AUTOMATED WRAPPER DESIGN FOR ISCAS '89 BENCHMARK CIRCUITS

By

MUHAMMAD ILIAS BIN MOHAMED IBRAHIM

FINAL PROJECT REPORT

Submitted to the Department of Electrical & Electronic Engineering
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

Universiti Teknologi PETRONAS

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

AUTOMATED WRAPPED DESIGN FOR ISCAS '89 BENCHMARK CIRCUITS

by

MUHAMMAD ILIAS BIN MOHAMED IBRAHIM

A project dissertation submitted to the
Department of Electrical & Electronic Engineering
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

Approved:

_____

Dr. Fawnizu Azmadi Hussin
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

September 2012

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

MUHAMMAD ILIAS BIN MOHAMED IBRAHIM

ABSTRACT

System-on-chip (SOC) enables the reuse of existing IP blocks in a system, thereby making it possible to design complex systems within a short period of time. With the complexity of the design comes the problem of testing the SOC. A typical SOC can integrate many modules, therefore making it difficult to test these individual modules by accessing from the primary interfaces of the chip. To aleviate this test access issue for SOC's, the IEEE 1500 standard has been introduced. There are commercial tools from main EDA players such as Synopsys that can help with the insertion of the IEEE 1500 wrapper. However, most researchers have no access to the expensive tool. Even if they do, the tools are protected so they do not allow researchers access to the internal features to explore potential enhancements. There is no open source tools that can assist test researchers with the 1500 wrapper insertion.

In this thesis, we illustrate our effort at automating the IEEE 1500 wrapper insertion. The design of the IEEE 1500 wrapper is done in Verilog and the automation is done using the Perl scripting language. The inserted wrapper modules are validated and an efficient approach of executing wrapper external tests is also illustrated in this thesis.

ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr. Fawnizu Azmadi Hussin, for the support and time that he had gave me throughout the final year project period. He had assisted me in answering to all my question and doubts and sharing ideas during the final year project. I also would like to thank Adel Eskandar, who had been helping me in getting the software tools and resources throughout the project.

I also would like to thank my friends especially Nazif Hawari and Annaitullah who had been supporting me and encouraging me throughout the project. Last but not least, my family. Thanks for being there for me!

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Background Studies

Electronic devices are getting smaller as we try to approach mobility. In achieving mobility, devices need to be portable and less consuming power. Electronic devices such smartphones, tablet, notebook, netbook, and etc. are the example of mobility devices. These products are able to build due to shrinking of the transistor (the smallest component in integrated circuit/device) and able to combine multiple single function module/circuit become a system that runs faster and consume less power in a single integrated device.

Combining multiple single function module/circuit in achieving a system, built in a single integrated device/circuit is called as System on a Chip (SoC). Example; previously, a chipsets or a memory or a graphic processor or a processors were in their own single chip acting as a single function module, but currently now these module such chipset, memory, graphic processor, and the processors are built in a single chip being a complete system in achieving smaller device with portability and runs faster with less power consumed.

This able to achieve due to shrinking in transistor, where we can put more function or modules in a single same size chip previously. Doing this would create complexity in a single chip, which leads to one of the problem which is how to detect defects, fault or problems after the chip is manufactured. Currently there are billions of transistors in a single chip.

Why testing are important, we know that, nothing in this world are perfect, same goes to the manufacturing stream in the microelectronic industry. There were faults or defects will happen, if there is not test to check the functionality, reliability and faults that happen, it may be a disaster for the company, when their product are in fault. The red ring of death in Xbox 360 is one of the example of general hardware failure where leads big loss for Microsoft (more than a billion) [11] [12].

To avoid these loss, testing are important. IEEE 1500 protocol has been suggested and become a standard to facilitate SOC testing, but inserting these test design manually are inefficient which would cost time and money.

1.2 Problem Statement

Every design needs to be tested; to check their functionality and fault that happens. Same goes to System on a Chip (SoC), it needs to be tested too. Testing SoC are quite complex and time consuming, since there are a lot of blocks need to be tested in a single SoC.

IEEE 1500 standard for embedded core test is widely used to test SoC's. This standard has been manually inserted to the SoC blocks for testing. Inserting wrapper manually would consume time and money which is not efficient. Currently there isn't any available open source automatic IEEE 1500 standard wrapper generator which could save Design for Testability (DfT) engineers or researcher's time and cost.

1.3 Objective of the project

Objective of the project are the milestone that need to be accomplished to solve the problem that arises. The objectives are:-

- To design IEEE 1500 Wrapper using Verilog and validate the functionality using software Quartus II.

- To design an automation script to automate the IEEE 1500 Wrapper insertion using Perl Script.

Final outcome of the project is the automated generated wrapped core using Verilog.

1.4 Relevance of the project

The IEEE 1500 standard for embedded core test which has been approved in 2005 defines the scalable and reusable wrapper architecture that allows the testing of and access for to embedded cores within the system on chip (SoC). There are a lot of Modular or SoC testing as the number of transistor decreases, and modules getting more in a single same size IC's, but there are no open source automation for generating and insertion of IEEE 1500 wrapper then the application to the real circuit. Most of them manually generate the IEEE 1500 core.

Here developing the automation for IEEE 1500 wrapper insertion would save engineer and researcher time, not only that, we could use as a tool to teach DfT to be engineer, so they can easily understand the process in a short term.

1.5 Feasibility of the project

The feasibility of this project is to complete the project within the scope and time frame,

During the FYP 1 period,

- Research on the IEEE 1500 wrapper
- Design the wrapper using Verilog Hardware Description Language
- Simulate the functionality of the wrapper design using Quartus II

During the FYP 2 period,

- Modification and improvise the wrapper design using Quartus II
- Validating the simulation result to check functionality of wrapper
- Developing the automation script for automatic wrapper design insertion.

CHAPTER 2: LITERATURE REVIEW

2.1 SoC – System on a Chip

System-on-Chip (SoC), combination of modules (cores previously were in a single chip) to be a full system in a single chip. Example, previously, chipsets, memory, graphic processor and the CPU were in their own single chip, becoming a complete computer (system). Currently now these modules such chipset, memory, graphic processor, and the CPU are built in a single chip to achieve smaller device for mobility; laptop, netbook, and tablet.

SoC design able to achieve due to shrinking in transistor, where there are more transistors (logic gates or modules) can fit in a single same size die or chip. When more transistors has been fitted to a single chip, the functionality of the chip is increased; as maintaining the previous functionality and adding up other/more functionality to be a system in a single chip is called as System on a Chip. SoC's design usually done by integrating reusable component or modules, and it had greatly reduces total time and cost in the industry [12]. Even reducing total time and cost, but the complexity of the design has increased.

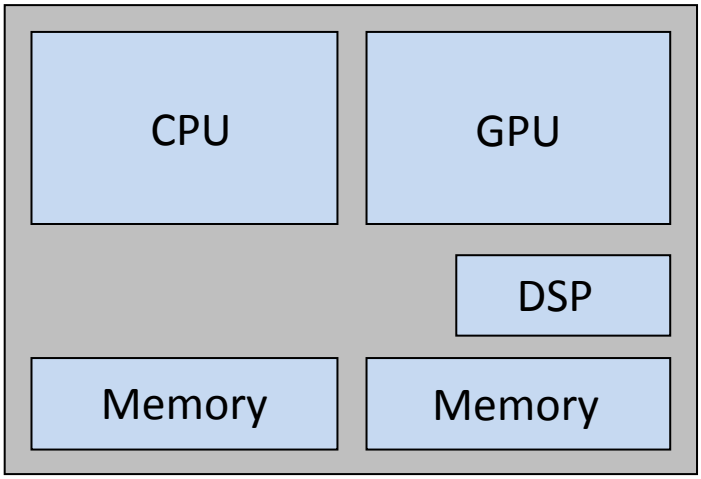


Figure 1: Basic System on a Chip (SoC)

In Figure 1, there is one module of CPU, one module of GPU, one module of DSP and two module of memory in a single chip/die. These modules combine become a system which is a complete Computer System.

2.2 Testing System on a Chip (SoC)

Testing a basic SoC is necessary, this to test the fault that happen in the SoC upon production. Testing this SoC's with normal fault analysis would be hard and time consuming due to the complexity of the SoC's. It's hard to access the modules from the chip pins, the modules are integrated in a single chip and not directly connected with the chip pin, it's connected to another modules or user define logic.

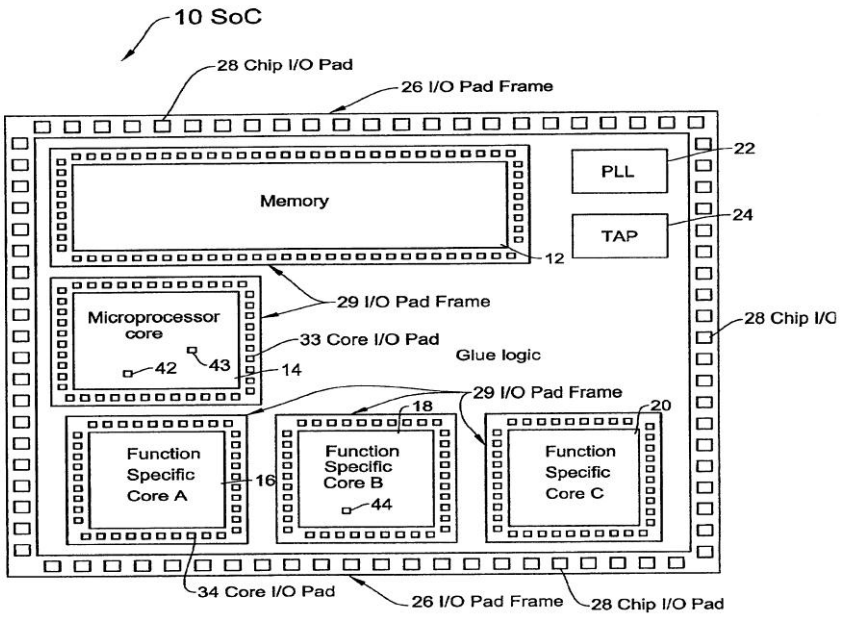


Figure 2: Modules in SoC are not connected directly to the chip pins

The idea of testing SoC is by treating each module separately (isolating the modules), in Figure 3, we are trying to test the GPU module in the SoC. We treat the GPU module separately by isolating the GPU module, where we access this module from the chip pins to the modules. Not all pins are directed connected to this module, but one or more pins that use to access this module (using the Test Access Mechanism).

The idea of above SoC testing is being widely used, but a protocol is needed where it defines a common test design or infrastructure amongst cores from different supplier (IP cores) [13]. IEEE 1500 Standard protocol had provide a standard in assisting the SoC Design for Testability (DfT) for a common test design for every cores which from different supplier. It also assists in reusing the wrapped core for further SoC design.

In Figure 2.2, illustrate one of the modules (GPU) from the basic SoC in Figure 3 is being tested. Here we can observer that the module GPU is being isolated with a wrapper surrounding it and connected to a Test Access Mechanism (TAM). TAM is connected to a sink and to a source where test stimuli are stored or generated at source while test response are stored or analyzed at the sink.

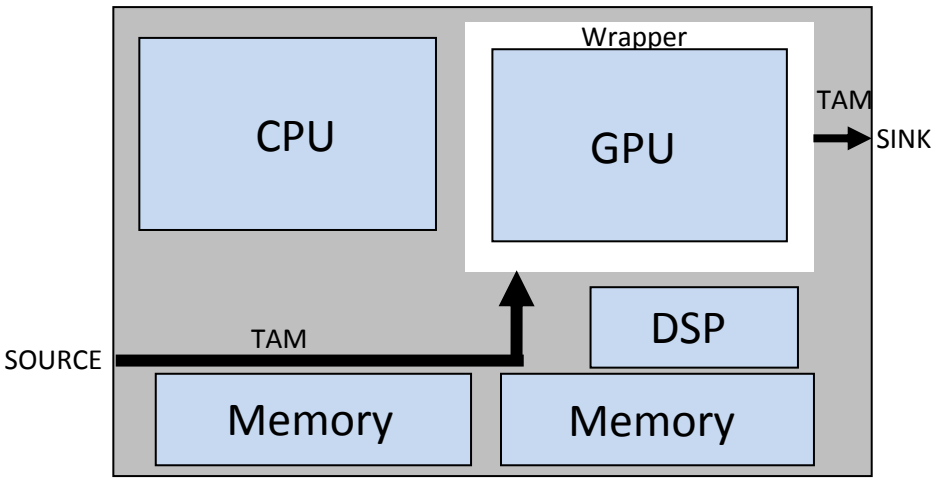


Figure 3: Simple SoC with GPU under test

2.3 IEEE 1500 Standard

How did the IEEE 1500 Standard Wrapper Core [1] introduced and what were problem arise before this standard introduced. Increasing use of SoC's, designers came up with the idea and solution in identifying the identical modules across the chip. Then they isolate this module and implement once, where at the beginning of the SoC's design, the re-use of previous module never done. Later, in the SoC's design, the reuse of the modules was done, where the core design was a fact

The reuse of the modules has led to a problem on how the DfT strategy fit into the SoC environment when the core user and the core provider may not be the same entity, this where the problem raised and my organizations came up with different specific solution, and the needs of a standard were essential [4]. IEEE 1500 Std Wrapper also has standardize the wrapper design where designers can re-use the wrapped core for the other SoC's design.

Specific language has been used to address the communication between the core provider and core user, which the IEEE 1450.6-Standard for Standard Test Interface Language (STIL) for digital Test Vector Data – Core Test Language [6]. The core user is the entity that is integrating the core to the SoC and the core provider is the entity that design and develops the core.

From the core test wrapper handbook [7] sates IEEE 1500 architecture consists of:

1. Both combinational and sequential core test strategies need to meet
2. CTL requirement that allow for the characteristic of the wrapper to be communicated to the core user

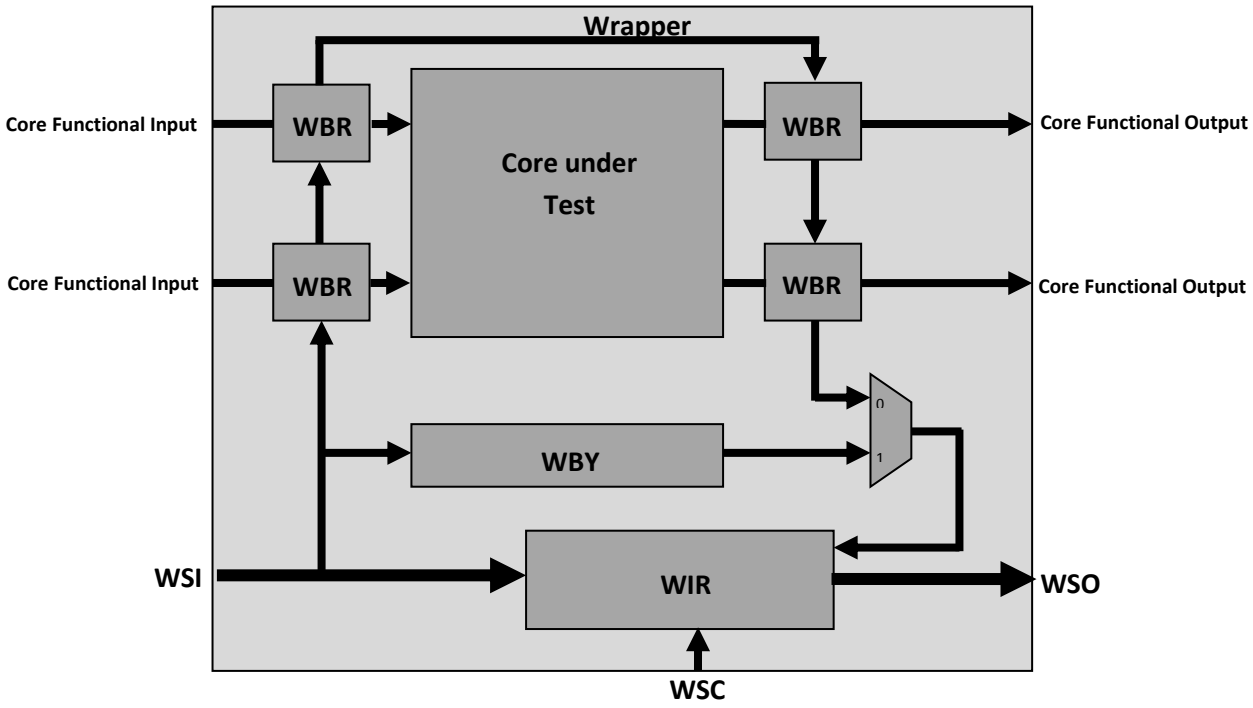


Figure 4: Wrapped core

IEEE 1500 wrapper architecture consist of the following components [7]:

- The Wrapper Interface Port (WIP), which interface the pins and the wrapped module across the SoC's.
- Wrapper Instruction Register (WIR), which consist control signal to control IEEE 1500 component and multiplexers across the wrapper for specific test mode.
- The Wrapper Boundary Register (WBR/WBI/WBO), registers that act to isolate the core/module from other module across the SoC's.
- The Wrapper Bypass Register (WBY), provide the shortest path in the wrapper when the wrapper is not used.

Figure 4 illustrate the basic block of a wrapped Core under Test (CUT) with wrapper main blocks.

2.3.1 WIP – Wrapper Interface Port

Wrapper Interface Port (WIP), collection of terminal that which pins of the chips that controls the wrapper through a Test Access Mechanism (TAM). It provides serial, parallel or hybrid (serial and parallel) access to the core. IEEE 1500 focus on the serial access, hence it is mandatory because the architecture is meant to be plug and play, this difficult to achieve with the parallel access interface [4]. The serial access provided to the wrapper serial port (WSP) while the parallel access provided through the wrapper parallel port (WPP).

As the wrapper serial is mandatory for IEEE 1500, but it also containing several control terminals which controls the wrapper. The control terminals are:

- Wrapper Clock Terminal (WRCK)
- Wrapper Reset Terminal (WRSTN)
- Wrapper Instruction Register Selection Terminal (SelectWIR)
- Wrapper Register Shift Control Terminal (ShiftWR)
- Wrapper Register Capture Control Terminal (CaptureWR)
- Wrapper Register Update Control Terminal (UpdateWR)

And the optional terminals:

- Wrapper Data Register Transfer Control Terminal (TransferDR)
- Auxiliary Wrapper Clock Terminal (AUXCK)

That were the control signal, WSP also consist of two data terminal which is Wrapper Serial Input (WSI) and the (Wrapper Serial Output)

WPP is optional, and there are no specified terminal in the standard, but is prohibited for the WPP use any of the WSP terminals with the exception clock terminal which is WRCK and AUXCK.

2.3.2 WIR – Wrapper Instruction Register

WIR is the brain of the IEEE 1500 wrapper, where it contains the control signals to control all the wrapper components such as the Wrapper Bypass Register (WBY), Wrapper Boundary Register (WBR) and the multiplexers which controls the signal path. The WIR controlled and clocked by signals from the WSC's terminal (Wrapper Serial Input (WSI) and WRCK (Wrapper Clock)).

The WIR should be of dedicated IEEE 1500 logic, and the shift path must be at least 2 bits which can contain up to 3 mandatory modes, (bypass, external test and internal test). No inversion on the logic values from the Wrapper Serial Input (WSI) to Wrapper Serial Output (WSO) during the shift operation. WIR is designed so that the current instruction is not interrupted with the loading if the new one, until the WIR update operation is called.

2.3.3 WBR – Wrapper Boundary Register

The WBR enables the separation of core internal testing from external interconnect or logic testing, it also provides isolation mechanism that allows test stimuli from wrapper input terminal to the wrapped core input terminal and for the response from core output terminal to wrapper output terminal. WBR data inputs are normally coupled with TAM's which typically have a reduce data bandwidth compared to data

bandwidth of the embedded core under test, resulting WBR serves as data bandwidth adaptation mechanism between the SOC environment and the embedded core.

IEEE 1500 mandates the presence of a single chain configuration of the WBR, but allows multiple chain implementations mainly to increase the test data bandwidth. [7]

If a core terminal is directly connected to a register without any combinatorial logic interconnects between them, the terminal is a registered port. If there is combinatorial logic between the register and terminal, the terminal would be unregistered port. Depending on if the port is registered or unregistered; you can use different types of WBR cells. The dedicated WBR cell is meant to be used on an unregistered port to ensured testability. For registered port, one may use the shared WBR cell to optimize the design.

IEEE 1500 mandates provision of a WBR cell on all core level digital terminals with the exception of the clocks, asynchronous set or reset and dedicated signals; this is to ensure isolation, control and observation at every core terminal, while the exemption allows signals such as clocks and scan inputs to be directly controlled from the SoC. [4]

2.3.5 WBY – Wrapper Bypass Register

The WBY is used to bypass the other Wrapper Data Register (WDR's). It consists of one or more register, and the simplest implementation using the D flip flop, it's a register connecting the WSI to WSO during the wrapper bypass mode (WS_BYPASS) [4]. It's basically the shortest path when the wrapper is not used but happen to be a medium before the signal arrived to the other core. The whole purpose is to reduce the test time.

2.3.6 IEEE 1500 Instructions

Internal Test, External Test and Bypass are the mandatory IEEE 1500 instructions as stated in the standard [1]. There also optional instruction, these instructions has it unique id or (called as opcode) to enable it. These instructions are shifted from the WSI to the shift register in the WIR (at least 2 bit). The instruction can be parallel or hybrid too, where parallel uses the wrapper parallel port (WPP) and hybrid uses both (WSI and WPP). Serial instruction are mostly preferable and mandatory, parallel and hybrid are optional but it reduces the test time and cost.

The mandatory instructions are:

- WS_BYPASS, allows core functional mode.
- WS_EXTEST, allows external test
- WX_INTEST, allows internal test.

The naming conventions of the instructions are based on the IEEE 1500 Std. here the first word ‘W’ represents the wrapper, then the second word ‘S’ , ‘P’, ‘H’ represents serial, parallel and hybrid respectively. The symbol ‘_’ to separate the mode and function in the naming convention, such in WS_BYPASS, means wrapper serial mode and the function is bypass. [4]

2.4 ISCAS 1989 Benchmark Circuits

ISCAS 1989 Benchmark Circuit is a set of 31 digital sequential circuits use for testing and benchmark purpose. The ’89 Benchmark Circuit is much bigger and complex compared to the previous released benchmark circuit which the ISCAS ’85 benchmark circuits.

The benchmark circuit as serves researchers for benchmark which interested in the sequential test generation, scans based test generation and mixed sequential or scan based design test generation using the scan techniques. Faults and behavior faults can be introduced (despite the benchmark circuits are sequential, synchronous and use only the D-Flip-Flops) by modifying of substituting some of the logic gates of flip flops with their appropriate function models. The standard model of the D-Flip-Flop provides a reference point that is independent of the faults particular to the flip flop implementation [8].

2.5 Stuck at Fault Analysis

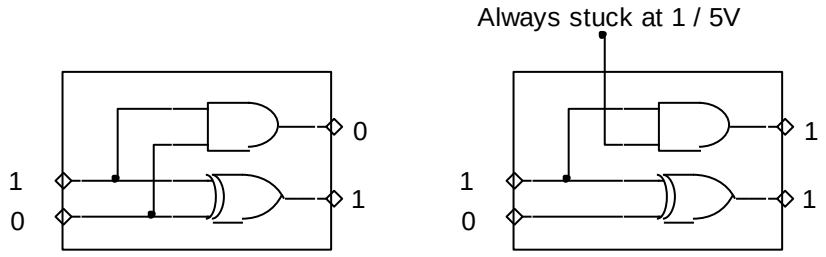


Figure 5: Stuck at Fault Model

Circuit after production would face some problem where there were wires interconnects to the HIGH logic or LOW logic. This fault are called as stuck at fault where stuck at fault at 1, wires always connected to HIGH logic even it's getting the LOW logic. While stuck at 0, wires always connected to LOW logic even it's getting the HIGH logic.

Figure 5 shows a half adder with and without the stuck at fault. When the fault happens, we observed that the output is different with the expected output (no fault output). We know that the output of the AND gate is effected, so here we know that along the path from the terminal to the AND gate, there are some stuck at fault. To check this fault we need to make sure that the fault is propagated by changing value of the input. In this case, to propagate the fault at AND gate, we need to set a HIGH logic to another pin of the AND gate. Depending on the output observed, in this case, the output should be LOW, but observed HIGH, so we can presume that there is stuck at 1 fault at the AND gate.

2.6 PERL Automation Script

PERL stands for Practical Extraction and Reporting Language. PERL is a scripting language, where the programs are written for a software environment that automates the execution whereby usually is done step by step by a human operator. PERL language it is simple (similar to C language) and powerful for regular expression. PERL is widely used in varied application such as military, manufacturing, genetics, finance, testing and etc; to process large data sets. PERL operates well in a UNIX system and Windows system, developing PERL in the UNIX system; it also can be portable to the Windows system [9] [14].

2.7 Need of Automation for Inserting the IEEE 1500 Wrapper Core

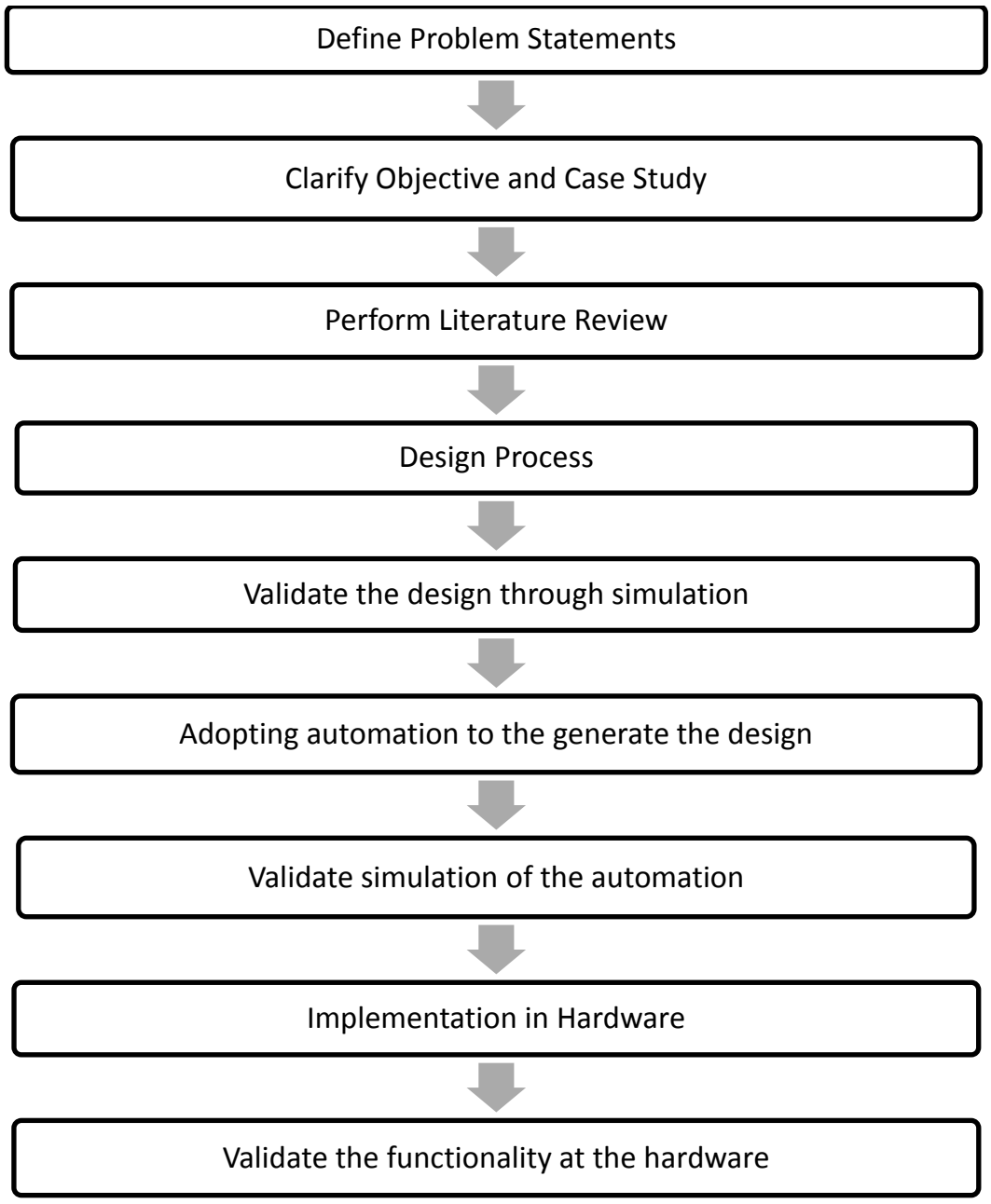
Current SoC design consist more modules (Core), inserting the wrapper manually to these unwrapped modules (Core) would consume time and cost. Automation had suggested in reducing time to insert the IEEE 1500 wrapper to each module. Current there are software such Sysnopsys [15] which assist in the automation, but it is too expensive. There are researchers that had built their automation tool in inserting the wrapper [4] [12] [13]. The automation of inserting the wrapper has been done using different script language such Phyton [2] and Tcl/Tk [8]. The design of the wrapper is mostly in VHDL [2] [7]. Here we have proposed that to build an automation inserting the wrapper and publish it open source. We also proposed that to design our automation script in Perl which is similar to C language. The wrapper is design using the Verilog HDL. This automation is compatible with modules that design in Verilog HDL because it is easy to understand (follow the structure of C language) and easy to troubleshoot (the design are in using behavior model).

CHAPTER 3: METHODOLOGY

Methodology; chapter that cover the flow of how the project will be conducted through flow chart, milestone that need to be achieved in a period of time by Gantt chart and tools and equipment that will be used in making the project successful.

3.1 Research Methodology

Method to be adopted;



3.2 Project Activities

These are the details task done throughout the project.

1. Reading and Understanding about the IEEE 1500 Wrapper.
2. Understanding each blocks in IEEE 1500 wrapper before performing the design stage
3. Designing the IEEE 1500 wrapper using Verilog HDL.
4. Validate the design through simulation using Quartus II
5. Creating an automation script using PERL for automating the generation of IEEE 1500 wrapper for ISCAS ’89 Benchmark Circuits
6. Validate the automated wrapper design using the Quartus II

3.3 Key Milestone

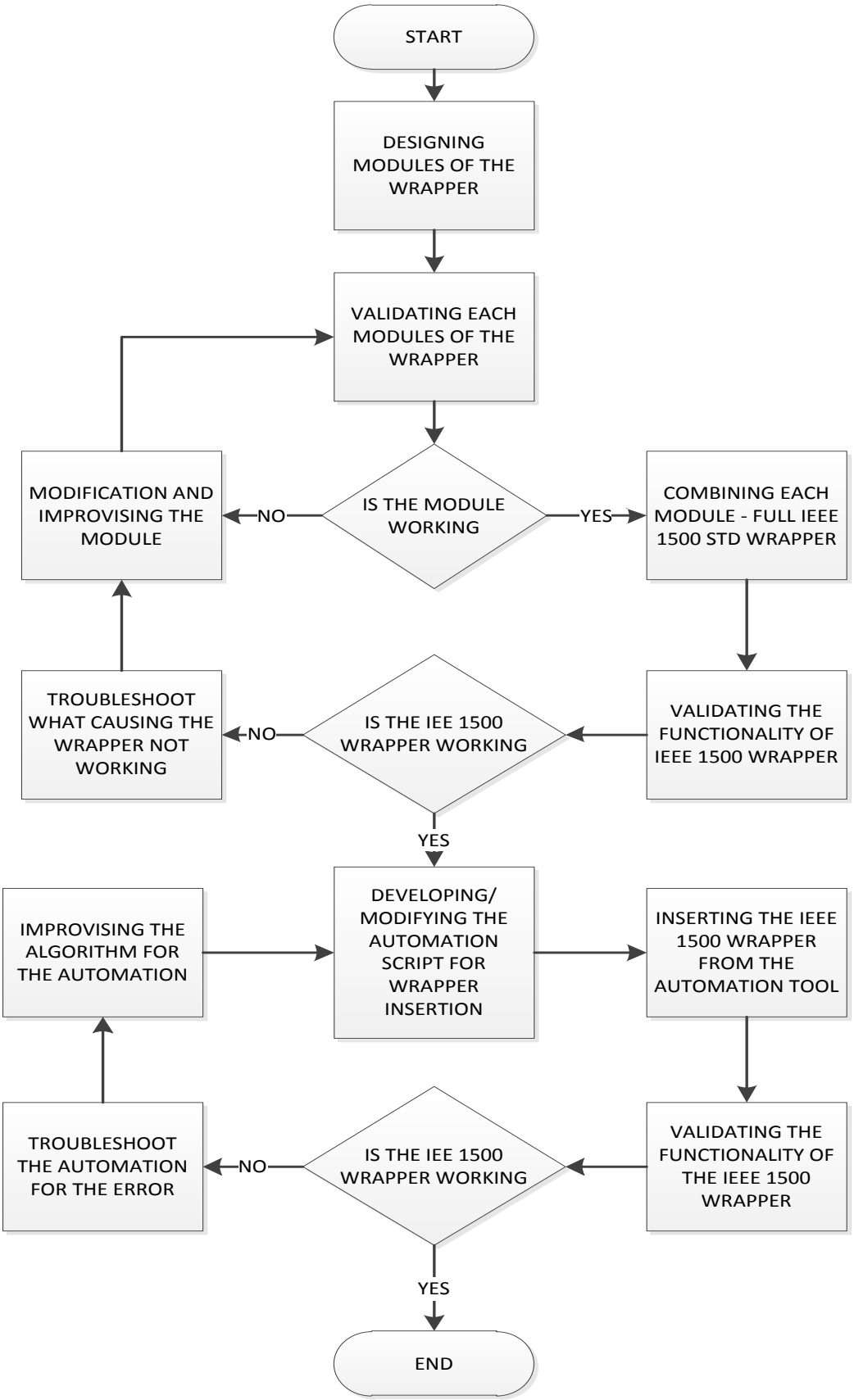
During the FYP 1 period,

- Research on the IEEE 1500 wrapper (week 1 to week 3)
- Design the wrapper using Verilog Hardware Description Language (week 3 to week 8)
- Simulate the functionality of the wrapper design using Quartus II (week 9)
- Validate the wrapper design using the FPGA (optional, week 10)
- Creating the automation script for IEEE 1500 wrapper generator. (week 10 to week 14)

During the FYP 2 period,

- Modification and improvise the wrapper design using Quartus II (week 1 – week 6)
- Validating the simulation result to check functionality of wrapper (week 5 – week 7)
- Developing the automation script for automatic wrapper design insertion. (week 8 – week 14)

3.3 Flow chart of Task Executed in Developing the Automation Script



3.4 Gantt Chart

Table 1 illustrates briefly the milestone need to be achieved versus the month frame.

Activity	FYP1				FYP2			
	JAN	FEB	MAR	APRIL	MAY	JUNE	JULY	AUG
Studying the IEEE 1500 wrapper								
Design the wrapper using Verilog Hardware Description Language								
Validate the functionality of the wrapper design using Quartus II								
Creating the automation script for inserting IEEE 1500 wrapper								
Simulating the output of automation script to validate functionality of the IEEE 1500 wrapper								
Validate the output of automation script on hardware using the FPGA								

Table 1: Gantt chart

3.5 Tools and Equipment

This project uses mostly software, and a little bit of hardware for validation. This are the tool (software and hardware) that been used.

- Mentor Graphics ModelSim
- Quratus II 4.0 Web Edition
- ActiveState Perl

3.6 IEEE 1500 Wrapper Design

3.6.1 Wrapper Instruction Register

Here in this design, the wrapper instruction register are divided to 4 main blocks, which are: WIR shift register (4 bits), WIR update register (13 bits – control signal), WIR Instruction Decode (decoding the opcode from shift register) and the WIR external circuit.

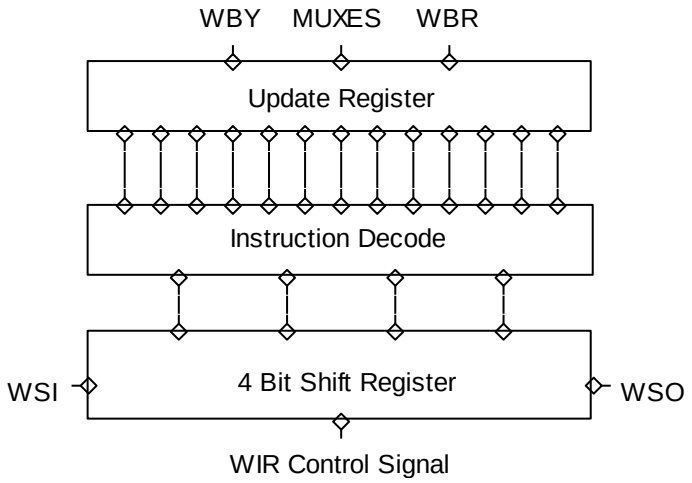


Figure 6: WIR model

In the current design, we have 4 bit shift register and 13 control signal. The 4 bit shift register is used to shift the serial input and updating the opcode to the WIR. Here this 4 bit opcode will be decoded to 13 bit control signal that controls other wrapper modules and the multiplexers.

There are several control signal which controlling the WIR process; WIR Shift, WIR Update, WIR Capture, Wrapper Select, Wrapper Clock, Wrapper Reset and the Wrapper Serial Input. When the WIR Shift is HIGH, the shift process is enabling, where the signal is shifted throughout the shift register from the WSI before it decoded and updated through the update register. When the shift process is done, it will be updated through the update register when WIR Update is HIGH. After the WIR is updated, the wrapper instantly is already in the specific test mode.

The 5 opcode that introduced in the wrapper design were the three mandatory instructions (internal test, external test and the bypass) and two newly introduced instruction in assisting the external test in reducing the test time (input bypass and output bypass).

Opcode	Instructions Mode
4'b 0001	Wrapper External Test Mode
4'b 0010	Wrapper Internal Test Mode
4'b 0000	Wrapper Bypass Mode
4'b 1100	Wrapper Input Bypass
4'b 1000	Wrapper Output Bypass

Table 2: Instruction List

The three mandatory instructions were design as followed in the wrapper handbook [7]. Two control signals were added to control addition multiplexer that design in assisting the newly two instructions (Wrapper Input Bypass and Wrapper Output Bypass).

Wrapper Internal Test (Intest) is a test to check faulty happen in the circuit/core. It is done by loading all the flip flops/registers across the wrapper boundary register (input) and the scan flip flop. After loading test value, capture process is done where the output of the circuit would be updated at the scan flip flops. After the capture process is done, the result in scan flip flop is shifted through the wrapper boundary register (output) and observed at the Wrapper Serial Output.

The wrapper input bypass and wrapper output bypass need to run on the same time to execute the wrapper external test, where the wrapper input bypass is set to the 1st core and the wrapper output bypass is set on the second core. The external test is done to test whether there are any faults in the user defined logic. Assuming the user defined logic is designed between these two cores, where it connects to the output of the 1st core and connected to input of the 2nd core as in **Figure 9**. It would take more cycle if the test stimuli are shifted through all the wrapper boundary register at the input of the 1st core and the scan flip flops. Setting the 1st core to input bypass where the test stimuli is bypass through the wrapper boundary register at the input and the scan flip

flop would save the clock cycle in loading the test stimuli at the output wrapper boundary register. Same goes to the setting the 2^{nd} core to output bypass where it bypass the scan flip flops and the wrapper boundary register at the output.

There are 13 bit control signal that generated from the WIR. These control signals need to go through wrapper external circuitry, it's a combinational logic in controlling the other wrapper modules (WBY and WBR) and multiplexers.

3.6.2 Wrapper Boundary Register

Wrapper Boundary Register is the register which controls the input and outputs of the circuit under test. It's a register that allows test mode or functional mode to be executed. All the wrapper boundary register are placed between the input and output of the circuit and the logics in the circuit. The wrapper boundary register can execute four different functions; functional, shift, capture and hold/apply. These operations are set by controlling two multiplexers in the register from the WIR.

In the design, we were focusing on the functional and shift operation. The functional allows functional operation of the Circuit/Core under Test (CuT), while the shift operation allows test vectors being shifted across the boundary registers and the scan flip flops.

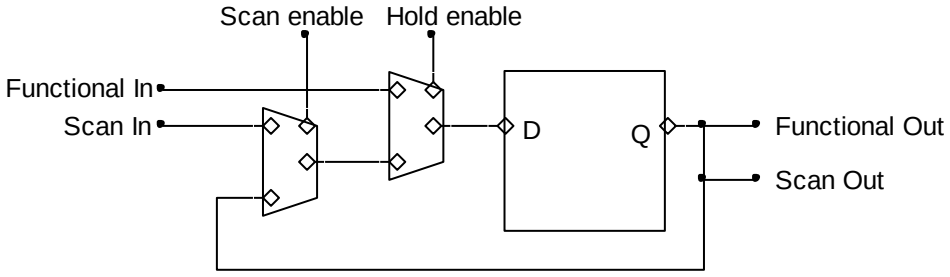


Figure 7: WBR Model

3.6.3 Wrapper Bypass Register

Wrapper Bypass Register, used as a function to bypass the circuit/core under test (CUT) if not being used. It saves time by reducing clock cycles to go through all the inputs of the circuit, scan flip flops in the circuit and the outputs of the circuit while the circuit is not used. The wrappers bypass register; consist of a d flip flop and a multiplexer to enable the bypass. To control the bypass register, control signal from the WIR would control the multiplexer in the register. Each circuit/core under test (CuT) must have a wrapper bypass register.

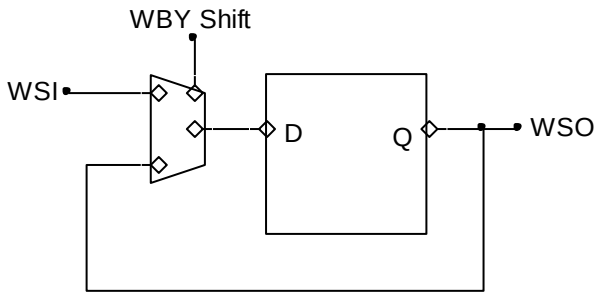


Figure 8: WBY Model

3.6.4 IEEE 1500 Verilog HDL Design

The IEEE 1500 Wrapper were design using Verilog HDL, and been validated using the Quartus II Waveform simulation tool. In the design we have break into several module of Verilog HDL in designing the IEEE 1500 Wrapper. The modules were:-

- dff1.v – D flip flop without reset
- async_dff.v – D flip flop with reset
- mux_2_to_1.v – 2 to 1 Multiplexer
- ins_dec_log.v – Instruction Decode (WIR)
- wir_update.v – WIR update register
- wir_shift.v – WIR shift register
- wir_ext_circ.v – WIR external circuitry
- wir.v – Wrapper Instruction Register

- WC_SF1_CII.v – Wrapper Boundary Register (Shared)
- wby.v – Wrapper Bypass Register
- wrapper.v – Higher level of Wrapper design

Most of the design followed as stated in the wrapper handbook [7]. There are some modifications in the design in reducing time in test. The modification was done in assisting the external test.

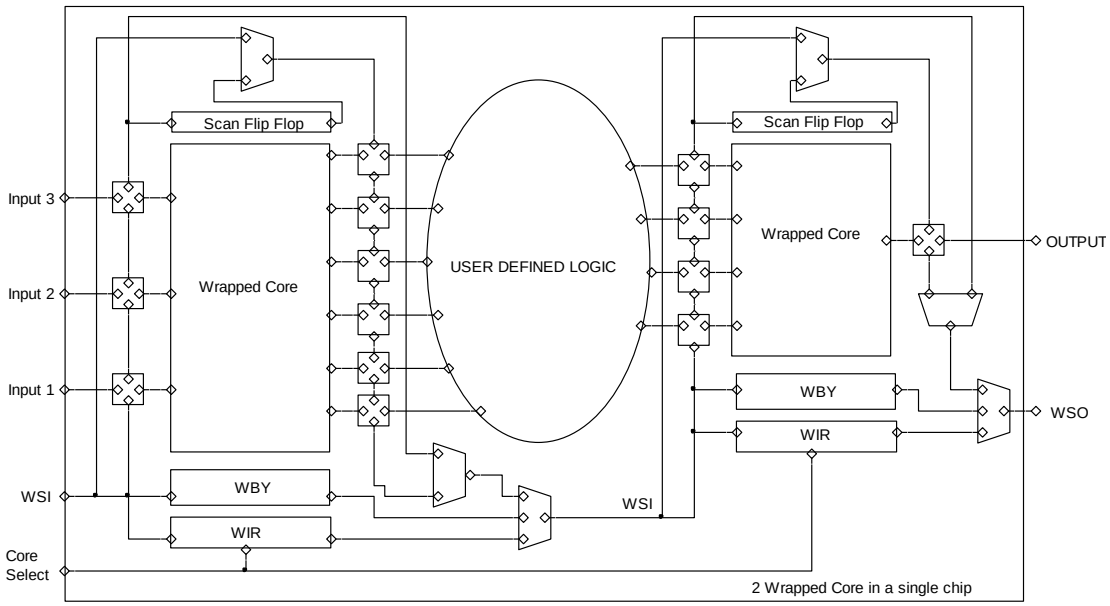


Figure 9 : The overall design of 2 Wrapped Core in a single chip

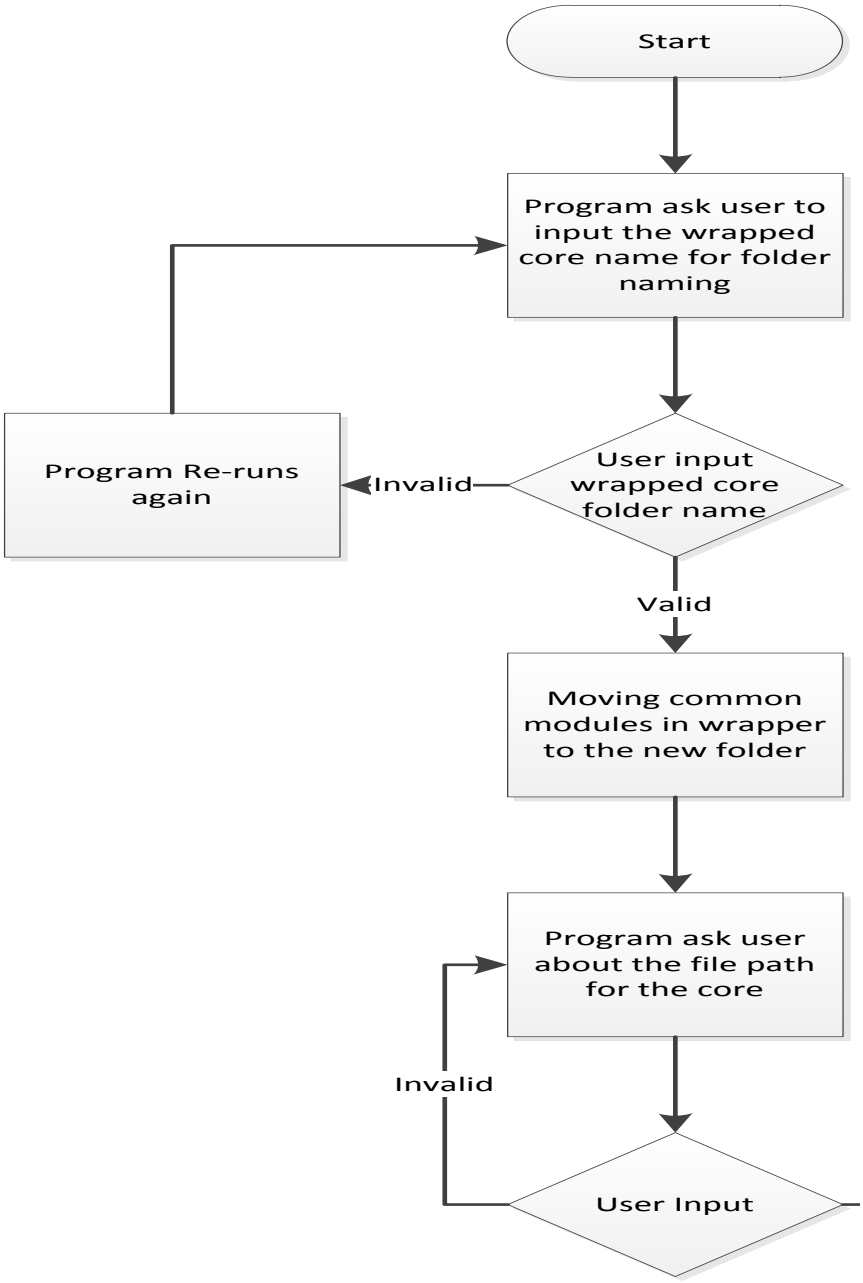
3.7 Perl Automation Script Design

The automation script in inserting the wrapper to the core was done using the Perl language and the software Active State Perl. The wrapper design was done in Verilog and been written in different modules base on its functionality in assisting the wrapper automation. The wrapper design consists of 12 modules with the wrapped core for full complete design of the wrapped core. 10 modules of the wrapper are similar in inserting the wrapper to any cores. The higher level of the wrapper design is the main part need to edit in wrapping the specific core. Information that needed in the automating the insertion of wrapper on the core are the inputs and outputs terminal of the core, the instantiation of the core, the clock terminal of the core, scan in terminal and the scan enable terminal. Information will be provided in the automation, user just need to select which is not belong to the group of information.

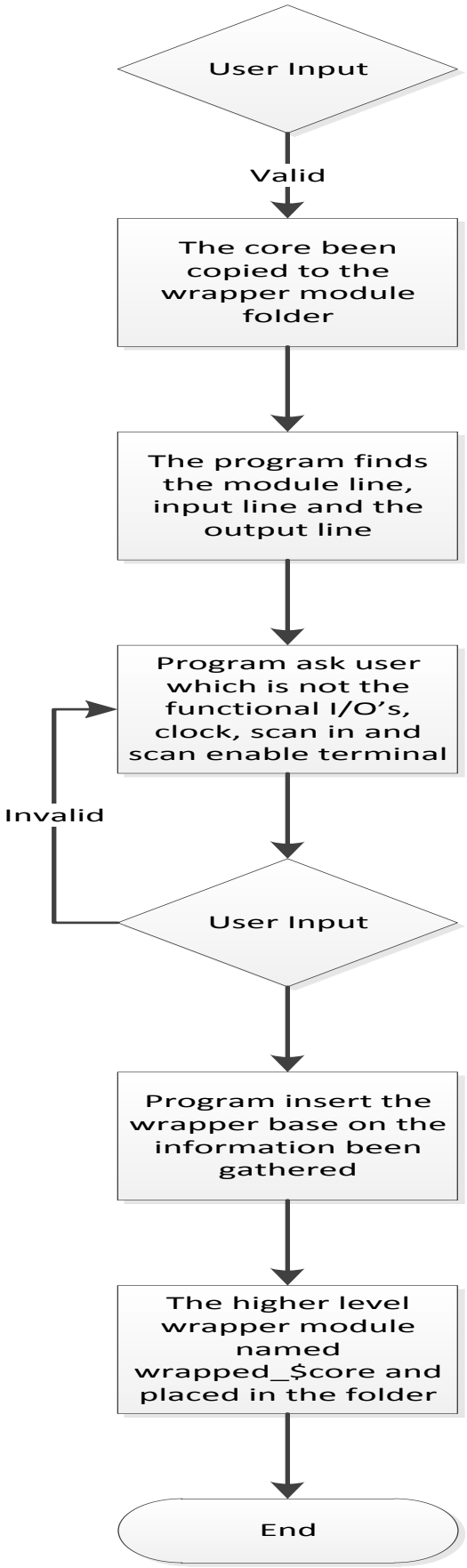
The flowchart on automating the IEEE 1500 wrapper insertion shown next page.

```
#module list
$end_module = shift (@symbol_module);
@core_module = @core_verilog[($start_module)..$end_module];
$join_module = join ("",@core_module);
$module_2nd = $join_module;

#input list
$end_input = shift (@symbol_input);
@core_inputs = @core_verilog[($start_input)..$end_input];
$join_inputs = join (" ",@core_inputs);
@input_1st = split (" ",$join_inputs);
shift(@input_1st);
$input_2nd = join ("",@input_1st);
@input_2nd_ar = split (';',$input_2nd);
@input_3rd_ar = split (',',$input_2nd_ar[0]);

#output list
$end_output = shift (@symbol_output);
@core_output = @core_verilog[($start_output)..$end_output];
$join_output = join (" ",@core_output);
@output_1st = split (" ",$join_output);
shift(@output_1st);
$output_2nd = join ("",@output_1st);
@output_2nd_ar = split (';',$output_2nd);
@output_3rd_ar = split (',',$output_2nd_ar[0]);

#input length
$input_length = $#input_3rd_ar + 1;
```

Figure 10: Illustration of the Perl Code in determining the instantiation of the core



Continue on next page;


User Input
Valid
The core been copied to the wrapper module folder
The program finds the module line, input line and the output line
Program ask user which is not the functional I/O's, clock, scan in and scan enable terminal
Invalid
User Input
Program insert the wrapper base on the information been gathered
The higher level wrapper module named wrapped_$core and placed in the folder
End

CHAPTER 4: CASE STUDY, RESULT AND DISCUSSION

4.1 Case Study

Throughout the process of constructing the automation wrapper design, we were designing the wrapper, for the use of automation in the future. In other to validate the functionality of wrapper, we have chosen sequential circuit from the ISCAS '89 Benchmark Circuit, s27 and s298 as circuit under test. These wrapper design and the s27 and s298 were in Verilog HDL, these wrapper design at 1st were done manually, in this design, modification been made while the functionality maintain the same for the purpose of ease in designing the automation script for wrapper automation insertion design.

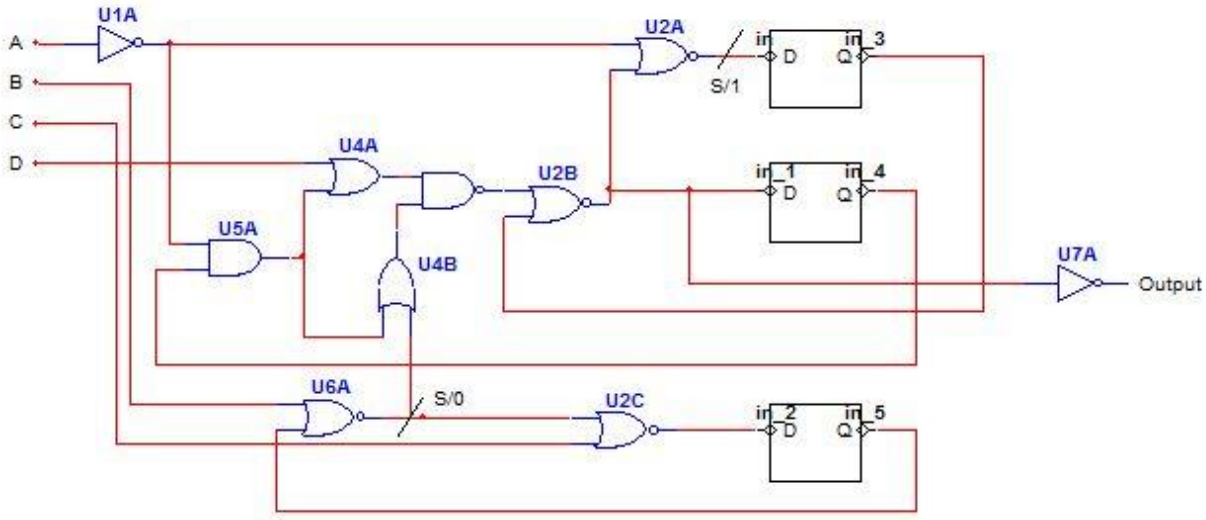


Figure 11 : S27 Sequential ISCAS'89 Benchmark Circuit

The wrapped s27 sequential benchmark test was validated by introducing two stuck at fault as **Figure 11**. During the validation process, we compared the wrapped core without stuck at fault and with stuck at fault to verify the functionality of the wrapper. This stuck at fault were tested for the Wrapper In test Mode, which to validate faulty happens in the circuit under test (CUT).

Wrapper In test mode, were done by inserting the scan test input to the Wrapper Serial Input (WSI) for setting the value for all the input boundary cells flip flops and the internal circuit flip flops. After setting the value of the flip flop, capture mode been done, where the new value updated base on the setting that been set in the

previous clock cycle to the internal flip flops and the output boundary cells. The capture mode is done in a single clock cycle. The scan out been observed at the Wrapper Serial Output (WSO). The output that been observed are the value updated during the capture mode, at the boundary cells and the internal flip flops and the setting value at the input boundary cells. The amounts of clock cycles are based on the number of flip flops used during the test and a single cycle of capture mode.

4.1.1 Wrapper In Test

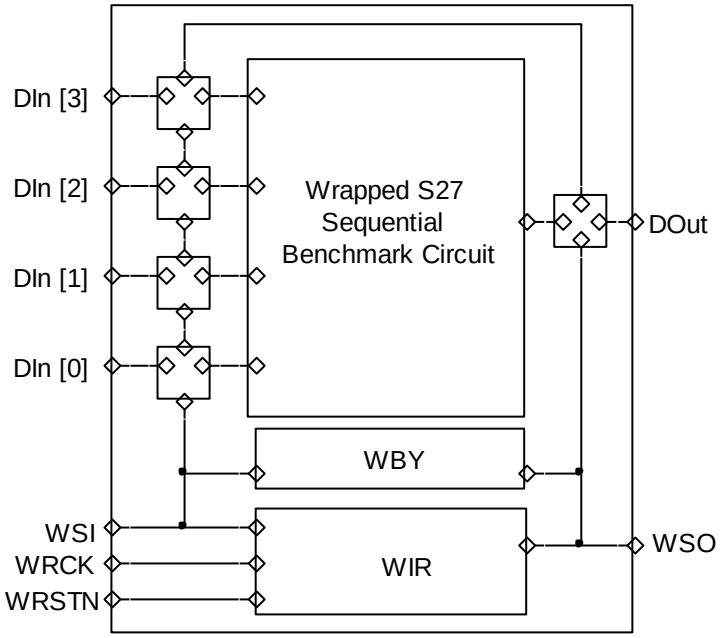


Figure 12 : Illustration of Wrapped 'S27 Sequential Benchmark Circuit'.

Figure shows the basic structure of the wrapped circuit under test. This design been used to validate the wrapper in test mode. The basic operations were scan in, capture and scan out. Before the In Test mode runs, we have to set the WIR to the wrapper In Test operation. To set the mode of WIR is done by shifting the input at the WIR. After setting the mode of operation, we need to set the value of internal flip flops and input boundary cells flip flop. To do so, we need to shift the input from WSI, to the boundary cells and the internal flip flops. In this case, we have 4 inputs and 3 internal flip flops, so we shift the input from WSI for 7 cycles (for 7 flip flops) than 1 cycle

for the capture mode, during the capture mode, at the WSO, we already can observe the value of output boundary cells. During the scan out, 7 clock cycles, where the 1st three clock cycles represent the captured value or the updated value during the faulty test and the other 4 clock cycles are the input that we have set.

4.1.2 Wrapper Ex Test

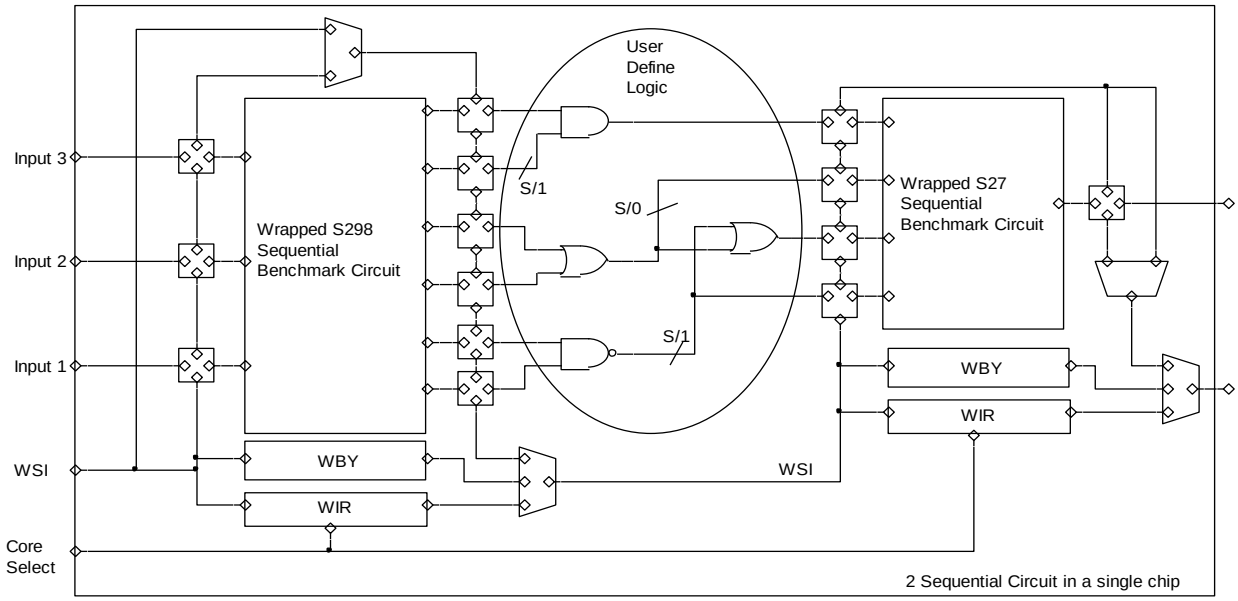


Figure 13 : Illustration of Wrapper Ex Test Design with 2 CUT

Here in the wrapper ex test mode, we have introduced S298 sequential ISCAS '89 benchmark circuit, the reason choosing s298 sequential circuit, is to have more output than the s27 sequential circuit inputs. Reason is to easy design the User Defined Logic (UDL) and stuck at fault placement. Before the validation of Wrapper Ex Test has been done, we have introduced three stuck at fault in the UDL as shown in **Figure 13**. During the validation we have tested the operation during no fault and faulty condition.

As we see in **Figure 13;** the design for the wrapper with 2 Circuit under Test (CUT) in a single chip, where it almost represent the similar concept of SOC. In the Ex Test Mode, the flip flop that are important are the output boundary cells of S298 CUT which is the input of the UDL and the input boundary cells of S27 CUT which is the output of the UDL. In the Ex Test mode, we are trying to test faulty that happen in the UDL. Setting the value in output boundary cells of S298 CUT and capturing the updated value at the input boundary cells of S27 CUT.

Here to set the Wrapper Ex Test Mode to operate, we need to set two wrappers with their own mode. For S298 CUT wrapper are set to the 'input and internal bypass mode', where it bypass the input from WSI to the boundary cells and the internal flip flops, it just loaded and shifted through the output boundary cells, where in this case for S298 CUT which has 14 internal flip flops and 3 input boundary cells, which we can save about 17 clock cycles for loading the value of output boundary cells. While in the S27 CUT wrapper are set to 'internal and output bypass mode'; where it bypass the internal flip flops and output boundary cells. Here in S27 CUT have 3 internal flip flops and 1 output boundary cells. Which in this mode, it can save up to 4 clock cycles. Overall it can save up to 21 clock cycles. To select the wrapper, we can set at the core select pin.

In this validation, we need to test for faulty UDL, as shown in **Figure 13;** we have introduced 3 stuck at fault. This stuck at fault will be test to verify the functionality of the wrapper.

4.1.3 Automation of IEEE 1500 Wrapper

The automation design using Perl scripting language, this language is almost similar to C language. The idea of automation of the wrapper possible since the design of wrapper involve in the core select and its route and the boundary register for each functional input and output. Other components of the wrapper are the same, since the design and meant for plug and play purpose. The illustration during the automation is running shown in Figure 14. In validating the automation script is working fine, the manually inserted wrapper has been compared with the automatically inserted wrapper. If the comparison is ~99% to 100% similar using the special tool in the Notepad ++, means the automation is working fine, since the manually inserted wrapper had been validated its functionality.

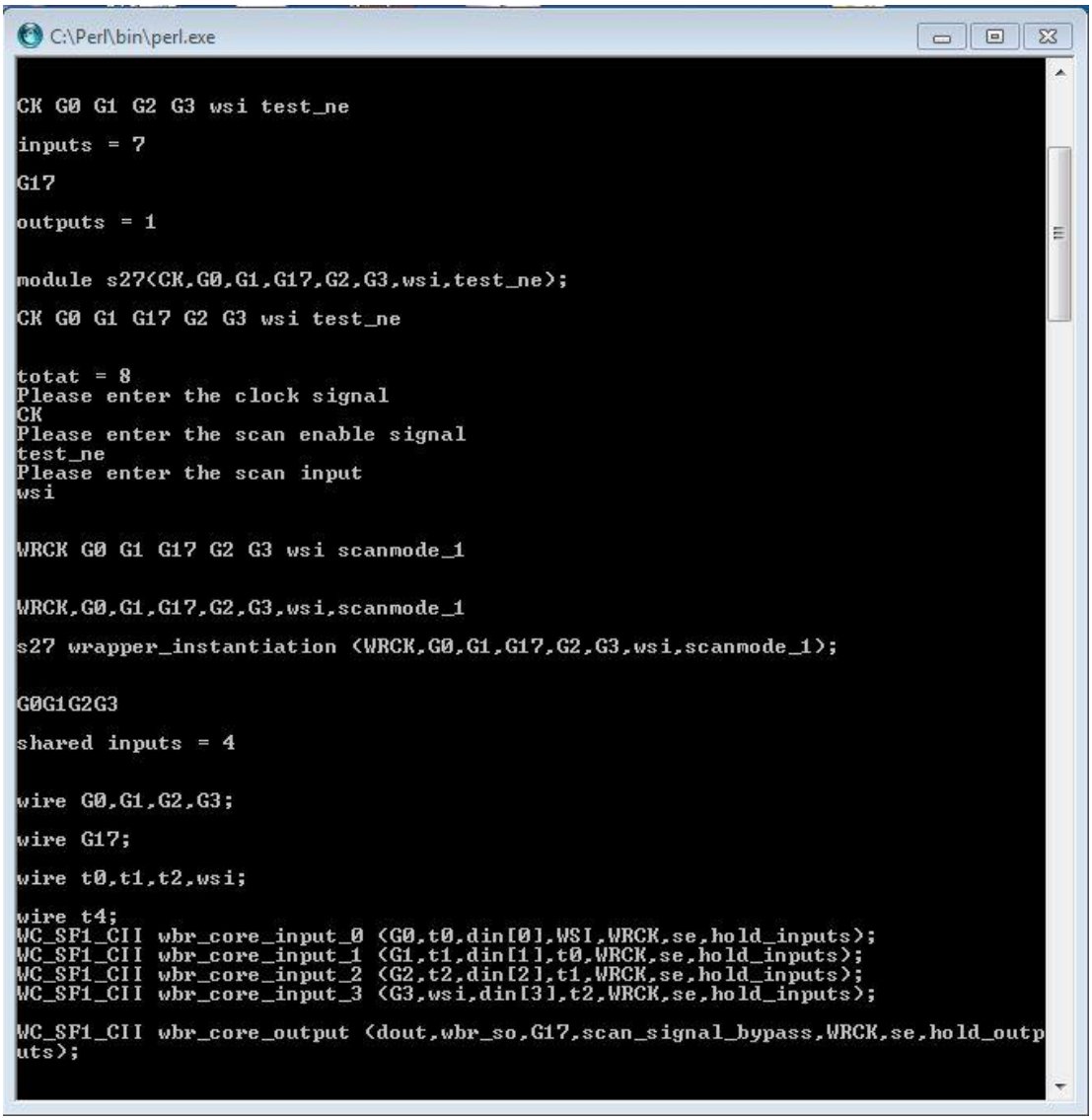


Figure 14 : Illustration during the automation is running

4.2 Results and Discussion

Throughout the design, we have validated the functionality of the wrapper for In Test and the Ex Test. This validation is done by comparing the expected result and the observed result during the free fault and fault test. The circuit and the fault that been introduced are shown in **Figure 11** and **Figure 13.** This fault is purposely set to high or low base on their respective stuck at fault at that point.

4.2.1 Wrapper In Test

In this mode, we have faulty and not faulty test, here there are two time of test been done, where we test only one fault for the 1st scan in, then for the second 2nd scan in were tested for 3 stuck at fault. The illustration shown in **Figure 15**,

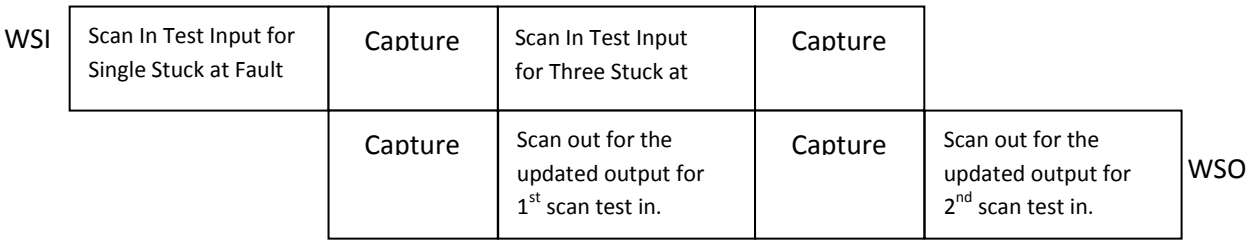


Figure 15 : Illustration on Scan In (WSI) and Scan Out (WSO)

Referring to the **Figure 11,** the stuck at fault (S/1) after the U2A Gate, is the first fault that been tested. Then, the three stuck at fault been tested in the second scan in test input.

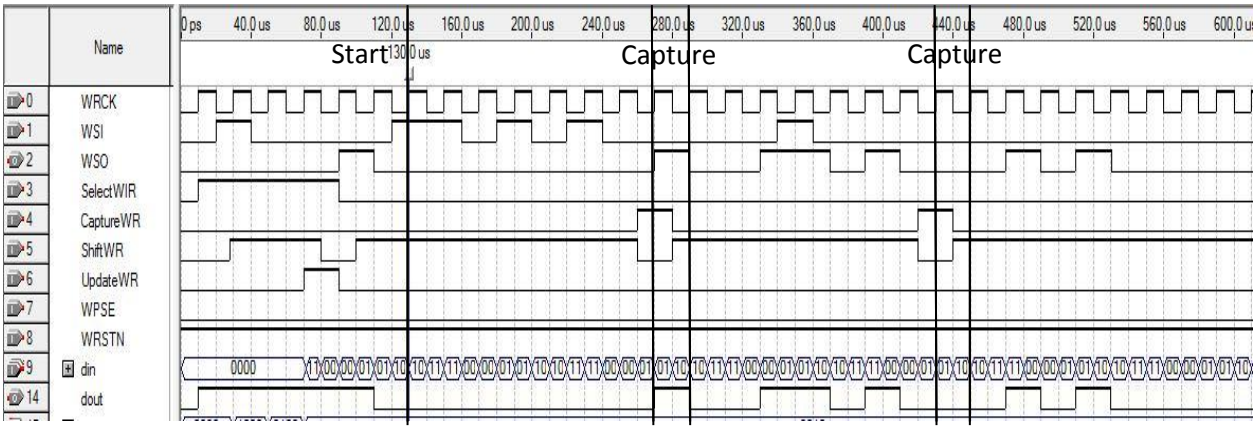


Figure 16 : Waveform form Wrapper In Test Simulation Fault Free

As observed in **Figure 16**, there are 7 clock cycle in each interval of capture, where represent all the 8 flip flops that involved. During the capture cycle, the output of WSO are the same as the output of output boundary cells. In the **Figure 16** waveform

are fault free CUT. At the 1st scan out the observed data are 10011010, the 2nd scan out the observed data are 00101000.

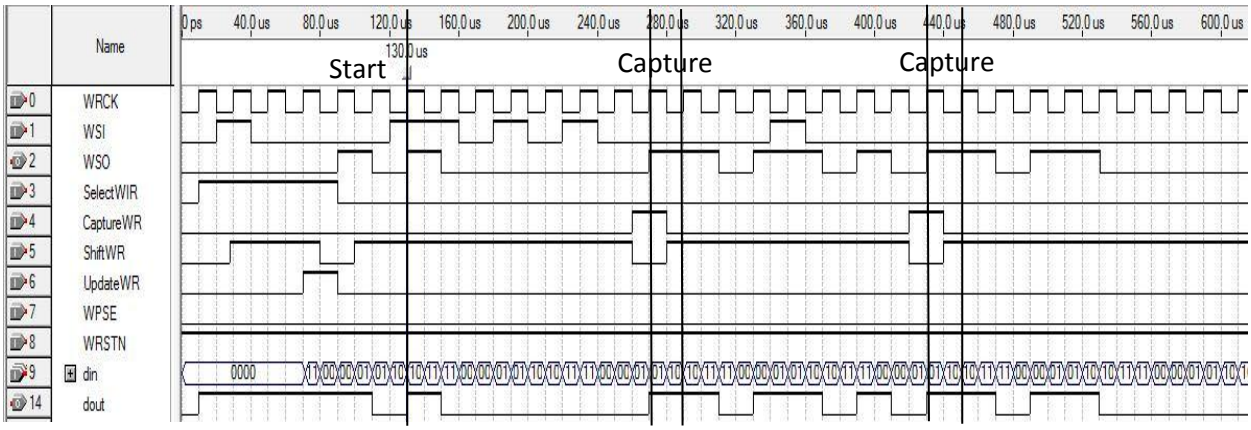


Figure 17 : Waveform form Wrapper In Test Simulation with Fault

In **Figure 17**, the waveform shows the faulty as we observed the 1st scan out, 11011010 and the 2nd scan out 11011000. Here we can observed or conclude that during the single stuck at fault during the single fault, the effected flip flops were the 1st flip flop shown in **Figure 11** (comparing 10011010 (no fault) with 11011010 (with fault)). By comparing the no fault and with the faulty one, we can know the path that causing the fault.

4.2.2 Wrapper Ex Test

As discuss earlier, wrapper ex test mode is to test the fault of the user define logic between the 2 CUT's. Here as shown in, **Figure 13,** there are 3 faults been introduced during the ex-test mode, here we have simulated the design, and observed the free fault and with fault waveform, to verify the functionality of the ex-test mode.

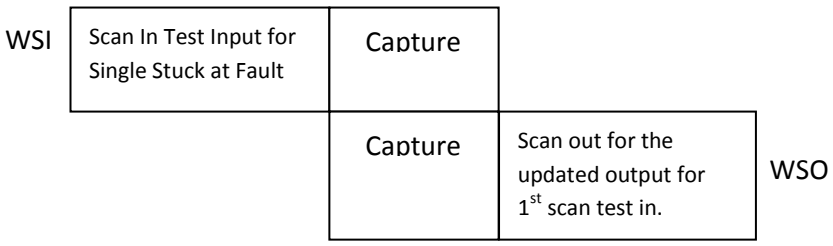


Figure 18 : Illustration on Scan In (WSI) and Scan Out (WSO)

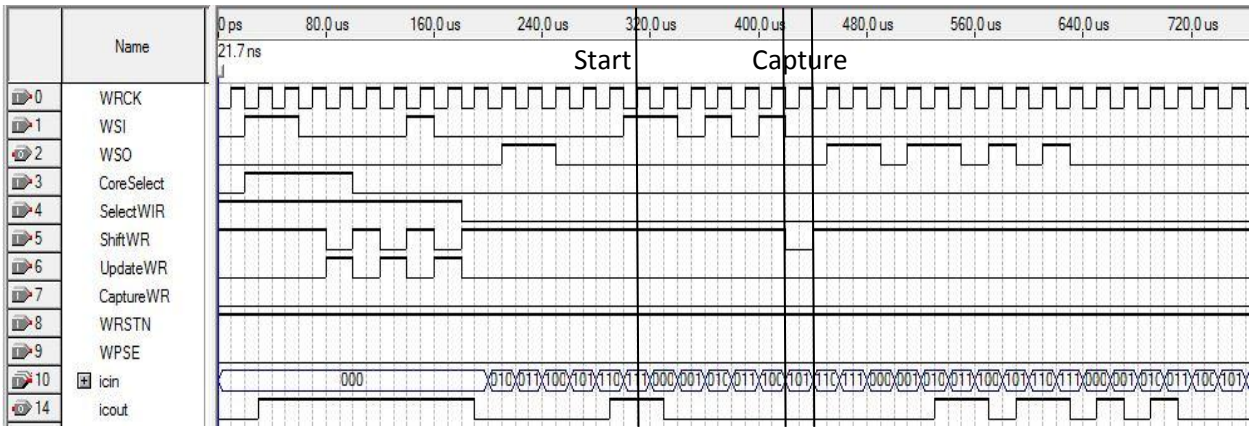


Figure 19 : Waveform form Wrapper Ex Test Simulation with Fault Free

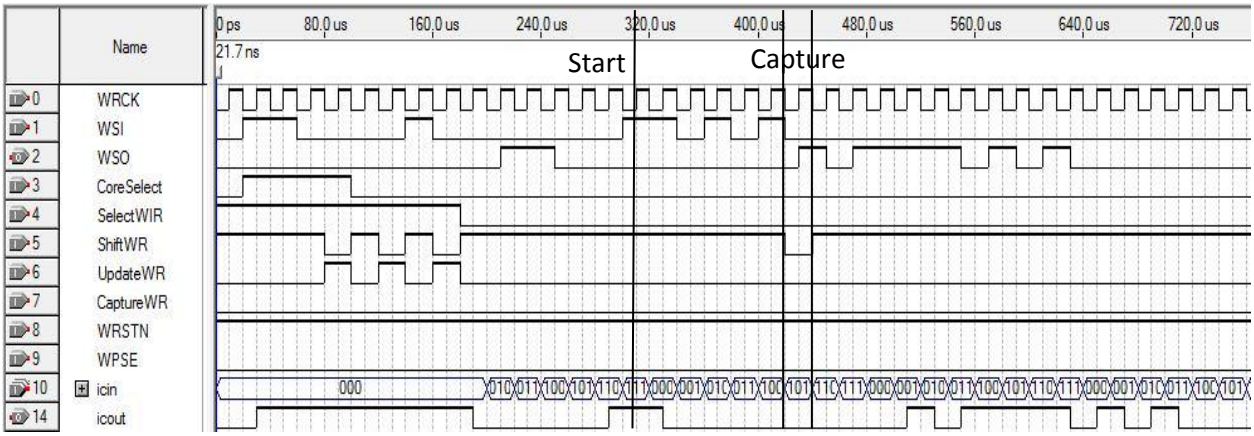


Figure 20 : Waveform form Wrapper Ex Test Simulation with Fault

The test was done by inserting scan input, then the capture cycle, then the scan out operation. In ex test, from the design, we have 6 clock cycles (6 output boundary cell) and during the scan out, we will be observing the updated value in the input boundary cell of S27 sequential benchmark circuit. The test input is 110101 for the fault and free fault case. Here the output that observed in free fault is 0110|110101, here the separation means, 0110 is the output value of the UDL while 110101 is the input that the user set for the UDL. For faulty case, the output, 1011|110101, means they were fault happen at the path leads to these 3 pins (comparing 0110 and 1011). The fault that tested in both Ex Test and In Test were verified. From these output, we can tell that the functionality of the wrapper valid.

CHAPTER 5: CONCLUSION AND RECOMMENDATION

In this thesis, we first proposed the needs of automating the IEEE 1500 wrapper insertion to the cores due to difficulty faced by test researchers. The IEEE 1500 wrapper core design has been done using Verilog and been validated using Quartus II simulation. During the IEEE 1500 wrapper validation, we found the design (which follows protocol and the wrapper handbook) are not efficient during the external test mode. Some modification had been done to increase the efficiency of the time to test (reducing clock cycles) during the external test. The overall wrapper design is made module base in assisting the development of automation script in inserting the wrapper. The automation script is done using the Perl scripting language.

The automation script have some limitation during the automation process, the automation can wrapped core with Verilog design, the core need to have shared or functional input outputs, need to have internal test enable pin and internal test in signal. The automation has tested to wrapped core with more than 30 input and outputs.

For future design of the automation script, the automation would overcome problem such the automation more flexible, the automation is done in graphical interface where it would be user friendly and editable. The validation need to be done in full SoC implementation and analyze using the CPLD/FPGA and the logic analyzer.

CHAPTER 6: REFERENCES

[1] IEEE standard 1500, Testability Method for Embedded Core-based Integrated Circuits. IEEE, 2005

[2] System on a Chip – Wikipedia, the free encyclopedia. [Online], February 2012. Available: http://en.wikipedia.org/wiki/System-on-a-chip

[3] Modular Design – Wikipedia, the free encyclopedia. [Online], February 2012. Available: http://en.wikipedia.org/wiki/Modular_design

[4] Niklas Huss, Automating IEEE 1500 wrapper insertion. Master's thesis, 2009. Available: http://liu.diva-portal.org/smash/get/diva2:282026/FULLTEXT01

[5] Teressa, L. McLaurin, IEEE Std. 1500 Compliant Wrapper Boundary Register Cell, 2005

[6] IEEE 1450.6 Core Test Language (CTL), February 2012. Available: http://grouper.ieee.org/groups/ctl/

[7] Francisco da Silva, Teresa L. McLaurin, and Tom Waayers. The Core Test Wrapper Handbook – Rationale and Application of IEEE Std. 1500™, Springer, 2006. ISBN-10 0-387-30751-6

[8] F Brglez, D Bryan, K Kozminsk, Combinational Profiles of Sequential Benchmark Circuits. In IEEE International Symposium Circuit and Systems, May 1989. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=100747

[9] PERL. [Online], February 2012. Available: http://en.wikipedia.org/wiki/Perl

[10] Wrapper Diagram [Online], March 2012.

Available: http://courses.cs.tamu.edu/cpsc680/walker/Slides/lecture4.ppt

[11] 3 Reds Lights of Death [Online], March 2012. Available: http://en.wikipedia.org/wiki/3_Red_Lights_of_Death

[12] S. Mlkhtonyuk, M. Davydov, R. Hwang and D. Shcherbin. IEEE 1500 Compliant Test Wrapper Generation Tool fo VHDL Models.

[13] B. Mullane, M Higgins and C. MacNamee. IEEE 1500 Core Wrapper Optimization Techniques and Implementation

[14] Perl Tutorial [Online], July 2012.
Available: http://www.perltutorial.org/introducing-to-perl.aspx

[15] "Sysnopsys's Tetramax" [Online], July 2012.
Available: http://www.sysnopsys.com