDL Testbench Generation\n");
printf($inF "--   Module    :  $file\n");
printf($inF "--   Author    :  $author\n");
printf($inF "--   Purpose   :  Year 4 FYP\n");
printf($inF "--\n");
#printf($inF "--   Roadmap   :\n");
printf($inF "-------------------------------------------------------------
------------------\n");
#printf($inF "--   Modification History :\n");
#printf($inF "--\tDate\t\tAuthor\t\tRevision\tComments\n");
#printf($inF "--\t%02d/%02d/%04d\t$author\tRev A\t\tCreation\n", $month+1,
$day, $year+1900);
#printf($inF "-------------------------------------------------------------
-------------------\n");
printf($inF "\n");

# Library
printf($inF "Library IEEE;\n");
printf($inF "use IEEE.STD_LOGIC_1164.all;\n");
printf($inF "use IEEE.std_logic_unsigned.all;\n");
printf($inF "use IEEE.std_logic_arith.all;\n");
printf($inF "use IEEE.Numeric_STD.all;\n");
#printf($inF "\n");
#printf($inF "library work;\n");
#my $new_text = join "_", $file, "pkgs.all";
#printf($inF "use work.$new_text;\n");
printf($inF "\n");
printf($inF "\n");

# Entity
printf($inF "-- Declare module entity. Declare module inputs, inouts, and
outputs.\n");
printf($inF "entity $new_file is\n");
printf($inF "end $new_file;\n");
printf($inF "\n");

# Architecture
```

```
printf($inF "-- Begin module architecture/code.\n");
printf($inF "ARCHITECTURE behavior OF $new_file IS\n");
printf($inF "\n");

# Component
print ($inF "COMPONENT $file\n"); #print first line
print ($inF " PORT(\n");          #print second line
my $out= join "\n\t", @ports;
print ($inF "$out\t\n\t\nEND COMPONENT;\n"); #print ports and last couple
of lines
print ($inF "\n");

# UUT Port Signals
#printf($inF "-- UUT Port Signals.\n");
#printf($inF "$out;\n"); #print ports and last couple of lines
#printf($inF "\n");

# Input and Outputs
printf($inF "-- Inputs & Outputs\n");
printf($inF "$out3\n");

# Clock Constant
printf($inF "-- *** Instantiate Constants ***\n");
if ($no_clock != 0)
{
      # Clock
      for (my $i=1 ; $i <= $no_clock ; $i++)
      {
            printf($inF "constant clk_PERIOD$i: time := $clock[$i-1]
ns;\n");
            printf($inF "\n");
      }
}

# Instantiate UUT
printf($inF "BEGIN\n");
printf($inF "\n");
printf($inF "-- Instantiate the UUT module.\n");
printf($inF "uut : $file\nport map (");       #print first line
printf($inF "\n\t$out2);\n\n");
printf($inF "\n");

# Generate Clock
if ($no_clock != 0)
{
printf($inF "-- Generate necessary clocks.\n");

      for (my $i=1 ; $i <= $no_clock ; $i++)
      {
            printf($inF "Clk_process$i: process\n");
            printf($inF "begin\n");
            printf($inF "\ttb_$clock_port[$i-1] <= '1';\n");
            printf($inF "\twait for clk_PERIOD$i / 2;\n");
            printf($inF "\ttb_$clock_port[$i-1] <= '0';\n");
```

```
            printf($inF "\twait for clk_PERIOD$i / 2;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
      }
}

# Reset
if ($no_reset != 0)
{
printf($inF "-- Toggle the resets.\n");

      for (my $i=1 ; $i <= $no_reset ; $i++)
      {
            printf($inF "reset$i: process\n");
            printf($inF "begin\n");

            for (my $j=0 ; $j <= $reset_cycle[$i-1] - 1 ; $j++)
            {
                  printf($inF "\ttb_$reset_port[$i-1] <= '1';\n");
                  printf($inF "\twait for $reset_on[$i-1][$j] ns;\n");
                  printf($inF "\ttb_$reset_port[$i-1] <= '0';\n");
                  printf($inF "\twait for $reset_off[$i-1][$j] ns;\n");
            }
            printf($inF "\ttb_$reset_port[$i-1] <= '1';\n");

            printf($inF "\twait;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
      }
}

# Stimulus process
if ($no_signal_input != 0)
{
printf($inF "-- Insert Processes and code here.\n");

      for (my $i=1 ; $i <= $no_signal_input ; $i++)
      {
            my $signal_match = '0';
            for (my $k=0 ; $k <= $no_truth_table - 1 ; $k++)
            {
                  if ($signal_input[$i-1] =~ $signal_truthTable[$k])
                  {
                        $signal_match = '1';
                  }
            }

            if ($signal_match =~ '0')
            {
                  printf($inF "-- Stimulus process$i\n");
                  printf($inF "$signal_input[$i-1]: process\n");
                  printf($inF "begin\n");

                  if ($signal_input_type[$i-1] =~ "Duty_Cycle")
```

```
                {
                        for (my $j=0 ; $j <= $signal_cycle_DC[$i-1] - 1 ;
$j++)
                        {
                                printf($inF "tb_$signal_input[$i-1] <=
'1';\n");
                                my $delay_on = ($signal_input_duty_cycle[$i-
1]/100)*$signal_input_timing[$i-1];
                                printf($inF "wait for $delay_on ns;\n");

                                printf($inF "tb_$signal_input[$i-1] <=
'0';\n");
                                my $delay_off = ((100 -
$signal_input_duty_cycle[$i-1])/100)*$signal_input_timing[$i-1];
                                printf($inF "wait for $delay_off ns;\n");
                        }
                }
                elsif ($signal_input_type[$i-1] =~ "Bits")
                {
                        for (my $j=0 ; $j <= $signal_cycle_bits[$i-1] - 1 ;
$j++)
                        {
                                if ($j > 0)
                                {
                                printf($inF "wait for $signal_input_delay[$i-
1][$j] ns;\n");
                                }

                                printf($inF "tb_$signal_input[$i-1] <=
\"$signal_input_bits[$i-1][$j]\";\n");
                        }
                }

                printf($inF "wait;\n");
                printf($inF "end process;\n");
                printf($inF "\n");
            }
        }

        if ($no_truth_table > 0)
        {
            printf($inF "-- Stimulus process\n");
            printf($inF "stim_proc: process\n");
            printf($inF "begin\n");

            for (my $i=0 ; $i <= $cycle_truthTable - 1 ; $i++)
            {
                if ($i > 0)
                {
                        printf($inF "wait for $input_delay[$i] ns;\n\n");
                }

                for (my $j=0 ; $j <= $no_truth_table - 1 ; $j++)
                {
```

```
                        printf($inF "tb_$signal_truthTable[$j] <=
'$input_truthTable[$i][$j]';\n");
                  }
            }
            printf($inF "wait;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
      }
}

printf($inF "END behavior; -- architecture\n");
printf($inF "\n");
printf($inF "\n");

#my $new_text2 = join "_", $new_file, "cfg";
#printf($inF "configuration $new_text2 of $new_file is\n");
#printf($inF "for behavior\n");
#printf($inF "end for;\n");
#printf($inF "end $new_text2;\n");

close(inF);

print("\nThe script has finished successfully! You can now use the file
$new_file_vhd.\n\n");


exit;


#-------------------------------------------------------------------------
-----
# Generic Error and Exit routine
#-------------------------------------------------------------------------
-----

sub dienice {
      my($errmsg) = @_;
      print"$errmsg\n";
      exit;
}
```

Perl Program for automatic testbench generator (GUI)

```
#!/usr/bin/perl -w
use strict;
use warnings;
use 5.010;

use Tk;
use Tk::Dialog;
use Tk::DialogBox;
use File::Basename;
use Tk::NoteBook;
use Tk::TableMatrix;
my $current_file=();

our $top = MainWindow->new;
$top->configure(-title=> "Automatic Testbench Generator");
$top->geometry("650x300+0+0");
my $nb = $top->NoteBook( )->pack(-expand => 1, -fill => 'both');
my $mw = $nb->add('page1', -label => 'Main');
my $mw2 = $nb->add('page2', -label => 'Truth Table');




######################### Ports
##################################################

my $input_ports = "Ports";

$mw->Label(-text => 'Input Ports :')->place(-x => 10, -y => 80);
my $fr1 = $mw->Frame()->place(-x => 75, -y => 75);
my $om1 = $fr1->Optionmenu(-variable => \$input_ports)->pack();
#print "$om1";




#################### File Name and Upload Button
##############################

$mw->Button(
    -text    => 'Upload',
    -command => \&open_vhdl,
)->place(-x => 510, -y => 13);

$mw->Label(-text => 'File Name :')->place(-x => 90, -y => 15);
```

```
my $text = $mw->Text(qw/-width 50 -height 1/) ->place(-x => 150, -y =>
15);

my ($file);
my $new_file_vhd;
my $year;
my $month;
my $day;
my $author;
my $new_file;
my @ports;
my $out3;
my $out2;

sub open_vhdl
{
      my @types =
                  (["VHDL files", [qw/.vhd /]],
            ["All files",        '*'],
            );
      $current_file = $mw->getOpenFile(-filetypes => \@types);

      $text->delete('1.0', 'end');
      $text->insert('end', "$current_file");

      open(inF, $current_file) or dienice ("file open failed");
      my @data = <inF>;
      close(inF);

      ($file) = fileparse $current_file;

      print "filename: $file\n";

      # Make Date int MM/DD/YYYY
      my $year      = 0;
      my $month     = 0;
      my $day       = 0;
      ($day, $month, $year) = (localtime)[3,4,5];


      # Grab username from PC:
      $author = "$^O user";
      if ($^O =~ /mswin/i)
      {
      $author= $ENV{USERNAME} if defined $ENV{USERNAME};
      }
      else
      {
      $author = getlogin();
      }


      #     Strip newlines
      foreach my $i (@data) {
```

```
            chomp($i);
            $i =~ s/--.*//;        #strip any trailing -- comments
      }

      #     initialize counters
      my $lines = scalar(@data);        #number of lines in file
      my $line = 0;
      my $entfound = -1;

      #     find 'entity' left justified in file
      for ($line = 0; $line < $lines; $line++) {
            if ($data[$line] =~ m/^entity/) {
                  $entfound = $line;
                  $line = $lines;  #break out of loop
            }
      }

      # find 'end $file', so that when we're searching for ports we don't
include local signals.
      my $entendfound = 0;
      $file =~ s/\.vhd$//;
      for ($line = 0; $line < $lines; $line++) {
            if ($data[$line] =~ m/^end $file/) {
                  $entendfound = $line;
                  $line = $lines;  #break out of loop
            }
      }


      #     if we didn't find 'entity' then quit
      if ($entfound == -1) {
            print("Unable to instantiate-no occurance of 'entity' left
justified in file.\n");
            exit;
      }

      #find opening paren for port list
      $entendfound = $entendfound + 1;
      my $pfound = -1;

      # Remove entity line and port (
      for ($line = $entfound; $line < $entendfound; $line++) { #start
looking from where we found module
            $data[$line] =~ s/--.*//;         #strip any trailing --
comment

            if ($data[$line] =~ m/\(/) {      # 0x28 is '('
                  $pfound = $line;
                  $data[$line] =~ s/.*\x28//;  # remove "port ("
                  print "$data[$line]\n";
                  $line = $entendfound;  # break out of loop
            }
      }
```

```
	#	if couldn't find '(', exit
	if ($pfound == -1) {
		print("Unable to instantiate-no occurance of '(' after module
keyword.\n");
		exit;
	}

	@ports = ();
	# print a(or b) : in  STD_LOGIC; sum(or carry) : out  STD_LOGIC;
	for ($line = $pfound; $line < $entendfound; $line++) {
		$data[$line] =~ s/--.*//;		#strip any trailing --
comment
		next if not $data[$line] =~ /:.*/;
		$data[$line] =~ s/^\s+|\s+$//; # trim right and left space
		push @ports , $data[$line];
		#print "$data[$line]";
	}

	my @portsInOut = ();
	@portsInOut = @ports;
	my @portlines1;
	my $count_ports = 0;
	foreach my $i (@portsInOut) {
		$i =~ s/ in//;
		$i =~ s/ out//;

		if( $count_ports == $#portsInOut ) {
			chop($i);
			chop($i);
			$i =~ s|/?$|;|;
		}
	push @portlines1, "\tsignal tb_$i";
	$count_ports++;
	}

	$out3 = ();
	$out3 = join "\n", @portlines1;

	#print out instantiation
	#print ("component $file\n");	#print first line
	#print " port (\n";		#print second line
	#my $out= join "\n", @ports;
	#print ("$out\t\n\t\nend component;\n"); #print ports and last
couple of lines

	# Create the module instantiation.  A future enhancement would be to
call the script vhdl_inst.pl instead.
	my @ports2;
	my @inOut;
	for ($line = $pfound; $line < $entendfound; $line++) {
		$data[$line] =~ s/--.*//;	# strip any trailing --comment

		#	next if not $data[$line] =~ /:.*;/;
```

```
            $data[$line] =~ s/^/ /mg;    # add space at the beginning of
line

            if ($data[$line] =~ /\s+(\w+)\s+:/)
            {
                  push @ports2, $1;
                  #print "\n$1";

            }

            if ($data[$line] =~ /\s+(\w+)\s+STD_LOGIC/)
            {
                  push @inOut, $1;
                  #print "\n$1";

            }
      }


      my @portlines2;
      foreach my $i (@ports2) {
       push @portlines2, "$i \t=> tb_$i";
      }

      $out2 = ();
      $out2 = join ",\n\t", @portlines2;

      # check to make sure that the file doesn't exist.
      $new_file = join "_", $file, "tb";
      $new_file_vhd = join ".",$new_file,"vhd";
      #die "Oops! A file called '$new_file.vhd' already exists.\n" if -e
$new_file_vhd;

      my $in_count = 1;
      my $out_count = 1;
      my $count = 0;
      my @sig_in;

      foreach my $i (@inOut) {
            if ($i =~ "in")
            {
                  print "\n Signals input $in_count: tb_$ports2[$count]";
                  $sig_in[$in_count - 1] = $ports2[$count];
                  $in_count = $in_count + 1;
            }
            elsif ($i =~ "out")
            {
                  print "\n Signals output $out_count: tb_$ports2[$count]";
                  $out_count = $out_count + 1;
            }
      $count = $count + 1;
      }
      print "\n";
```

```
      $fr1 = $mw->Frame()->place(-x => 75, -y => 75);
      $om1 = $fr1->Optionmenu(
      -variable => \$input_ports,
      -options => \@sig_in,
      -command  => \&option_menu_changed,
      )->pack;
}

sub option_menu_changed {
    #say "\nInput Ports: $input_ports"
}




######################### Signal Checkbox
##################################################

$mw->Label(-text => 'Input Type :')->place(-x => 240, -y => 55);

my @types1 = ('Clock','Reset','Signal');
my $input_type;
for my $itype1 (@types1) {
    my $cb = $mw->Radiobutton(
        -text     => $itype1,
        -variable => \$input_type,
        -value    => $itype1,
        -font     => ['fixed', 10],
      -command  => \&do_on_select1,
    );
    $cb->pack(-side => 'right',-anchor => 'ne',-padx => 45, -pady => 75);
}

sub do_on_select1 {
      #say "\nInput Type : $input_type";
}



################### Signal Type (DC,Bits,Truth_Table)
#####################################

$mw->Label(-text => 'Signal', -font => ['bold',10])->place(-x => 60, -y =>
120);
### No. of Cycle ###
$mw->Label(-text => 'No. Cycle :')->place(-x => 10, -y => 150);
my $entry = $mw->Entry(
    -font => ['fixed', 10],
    -width   => 3,
);
$entry->place(-x => 70, -y => 150);

my $btn = $mw->Button(
```

```
    -text    => 'Saved',
    -font    => ['fixed', 10],
    -command => \&do_on_clicked_S,
);
$btn->place(-x => 100, -y => 147);

sub do_on_clicked_S {
    #print("\nNo. Signal Cycle: ",$entry->get);

    return $entry->get;
}

$mw->Label(-text => 'Signal Type :')->place(-x => 10, -y => 185);




################# Duty Cycle, Bits, Truth Table
##########################################

my @types2 = ('Truth Table','Bits','Duty Cycle');
my $signal_type;
for my $itype2 (@types2) {
    my $cb = $mw->Radiobutton(
        -text     => $itype2,
        -variable => \$signal_type,
        -value    => $itype2,
        -font     => ['fixed', 10],
      -command  => \&do_on_select2,
   );
    $cb->pack(-side => 'bottom',-anchor => 'w',-pady => 3,-padx => 10);
#place(-x => 10, -y => 210);
}


my @entry_Bit_Bits;
my @entry_Bit_Delay;
my $entryDC1;
my $entryDC2;

sub do_on_select2 {
      #print "\nSignal Type: $signal_type\n";

######################## Duty Cycle
#######################################
      if ($signal_type =~ "Duty Cycle"){

            my $btn1 = $mw->Button(
                  -text    => 'Timing Details',
                  -font    => ['fixed', 10],
                  -command => \&do_on_click1,
            );
            $btn1->place(-x => 110, -y => 225);
```

```
            sub do_on_click1 {
                  my $dialog1 = $mw->DialogBox(
                        -title   => 'Timing Details',
                        -popover => $mw,
                        -buttons => ['Confirm', 'No'],
                  );

                  $dialog1->add("Label", -text => 'Signal Input Timing
Details in (ns) : ', -font => ['fixed', 10])->pack();
                  $entryDC1 = $dialog1->add("Entry", -font => ['fixed',
10],)->pack();

                  $dialog1->add("Label", -text => 'Signal Input Duty Cycle
(out of 100) : ', -font => ['fixed', 10])->pack();
                  $entryDC2 = $dialog1->add("Entry", -font => ['fixed',
10],)->pack();
                  my $res1 = $dialog1->Show;

                  #if ($res1) {
                        #say "$res1";
                        #say $entryDC1->get;
                        #say $entryDC2->get;
                  #}
            }
      }
############################## Bits
########################################
      elsif ($signal_type =~ "Bits"){

            my $btn1 = $mw->Button(
                  -text    => 'Timing Details',
                  -font    => ['fixed', 10],
                  -command => \&do_on_click2,
            );
            $btn1->place(-x => 110, -y => 225);

            sub do_on_click2 {
                  my $dialog2 = $mw->Dialog(
                        -title   => 'Timing Details',
                        -popover => $mw,
                        -buttons => ['Confirm', 'No'],
                   );

                  my $cycle_ports = do_on_clicked_S();

                  for (my $i = 0 ; $i <= $cycle_ports - 1 ; $i++) {
                        $dialog2->add("Label", -text => 'Bits : ', -font =>
['fixed', 10])->pack();
                        $entry_Bit_Bits[$i] = $dialog2->add("Entry", -font
=> ['fixed', 10],)->pack();
                        if ($i < $cycle_ports - 1) {
                              $dialog2->add("Label", -text => 'Delay
Timing : ', -font => ['fixed', 10])->pack();
```

```
                              $entry_Bit_Delay[$i] = $dialog2->add("Entry",
-font => ['fixed', 10],)->pack();
                        }
                  }

                  my $res2 = $dialog2->Show;

                  #for (my $i = 0 ; $i <= $cycle_ports - 1 ; $i++) {
                        #say $entry_Bit_Bits[$i]->get;

                  #if ($i < $cycle_ports - 1) {
                        #say $entry_Bit_Delay[$i]->get; }
                  #}
            }
      }
######################## Truth Table
########################################
      elsif ($signal_type =~ "Truth Table"){
            my $btn1 = $mw->Button(
                  -text    => 'Timing Details',
                  -font    => ['fixed', 10],
            );
            $btn1->place(-x => 110, -y => 225);
      }
}

my $btn1 = $mw->Button(
    -text    => 'Timing Details',
    -font    => ['fixed', 10],
);
$btn1->place(-x => 110, -y => 225);




######################## Reset
#############################################

$mw->Label(-text => 'Reset', -font => ['bold',10])->place(-x => 310, -y =>
120);
### No. of Cycle ###
$mw->Label(-text => 'No. Cycle :')->place(-x => 260, -y => 150);
my $entry_R = $mw->Entry(
    -font => ['fixed', 10],
    -width   => 3,
);
$entry_R->place(-x => 320, -y => 150);

my $btn_R = $mw->Button(
    -text    => 'Saved',
    -font    => ['fixed', 10],
    -command => \&do_on_clicked_R,
);
```

```
$btn_R->place(-x => 350, -y => 147);

sub do_on_clicked_R {

    #print("\nNo. Reset Cycle: ",$entry_R->get);

    return $entry_R->get;
}


my $btn4 = $mw->Button(
    -text    => 'Timing Details',
    -font    => ['fixed', 10],
    -command => \&do_on_click4,
);
$btn4->place(-x => 260, -y => 180);


my @entry_Reset_Bits;
my @entry_Reset_Delay;

sub do_on_click4 {
    my $dialog3 = $mw->Dialog(
        -title   => 'Timing Details',
        -popover => $mw,
        -buttons => ['Confirm', 'No'],
    );

    my $reset_c = do_on_clicked_R();

    for (my $i = 0 ; $i <= $reset_c - 1 ; $i++) {
      $dialog3->add("Label", -text => 'Clock On Timing in (ns) : ', -font
=> ['fixed', 10])->pack();
      $entry_Reset_Bits[$i] = $dialog3->add("Entry", -font => ['fixed',
10],)->pack();

      $dialog3->add("Label", -text => 'Clock Off Timing in (ns) : ', -font
=> ['fixed', 10])->pack();
      $entry_Reset_Delay[$i] = $dialog3->add("Entry", -font => ['fixed',
10],)->pack();

    }

    my $res3 = $dialog3->Show;

   #for (my $i = 0 ; $i <= $reset_c - 1 ; $i++) {
      #say $entry_Reset_Bits[$i]->get;
      #say $entry_Reset_Delay[$i]->get;
    #}
}
```

```
######################## Clock
##############################################

$mw->Label(-text => 'Clock', -font => ['bold',10])->place(-x => 540, -y =>
120);
$mw->Label(-text => 'Clock Timing (ns) :')->place(-x => 500, -y => 150);
#my $text5 = $mw->Text(qw/-width 5 -height 1/) ->place(-x => 595, -y =>
150);
my $entry_C = $mw->Entry(
    -font => ['fixed', 10],
    -width   => 5,
);
$entry_C->place(-x => 595, -y => 150);




######################## Saved
##############################################

$mw->Button(
    -text    => 'Saved',
    -font    => ['fixed', 10],
    -width   => 15,
    -command => \&saved,
)->place(-x => 500, -y => 200);



my $no_signal_input = 0;
my @signal_input;
my $no_signal_cycle = 0;
my @signal_cycle;
my @signal_input_timing;
my @signal_input_type;
my @signal_input_duty_cycle;
my @signal_input_bits;
my @signal_input_delay;
my $no_truth_table = 0;
my @signal_truthTable;

my $no_reset = 0;
my @reset_port;
my @reset_cycle;
my @reset_on;
my @reset_off;

my $no_clock = 0;
my @clock;
my @clock_port;

sub saved{
```

```
	if ($input_type =~ "Signal") {

		$signal_input[$no_signal_input] = $input_ports;
		#print "\n\nInput Ports: $signal_input[$no_signal_input]";

		$signal_input_type[$no_signal_input] = $signal_type;

		my $cycle_ports = do_on_clicked_S();

		$signal_cycle[$no_signal_cycle] = $cycle_ports;
		#print "\nNo. Signal Cycle: $signal_cycle[$no_signal_cycle]";
		$no_signal_cycle = $no_signal_cycle + 1;

		if ($signal_type =~ "Duty Cycle") {
			$signal_input_timing[$no_signal_input] = $entryDC1->get;

			$signal_input_duty_cycle[$no_signal_input] = $entryDC2-
>get;

			print "\nTiming: $signal_input_timing[$no_signal_input]
|| DC: $signal_input_duty_cycle[$no_signal_input]";
		}
		elsif ($signal_type =~ "Bits") {
			for (my $i = 0 ; $i <= $cycle_ports - 1 ; $i++) {
				$signal_input_bits[$no_signal_input][$i] =
$entry_Bit_Bits[$i]->get;
				print "\nBits:
$signal_input_bits[$no_signal_input][$i]";

				if ($i < $cycle_ports - 1) {
					$signal_input_delay[$no_signal_input][$i] =
$entry_Bit_Delay[$i]->get;
					print "\nDelay:
$signal_input_delay[$no_signal_input][$i]";
				}
			}
		}
		elsif ($signal_type =~ "Truth Table") {
			$signal_truthTable[$no_truth_table] =
$signal_input[$no_signal_input];
			print "\nTruth Table:
$signal_truthTable[$no_truth_table]";
			$no_truth_table = $no_truth_table + 1;
		}

		$no_signal_input = $no_signal_input + 1;
	}
	elsif ($input_type =~ "Reset") {
		$reset_port[$no_reset] = $input_ports;
		print "\nReset Port: $reset_port[$no_reset]";

		my $reset_c = do_on_clicked_R();
		$reset_cycle[$no_signal_input] = $reset_c;
		print "\nReset Cycle: $reset_cycle[$no_signal_input]";
```

```
            for (my $i = 0 ; $i <= $reset_c - 1 ; $i++) {
                  $reset_on[$no_reset][$i] = $entry_Reset_Bits[$i]->get;
                  $reset_off[$no_reset][$i] = $entry_Reset_Delay[$i]->get;

                  print "\nClock On: $reset_on[$no_reset][$i] || Clock Off:
$reset_off[$no_reset][$i]";
            }

            $no_reset = $no_reset + 1;
      }
      elsif ($input_type =~ "Clock") {
            $clock_port[$no_clock] = $input_ports;
            print "\nClock Port: $input_ports";

            $clock[$no_clock] = $entry_C->get;
            print "\nClock Cycle: $clock[$no_clock]";

            $no_clock = $no_clock + 1;
      }
}




############################## Truth Table Generation
##############################

$mw2->Button(-text => "Update", -command => \&update_table)
               ->pack(-side => 'bottom',-anchor => 's');

my $t;
my ($rows,$cols);
my $arrayVar = {};
my @combinations=();
my $count2 = 1;

sub update_table {

      $arrayVar = {};
      @combinations=();
      $count2 = 1;

      show_combinations($no_truth_table);

      ($rows,$cols) = ($count2 , $no_truth_table);

      foreach my $col (0..($cols-1)){
            $arrayVar->{"0,$col"} = "$signal_truthTable[$col]";

            foreach my $row  (1..($rows-1)){
```

```
                        $arrayVar->{"$row,$col"} =
"$combinations[$row][$col]";

            }
      }

      $t = $mw2->Scrolled('TableMatrix', -rows => $rows, -cols => $cols,
      -width => 6, -height => 6,
      -titlerows => 1,
      -variable => $arrayVar,
      -selecttitles => 0,
      -drawmode => 'slow',
      -scrollbars=>'se'
      );


      # Color definitions here:
      $t->tagConfigure('title', -bg => 'lightblue', -fg => 'black', -
relief
      => 'sunken');
      $t->tagConfigure('dis', -state => 'disabled');
      $t->pack(-expand => 1);
      $t->focus;

      sub show_combinations { my($n,@prefix)=@_;
            if($n > 0) {
                  show_combinations( $n-1, @prefix, 0);
                  show_combinations( $n-1, @prefix, 1);
            }
            else {
                  #print " @prefix \n";

                  for (my $j = 0 ; $j <= $no_truth_table - 1 ; $j++) {
                        $combinations[$count2][$j] =  $prefix[$j];
                  }
            $count2 = $count2 + 1;
            }
      }
}


$mw2->Button(-text => "Saved", -command => \&save_table)
               ->pack(-side => 'bottom',-anchor => 's');

my $entry_T = $mw2->Entry(
    -font => ['fixed', 10],
    -width   => 3,
);
$entry_T->pack(-side => 'bottom',-anchor => 's');
$mw2->Label(-text => 'Delay Timing :')->pack(-side => 'bottom',-anchor =>
's');

my @input_truthTable;
my $input_delay;
```

```
sub save_table {
      foreach my $row  (1..($rows-1)){
            foreach my $col (0..($cols-1)){
                  $input_truthTable[$row][$col] = $t->get("$row,$col");
                  #print("\nTruth Table Value: ",$t->get("$row,$col"));
            }
      }
      #print ("\n Truth Table Delay Timing :",$entry_T->get);
      $input_delay = $entry_T->get;
}




############################## Generate Testbench
##############################

$mw->Button(
    -text    => 'Generate',
    -font    => ['fixed', 10],
    -width   => 15,
    -command => \&generate,
)->place(-x => 500, -y => 230);

sub generate{

open(my $inF, ">", $new_file_vhd);

# Print title
printf($inF "-----------------------------------------------------------------
------------------\n");
printf($inF "--
Revision: 1.1 \n");
#printf($inF "--
Date: %02d/%02d/%04d \n", $month+1, $day, $year+1900);
printf($inF "-----------------------------------------------------------------
------------------\n");
#printf($inF "--\t\t\t\t My Company Confidential Copyright © %04d My
Company, Inc.\n", $year+1900);
printf($inF "--\n");
printf($inF "--   File name :  $file.vhd\n");
printf($inF "--   Title     :  Automatic HDL Testbench Generation\n");
printf($inF "--   Module    :  $file\n");
printf($inF "--   Author    :  $author\n");
printf($inF "--   Purpose   :  Year 4 FYP\n");
printf($inF "--\n");
#printf($inF "--   Roadmap   :\n");
printf($inF "-----------------------------------------------------------------
------------------\n");
#printf($inF "--   Modification History :\n");
#printf($inF "--\tDate\t\tAuthor\t\tRevision\tComments\n");
```

```
#printf($inF "--\t%02d/%02d/%04d\t$author\tRev A\t\tCreation\n", $month+1,
$day, $year+1900);
#printf($inF "-------------------------------------------------------------
-------------------\n");
printf($inF "\n");

# Library
printf($inF "Library IEEE;\n");
printf($inF "use IEEE.STD_LOGIC_1164.all;\n");
#printf($inF "use IEEE.std_logic_unsigned.all;\n");
#printf($inF "use IEEE.std_logic_arith.all;\n");
#printf($inF "use IEEE.Numeric_STD.all;\n");
#printf($inF "\n");
#printf($inF "library work;\n");
#my $new_text = join "_", $file, "pkgs.all";
#printf($inF "use work.$new_text;\n");
printf($inF "\n");
printf($inF "\n");

# Entity
printf($inF "-- Declare module entity. Declare module inputs, inouts, and
outputs.\n");
printf($inF "entity $new_file is\n");
printf($inF "end $new_file;\n");
printf($inF "\n");

# Architecture
printf($inF "-- Begin module architecture/code.\n");
printf($inF "ARCHITECTURE behavior OF $new_file IS\n");
printf($inF "\n");

# Component
print ($inF "COMPONENT $file\n"); #print first line
print $inF " PORT(\n";       #print second line
my $out= join "\n\t", @ports;
print ($inF "$out\t\n\t\nEND COMPONENT;\n"); #print ports and last couple
of lines
print ($inF "\n");

# UUT Port Signals
#printf($inF "-- UUT Port Signals.\n");
#printf($inF "$out;\n"); #print ports and last couple of lines
#printf($inF "\n");

printf($inF "-- Inputs & Outputs\n");
printf($inF "$out3\n");


printf($inF "\n-- Local parameter, wire, and register declarations go
here.\n");
printf($inF "-- N/A\n");
printf($inF "-- general signals\n");
printf($inF "-- N/A\n");
printf($inF "\n");
```

```
printf($inF "-- *** Instantiate Constants ***\n");

if ($no_clock != 0)
{
# Clock
      for (my $i=1 ; $i <= $no_clock ; $i++)
      {
            printf($inF "constant clk_PERIOD$i: time := $clock[$i-1]
ns;\n");
            printf($inF "\n");
      }
}

printf($inF "BEGIN\n");
printf($inF "\n");
printf($inF "-- Instantiate the UUT module.\n");
printf($inF "uut : $file\nport map (");       #print first line
printf($inF "\n\t$out2);\n\n");
printf($inF "\n");

# Generate Clock
if ($no_clock != 0)
{
printf($inF "-- Generate necessary clocks.\n");

      for (my $i=1 ; $i <= $no_clock ; $i++)
      {
            printf($inF "Clk_process$i: process\n");
            printf($inF "begin\n");
            printf($inF "\ttb_$clock_port[$i-1] <= '1';\n");
            printf($inF "\twait for clk_PERIOD$i / 2;\n");
            printf($inF "\ttb_$clock_port[$i-1] <= '0';\n");
            printf($inF "\twait for clk_PERIOD$i / 2;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
      }
}

# Reset
if ($no_reset != 0)
{
printf($inF "-- Toggle the resets.\n");

      for (my $i=1 ; $i <= $no_reset ; $i++)
      {
            printf($inF "reset$i: process\n");
            printf($inF "begin\n");

            for (my $j=0 ; $j <= $reset_cycle[$i-1] - 1 ; $j++)
            {
                  printf($inF "\ttb_$reset_port[$i-1] <= '1';\n");
                  printf($inF "\twait for $reset_on[$i-1][$j] ns;\n");
```

```
                  printf($inF "\ttb_$reset_port[$i-1] <= '0';\n");
                  printf($inF "\twait for $reset_off[$i-1][$j] ns;\n");
            }
            #printf($inF "\ttb_$reset_port[$i-1] <= '1';\n");

            printf($inF "\twait;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
      }
}


# Stimulus process

if ($no_signal_input != 0)
{
printf($inF "-- Insert Processes and code here.\n");
      for (my $i=1 ; $i <= $no_signal_input ; $i++)
      {
            my $signal_match = '0';
            for (my $k=0 ; $k <= $no_truth_table - 1 ; $k++)
            {
                  if ($signal_input[$i-1] =~ $signal_truthTable[$k])
                  {
                        $signal_match = '1';
                  }
            }

            if ($signal_match =~ '0')
            {
                  printf($inF "-- Stimulus process$i\n");
                  printf($inF "$signal_input[$i-1]: process\n");
                  printf($inF "begin\n");

                  if ($signal_input_type[$i-1] =~ "Duty Cycle")
                  {
                        for (my $j=0 ; $j <= $signal_cycle[$i-1] - 1 ;
$j++)
                        {
                              printf($inF "tb_$signal_input[$i-1] <=
'1';\n");
                              my $delay_on = ($signal_input_duty_cycle[$i-
1]/100)*$signal_input_timing[$i-1];
                              printf($inF "wait for $delay_on ns;\n");

                              printf($inF "tb_$signal_input[$i-1] <=
'0';\n");
                              my $delay_off = ((100 -
$signal_input_duty_cycle[$i-1])/100)*$signal_input_timing[$i-1];
                              printf($inF "wait for $delay_off ns;\n");
                        }
                  }
                  elsif ($signal_input_type[$i-1] =~ "Bits")
                  {
```

```
                        for (my $j=0 ; $j <= $signal_cycle[$i-1] - 1 ;
$j++)
                        {

                              printf($inF "tb_$signal_input[$i-1] <=
'$signal_input_bits[$i-1][$j]';\n");

                              if ($j < $signal_cycle[$i-1] - 1) {
                                    printf($inF "wait for
$signal_input_delay[$i-1][$j] ns;\n");
                              }
                        }
                  }

                  printf($inF "wait;\n");
                  printf($inF "end process;\n");
                  printf($inF "\n");
            }
      }

      if ($no_truth_table > 0)
      {
            printf($inF "-- Stimulus process\n");
            printf($inF "stim_proc: process\n");
            printf($inF "begin\n");

            for (my $i=1 ; $i <= $rows - 1 ; $i++)
            {
                  if ($i > 0)
                  {
                        printf($inF "wait for $input_delay ns;\n\n");
                  }

                  for (my $j=0 ; $j <= $no_truth_table - 1 ; $j++)
                  {
                        printf($inF "tb_$signal_truthTable[$j] <=
\"$input_truthTable[$i][$j]\";\n");
                  }
            }
            printf($inF "wait;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
      }

}

printf($inF "END behavior; -- architecture\n");
printf($inF "\n");
printf($inF "\n");

#my $new_text2 = join "_", $new_file, "cfg";
#printf($inF "configuration $new_text2 of $new_file is\n");
#printf($inF "for behavior\n");
#printf($inF "end for;\n");
```

```
#printf($inF "end $new_text2;\n");

close(inF);

print("\nThe script has finished successfully! You can now use the file
$new_file_vhd.\n\n");


##################### Reset all the variable
################################
$no_signal_input = 0;
@signal_input = ();
$no_signal_cycle = 0;
@signal_cycle = ();
@signal_input_timing = ();
@signal_input_type = ();
@signal_input_duty_cycle = ();
@signal_input_bits = ();
@signal_input_delay = ();
$no_truth_table = 0;
@signal_truthTable = ();

$no_reset = 0;
@reset_port = ();
@reset_cycle = ();
@reset_on = ();
@reset_off = ();

$no_clock = 0;
@clock = ();
@clock_port = ();

@input_truthTable = ();
$input_delay = 0;
$t->destroy;

}

MainLoop();

tie *STDOUT, ref $text, $text;

#-----------------------------------------------------------------------
-----
# Generic Error and Exit routine
#-----------------------------------------------------------------------
-----

sub dienice {
      my($errmsg) = @_;
      print"$errmsg\n";
      exit;
}
```

Perl program for reading design and generating text file

```
# Read file and extract ports name and its features (input/output)

use strict;
use warnings;
use Text::Table;
#my $file = $ARGV[0];
my $file = "Rs232Txd.vhd";

# check to see if the user entered a file name.
#die "syntax: [perl] vhdl_tb.pl existing_file.vhd\n" if ($file eq "");

# Read in the target file into an array of lines
open(inF, $file) or dienice ("file open failed");
my @data = <inF>;
close(inF);

# Make Date int MM/DD/YYYY
my $year      = 0;
my $month     = 0;
my $day       = 0;
($day, $month, $year) = (localtime)[3,4,5];


# Grab username from PC:
my $author= "$^O user";
if ($^O =~ /mswin/i)
{
  $author= $ENV{USERNAME} if defined $ENV{USERNAME};
}
else
{
  $author = getlogin();
}


#     Strip newlines
foreach my $i (@data) {
      chomp($i);
      $i =~ s/--.*//;        #strip any trailing -- comments
}

#     initialize counters
my $lines = scalar(@data);         #number of lines in file
my $line = 0;
```

```
my $entfound = -1;

#     find 'entity' left justified in file
for ($line = 0; $line < $lines; $line++) {
      if ($data[$line] =~ m/^entity/) {
            $entfound = $line;
            $line = $lines;  #break out of loop
      }
}

# find 'end $file', so that when we're searching for ports we don't
include local signals.
my $entendfound = 0;
$file =~ s/\.vhd$//;
for ($line = 0; $line < $lines; $line++) {
      if ($data[$line] =~ m/^end $file/) {
            $entendfound = $line;
            $line = $lines;  #break out of loop
      }
}


#     if we didn't find 'entity' then quit
if ($entfound == -1) {
      print("Unable to instantiate-no occurance of 'entity' left justified
in file.\n");
      exit;
}

#find opening paren for port list
$entendfound = $entendfound + 1;
my $pfound = -1;

# Remove entity line and port (
for ($line = $entfound; $line < $entendfound; $line++) { #start looking
from where we found module
      $data[$line] =~ s/--.*//;         #strip any trailing --comment

        if ($data[$line] =~ m/\(/) {          # 0x28 is '('
            $pfound = $line;
                 $data[$line] =~ s/.*\x28//;   # remove "port ("
            #print "$data[$line]\n";
            $line = $entendfound;  # break out of loop
      }
}

#     if couldn't find '(', exit
if ($pfound == -1) {
      print("Unable to instantiate-no occurance of '(' after module
keyword.\n");
      exit;
}

#collect port names
```

```
my @ports;
my @ports3;

# print a(or b) : in  STD_LOGIC; sum(or carry) : out  STD_LOGIC;
for ($line = $pfound; $line < $entendfound; $line++) {
      $data[$line] =~ s/--.*//;         #strip any trailing --comment
      next if not $data[$line] =~ /:.*/;
      $data[$line] =~ s/^\s+|\s+$//; # trim right and left space
        push @ports , $data[$line];
      #print "$data[$line]";
}


#print out instantiation
print ("component $file\n"); #print first line
print (" port (\n");        #print second line
my $output= join "\n", @ports;
print ("$output\t\n\t\nend component;\n"); #print ports and last couple of
lines

# Create the module instantiation.  A future enhancement would be to call
the script vhdl_inst.pl instead.
my @ports2;
my @inOut;
for ($line = $pfound; $line < $entendfound; $line++) {
      $data[$line] =~ s/--.*//;    # strip any trailing --comment

      #   next if not $data[$line] =~ /:.*;/;
      $data[$line] =~ s/^/ /mg;    # add space at the beginning of line

      if ($data[$line] =~ /\s+(\w+)\s+:/)
      {
            push @ports2, $1;
            print "\n$1";

      }

      if ($data[$line] =~ /\s+(\w+)\s+STD_LOGIC/)
      {
            push @inOut, $1;
            print "\n$1";

      }
}




##########################################################################
##############################
# Create a txt file and ask for test stimulus

my $in = 1;
my $out = 1;
```

```
my $count = 0;
my @input_type;

my $no_signal_input = 0;
my $no_truth_table = 0;
my @signal_input;
my @signal_input_timing;
my @signal_input_duty_cycle;
my @signal_cycle_DC;
my @signal_cycle_bits;
my @signal_input_type;
my @signal_input_bits;
my @signal_input_delay;
my @signal_truthTable;

my $no_reset = 0;
my @reset_on;
my @reset_off;
my @reset_port;
my @reset_cycle;

my $no_clock = 0;
my @clock;
my @clock_port;

my $Reset_Or_Clock = 0;

# check to make sure that the file doesn't exist.
$file =~ s/\.vhd$//;
my $new_file = join "_", $file, "stimulus";
my $new_file_vhd = join ".",$new_file,"txt";

open(FH, '>', $new_file_vhd) or die $!;

foreach my $i (@inOut) {
      if ($i =~ "in")
      {
            print FH "\nSignal input $in =: $ports[$count]";


            if (index($ports2[$count],"Rst") >= 0 or
index($ports2[$count],"Reset") >= 0 or index($ports2[$count],"rst") >= 0
or index($ports2[$count],"reset") >= 0)
            {
                  print FH
"\n\n+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n";
                  print FH "\n-- Reset Input Type --";
                  print FH "\n\n-----------------------------------------
---------------------\n";
                  print FH "\nHow many Cycle : ";
                  print FH "\nClock On Timing in (ns) : ";
                  print FH "\nClock Off Timing in (ns) : ";
                  $Reset_Or_Clock = 1;
            }
```

```
            if (index($ports2[$count],"Clk") >= 0 or
index($ports2[$count],"Clock") >= 0 or index($ports2[$count],"clk") >= 0
or index($ports2[$count],"clock") >= 0)
            {
                  print FH
"\n\n+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n";
                  print FH "\n-- Clock Input Type --";
                  print FH "\n\n-----------------------------------------
---------------------\n";
                  print FH "\nClock Timing in (ns) : ";
                  $Reset_Or_Clock = 1;
            }

            if ($Reset_Or_Clock =~ 0)
            {
                  print FH
"\n\n+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n";
                  print FH "\n-- Signal Input Type --";
                  print FH "\n\n-----------------------------------------
---------------------\n";
                  print FH "\nInput Type (Duty_Cycle/Bits/Truth_Table) :";
                  print FH "\n\n-----------------------------------------
---------------------\n";
                  print FH "\nDuty Cycle Input Type";
                  print FH "\nHow many Cycle : ";
                  print FH "\nSignals input Period in (ns) : ";
                  print FH "\nSignals input $in duty cycle (out of 100) :
";
                  print FH "\n\n-----------------------------------------
---------------------\n";
                  print FH "\n-- Bits Input Type --";
                  print FH "\nBits1 : ";
                  print FH "\nDelay Timing in (ns) : ";
                  print FH "\nBits2 : ";
                  print FH "\nDelay Timing in (ns) : ";
                  print FH "\nBits3 : ";

                  $signal_truthTable[0][$no_truth_table] = $ports2[$count];

                  $no_truth_table = $no_truth_table + 1;
            }

            print FH
"\n\n\n\n\n\n\n*********************************************************
*****\n";
            print FH
"*************************************************************\n";

            $Reset_Or_Clock = 0;
            $in = $in + 1;
      }
      elsif ($i =~ "out")
      {
```

```
            print FH "\nSignal output $out == $ports[$count]";
            $out = $out + 1;
      }
      else
      {
            print "Error in file";
      }
$count = $count + 1;
}


############################## Truth Table Input
################################
my @input_truthTable;
my @input_delay;
my $cycle_truthTable;
my @combinations;
my $count2 = 0;
my $tb = Text::Table->new(
);

if ($no_truth_table > 0)
{
      print FH
"\n\n\n\n\n\n\n*************************************************************
*****\n";
      print FH
"*****************************************************************\n";
      print FH "\n-- Truth Table Input Type --\n";
      print FH
"\n+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n";
      print FH "\nDo You Want Truth Table Inputs (Yes/No) : \n";

      show_combinations($no_truth_table);

      $tb->load(
            @signal_truthTable
      );

      $tb->load(
            @combinations

      );

      print FH "\n";

      $tb->load(
            "Time : "
      );

}

print FH $tb;
```

```
sub show_combinations { my($n,@prefix)=@_;
      if($n > 0) {
            show_combinations( $n-1, @prefix, 0);
            show_combinations( $n-1, @prefix, 1);
      }
      else {
            #print FH " @prefix \n";

            for (my $j = 0 ; $j <= $no_truth_table - 1 ; $j++) {
                  $combinations[$count2][$j] =  $prefix[$j];
            }

      $count2 = $count2 + 1;
      }
}

close(FH);

print "\nWriting to file successfully!\n";
```

Read text file and generate testbench

```
# Read file and extract ports name and its features (input/output)

use strict;
use warnings;
use Text::Table;
#my $file = $ARGV[0];
my $file = "half_adder_stimulus.txt";

# Read in the target file into an array of lines
open(inF, $file) or dienice ("file open failed");
my @data = <inF>;
close(inF);

# Make Date int MM/DD/YYYY
my $year      = 0;
my $month     = 0;
my $day       = 0;
($day, $month, $year) = (localtime)[3,4,5];



# Grab username from PC:
my $author= "$^O user";
if ($^O =~ /mswin/i)
{
  $author= $ENV{USERNAME} if defined $ENV{USERNAME};
}
```

```
else
{
  $author = getlogin();
}


#     Strip newlines
#foreach my $i (@data) {
#     chomp($i);
#     $i =~ s/--.*//;        #strip any trailing -- comments
#}

#     initialize counters
my $lines = scalar(@data);        #number of lines in file
my $line = 0;
my $count_signal = 0;
my $Truth_Table_Line;
my @Input_Line = -1;
my @Input_ports_STD;
my @Input_ports;
my @Output_ports;
my @Output_ports_STD;

#     find 'Signal input' left justified in file
for ($line = 0; $line < $lines; $line++) {
      if ($data[$line] =~ m/^Signal input/) {
            $Input_Line[$count_signal] = $line;
            #print "\n$Input_Line[$count_signal]";

            if ($data[$line] =~ /=:\s*(.+)$/)
            {
                  push @Input_ports_STD, $1;

                  if ($data[$line] =~ /\s+(\w+)\s+:/)
                  {
                        push @Input_ports, $1;
                        print "\n$1\n";
                  }
            }
            $count_signal = $count_signal + 1;
      }
      elsif ($data[$line] =~ m/^Signal output/) {
            $Input_Line[$count_signal] = $line - 1;

            if ($data[$line] =~ /==\s*(.+)$/)
            {
                  push @Output_ports_STD, $1;

                  if ($data[$line] =~ /\s+(\w+)\s+:/)
                  {
                        push @Output_ports, $1;
                        print "\n$1\n";
                  }
            }
```

```
	}
	elsif ($data[$line] =~ m/^-- Truth Table/) {
		$Truth_Table_Line = $line;
		print "\nTruth Table is Found \n\n";
	}

}


#	if we didn't find 'Signal input' then quit
if ($Input_Line[0] == -1) {
	print("Unable to find 'Signal input' in the file.\n");
	exit;
}



my $count_data = 0;
my $type;
my $data_type;
my @data_rst;
my @data_clk;
my @data_signal;
my @data_signal_test;

my $no_signal_input = 0;
my $no_signal_input2 = 0;
my @stored_input2;
my @signal_input;

my $no_reset = 0;
my @reset_port;

my $no_clock = 0;
my @clock_port;

for ($count_data = 0 ; $count_data <= $count_signal - 1 ; $count_data++){
	if (index($Input_ports[$count_data],"Rst") >= 0 or
index($Input_ports[$count_data],"Reset") >= 0 or
index($Input_ports[$count_data],"rst") >= 0 or
index($Input_ports[$count_data],"reset") >= 0)
	{
		$type = "Reset";
		$reset_port[$no_reset] = $Input_ports[$count_data];
		$no_reset = $no_reset + 1;
	}
	elsif (index($Input_ports[$count_data],"Clk") >= 0 or
index($Input_ports[$count_data],"Clock") >= 0 or
index($Input_ports[$count_data],"clk") >= 0 or
index($Input_ports[$count_data],"clock") >= 0)
	{
		$type = "Clock";
		$clock_port[$no_clock] = $Input_ports[$count_data];
		$no_clock = $no_clock + 1;
```

```
	}
	else
	{
		$type = "Signal";
		$signal_input[$no_signal_input] = $Input_ports[$count_data];
		$no_signal_input = $no_signal_input + 1;

		$no_signal_input2 = 0;
	}

	for ($line = $Input_Line[$count_data] + 1 ; $line <
$Input_Line[$count_data + 1] ; $line++) {
		if ($type =~ "Reset"){
			if ($data[$line] =~ /\s+:\s+(\w+)/)
			{
				push @data_rst, $1;
				print "\n$1\n";
			}
		}
		elsif ($type =~ "Clock"){
			if ($data[$line] =~ /\s+:\s+(\w+)/)
			{
				push @data_clk, $1;
				print "\n$1\n";
			}
		}
		elsif ($type =~ "Signal"){

			if ($data[$line] =~ /\s+:\s+(\w+)/)
			{
				$data_signal[$no_signal_input -
1][$no_signal_input2] =  $1;
				$no_signal_input2 = $no_signal_input2 + 1;
				print "\n$1\n";
			}
		}
	}

	if ($no_signal_input2 != 0) {
		$stored_input2[$no_signal_input - 1] = $no_signal_input2;
		print "\nStored: $stored_input2[$no_signal_input - 1]\n\n";
	}
}



my @reset_sig;
my $count1_reset = 0;

for (my $i = 0 ; $i <= $no_reset - 1 ; $i++){
	print "\n\n$reset_port[$i]\n";

	for (my $j = 0 ; $j <=  2 ; $j++){
```

```
            $reset_sig[$i][$count1_reset] = $data_rst[$j];
            print "$reset_sig[$i][$count1_reset]\n";

            $count1_reset = $count1_reset + 1;
      }

      $count1_reset = 0;
}


my @clock = @data_clk;


my @signal_sig_DC;
my @signal_sig_Bits;
my $type_sig;
my $type_sig_line;
my $signal_DC_count = 0;
my $signal_Bits_count = 0;
my $no_truth_table = 0;
my @signal_truthTable;

for (my $i = 0 ; $i <= $no_signal_input - 1 ; $i++){
      print "\n\n$signal_input[$i]\n";

      if ($data_signal[$i][0] =~ "Duty_Cycle") {
            for (my $j = 1 ; $j <= $stored_input2[$i] - 1 ; $j++){
                  $signal_sig_DC[$signal_DC_count][$j - 1] =
$data_signal[$i][$j];
                  print "$signal_sig_DC[$signal_DC_count][$j - 1]\n";
            }
            $signal_DC_count = $signal_DC_count + 1;
      }
      elsif ($data_signal[$i][0] =~ "Bits") {
            for (my $j = 1 ; $j <= $stored_input2[$i] - 1 ; $j++){
                  $signal_sig_Bits[$signal_Bits_count][$j - 1] =
$data_signal[$i][$j];
                  print "$signal_sig_Bits[$signal_Bits_count][$j - 1]\n";
            }
            $signal_Bits_count = $signal_Bits_count + 1;
      }
      elsif ($data_signal[$i][0] =~ "Truth_Table") {
            $signal_truthTable[$no_truth_table] = $signal_input[$i];
            print "\nTruth Table: $signal_truthTable[$no_truth_table]";
            $no_truth_table = $no_truth_table + 1;
      }
}


my $count_spl = 0;
my $length_spl = 0;
my @data_truth_table;
my $data_truth_table_delay;
for (my $line = $Truth_Table_Line + 7 ; $line < $lines ; $line++){
```

```
	if ($data[$line] =~ m/^Time/) {
		if ($data[$line] =~ /\s+:\s+(\w+)/){
				$data_truth_table_delay = $1;
				print "\nDelay Timing: $1\n";
		}
	}
	else{
		my @spl = split(' ', $data[$line]);

		foreach my $i (@spl)
		{
			$data_truth_table[$count_spl][$length_spl] = ${i};
			print "\n\nSplit Line ($count_spl)($length_spl): ${i}\n";
			$length_spl = $length_spl + 1;
		}
		$length_spl = 0;
		$count_spl = $count_spl + 1;
	}
}



my @portlines;
foreach my $i (@Input_ports_STD) {
	$i =~ s/ in//;
	push @portlines, "\tsignal tb_$i";
}

my @portlinesInst;
foreach my $i (@Input_ports) {
	push @portlinesInst, "$i \t=> tb_$i";
}

my $out2 = join "\n", @portlines;
my $out3 = join "\n", @portlinesInst;


my @portlines1;
my $count_STD = 0;
foreach my $i (@Output_ports_STD) {
	$i =~ s/ out//;
	$count_STD++;
	push @portlines1, "\tsignal tb_$i";
}

my @portlinesInst1;
foreach my $i (@Output_ports) {
	push @portlinesInst1, "$i \t=> tb_$i";
}

$portlines1[$count_STD - 1] =~ tr/)//d;
my $out4 = join "\n", @portlines1;
my $out5 = join "\n", @portlinesInst1;
```

```
# check to make sure that the file doesn't exist.
$file =~ s/\_stimulus.txt$//;
my $new_file = join "_", $file, "tb";
my $new_file_vhd = join ".",$new_file,"vhd";

open(my $inF, ">", $new_file_vhd);

# Print title
printf($inF "-----------------------------------------------------------------
------------------\n");
printf($inF "--
Revision: 1.1 \n");
printf($inF "--
Date: %02d/%02d/%04d \n", $month+1, $day, $year+1900);
printf($inF "-----------------------------------------------------------------
------------------\n");
#printf($inF "--\t\t\t\t My Company Confidential Copyright © %04d My
Company, Inc.\n", $year+1900);
printf($inF "--\n");
printf($inF "--   File name :  $file.vhd\n");
printf($inF "--   Title     :  Automatic HDL Testbench Generation\n");
printf($inF "--   Module    :  $file\n");
printf($inF "--   Author    :  $author\n");
printf($inF "--   Purpose   :  Year 4 FYP\n");
printf($inF "--\n");
#printf($inF "--   Roadmap   :\n");
printf($inF "-----------------------------------------------------------------
------------------\n");
#printf($inF "--   Modification History :\n");
#printf($inF "--\tDate\t\tAuthor\t\tRevision\tComments\n");
#printf($inF "--\t%02d/%02d/%04d\t$author\tRev A\t\tCreation\n", $month+1,
$day, $year+1900);
#printf($inF "-----------------------------------------------------------------
-------------------\n");
printf($inF "\n");

# Library
printf($inF "Library IEEE;\n");
printf($inF "use IEEE.STD_LOGIC_1164.all;\n");
#printf($inF "use IEEE.std_logic_unsigned.all;\n");
#printf($inF "use IEEE.std_logic_arith.all;\n");
#printf($inF "use IEEE.Numeric_STD.all;\n");
#printf($inF "\n");
#printf($inF "library work;\n");
#my $new_text = join "_", $file, "pkgs.all";
#printf($inF "use work.$new_text;\n");
printf($inF "\n");
printf($inF "\n");

# Entity
printf($inF "-- Declare module entity. Declare module inputs, inouts, and
outputs.\n");
printf($inF "entity $file is\n");
printf($inF "end $file;\n");
```

```
printf($inF "\n");

# Architecture
printf($inF "-- Begin module architecture/code.\n");
printf($inF "ARCHITECTURE behavior OF $file IS\n");
printf($inF "\n");

# Component
print ($inF "COMPONENT $file\n"); #print first line
print $inF "PORT(\n";        #print second line
my $InSTD = join "\n\t", @Input_ports_STD;
my $OutSTD = join "\n\t", @Output_ports_STD;
printf($inF "$InSTD\n");
print($inF "\n");
printf($inF "$OutSTD\n");
print ($inF "  END COMPONENT;\n"); #print Input_ports and last couple of
lines
print ($inF "\n");

# UUT Port Signals
#printf($inF "-- UUT Port Signals.\n");
#printf($inF "$out;\n"); #print ports and last couple of lines
#printf($inF "\n");

printf($inF "-- Inputs\n");
printf($inF "$out2\n");
printf($inF "\n");

printf($inF "-- Outputs\n");
printf($inF "$out4\n");
printf($inF "\n");


printf($inF "-- Local parameter, wire, and register declarations go
here.\n");
printf($inF "-- N/A\n");
printf($inF "-- general signals\n");
printf($inF "-- N/A\n");
printf($inF "\n");


printf($inF "-- *** Instantiate Constants ***\n");

if ($no_clock != 0)
{
# Clock
      for (my $i=1 ; $i <= $no_clock ; $i++)
      {
            printf($inF "constant clk_PERIOD$i: time := $clock[$i-1]
ns;\n");
            printf($inF "\n");
      }
}
```

```
printf($inF "BEGIN\n");
printf($inF "\n");
printf($inF "-- Instantiate the UUT module.\n");
printf($inF "uut : $file\nport map (");       #print first line
printf($inF "\n\t$out3");
printf($inF "\n\t$out5\n\n");
printf($inF "\n");

# Generate Clock
if ($no_clock != 0)
{
printf($inF "-- Generate necessary clocks.\n");

      for (my $i=1 ; $i <= $no_clock ; $i++)
      {
            printf($inF "Clk_process$i: process\n");
            printf($inF "begin\n");
            printf($inF "\ttb_$clock_port[$i-1] <= '1';\n");
            printf($inF "\twait for clk_PERIOD$i / 2;\n");
            printf($inF "\ttb_$clock_port[$i-1] <= '0';\n");
            printf($inF "\twait for clk_PERIOD$i / 2;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
      }
}


# Reset
if ($no_reset != 0)
{
printf($inF "-- Toggle the resets.\n");

      for (my $i=1 ; $i <= $no_reset ; $i++)
      {
            printf($inF "reset$i: process\n");
            printf($inF "begin\n");

            for (my $j=0 ; $j <= $reset_sig[$i-1][0] - 1 ; $j++)
            {
                  printf($inF "\ttb_$reset_port[$i-1] <= '1';\n");
                  printf($inF "\twait for $reset_sig[$i-1][1] ns;\n");
                  printf($inF "\ttb_$reset_port[$i-1] <= '0';\n");
                  printf($inF "\twait for $reset_sig[$i-1][2] ns;\n");

            }
            printf($inF "\ttb_$reset_port[$i-1] <= '1';\n");

            printf($inF "\twait;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
      }
}
```

```
# Stimulus process
if ($no_signal_input != 0)
{
printf($inF "-- Insert Processes and code here.\n");

      for (my $i=0 ; $i <= $no_signal_input - 1 ; $i++)
      {
            my $signal_match = '0';
            for (my $k=0 ; $k <= $no_truth_table - 1 ; $k++)
            {
                  if ($signal_input[$i-1] =~ $signal_truthTable[$k])
                  {
                        $signal_match = '1';
                  }
            }

            if ($signal_match =~ '0')
            {
                  printf($inF "-- Stimulus process$i\n");
                  printf($inF "$signal_input[$i]: process\n");
                  printf($inF "begin\n");

                  if ($data_signal[$i][0] =~ "Duty_Cycle") {

                        for (my $j = 0 ; $j <= $data_signal[$i][1] - 1 ;
$j++)
                        {
                              printf($inF "\ttb_$signal_input[$i] <=
'1';\n");
                              my $delay_on =
($data_signal[$i][3]/100)*$data_signal[$i][2];
                              printf($inF "\twait for $delay_on ns;\n");

                              printf($inF "\ttb_$signal_input[$i] <=
'0';\n");
                              my $delay_off = ((100 -
$data_signal[$i][3])/100)*$data_signal[$i][2];
                              printf($inF "\twait for $delay_off ns;\n");

                        }
                  }
                  elsif ($data_signal[$i][0] =~ "Bits") {
                        for (my $j = 1 ; $j <= $stored_input2[$i] - 1 ;
$j++){
                              if ($j % 2 == 0) {
                                    printf($inF "\twait for
$data_signal[$i][$j] ns;\n");
                              }
                              else {
                                    printf($inF "\ttb_$signal_input[$i] <=
'$data_signal[$i][$j]';\n");
                              }
                        }
                  }
```

```
                printf($inF "\twait;\n");
                printf($inF "end process;\n");
                printf($inF "\n");
            }
        }
        if ($no_truth_table > 0)
        {
            printf($inF "-- Stimulus process\n");
            printf($inF "stim_proc: process\n");
            printf($inF "begin\n");

            for (my $i=0 ; $i <= $count_spl - 1 ; $i++)
            {
                if ($i > 0)
                {
                    printf($inF "wait for $data_truth_table_delay
ns;\n\n");
                }

                for (my $j=0 ; $j <= $no_truth_table - 1 ; $j++)
                {
                    printf($inF "tb_$signal_truthTable[$j] <=
\"$data_truth_table[$i][$j]\";\n");
                }
            }
            printf($inF "wait;\n");
            printf($inF "end process;\n");
            printf($inF "\n");
        }
}



printf($inF "END behavior; -- architecture\n");
printf($inF "\n");
printf($inF "\n");

#my $new_text2 = join "_", $new_file, "cfg";
#printf($inF "configuration $new_text2 of $new_file is\n");
#printf($inF "for behavior\n");
#printf($inF "end for;\n");
#printf($inF "end $new_text2;\n");

close(inF);

print("\nThe script has finished successfully! You can now use the file
$new_file_vhd.\n\n");


exit;
```

```
#-------------------------------------------------------------------------
-----
# Generic Error and Exit routine
#-------------------------------------------------------------------------
-----

sub dienice {
      my($errmsg) = @_;
      print"$errmsg\n";
      exit;
}
```