SELF TESTING RISC PROCESSOR USING JTAG

A DISSERTATION
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF
MASTER OF TECHNOLOGY
IN
VLSI DESIGN AND EMBEDDED SYSTEMS

Submitted by:

HARSH (2K20/VLS/05)

Under the supervision
of

Dr. MALTI BANSAL

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

MAY, 2022

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, HARSH (2K20/VLS/05) student of M.Tech (VLSI Design and Embedded Systems), hereby declare that the dissertation report titled "**SELF TESTING RISC PROCESSOR USING JTAG**" which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: DELHI **HARSH**

Date: 31/05/2022

ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the minor Report titled "SELF TESTING RISC PROCESSOR USING JTAG" which is submitted by HARSH, 2K20/VLS/05 of Electronics and Communication Department, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: DELHI **SUPERVISOR**
Date: 31/05/2022 **Dr. Malti Bansal**

ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

A successful project can never be prepared by the efforts of the person to whom the project is assigned, but it also demands the help and guardianship of people who helped in completion of the project.

With profound sense of gratitude, we thank Prof. Malti Bansal, my supervisor, for their encouragement, support, patience and their guidance in this minor work. We take immense delight in extending my acknowledgement to my family and friends who have helped me throughout this project work.

HARSH
M.Tech (VLSI AND EMBEDDED SYSTEM)

ABSTRACT

Modern technologies are built to solve complicated issues in real time and it seems like the IC only includes the circuitry for the dedicated logic but there is some circuitry that is added which is equally important which we called testing circuitry. This kind of circuits are basically used to test the IC in less time and fabrication process. All the work, comes under the profile DFT (Design for Testability). DFT also works on making a design, a self-testing design because there are many devices in the field which needs to be test after a particular time regularly like radar system of automobiles. The pros of DFT is that it eases the testing process of a design in general but the cons are that because of the logic, hardware requirement increase which leads to rise in two factors on which whole VLSI industry focusing on: Area and Power. The DFT logic increase these two factors which is not preferable but there is a trade-off between testing time, area and power because the end goal is to get a efficient device in less price.

This project is focused on providing a basic knowledge about Design for Testability (DFT). Then, we are going to implement same methodology studied in RISC processor and make it a self-testing processor. At last, we also going to study the implement design in AMD Xilinx VIVADO 2020.2 and perform a power analysis on the basis of which we will conclude the result.

LIST OF FIQURES

LIST OF TABLES

LIST OF ABBREVIATIONS

- **DFT –** Design for Testability
- **RISC –** Reduced Instruction Set Computer
- **JTAG –** Joint Test Action Group
- **DUT –** Design Under Test
- **SI –** SCAN_IN
- **SE –** SCAN_ENABLE
- **IR –** Instruction Register
- **DR –** Data Register
- **TDI –** Test Data Input
- **TDO –** Test Data Output
- **HDL –** Hardware Description Language
- **RTL –** Register Transfer Logic
- **PC-** Program Counter
- **IF –** Instruction Fetch
- **EX –** Execution
- **WB –** Write Back
- **NPC –** Next Program Counter
- **SA –** Stuck At
- **BSR –** Boundary Scan Register
- **TMS –** Test Mode Select
- **SD –** Scan Data
- **DM –** Data Memory
- **LPG –** Low Power Gating
- **CM –** Channel Masking

TABLE OF CONTENT

CHAPTER 1: DESIGN FOR TESTABILITY

1.1 INTRODUCTION

The demand for chips with more functionality and reduced size is increasing day by day because of this the work of VLSI engineers becoming more complex in aspect of every profile. As we know that in the designing of an IC from architecture to silicon, verification is the most crucial part which is on the highest priority to be taken care of. According to the Application Specific Integrated Circuit (ASIC) Design Flow, after designing the logic on the Verilog we go the verification of the logic and the loop between the both continues until the verification team gives a green signal. After that we create the netlist from the verified logic design in the Synthetization.

Now, the netlist only defines the logic of the required IC but before going for the physical design flow which basically include the conversion of the netlist into physical gates; let's take an example. Let say the IC we designed till yet is going to be used as a radar system IC in a car. So, basic requirement is that it should sense the distance every time when the car is moving but let say because of heat or an impact the chip got damaged and the data is giving is wrong. Then in this case the damage could also cost someone's life. So, it is very important that the chip should have the capability to test itself when the car starts and after every defined interval. Now, in order to make the chip testable we need to add some logic in it and the work of adding logic that will test a chip in the field is done by DFT or Design for testability team. So, role of DFT in ASIC design flow is to add some logic to the design which will reduce the testing time which leads to reduction in the cost of the IC.

Design for Testability (or DFT) is a term used to describe design processes that include testability characteristics. DFT makes it easier to develop and apply manufacturing tests for the design hardware. DFT is not only provide the ease in testing an IC but also have another role of making a design, a self-testing design in the cost of increase in space and will require more power to test the IC. Also, DFT focus on the defect that cause because of the fabrication process so what DFT do is add the testing circuit into the design and generate some golden responses according to some patterns. Now, after the fabrication process same patterns are applied to the physical chip and the output is match with golden responses. If they match then the chip is fine but if not because of the patterns they applied, it not difficult to find which part of the chip have defects.

As shown in above figure, if we try to test a 32-Bit adder it will approx. take 5850 years to get tested when we apply the input at the frequency of 100 MHz that's why the designer major focus is on to reduce the testing time because if this thing is not been taken care off, it will increase the cost of the chip exponentially.

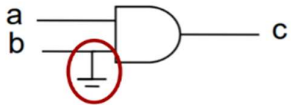
1.2 BASIC TERMINOLOGIES

Defect - The inadvertent discrepancy between the implemented hardware and its planned design is referred to as a defect. Defects might develop during the manufacturing process or when the IC is in operation.

Fault is a way of defining defect on the paper or we can say it is a different form of defect but in abstraction level.

Error is a wrong output signal produced by a defective system. Example:

Defect: Short to ground
Fault: Signal b stuck at logic 0
Error: a=1, b=1, c=0 (correct output c=1)



Fault Model is an engineering representation of something that could go wrong in the production, development, or operation of the designed hardware. Types are as follows:

- Stuck-at Faults.
- RAM/ROM Seq. Faults.
- Delay (Transition) Faults.
- Cell Aware Faults.
- Transistor Open and Short Faults (IDDQ).
- SDD Faults.
- Memory Faults.
- Bridge faults.
- Path Delay Faults.

Types of Testing: In a design, there are many parts which we cannot test together. For example: I/O pins and the core logic, it is not possible to test them together. Because of this reason we have categorized the testing as follows:

- Scan Test
- StuckAt (Slow Frequency – 10 to 50 MHz)
    - At-Speed (Functional Frequency)
    - Path Delay (Critical paths will be tested)
    - IDDQ (Leakage test)
- Boundary Scan testing
- Memory Testing.
- IP testing.
- Stress Test (Burn-in)

We are performing boundary testing in the project of this thesis.

Boundary Scan Testing: This type of testing is primarily focus on the Input and output pins of the module. Let say, we fabricate the whole design and because of the process one of the input pins gets connected to the ground. Now, the logic input it will give to the core design is logic '0' no matter what input we are providing from outside. So, it is very important to check both input and output pins. In order to accomplish this kind of testing, both inputs and outputs are required to be controllable and observable.

Controllable means that a particular pin or ports input can be changes directly or through minimum logic by the designer and observable means that a particular pin or ports values are under the observation of the designer or tool. Now, for performing scan boundary we connect the outer input pins with inner core input pins of module via scan flop same connections are performed on output side. These scan flops are also connected each other in the form of a register. The same testing, we are going to perform in this thesis.

Scan Insertion: The concept of a scan chain testing is based on a detecting a known set of manufacturing defects (or faults) in silicon. To illustrate the scan chain concept, we focus on detecting simple faults such as shorts and opens.

A Scan Flip-Flop is shown in Figure 1.2. At the flip flop-input, a multiplexer is used, with one input of MUX working as the functional input D and the second as Scan-In (SI). The Scan Enable (SE) signal controls which mode is selected among D and SI.

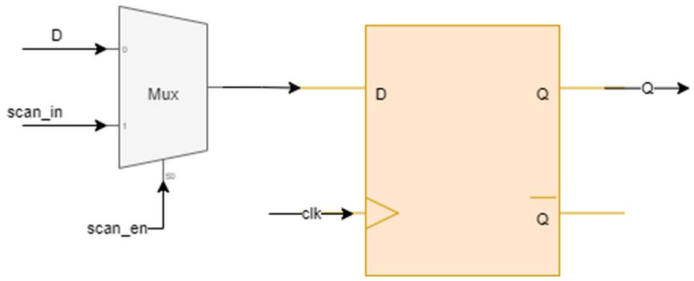


Fig 1.2 Scan flipflop (Reproduced from [33])

All of the flops are connected in a chain that acts as a shift register, with the basic block being a simple Scan Flip-Flop. The first flop in the scan chain is connected to a scan-in port, while the last flip is connected to a scan-out terminal. In Figure 1.3, the clock signal is depicted in red, the scan chain in yellow, and the functional path in black. A scan test is performed on the combinatorial logic unit to identify any manufacturing faults. To do so, the ATPG tool attempts to stimulate every node inside the combinatorial logic unit by using input vectors at each scan chain's flops.

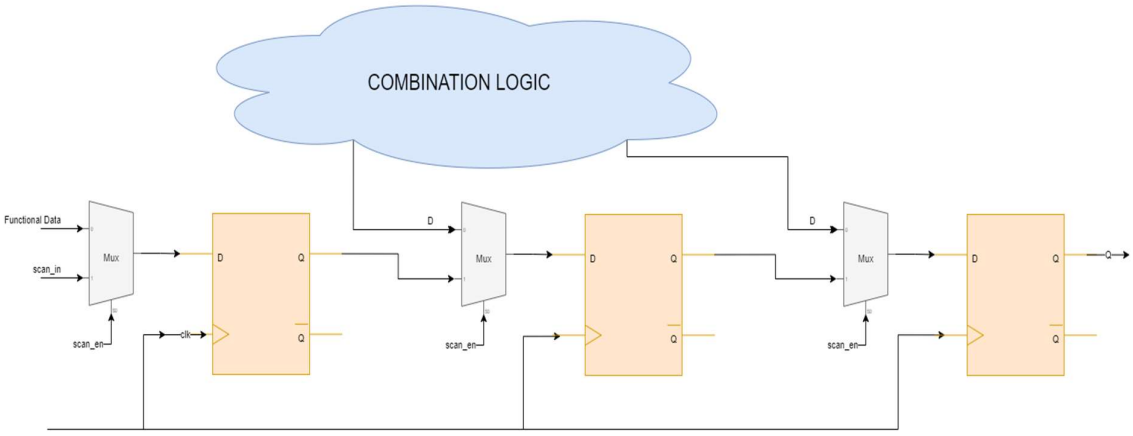


Fig 1.3 Structure of scan chain (Reproduced from [33])

Scan-in, Scan-capture, and Scan-out are the three steps of the scan chain. Moving in and loading an input vector into all of the flip-flops is called scan-in. Similar to shift register, data travels from a flop's output to scan-input of its subsequent flop during scan-in. After loading the sequence, single clock pulse (also known as the capture pulse) is permitted to activate the combinatorial logic unit, and the result is caught at the subsequent flop. The signature is matched to the predicted signature when the data is pushed out. The collected sequence can be used as the second input vector in the shift-in loop by modern ATPG technologies. Furthermore, in the event of a mismatch, these can indicate to the nodes as to where a manufacturing flaw can be found. The events of scan-shifting with scan-capture are depicted in Figure 1.4

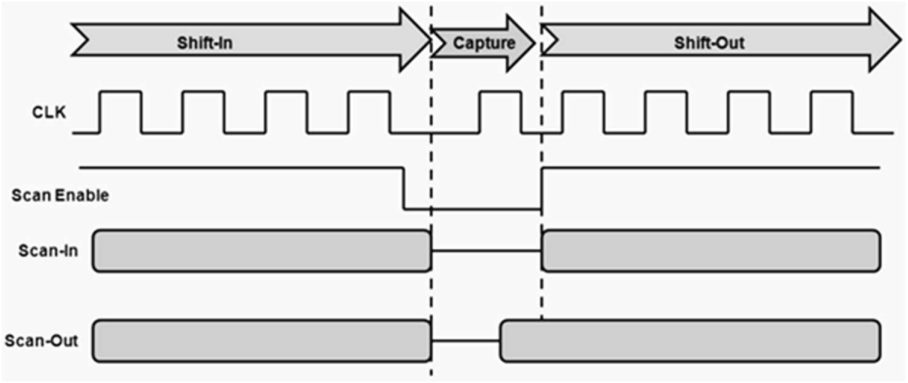


Fig 1.4 Waveform indicating scan data insertion (Reproduced from [33])

1.2.1 Scan Chain Operation for StuckAt Fault

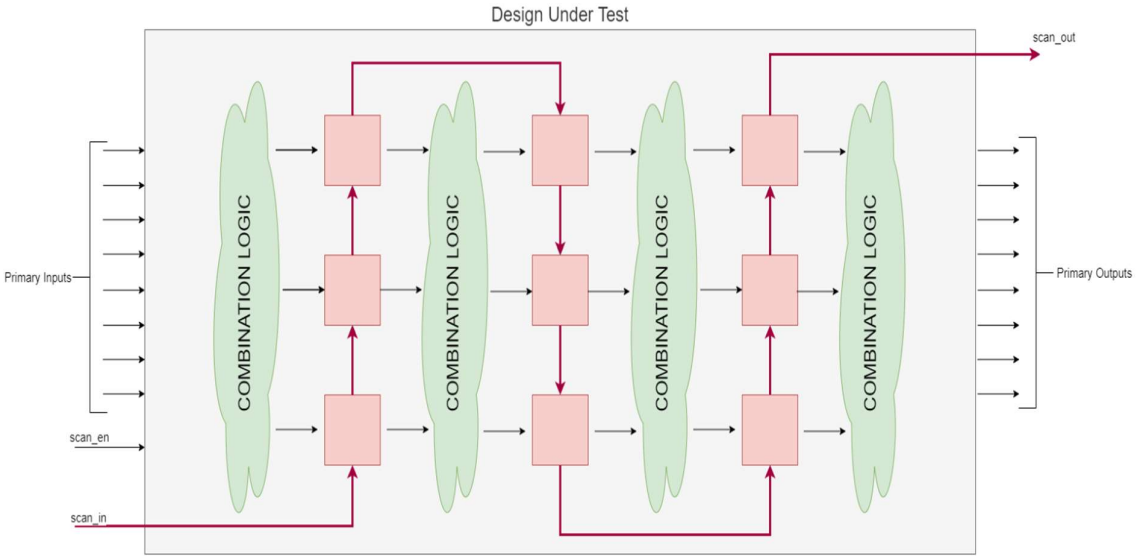


Fig 1.5 Schematic indicating scan chain structure (Reproduced from [34])

This is an instance of a design that's being tested (DUT). In the circuit, I have illustrated a singular scan chain (in red) with Scan In and Scan Out terminals. Assume that the Scan Enabled signal controls every scan flipflops.

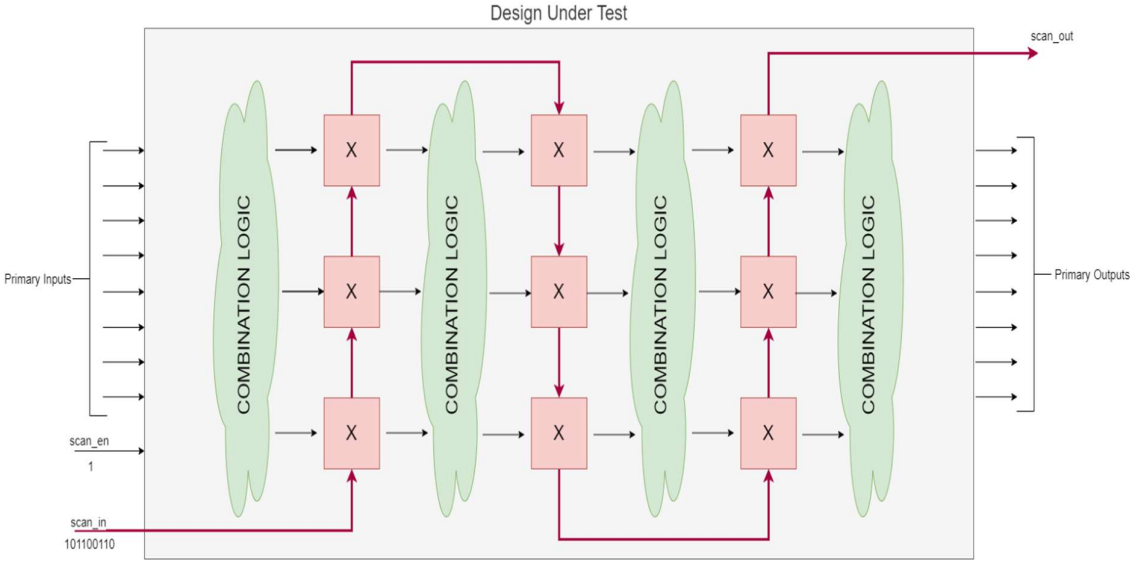


Fig 1.6 Scan Chain Operation for StuckAt Fault – Step 2 (Reproduced from [34])

The very first step we need to do is set the scan flipflops to scan mode. The Scan Enable pin is used to do this. The scan mode is enabled in this situation by setting Scan Enable to 1. Remember that all of the scan flops were originally in an unknown condition (X). There are design solutions to initialise all flipflops to specified states in industrial devices if required.

Suppose, however, if every scan flop was initially at undetermined state X in this scenario.

The below vector is what we desire to scan: 101100110

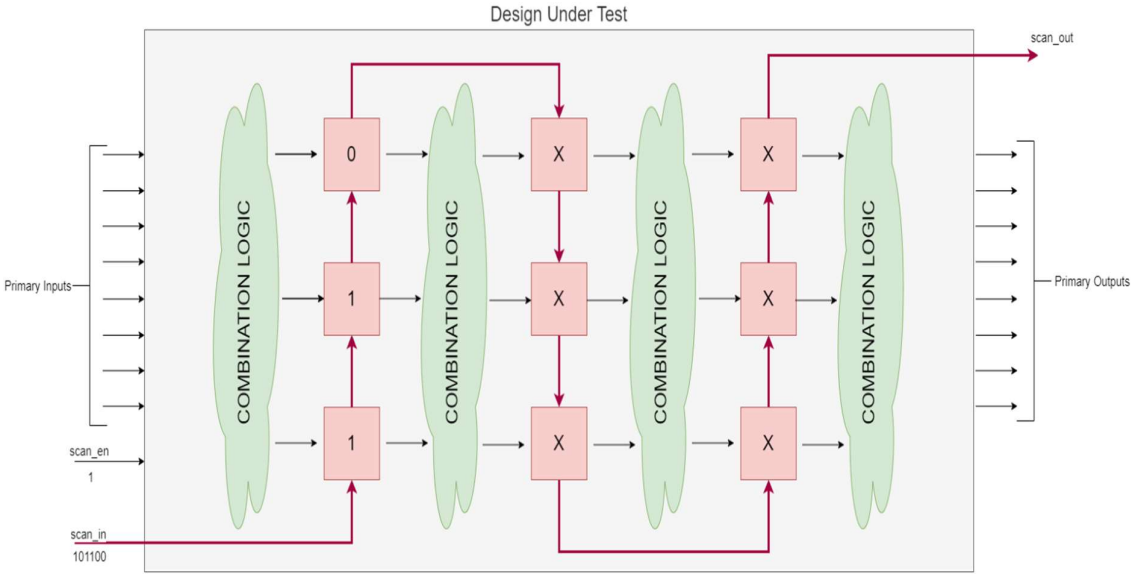


Fig 1.7 Scan Chain Operation for StuckAt Fault – Step 3 (Reproduced from [34])

And then we begin scanning the testing vector that we wish to use. The first three bits get scanned in, as seen in the diagram above. Per clock cycle, we shift inside one bit. The scan shift frequencies is usually rather low, substantially smaller than the circuit's useful frequency. Many ASIC circuits now operate at a frequency of around 100MHz. AMD employs a shift frequency of 400MHz, which is rather high for this purpose. The lower the exam period, the greater the test frequency.

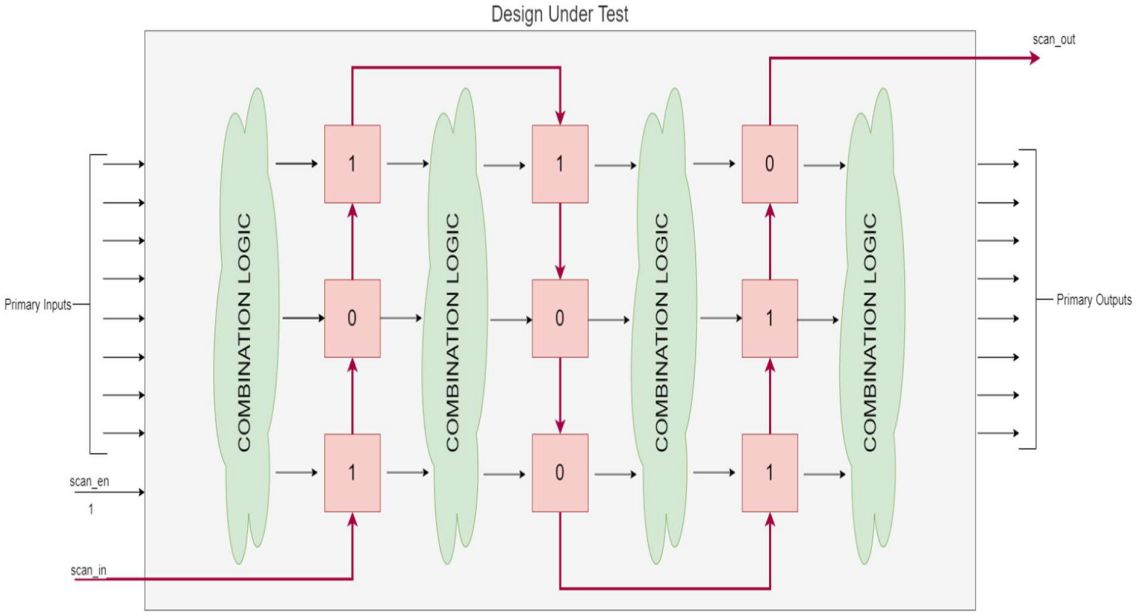


Fig 1.8 Scan Chain Operation for StuckAt Fault – Step 3.1 (Reproduced from [34])

At about this moment, the entire test vector '100101011' has been moved in. We've finished moving in. Scan mode will be disabled by setting Scan Enable to just 0. The shifted in test vector is presently applied on combinational circuitry parts powered by scan flipflops. The second, third, and fourth combinational logic units are now required test inputs.

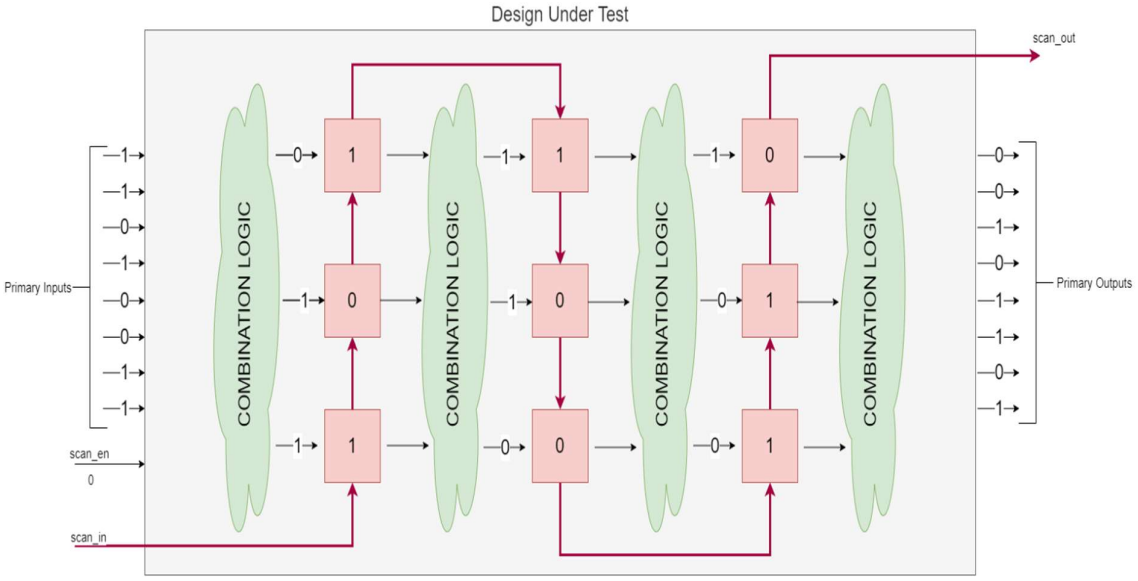


Fig 1.9 Scan Chain Operation for StuckAt Fault – Step 4 (Reproduced from [34])

Now next stage is to use force PI and measure PO to compel principal input (PI) value and measure principal output (PO) values. The shifted in testing vector was previously given to the combinational circuit components that are controlled by scan flipflops in the previous stage. The second, third, and fourth combinational logic blocks had previously been forced test inputs. The outputs of such combinational logic units have now been created. The first combinational block has its results available since we pushed values to PI. Additionally, POs may now see the outputs of the fourth combinational block. By measuring POs, we will obtain the results of combinational block 4. The output values must

be pushed into scan flipflops and afterwards shifted out for the remaining combinational blocks (1,2, & 3).

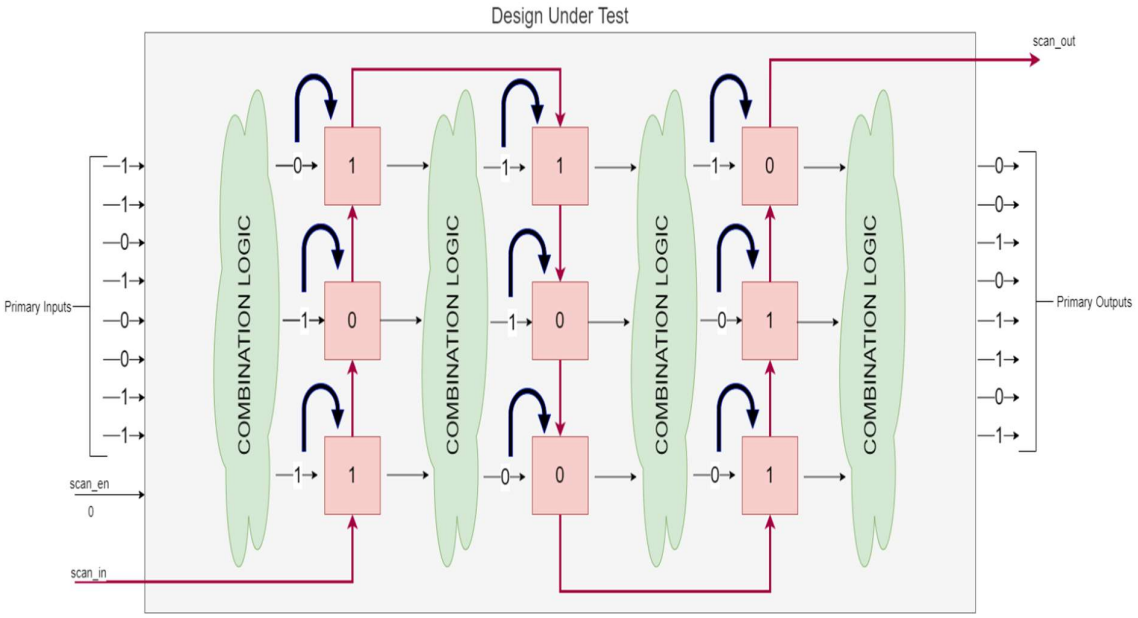


Fig 1.10 Scan Chain Operation for StuckAt Fault – Step 5 (Reproduced from [34])

Toggling the system clock is expected to move the output results of combinatorial blocks 1,2, & 3 into scan flipflops. All D flipflops (scan flipflops) will collect the data at respective D input after the system clock is toggled. The capturing event is depicted in figure 6 above.

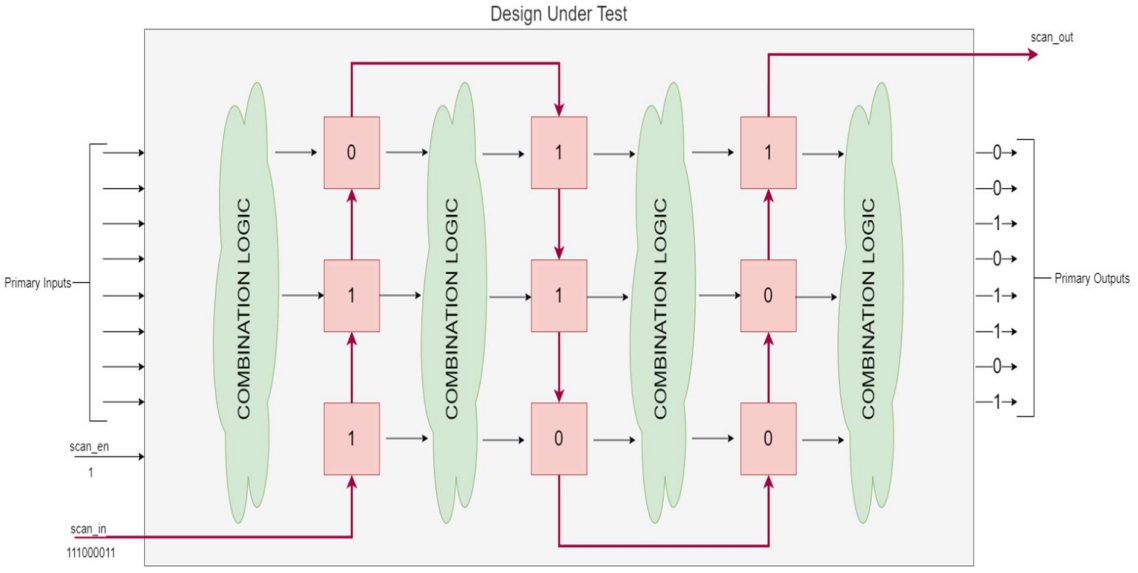


Fig 1.11 Scan Chain Operation for StuckAt Fault – Step 6 (Reproduced from [34])

We can now transfer out the collected combinational logic replies. We will, though, shift in the next testing vector when doing so. '111100111' will be the next testing vector. To allow shifting, we've reset the Scan Enable signal at 1. Here's a look at the change in action. As you see, we've shifted out four bits of the prior test answer while simultaneously shifting in four bits of the latest testing vector input. In the diagram above, the latest testing vector bits are seen in bold red.

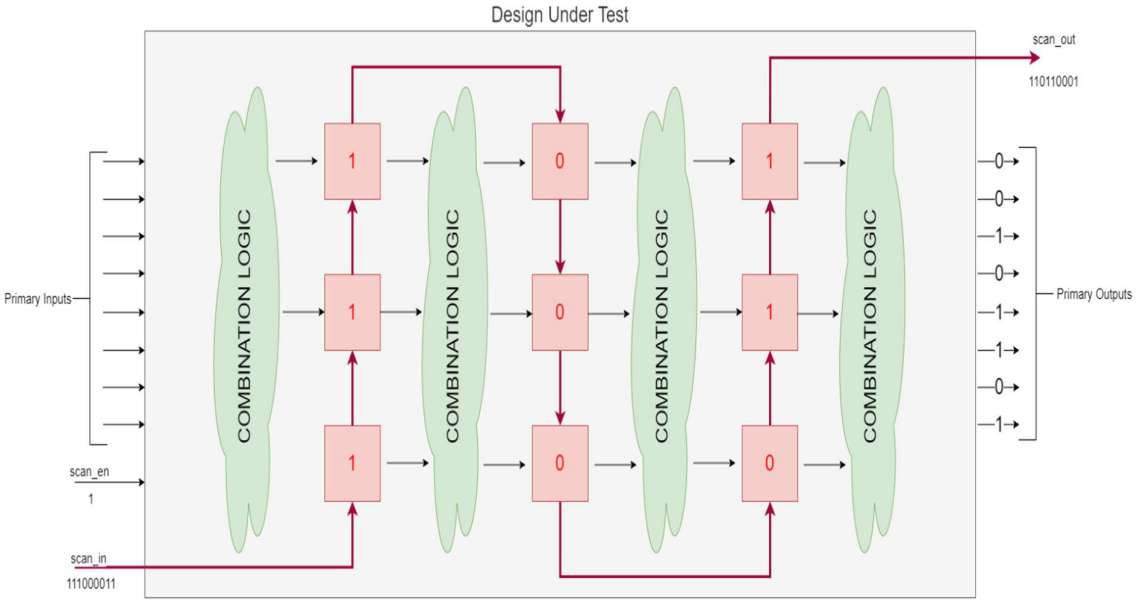


Fig 1.12 Scan Chain Operation for StuckAt Fault – Step 7 (Reproduced from [34])

At this point, the testing output for the prior testing vector has been entirely scanned out, and the testing vector input has been scanned in. This is repeated till every test vector have been applied.

CHAPTER 2: JOINT TEST ACTION GROUP (JTAG)

2.1 INTRODUCTION

Design complexity of a chip is increasing day by day and because of which cost of testing is also increasing exponentially in terms of time and money and the remedy of this problem is DFT – Design for testability which plays an important part in the ASIC design flow by reducing testing time and providing high fault coverage. Testing can be of two type Intest and Extest. In order to perform extest, we have to perform the boundary scan and for that we have an IEEE standard 1149.X in DFT known as JTAG – Joint Test Action Group. JTAG is basically a design which we add as an additional circuitry to the system which makes input as controllable and output as observable. In DFT, Internal scan is typically used for testing, however in some circumstances, such as board-level test and diagnostic, test on-board connectivity among chips, and test on-chip system logic, boundary scan is required, making JTAG a major element of DFT.

JTAG is basically an interfacing device via which a testing designer can interact with all the chips coming from different vendors. The first JTAG was developed consortium, and the main purpose was to reduce the testing time of PCBs. After that many JTAG versions were developed as shown in the figure.

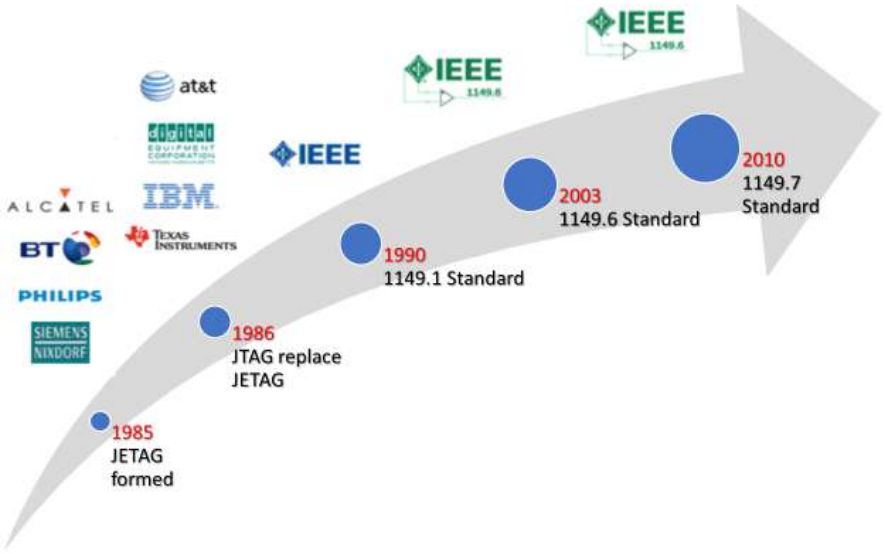


Fig 2.1 History of JTAG / Boundary Scan (IEEE1149.x)

JTAG is mostly use for performing boundary scan testing in the chip. The scan flops used in the boundary scan testing are serially connected to each other as a register and the output and input of this register is connected to the JTAG via pin TPO and TPI respectively. Now, the reason why we use JTAG instead of applying the input and observe the outputs manually is that in JTAG we have an instruction register through which we can test many chips parallelly or if we have many hierarchal modules in one module then also JTAG can use and we just need to apply instruction only one time and whole module will get tested. So, the conclusion is that using JTAG we are automating the boundary scan testing.

2.2 JTAG ARCHITECTURE AND COMPONENTS

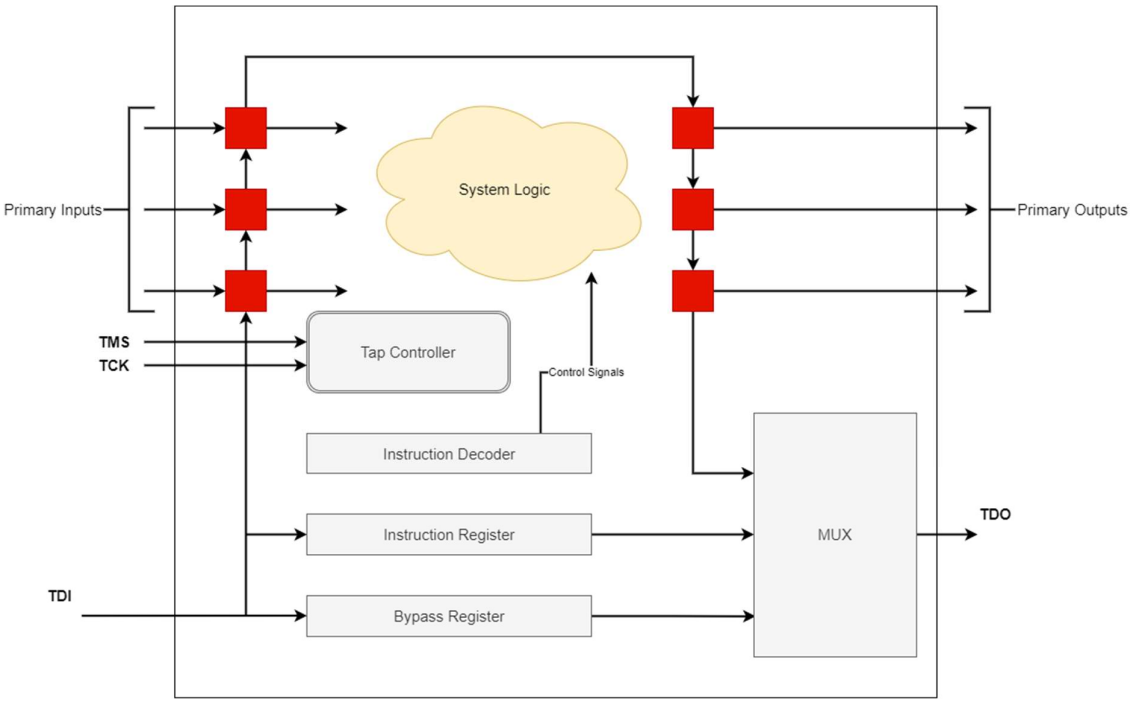


Fig 2.2 JTAG Architecture (Reproduced from [35])

The Joint test action group (JTAG) majorly consist of four components which are as follows:

- Test Access Port (TAP)
- TAP controller
- Registers
    - Instruction registers
    - Boundary scan registers
    - Bypass registers
- Instruction Decoder

2.2.1 TEST ACCESS PORT (TAP)

TAP as its name suggest defines the inputs and outputs pins of JTAG. The TAP consists of following signals.

Signal name	I/O	Description
tck	Input	This port is the JTAG Test Clock.
tms	Input	This port is for mode selection and it is an input signal for the TAP controller. It is defined as TEST MODE SELECT (TMS)
tdi	Input	This port is used to feed bit stream to all Gives a detailed data and instruction registers.
trst	Input	This port is the JTAG Test reset. The JTAG reset can also be done by driving TMS high for five clock cycle
tdo	Output	This port is serial data output to all JTAG data registers and instruction registers.

Table 2.1: JTAG signals description

2.2.2 TAP CONTROLLER

The Test Mode Select (TMS) and Testing Clock (TCK) inputs address a TAP controller, which is a 16-state device that governs the movement of information bits to the Program Counter (IR) and Information Registers (DR).The Instruction Register column is labelled IR. The function and properties of the IR, which is a necessary component of every defined register, are affected by data instructions delivered through this column. DR stands for Data Registers. The functioning and values of the DR are affected by data commands supplied via this column. All boundary-scan systems must have the Boundary Register or Bypass Register, but they can also be the optional IDCODE or USERCODE Registers, or even a designer-specified register (also known as Configuration Register).

I/O SIGNALS OF TAP CONTROLLER

The below diagram shows the I/O signals for the TAP controller each pin represents an operation for this TAP controller and its related modules.

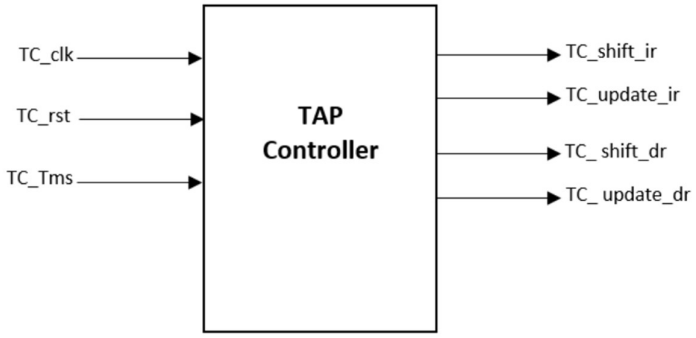


Fig 2.3 TAP Controller

SIGNAL NAME	I/O	DESCRIPTION
tc_clk	INPUT	tclk (Test Clock) JTAG clock used for the TAP controller to operate the states as per instruction.
tc_rst	INPUT	Reset is used separately in TAP controller to reset system when necessary (or) Driving the TMS high for five clock cycle makes the system to reset
tc_tms	INPUT	TMS (Test Mode Select) this is used to drive the states based on the control signals
tc_shift_ir	OUTPUT	This pin instructs the instruction register that there is a valid instruction on the TDI line and this signal is given as input to the instruction register
tc_update_ir	OUTPUT	This output line represents the instruction block that the data transfer from JTAG is done
tc_shift_dr	OUTPUT	This pin instructs the configuration register that there is a valid data on the TDI line and this signal is given as input to the configuration register
tc_update_dr	OUTPUT	This output line represents the configuration block that the data transfer from jtag is done

Table 2.2: JTAG signals description

MICROARCHITECTURE OF TAP CONTROLLER

This TAP controller is used to control the data movement from the Instruction set to the data configuration block and it controls the transmission of data from and to the JTAG.

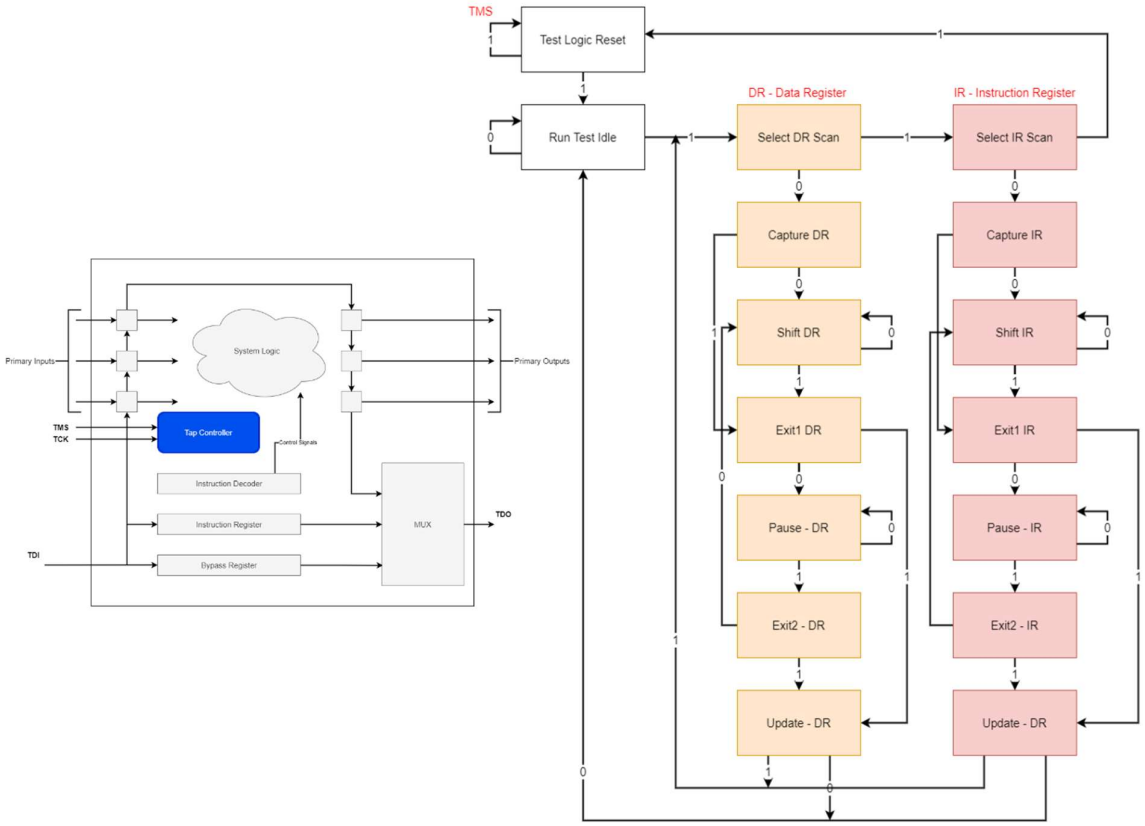


Fig 2.4 TAP Controller Internal Architecture (Reproduced from [35])

2.2.3 INSTRUCTION REGISTER

Instruction register basically works on the signals received from the TAP controller based on tck. The inputs and outputs given to the instruction register are shown in the I/O pin diagram given below. The description of each pin is given in the I/O description table. The serial data tdi is converted into 3-bit parallel data which is given as an address to various data registers present.

In the case of JTAG to AHB Bridge it is given to the configuration register, in order to configure the data register to be used. This Instruction register captures an address bit of 3 which is “100” and is sent as an address to the configuration module. This block is used to transfer a set of input signal to AHB signal generator which is in the form of frame structure. Once the data frame is received through the tdi pin, it stores the data in a buffer which consist of 162 bits, which has a Address bits, data bits, and control information. The Configuration register is split into 34 bits of Address_out, and remaining bits for data_out and it generates a valid signal along with it to identify the valid data available in the transmission pad.

I/O SIGNALS OF INSTRUCTION REGISTER

Below pin diagram shows the I/O pin diagram for the instruction register.

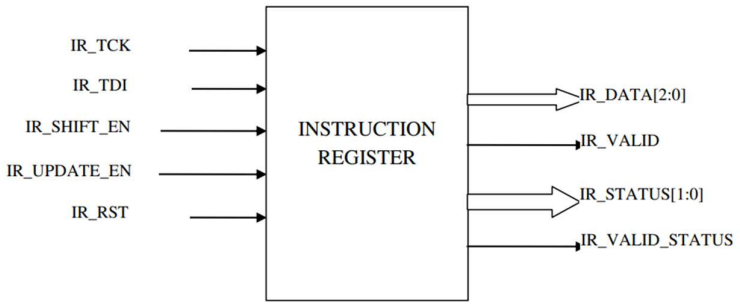


Fig 2.5 Instruction Register

I/O SIGNALS DESCRIPTION

The input and output pin description for the instruction register is given in the below table.

SIGNAL NAME	I/O	DESCRIPTION
ir_tck	INPUT	JTAG clock used for the instruction register in order to shift data serially.
ir_tdi	INPUT	Single bit serial input for instruction. Captured on the rising edge of tck
ir_shift_en	INPUT	Its high for three clock cycles, which receives the IR data through tdi pin
ir_update_en	INPUT	Its high for one clock cycle, which denotes that the data is shifted on the id_data
ir_rst	INPUT	Active low input which asynchronously reset the instruction register
ir_data	OUTPUT	3 bits parallel output of the instruction register. Used to select a particular data register.
ir_valid	OUTPUT	High for one tck cycle. Sends 1 when the data is valid else 0
ir_status	OUTPUT	This consist of address of status register and is of 2bits. If the address value is 10, it selects the status register
ir_valid_status	OUTPUT	This is a valid signal generated for the status register

Table 2.3: Instruction Register signals description

The main function of the instruction register is to convert the serial single bit input into a 3-bit parallel output. The shift operation in the register is done only when the ir_shift_en is high else the previous value is maintained. The output of the instruction register is an ir_valid and the other is the 3-bit parallel

Ir_data. The ir_valid depends on the ir_update that is, it is high only when the ir_update_en is high else it is maintained low. The system can be reset using the ir_trst which is actually the JTAG reset which is used to reset the bridge. The ir_data transmitted through this instruction register selects the particular data register which can perform the required functions.

2.2.4 BOUNDARY SCAN REGISTER

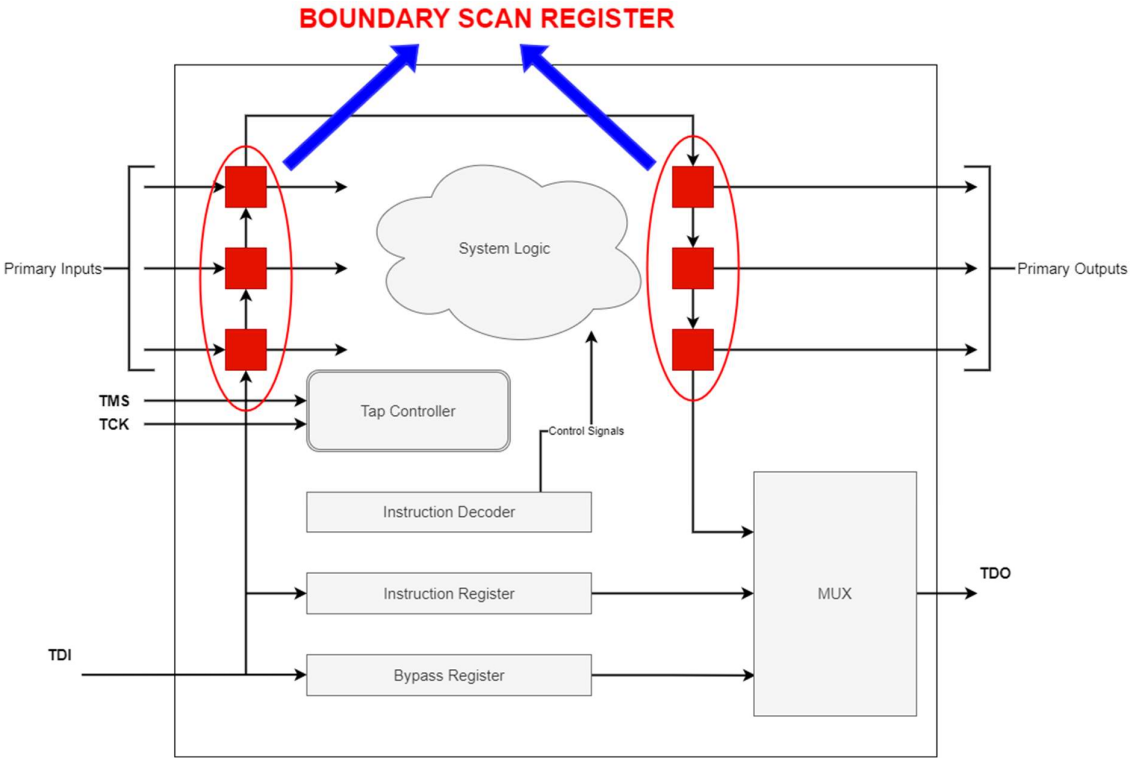


Fig 2.6 Boundary Scan Register (Reproduced from [35])

The boundary scan registers are similar to the normal register but the only difference is that they have two inputs and two outputs. The flipflop which is a fundamental unit of any register is also replaced by the scan flipflop. The boundary scan registers are warp around the testing module and one input is connected to the outer input of the module. For the output also, the connections are similar. Another input is connected to the TDI pin of the JTAG and another output of the register is connected to the TDO pin of the JTAG.

In spite of data I/O pins, there are two clock input pins are also present. One clock input is normal clock for data and another is scan clock for defining the frequency of the testing.

I/O SIGNALS OF INSTRUCTION REGISTER

Below pin diagram shows the I/O pin diagram for the instruction register.

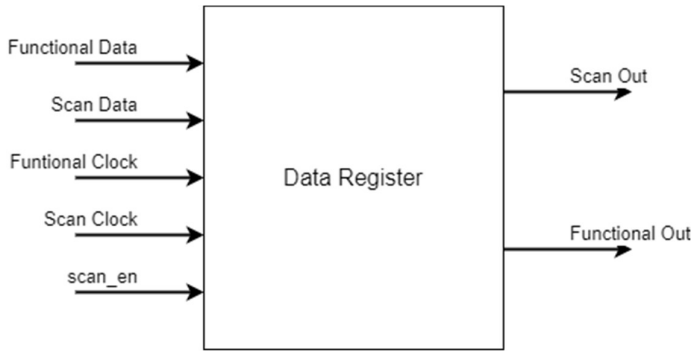


Fig 2.7 Data Register

I/O SIGNALS DESCRIPTION

The input and output pin description for the instruction register is given in the below table.

SIGNAL NAME	I/O	DESCRIPTION
Fun_Data	INPUT	Fun_Data pin is used to process data in normal mode.
Scan_Data	INPUT	Scan_Data pin is used to process data in test mode.
Fun_clk	INPUT	Fun_clk is used to provide clock pulses in normal mode.
Scan_clk	INPUT	Scan_clk is used to provide clock pulses in test mode.
Scan_en	INPUT	Scan_en pin is used to switch mode from functional to test mode.
Fun_out	OUTPUT	Fun_out pin is used to dump functional data out.
Scan_out	OUTPUT	Scan_out pin is used to dump test reponse data out.

Table 2.4: Data Register signals description

MICROARCHITECTURE OF BOUNDARY SCAN REGISTER

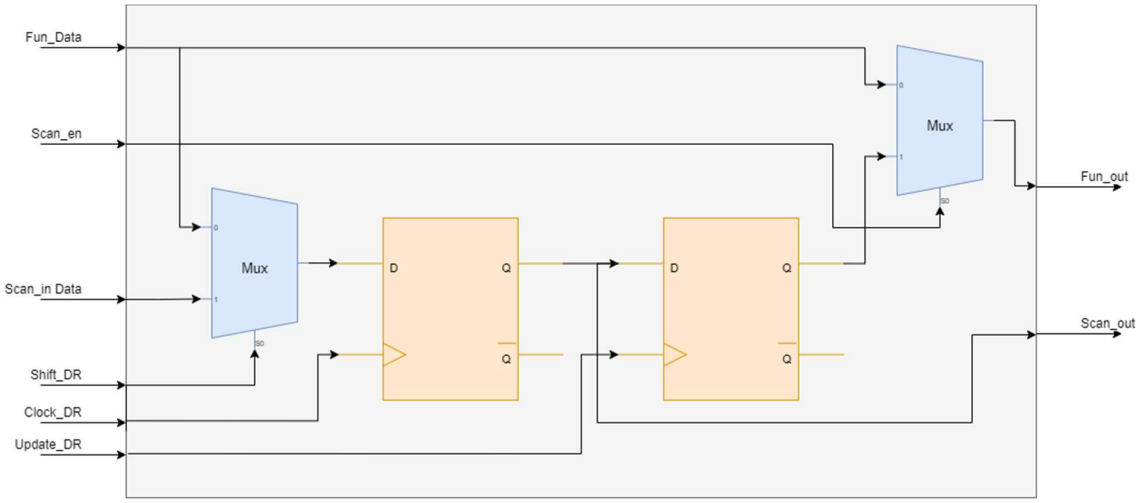


Fig 2.8 Data Register Internal Architecture (Reproduced from [35])

CHAPTER 3: REDUCED INSTRUCTION SET COMPUTER

3.1 INTRODUCTION

We know that a system particular to an application has various input signals, so we need the correct results based on these input signals. If the device is relatively tiny, we can configure it, but for big devices, we need a microprocessor that can efficiently process the input data and generate error-free results. It must also be capable of dealing with real time interruptions. As a result, the microprocessor may be defined as an FSM that executes instructions saved in memory. The instructions saved in the storage and the instructions saved in the microprocessor's registers are used to understand the state of a program. Each instruction describes how the condition should change as well as what instruction are sent in next cycle.

Now, in order to create a microprocessor, we must first determine exactly what sort commands we will transmit and then create each component accordingly. If we wish to do an adding operation, the sequence should be as follows: get the command from storage -> decode the received instructions -> produce execution signals such as activate the basic arithmetic units and transfer operand to it -> conduct addition -> save the output in storage -> receive next command. As a result, we must construct various units, such as a program counter to acquire command from storage, a decoding unit to decode, in accordance with the following approach. As a result, we need a fundamental layout that anybody may use to create their customized microprocessor, with adjustments and enhancements based on the requirements.

Set of Reduced Instructions are utilized because of its numerous benefits over other architectures, computers are frequently utilised as the foundation for constructing microprocessor. To acquire data and comma from storage, RISC needs only load / store operations. Registers are used for any commands that aren't memory-dependent. This makes the instruction set architecture, also known as ISA, more intelligible and easier to use since we don't have to read memory every time, we run a program. It also allows us to execute one instruction at a time, this is helpful in increasing performance. In order to achieve faster processing simple addressing modes are used. Usually commands employ register-based tackling since Reduced Instruction Set Device related computers exercise register to register type commands. Another significant benefit is the big memory set. A large memory set allows the compiler to do a large set of operations while also reducing the problems related with operation and call returning. In comparison to other architectures, the RISC command format is also simplified. All of these features contribute to increased performance and decreased complexities, which are the major factors that a developer aims to enhance in order to maximize productivity.

Complicated Instruction Set Computers, or CISCs, led the market in the mid-1980s because they used detailed instructions. However, why do designers like to adopt such a complicated design? The reason for this is that storage and microprocessor were the most important information systems components at that time. In actuality, 16KB of RAM costs around $500. This pushes designers to consider a design that uses large density programming, in which every command is allocated to perform additional task in order to minimise program size. Complex instructions, on the other hand, add to the hardware's complexity. Wilkes developed a micro-programmed controller in the mid-1950s that works as a translator between ISA and peripherals. The complex instruction is executed by the micro-program in a series of short and basic instructions that do not require any complicated circuitry. Modification of ISA could also be accomplished using various micro-programs, which reduces the translation space between computer and higher language.

A question emerges now, what motivates programmers to consider RISC technique? So, CISC was built to make devices more economical while working within the limits of tiny and costly storage. However, as years progresses, technological improvements in hardware and software components happened. These types of incentive lead designers to consider CISC architecture alternatives. Patterson and Ditzel released a paper titled "The Case for the Reduced Instruction Set Computer" in the year 1980. They argue in their paper that multi-chip microprocessor designs will not be the same as single-chip microprocessor architectures in terms of speed and reliability.

The efficiency of the RISC is comparable to that of the CISC, but the effort needed is far lower. The suggested RISC design has a large number of memory banks, each of which is 32 bits in length. Memory instructions are implemented using the load / store model. The length of RISC commands is set at 32 bits, unlike CISC commands, thus RISC standards are substantially smaller than CISC standards.

Including all of the previous studies, the RISC architecture balances between power efficiency and pricing. Because of the advantages described above, RISC quickly gained traction in the industry, and many businesses began developing their own processors based on the RISC technique for specific applications. ARM was the very first business to create its own microprocessor using the RISC idea as a foundation. Following that, several businesses began to employ RISC to varying degrees of success.

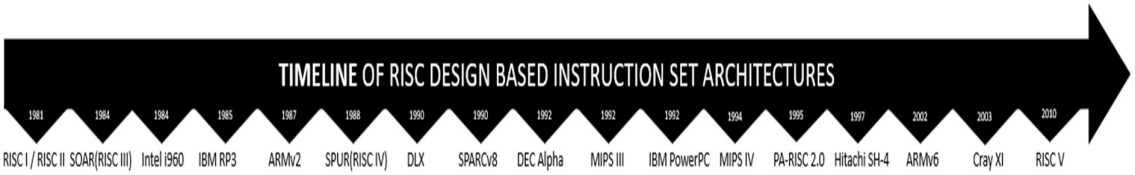


Fig 3.1: Timeline of RISC based instruction set architectures

The idea of RISC architecture and approaches for improving a chip's features (both in terms of usability and efficiency). Along the way, we gain a sense of the breadth of RISC-based applications, which includes convolution, data analysis, commercial computation, and a larger basis for embedded systems like smartphones and tablets [18]. So it is time to go into the application side of RISC architecture so that a developer may decide whether to use RISC as the foundation for their work based on available data. This section's goal is to give an overview of recently released applications-based published studies so that the reader may understand the influence of RISC architecture in many sectors.

Because of its open ISA characteristic, the RISC V Instruction Design is becoming increasingly famous. The term "open source" refers to the ability of a designer to modify the rules and add enhancements as needed for a specific requirement. Other RISC characteristics include ISA flexibility and numerous industrial property sources [19]. RISC-V is also supported by highly effici Chipsets in the micro computing and data transmission fields, as well as integrated CPUs in the IoT and smartphone sectors. The RISC V has already been implemented by the SoC market and developers because the main goal of SoC is to minimise production costs, while operating on black box equipment is a major stumbling block in research.

3.2 REDUCED INSTRUCTION SET COMPUTER (RISC) – GENERAL PRINCIPALS

The best way to understand the RISC concept is to think of it as a concept rather than an architecture. Furthermore, previous RISC microprocessors had a limited number of commands, but today's RISC-based microprocessors include hundreds of commands, some of which are more advanced than CISC microprocessors, but all RISC-based microprocessors obey the following principles:

3.2.1 SIMPLE INSTRUCTION SET ARCHITECTURE

The primary goal of the RISC technique is to provide a basic machine code that can be executed in an one period. This attribute makes the processor easier and more effective by simplifying the architecture. The significant channel latency is also impacted by the intricacy of a command. The operating frequency, or in plain terms, the processor's speed, is a key characteristic that may be simply computed using the significant channel idea. Critical route should have a low value. [3]. Smaller datasets and inbuilt functions are no longer required thanks to the simple command base. Dedicated storage device will improve average processing speed by storing these commands.

3.2.2 LOAD AND STORE ARCHITECTURE

This design is utilised to perform instructions that need storage accessibility, whereas the remainder of the operations use the inbuilt register library to transmit information. Since each command needs to complete in single period, the speed of the system increases. Unlike CISC, which has several different kinds of actions, [1] only adds to the intricacy in most circumstances because the commonly used commands in any microprocessor are storage to reg and reg to reg. We may also utilise this simple framework to create more complicated ones.

3.2.3 SIMPLE ADDRESSING MODES

Simple addressing modes are included into the RISC methodology's design. Register addressing and memory addressing are two of them. As we understand, RISC has a large number of registers, each of which has a defined size. Every one of these big register sets are utilised for register mapping, which is primarily used to move distinct variable values through one operation unit to another. Memory addressing is only required for loading and retrieving commands. Another benefit of having a broad register set is because developers can cut down on additional call and return operations. We can utilise register to maintain local dynamic and transferring problems in mind while speeding up program operations. Local modification can only be established through the declaration procedure. This mutation develops throughout the appealing procedure and is killed after the procedure is completed.[15]

3.2.4 INSTRUCTION FORMATS WITH SPECIFIED LENGTHS

The most significant benefit of employing predefined command forms is the decrease in command runtime. Because all commands have the relatively similar length, the compiler does not require to examine the length of each command and differentiate them based on it each time. Furthermore, each opcode is the relatively similar size. All of this speeds up the decode operation. Because of the fixed set size, trailing is reduced. In order to have a better understanding of RISC, we'll look at two configurations: MIPS and ARM.

Microprocessor without Interlocked Pipelined Stages (MIPS)

Without an interlocked processor, Pipelined Modules is a commonly used architecture whose primary objective is to offer fast execution of generated programs. MIPS's primary concept is to present a collection of commands that may be used to control a modest processor. As a result, there is less or no need for precision, and the instructions are quite similar to the small program instructions. Although the CPU has a pipe, it lacks the circuitry required to connect it; this role should be handled by soft programming. It has expanded tremendously as a result of Stanford studies. These initiatives, along with Berkeley's RISC work, have affected the advancement of the microprocessor in connection to the RISC framework. MIPS is critical for preinstalled programs like as dslrs, smart tvs, gaming consoles, wireless routers, and other similar devices. MIPS standardized its architecture around two fundamental architectures in the 1990s. They are MIPS32 and MIPS64.

As RISC microprocessors, MIPS uses a store/store architecture to enable memory operations. This configuration offers a variety of uploading and save commands for data transfers of different sized, such as bytes, halfwords, etc. MIPS32 was designed with 32 commonly used registers, a Program Counter (PC) and two specially used registers. These registers are designated as $ 0, $ 1,..., $ 31 in the meeting's terminology. The first and last commonly used registers are dedicated for a particular work: The last $31 register is utilized as a connecting register for the communication and connecting sequence, which is required to initiate the program, and register $0 is utilized for zero values. Apart from that, the outcomes of numeric multiplications commands are stored in two specially used registers named HI and LO.

Because MIPS is a RISC CPU, it has a constrained teaching structure. Each instruction has a length of 32 cubits. It teaches in three distinct ways: Instant (Type), Jump (J-type), and Register (R-type). Command processing is made easier by using a small number of instruction forms. However, because there are three command formats and one speaking mode, the compiler will have to include sophisticated processes and coping mechanisms. Designers might not be able to pay the whole consequence if these programs and local methods are not extensively adopted. This is a positive message from RISC microprocessors.

Advanced RISC Machine (ARM)

ARM's advancement began by watching the progress of Stanford's MIPS and Berkeley's RISC computers at the time. Before such topologies, several architectures, such as IBM, Cray-I and Digital PDP-8 processed many features that were later associated to RISC.

Berkeley RISC was considered as an ARM antecedent because it had numerous characteristics with it, including as fixed 32-bit instruction size, load and store architectural style, and three addressing instruction modes. A couple of them were also denied, such as not running all commands in one clock period. ARM also avoids using deferred splitting since it compromises the concurrency of each specific command. Furthermore, unlike Berkeley RISC, ARM does not employ big Register Windows because they maximise chip space and hence pricing.

3.3 TECHNIQUES

Handling real-time operations is the primary issue in today's rapidly evolving technology, and complicated processors are necessary to accomplish so. We can develop such processors in two ways: first, we can design one from scratch, and second, we can change existing processor architecture by adding performance optimization techniques. Furthermore, while both approaches aid in boosting functional capabilities, they also raise a processor's power consumption, posing a challenge; hence, a trade-off among both low-power strategies is required. Some of the more efficient ways for designing a processor are discussed further down.

3.3.1 FIVE STAGE PIPELINE ARCHITECUTURE

Pipelining, a common feature of RISC-based general-purpose processors, is an effective approach to use concurrency. It's a way in which the processor finishes the expert advice before going on to the next. It's a worthwhile strategy for maximising hardware resources. To appreciate the core concept of pipelining, consider a washing scenario. Assume we have three items to clean, dry, and fold; each task will take 10 minutes. One way to finish the procedure is to start with the first towel and complete all three steps in around 30 minutes. Then repeat the first three steps with the other cloth. The entire process requires about 90 minutes to finish. This, though, is not the best method. The wiser option would be to wash the second cloth after the first has been washed and then spin it in the machine. As a result, the first load of laundry will be sorted, followed by the second load drying, and finally the third load. The entire procedure will require only 50 minutes if you do it this way.

In much similar way, pipeline operations occur in RISC processors, though the steps of the pipeline alter based on the processor kind. The CPU executes each instruction in a sequential manner. While various numbers of steps may have varying processors, several of the following are listed in the order of five stages: Decode Instructions ; Fetch Instructions ; Execute Instructions ; Memory Back; Write Access As a result, we named it Five stage pipelining since we execute instructions in five phases [18]. We can create N stage pipelining in general by processing an operation in N phases. The more phases there are, better effectively the processor can function. Figures 4.1 and 3.2 depict the five-stage pipeline with and sans MIPS.

A processor with pipelining executes numerous commands in the single clock pulse. These commands are processed independently at each level. These steps, especially five phases in the context of a 5 stage pipeline, function as separate units that collaborate and produce outcomes in a synchronous manner. It is critical for these distinct units to communicate synchronously with one another and convey information in a timely manner in order to effectively execute the commands, thus it is necessary to address all of the risks associated with the pipeline feature.

Like we understand, pipeline processors perform many commands at various levels at the same time, and also the outputs are only saved in general purpose registers during the write back stage. Thus, if the modified result of the preceding command is to become the following command to be performed, the provided command may now receive incorrect data, necessitating the updating of values. Data risk is the name given to this type of threat. We may easily fix the data hazard by simply forwarding with stall induction somewhere between the commands or by simply moving one step to the next [4]. As Amity Pandey explains in his work that forwarding mechanism may easily avoid data hazard using eight instances.

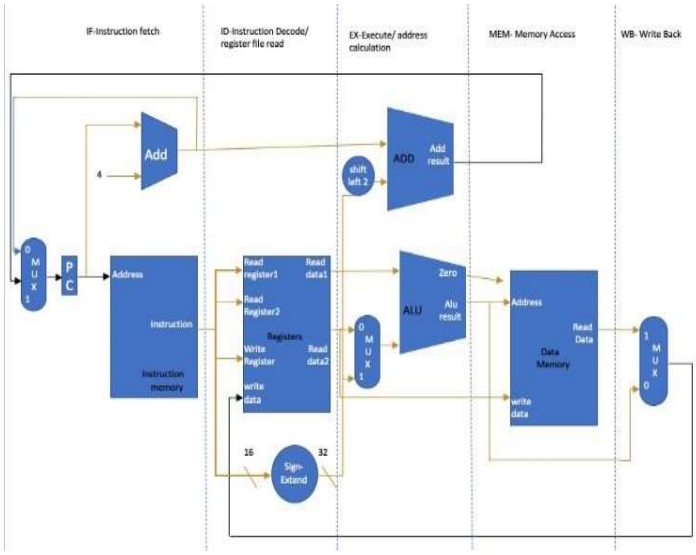


Fig 3.2: MIPS design without a five-stage pipeline (Reproduce from [3])

The control hazard will be discussed second, that happens as a result of branch instructions. At the finish, the instructions are decoded, and branching instructions are handled. So, unless the branch instruction hits the instruction decoding phase, but the pipeline instruction beside it gets processed, causing a difficulty, it is not determined which instruction will be retrieved next. Control Hazards are

the name for this type of danger. Amit Pandey offered Static Branch Predictions and Dynamic Branch Prediction as two strategies for removing this issue [4].

3.3.2 POWER OPTIMIZATION TECHNIQUES

When it comes to the ASIC design, two words come up often. And those are Power and Performance. These variables are inextricably related, and there is a trade-off among them. As a result, it is the creator's task to optimise the design for such optimum equilibrium among the two. Several solutions, including as clock gating, multiple threshold techniques, are being developed over the decade to overcome these challenges. The results of the experiments reveal that VDD/2 is a good option for a second VDD. The study investigates various combinations of the aforesaid strategies in order to find an appropriate one. These were put through their paces on to an 8-bit RISC CPU [8].

Clock Gating: Turning off the clock when the device is not in use is a well-known method of decreasing power consumption, that is mostly dynamic. The clock circuit is one of the most significant sources of dynamic power in synchronous systems. Buffers and clock tress consume about 30% of total power. [11]. When the load signal is activated, a register is loaded with fresh data at the clock edge; otherwise, it retains its prior values. This indicates that whenever no load is applied, the border is still necessary. This costs energy and may be prevented by establishing an activated clock by ANDing the clock and the enable signal.

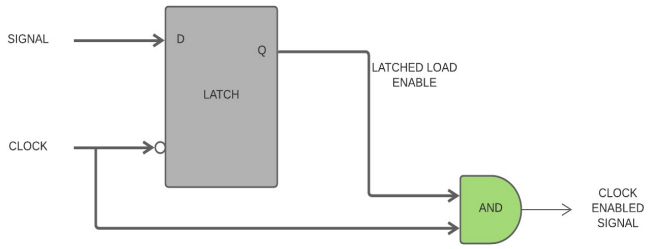


Fig 3.4 Clock Gating schematic is seen in this diagram (Reproduce from **[8]**)

Multi-Voltage: Because dynamic power is proportional to supply voltage VDD, reducing VDD reduces power consumption. However, there is once again a cost. Because low VDD increases latency, suitable optimization is required. The supply voltages in the presented work are 1V and 0.7 V.

Multi Threshold: Reduced VTH reduces latency as current rises, but it raises subthreshold leakage current, resulting in increased power dissipation. There are two threshold levels employed in the presented study.

Pritesh Kumar Yadav [14] utilized Xilinx 14.5 to construct a 32-bit 5 step pipelined RISC processor utilising conventional 180nm CMOS technology. A report showing power usage at various frequencies prior and post clock gating is created to provide a clear picture. The range of frequencies is 100MHz to 500MHz. Table 3.1 shows the results of a comparative analysis of power for adding instructions at various clock frequencies. Table 3.2 shows the results of the same study for multiplication teaching.

Multiplication operation consumes 56 percent of total processing power, which is due to the huge iteration number when compared to the addition instruction. When comparing addition sans clock gating against multiplication, the observation indicates that multiplication consumes 13.6 times the amount of electricity. This factor was lowered to 6.6 by employing clock gating. This demonstrates that the clock gating approach significantly reduces power usage.

FREQUENCY (GHz)	POWER (μW)	
	WITHOUT CLOCK GATING	WITH CLOCK GATING
0.1	7500	5250
0.2	13750	11000
0.3	19230	18860
0.4	24840	23100
0.5	31.50	30030

Table 3.1: Power analysis of addition unit without and with clock gating

FREQUENCY (GHz)	POWER (μW)	
	WITHOUT CLOCK GATING	WITH CLOCK GATING
0.1	92.450	49.610
0.2	200.030	90.280
0.3	282.560	112.980
0.4	374.660	146.700
0.5	451.340	192.210

Table 3.2: Power analysis of multiplication unit without and with clock gating

3.3.3 POWER OPTIMIZATION BASED ON HDL MODIFICATIONS

In the research, there are several approaches for reducing power usage in RISC processors. Several of them are structural improvements, such as clock gating, Wallace multipliers, and advanced branching algorithms. In contrast to this, Soumya Murthy with Usha Verma's method focuses on HDL optimization to reduce power loss in their study. This is an algorithmic alteration rather than architectural adjustment.

Controlling static or idle power is tough. It is also feasible to lower power consumption by modifying HDL procedures. IOs are another high-power-consumption item in FPGA. This is dynamic power as well. It must also be lowered in order to maximise the effective usage of electricity.

Battery backup (for mobile devices), environmental conditions, digital noise tolerance, and coolant and packaging costs are all issues in FPGA power efficiency. Also, as previously stated, there are primarily two types of power dissipation sources: - 1) Power dissipation during idle/standby mode is known as leakage power consumption. 2) Dynamic power usage: This occurs when either logic gates or flip-flops make a logic transition [22].

The following are the many VERILOG coding approaches used to create low power RISC:

- Controlling the clock allows for more flexibility in the design.
- The design does not have a reset since the FPGA resets itself.
- Multi Block RAM array employing a low-power architecture (using CORE generator).
- Block RAM is only enabled while active read/write tasks.
- Synchronous reset is used in the design.
- Internal resets are minimised.
- Small memory blocks based on LUTs (4k bits).

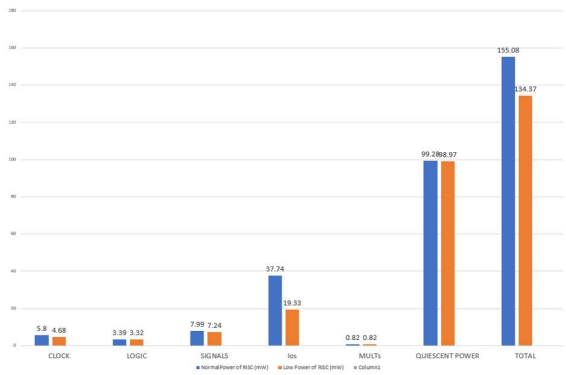


Fig 3.5: The relative power for various units with and without HDL changes is shown in a bar graph.

Ultimately, the simulated and synthesised design demonstrates that adopting the aforementioned strategies saves 22mW of power. As a result, we can state that an overall decrease of 13.33 percent has been achieved. [22].

3.4 IMPLEMENTATION AND RESULTS

RISC, because of its numerous benefits over other architectures, computers are frequently used as a fundamental architecture for constructing processors. Just load and store operations are used in RISC to access data and instructions from memory. Apart from memory dependent commands, registers are used in them. This provides the instruction set architecture, also known as ISA, more intelligible and simpler to use since we don't have to enter memories every time we execute a command, and it allows us to execute single command every clock cycle, thereby increasing speed.

It has been created, a 16-bit RISC processor having 15 instructions. Extremely thorough simulations are used to verify the design. The design will be built on a Spartan 6- FPGA board. The design has also been checked. VIVADO 2020. 1 SUIT has implemented this design.

3.4.1 SIMULATIONS

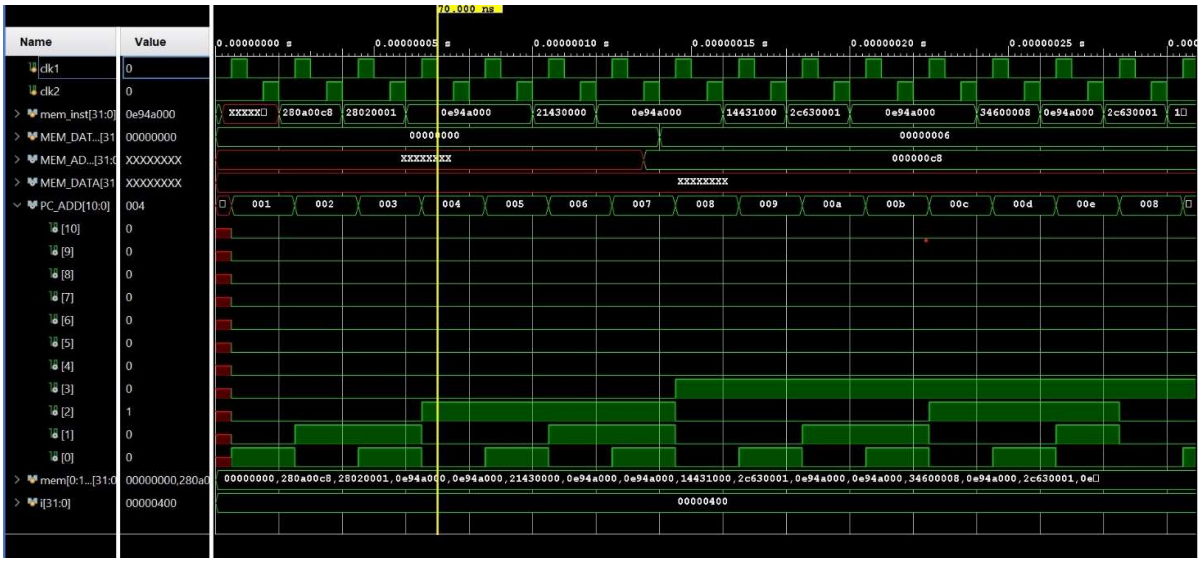


Fig 3.6: Simulation 1

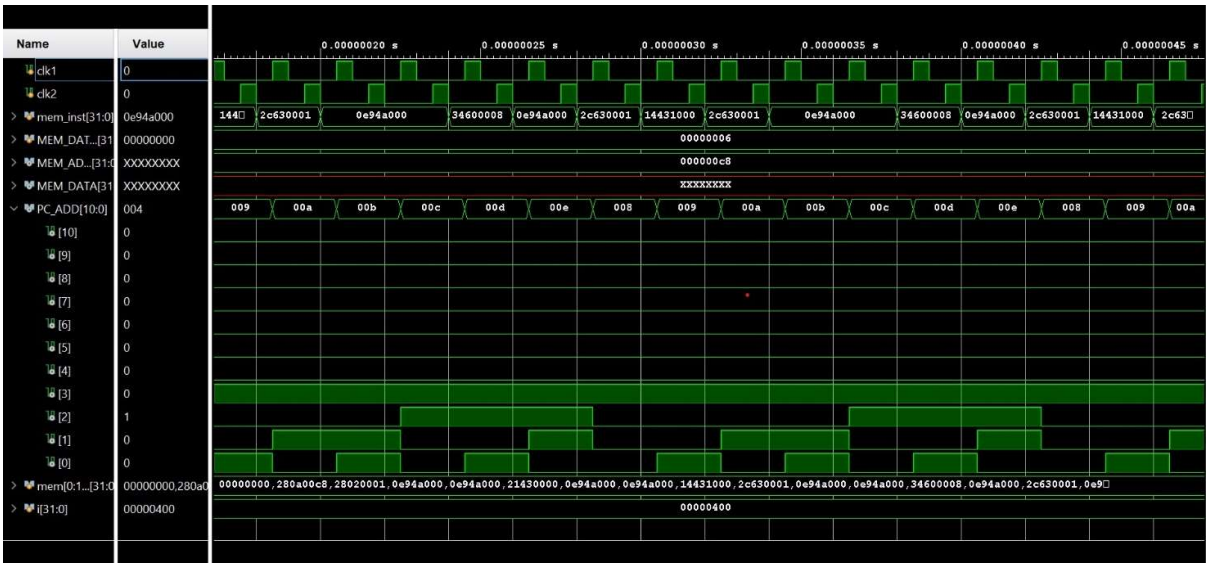


Fig 3.7 Simulation 2

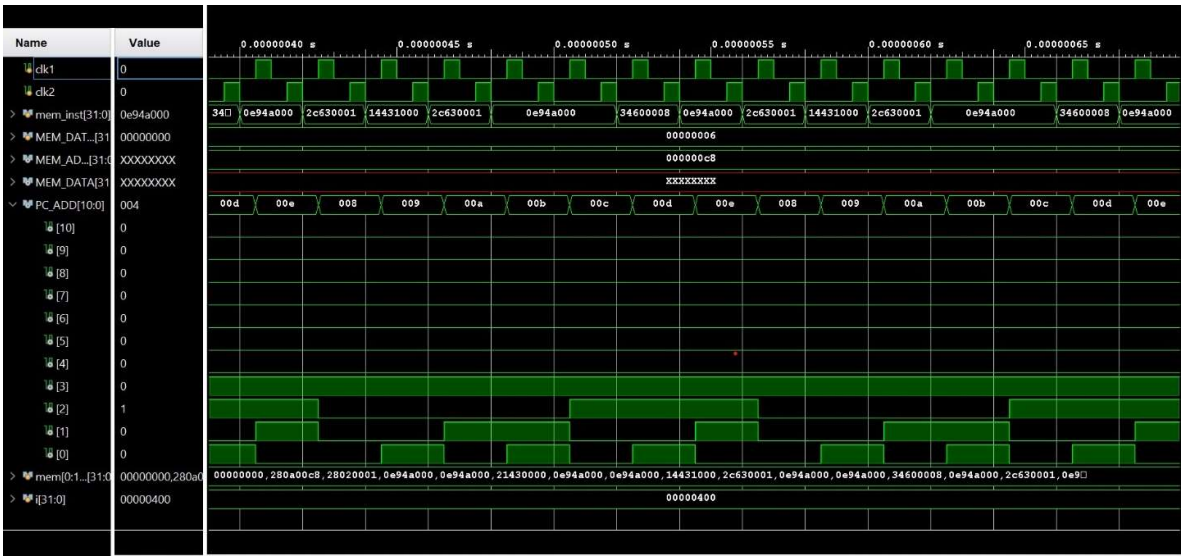


Fig 3.8: Simulation 3

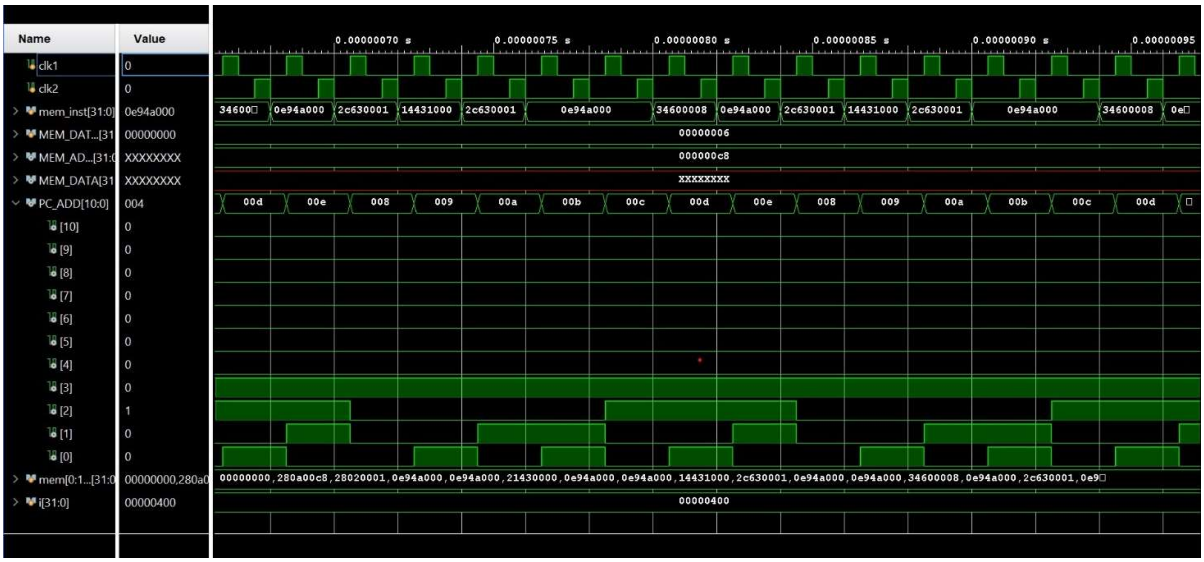


Fig 3.9: Simulation 4

3.4.2 HARDWARE IMPLEMENTATION

- **RTL SCHEMATIC**

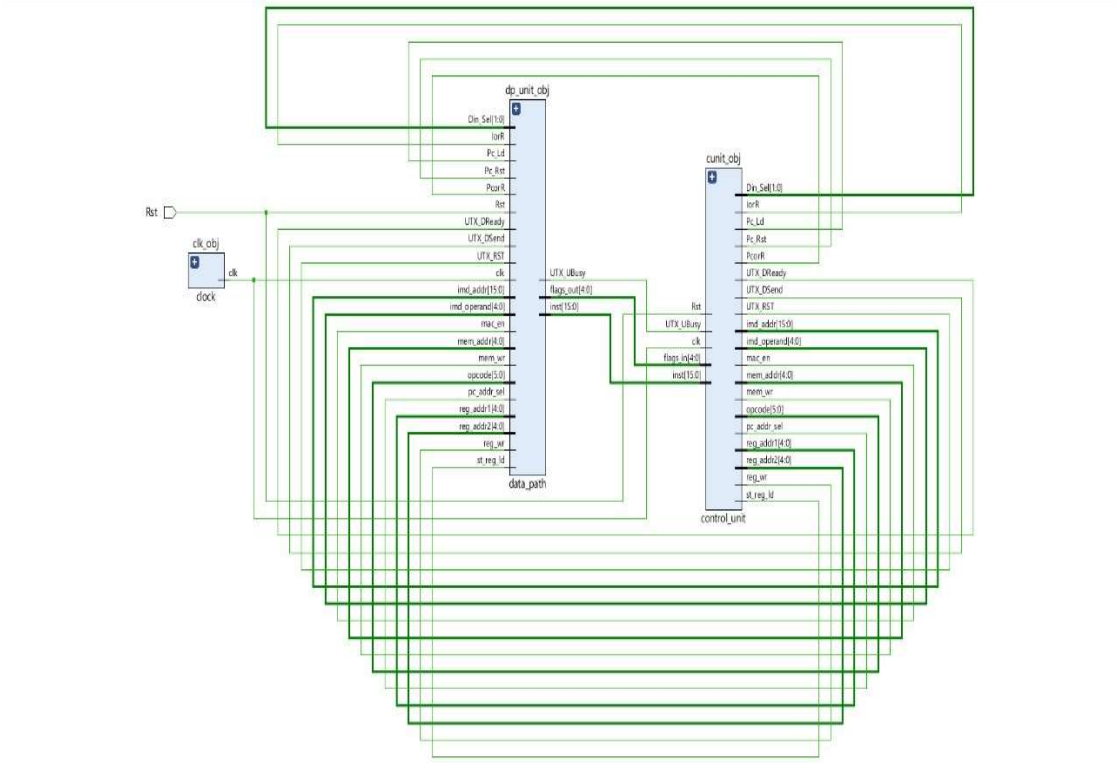


Fig 3.10: Schematic 1

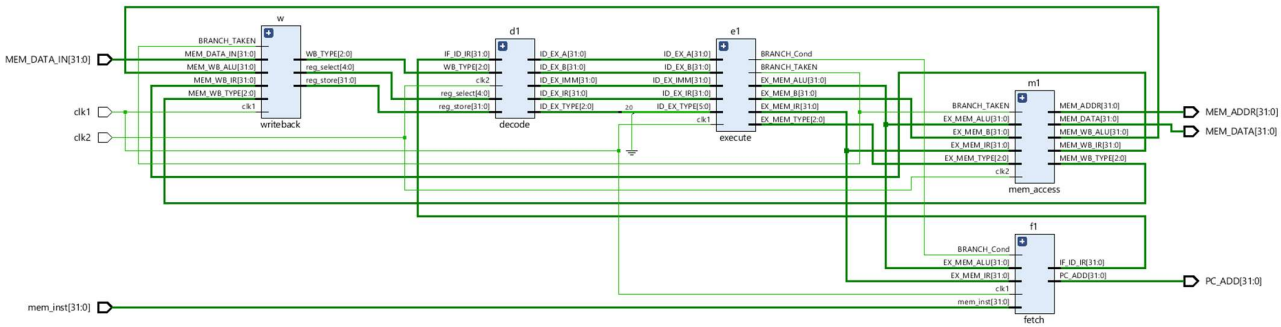


Fig 3.11: Schematic 2

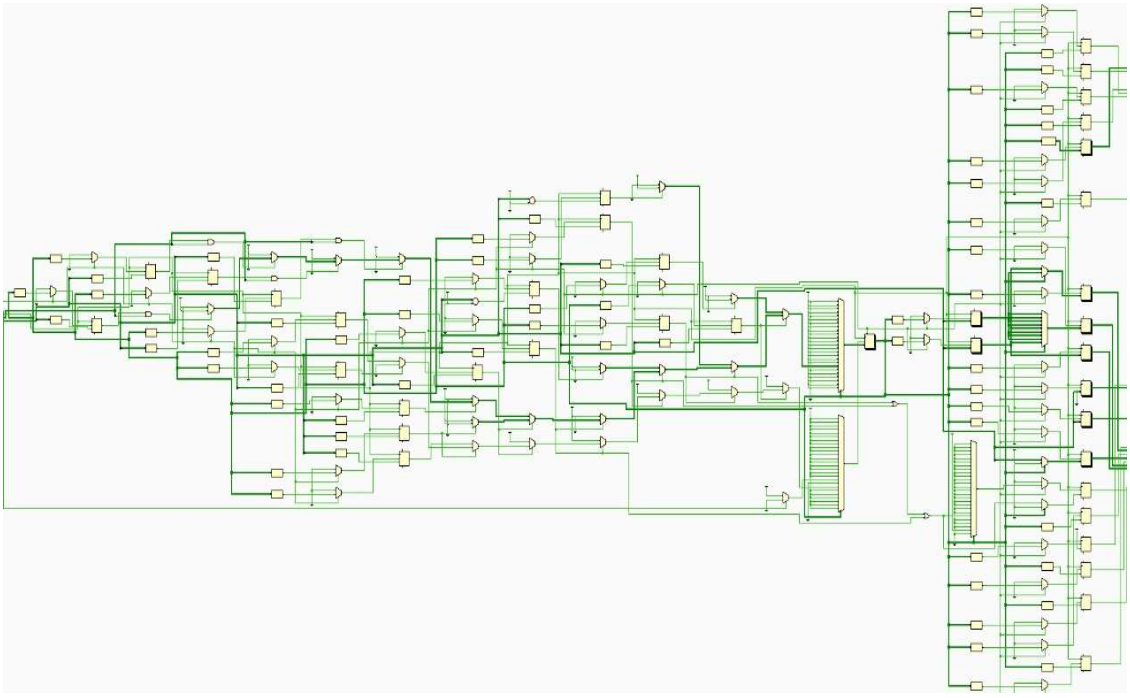


Fig 3.12: Schematic 3

- **POWER REPORT**

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	25.002 W
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	125.0°C
Thermal Margin:	-213.4°C (-17.5 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

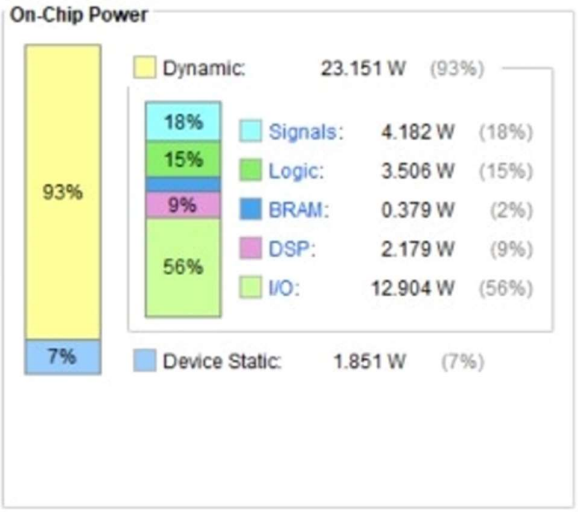


Fig 3.13 Power Report

CHAPTER 4: SELF TESTING RISC PROCESSOR

4.1 INTRODUCTION

Due to its free and accessible ISA characteristic, the RISC Instruction Set Architecture has become increasingly popular. The term "open source" refers to the ability of a designer to modify the instructions plus concatenate extensions as needed for a particular application. RISC also has modular ISA and different intellectual copyright suppliers. RISC has been accepted by the SoC industries and academia since the main goal of SoC is to reduce manufacturing costs, whereas researching on black box equipment is a big stumbling block in the research sector. RISC also aids developers by providing a perfect ecosystem in which they may execute their entire hardware/software application onto an FPGA.

Software-defined radio (SDR) is another key discipline that is partnering with RISC for the realisation of its applications Accetti R. et al. describe a processor architecture for SDR applications that is essentially a RISC V ISA along with an expansion of instruction set [23]. The design incorporates a conventional multiplication extension core processor, a 32-bit RISC-V processor with integer base instructions, and an Oolong baseband computing accelerators unit.

The primary processor core is a three-stage in-order pipeline Harvard design that employs the RV32M and RISC-V integer base commands for integer divisions and multiplications (RV321). To examine the performance of RISC-V, this study was written. The Dhrystone 2.1 benchmarked results were compared to the existing 3-stage pipeline ARM core results. Following the high demands for RISC in many industries, particularly if the end products are utilised in various fields, it is critical that RISC processors be self-testable.

The motive of this project is to make the RISC a self-testing processor with minimum logic to be added. The new logic is added in such a way that it will take minimum space and consume minimal power and for the same to proof, a power analysis is also performed in order to check the increase in power and area of the new design. Now, In order to upgrade the RISC processor in a self-testing processor, DFT (Design for Testability) is going to play a very important role. If we want to comment on the testability feature of any device, it is only possible if we know how much of the design is controllable and observable.

Controllability in DFT points towards the controllable feature of the flipflop in the design and same with the observability. So, the flow that we are going to use in order to make the RISC processor, a self-testing processor is as follow:

First, we are going to perform the scan insertion in which we will add scan flops at each input and output of every inner module.

Next, we will do the connection between these flops in the way that they will work as a register with one input and one output. After the connection, a wrap kind of structure will form around every inner module. All these wrappers are connected in a serial fashion and the only input of this chain will going to connect with TDI of JTAG and similarly the output will connect to the TPO of JTAG.

Next step is to Implement the Joint Test Action Group (JTAG) in the main design. The work of JTAG will to send patterns into the RISC and get the response out of it. After that we will compare the output response with the golden response with the help of comparator. Also, this comparator is attached with the memory in which the golden responses are present.

4.2 PROPOSED MODEL

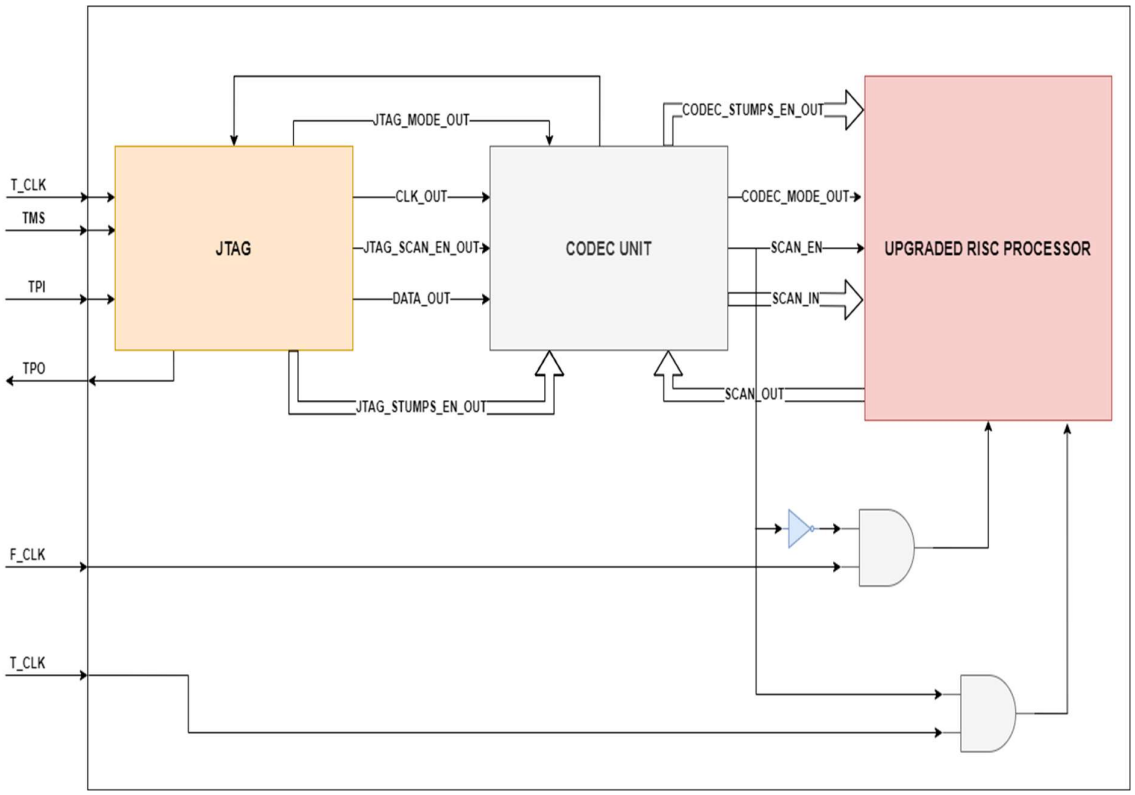


Fig 4.1: Proposed Architecture of Self-Test RISC Processor

In the proposed model, first we have designed the 5 Stage RISC processor and under how the data flow in the design. Now, the motive of testing is to get whether the data we are providing is processing correctly or not and to achieve the same we require to define the testing points in the design of the RISC processor.

The Test points are defined as the points in the design where we insert the scan registers and at the I/Os of all the registers, we can have the scan chain break point.

In our design, the main modules are the different five stages which transfers data from one stage to other. So, if the data that is present at the input of one stage is wrong then the output of that particular stage will also be wrong and definitely we will get wrong output at the final stage. That's why the main test points are the inputs and outputs of every five stage. These are the points where scan registers are present.

Now, the overall design is consisting of three main modules named as:

- RISC Processor
- Codec Unit
- JTAG Unit

The RISC processor is already designed and implemented in previous session. The only difference or the upgradation that is implemented is the addition of scan registers at the I/Os of every state and all these registers are connected in the serial fashion that's why three types of pins are also added in the RISC design. The total number of pins added to the RISC processor are thirteen pins consists of six Scan_in pins, six Scan_out pins and one Test_clk pin

Three types of pins are added to the design of RISC processor names as:

- Scan_en pin: This pin enables the test mode of the module under observation.
- Scan_in pins: This pin is basically defined as the input of scan registers.
- Scan_out pins: This pin is basically defined as the output of scan registers.
- Test_clk: The clock on which the testing is happening.

The thirteen pins added to the RISC processor are interacting with the Codec Unit.

Modes of Testing

There are Two modes of testing in our design named as:

- Fullscan Mode
- Compression Mode

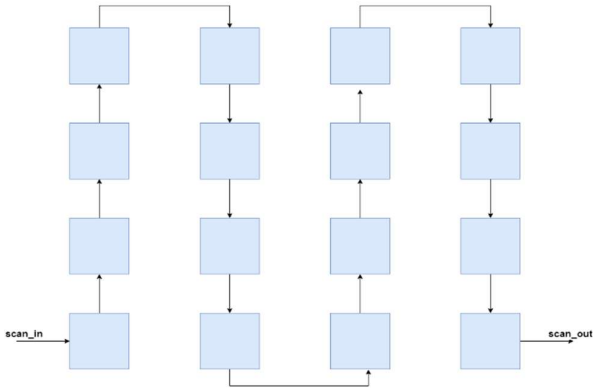


Fig 4.2: Scan chain configuration in Fullscan mode

In Fullscan mode, the whole chain participates in testing and we need to apply on one pattern for testing the whole design. The pros of Fullscan mode is that only pattern is required so if we need to testing whole design then Fullscan mode is the best option but cons are that it takes large number of clock cycles that results in more testing time and if we need to check or test only some part of the design then it will take more time than required. The above figure shows the Fullscan mode.

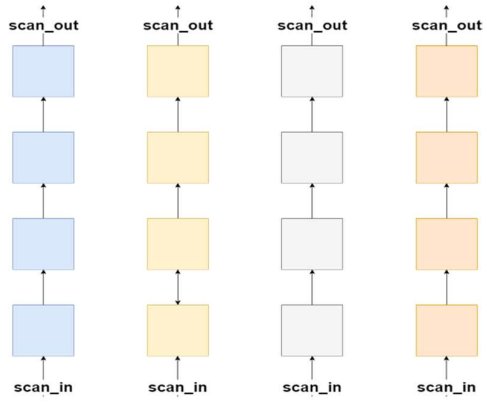


Fig 4.3: Scan chain configuration in Compression mode

In Compression Mode, the scan chain is divided into some parts according to the design. In our design compression mode divide the scan chain into four parts. Basically, if we need to test only some parts of the design then it is the best option to opt and it takes only required amount of clock cycles. But if we talk about the cons then it adds extra logic to the design which we called Codec Unit. The Codec Unit

manage all the small scan chains in order to perform the compression mode. The above figure shows the Fullscan mode.

The Codec Unit is basically used to perform the Compression mode of DFT. The hierarchal chain of scan chains is required to be manage properly in the way of providing different patterns to different chains and the outputs getting out are required to be processed. All this work is handled by the codec unit.

4.3 INTERNAL ARCHITECTURE

There are three components present in the proposed model named as:

- Upgraded RISC Processor
- Codec Unit
- JTAG Unit

4.3.1 UPGRADED RISC PROCESSOR

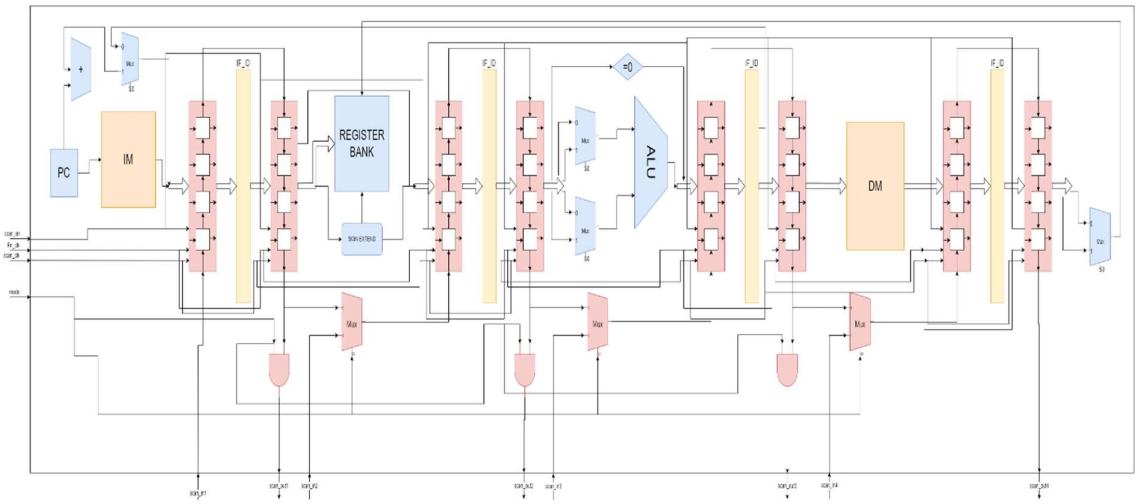


Fig 4.4: Internal Architecture of Upgraded RISC Processor

As shown in the red boxes, the scan flops are present and the number of scan flops to be present at the input and the output of all the modules depends upon the design. In our design every stage process 32 bits of data at one time that why the size of scan registers is also 32 bits.

As discussed earlier, the major modules of the previous RISC design should be wrapped around by the scan registers in order to make them controllable and observable. The main modules are the five stage blocks which act as transparent when clock is provided else they remain opaque. The reason of choosing five stage blocks is because they are the only main modules present between the inputs and outputs. The process of inserting the scan registers in the design is called scan insertion.

Because of scan insertion some pins are added in the design. Total 12 pins are added and their detailed information is given in below Table 4.1:

SIGNALS	I/O DESCRIPTION	PIN DESCRIPTION
scan_en	INPUT	This pin is basically used to switch mode from normal to test mode.
mode	INPUT	This pin is used to define the type of configuration used in scan chains
Fun_clk	INPUT	This pin indicates the normal clock.
Scan_clk	INPUT	This pin indicates the test clock.
Scan_in [1,2,3,4]	INPUT	This pin is used to supply patterns to the processor.
Scan_out [1,2,3,4]	OUTPUT	This pin is used to get response out from the processor.

Table 4.1: Pin description of additional signals added to the normal RISC processor

These pins are basically the part of the scan registers. There are two scan registers for individual stage block. So, in total 8 scan registers are added. As per the figure shown above, it is clear that the logics and upgradation are definitely adding the testing attribute to the old RISC processor but it also increasing the hardware requirement. This hardware requirement also leads to the more power consumption by the design. This is also seen as the disadvantage of the DFT in the ASIC design flow.

4.3.2 CODEC UNIT

The codec plays a very important role in DFT. After scan insertion, we have scan chains present but all these scan chains required a unit or module to manage them. In general, codec works is to select some of the scan chains according to the requirement and provide the patterns. In complex designs, some of the scan chains are connected to the analog devices whose output values are very dynamic in nature so this is also the responsibility of codec to mask the values of that particular flop to the desired value. This is called masking. Now in order to understand the functioning of codec in detail, let's take an example.

Example: Let's say we have 100 chains available to cover whole design but we need to test only 25% of the design. In this case, codec will gate all the chains who are not mean to cover the desired area and by mean gating it will provide a constant value to the flops so that toggle will not occur. This is one function of codec.

Second, responsibility of codec is to manage the responses from the scan chain. The responses the multiple chains that are active for 25 % coverage will send the data parallelly to the codec. Now, it is the duty of codec to dump the data at the output in serial fashion.

Third task of the codec is to mask the targeted flops it they are going to communicate with analog modules.

ARCHITECTURAL DIFFERENC BETWEEN THE RISC PROCESSOR AND SELFT TESTING RISC PROCESSOR

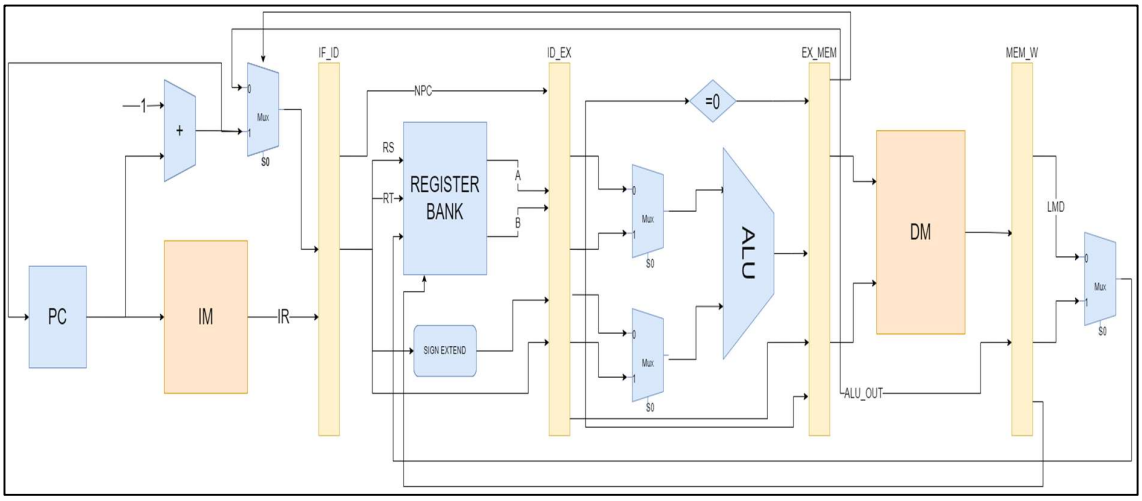


Fig 4.5: RISC Processor (Reproduced from [3])

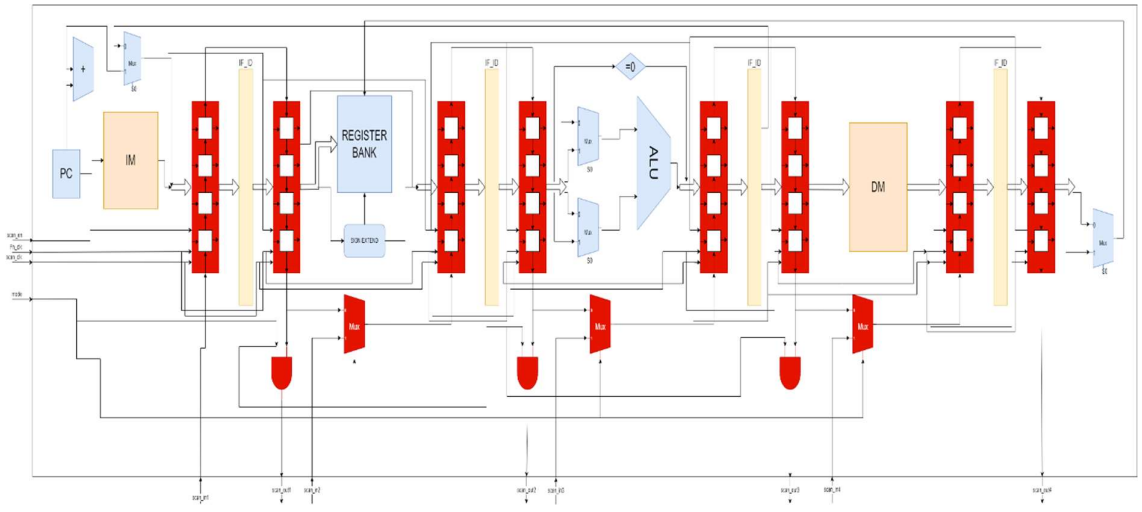
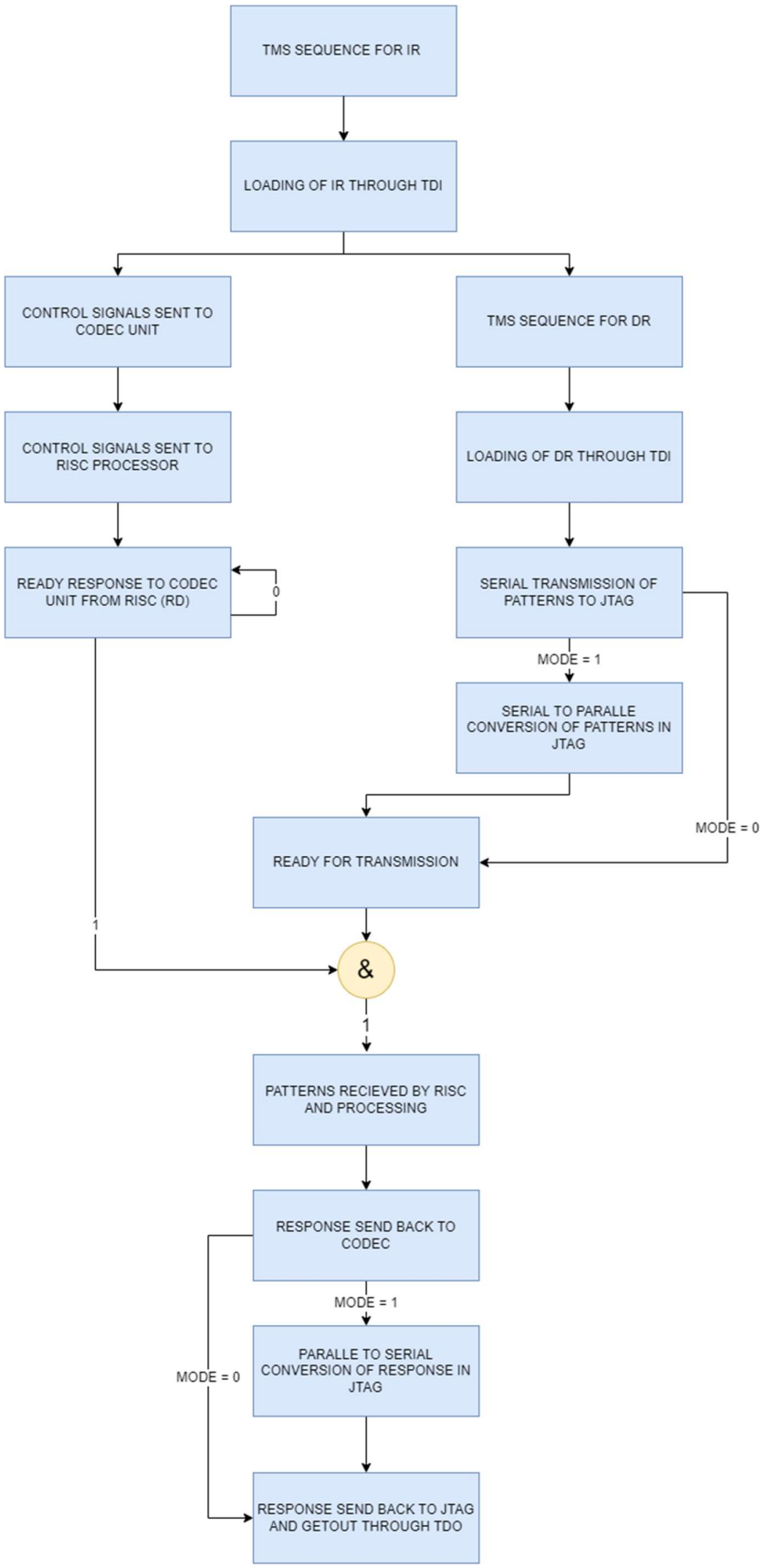


Fig 4.6: Updated RISC Processor

The extra logic added for making testing faster leads to an increase in hardware requirement. This is the disadvantage of DFT which results in the trade-off between the testing time and power consumption because more hardware requires more power.

4.4 OPERATIONAL FLOW



The flow of the operation of testing start with the JTAG (Joint Test Action Group). First, the instruction register is loaded. The instruction contains following information:

- 1-bit of Fullscan or Compression mode.
- 4-bit for selecting the pipeline stage registers, this is only possible in compression mode else the value remains zero for all four bits in Fullscan mode.

After loading the instruction register with desired information, The JTAG control unit generate some signals which goes as an input to the codec unit. Basically, these signal which act an input in codec, perform the connections in the codec. For example, let's say the instruction register was loaded with the information of compression mode and only two of the pipeline registers are selected for the testing. Now the signals are generated in such a way that it only activates the scan chain that are connected with the desired pipeline registers.

Now, the connections are ready for processing the patterns. Next thing is to switch the control from instruction register to data register using the control unit of JTAG. The sequence for selecting the instruction register is:

TC_CLK: JTAG Clock

TC_TMS:

- 0 -> Move to "Run_test_idle"
- 1 -> Move to "Select_dr_scan"
- 1 -> Move to "Select_ir_scan"
- 1 -> Move to "Capture_ir"
- 0 -> Move to "Shift_ir"
- 0 -> Remain in "Shift_ir" (Data bit 0)
- 0 -> Remain in "Shift_ir" (Data bit 0)
- 0 -> Remain in "Shift_ir" (Data bit 1)
- 1 -> Exit1_ir
- 1 -> Update_ir

The sequence is applied at the tms pin of the JTAG. Next for the switching to data register, sequence is as follow:

TC_CLK: JTAG clock

TC_TMS:

- 1 -> Move to "select_dr_scan"
- 0 -> Move to "capture_dr"
- 0 -> Move to "Shift_dr"
- 0 -> Remain in "Shift_dr" (this will be in same state for 162 clock)
- 1 -> Move to "exit1_dr"
- 1 -> Move to "update_dr"

The patterns are sent through TDI of JTAG to the codec. Now, in codec these patterns are stored in registers and send parallelly to the specific scan chains of the RISC processor in serial fashion through output pin of the codec.

Now, before sending the patterns to the RISC processor the scan enable becomes high which shows that RISC is now in testing mode. All the scan flops are getting the scan_en input as logic '1'. The patterns

are now feed to the processor. The number of clock pulse is will take depends upon the design. In this design, the size of one pattern is 256 bits.

Now, the scan_en is set to logic ‘0’ means that RISC is in functional mode and one functional clock is applied. Because of this functional clock, output of all the pipeline registers will get the new value. These outputs are now required to be sent out through scan chains only. This we called scan response. The response is sent out of the processor to codec in parallel fashion through the output of each individual scan chains. The codec performs a parallel to serial conversion and send out the data to serially to the data registers of JTAG where we can store it or send the response out through the TPO pin and after that we can compare the response with the golden response to check whether the design have any faults or not and if it has any then in which part of the design fault occurs.

Each bits of the response that we get at the end indicates each input and output pin of the pipeline register so if any bit doesn’t match with the golden response that means that particular bit has a stuck-at fault. For example, if the faulty bit as logic ‘1’ that means the faulty pin got connected with voltage supply because of fabrication process. That’s how boundary scan testing of RISC processor is performed.

CHAPTER 5: HARDWARE IMPLEMENTATION AND SIMULATIONS

5.1 INTRODUCTION

This chapter primarily focus on the simulations and hardware implementation based on the operation flow discussed in previous chapter. The simulations are divided into four parts:

1. JTAG loading.
2. Codec unit loading.
3. Pattern processing in Codec and response generation
4. Pattern processing in RISC and sending response back to JTAG (TDO)

After simulations, hardware implementation is discussed. This involves the RTL schematic of overall design showing all the signals and three main modules. Then, netlists of upgraded RISC processor and important units are shown. At last power analysis is also performed in order to study the rise in power consumption after implementing testing logic in RISC processor.

All simulation and schematic generations are performed in AMD Xilinx VIVADO 2020.2.

5.2 SIMUALTIONS

- **JTAG Loading**

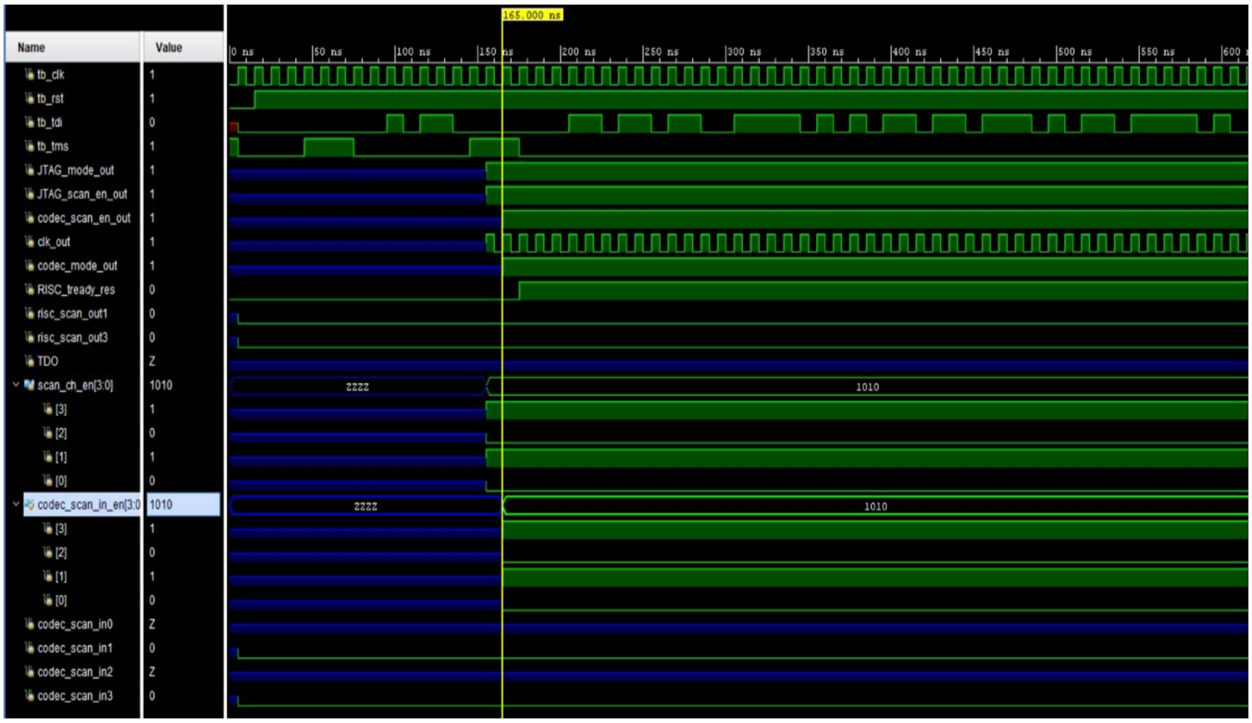


Fig 5.1: Simulation showing the loading of instruction register through TDI. The marker shows the generation of control signals according to instruction.

- **Codec unit loading**

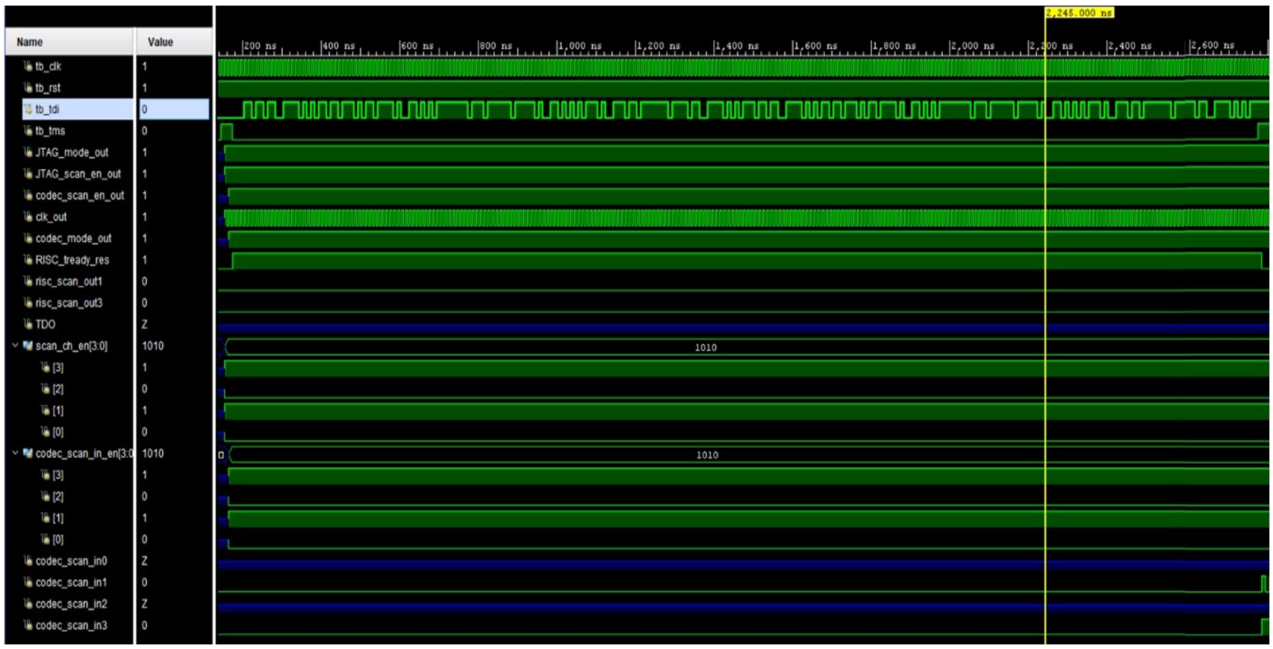


Fig 5.2: Simulation showing the loading of data register through TDI

- **Pattern processing in Codec and response generation**

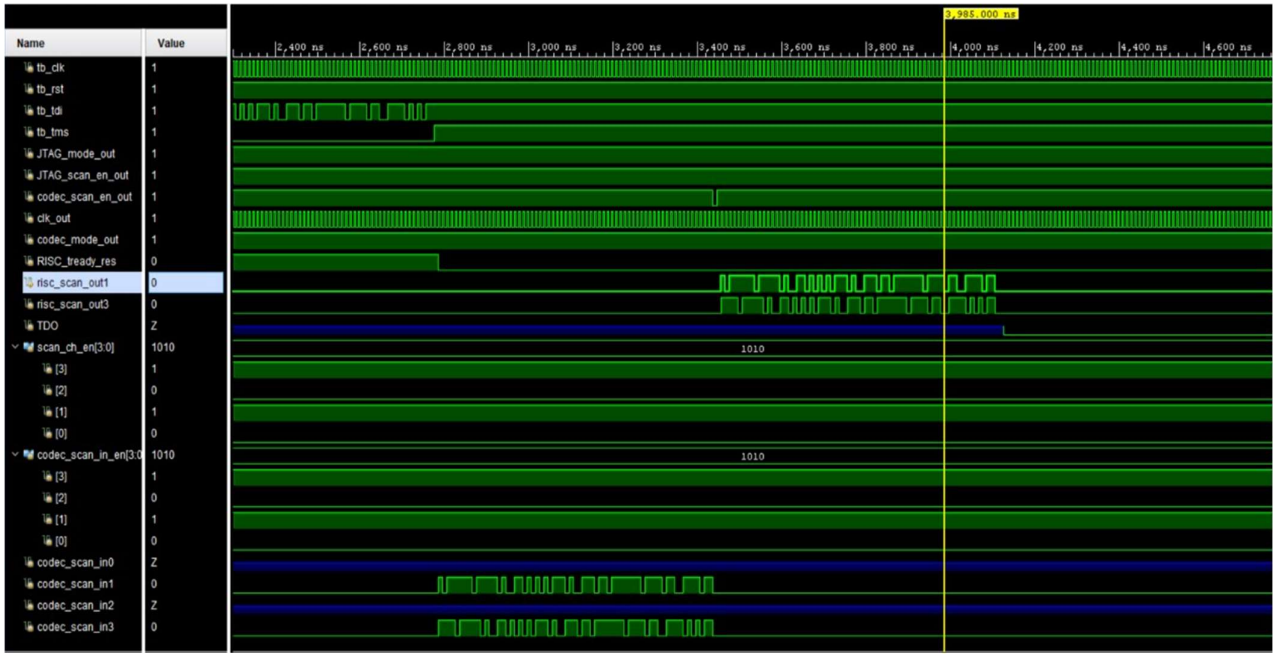


Fig 5.3: Simulation showing the serial to parallel conversion of data according to control signals from JTAG

- **Pattern processing in RISC and sending response back to JTAG (TDO)**

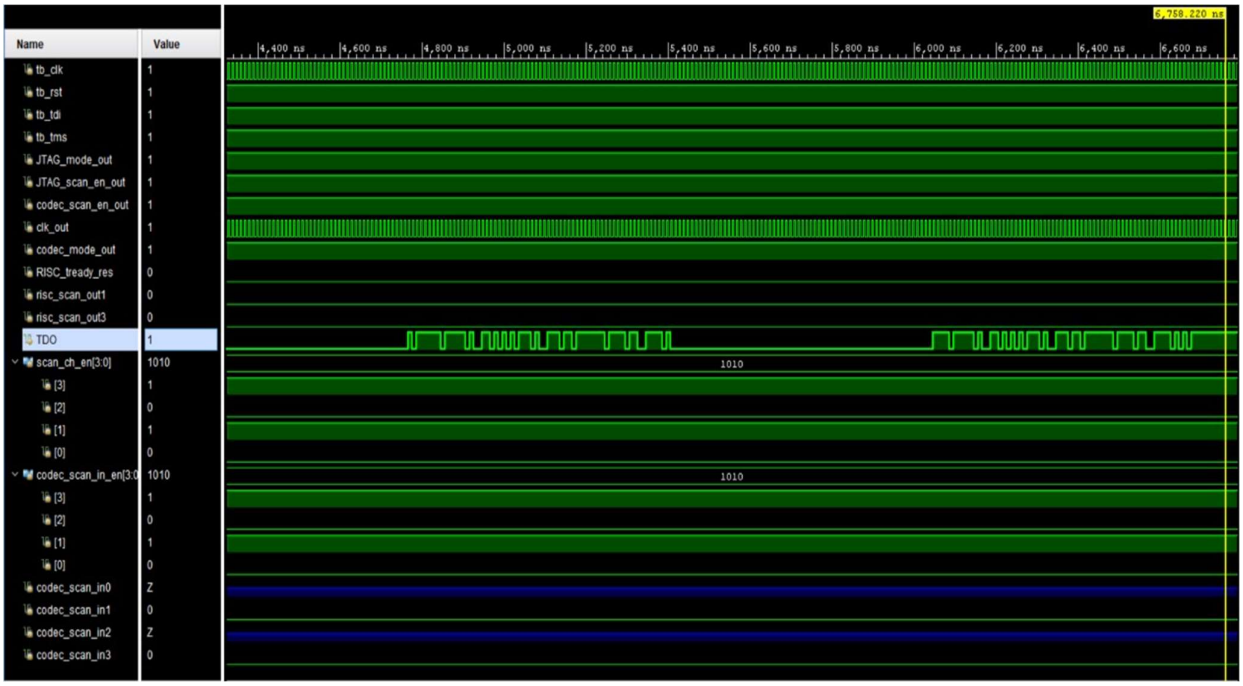


Fig 5.4: Simulation showing the serial transfer of response out from TDO

5.3 HARDWARE IMPLEMENTATION

- **Overall Design**

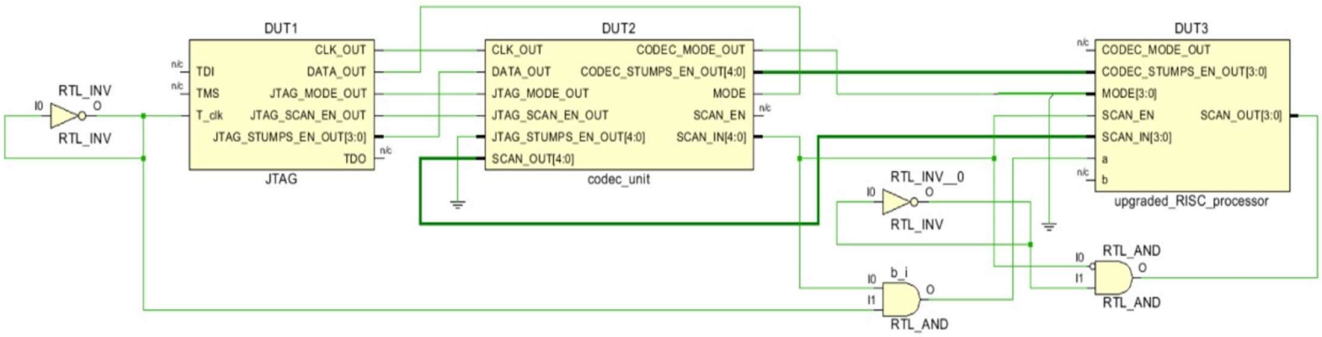


Fig 5.5: RTL schematic showing overall design

- **Scan Flipflop**

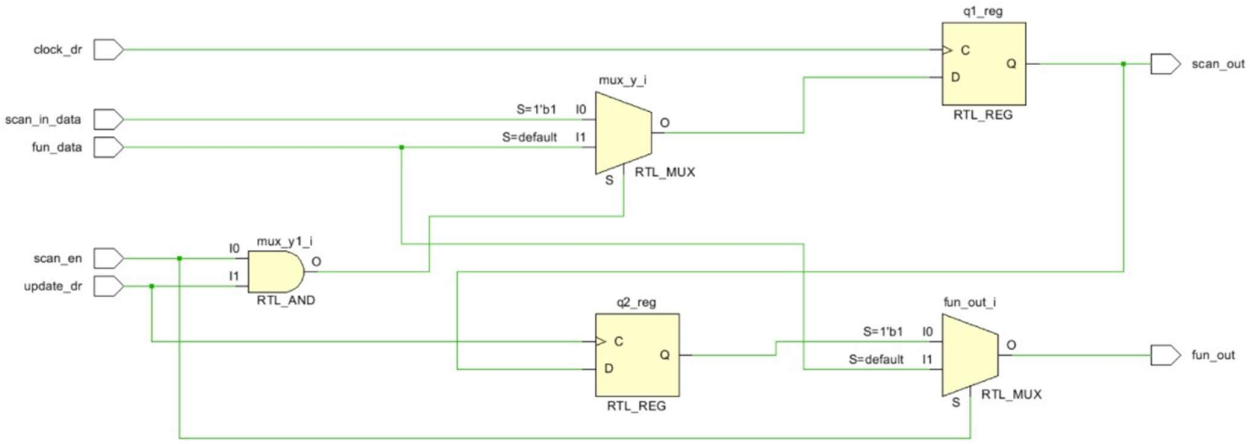


Fig 5.6: RTL schematic showing scan flipflop

- **Scan Register**

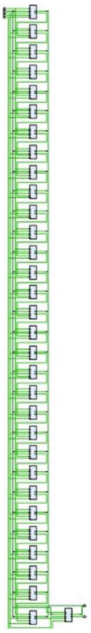


Fig 5.7: RTL schematic showing scan register

- **Upgraded RISC processor**

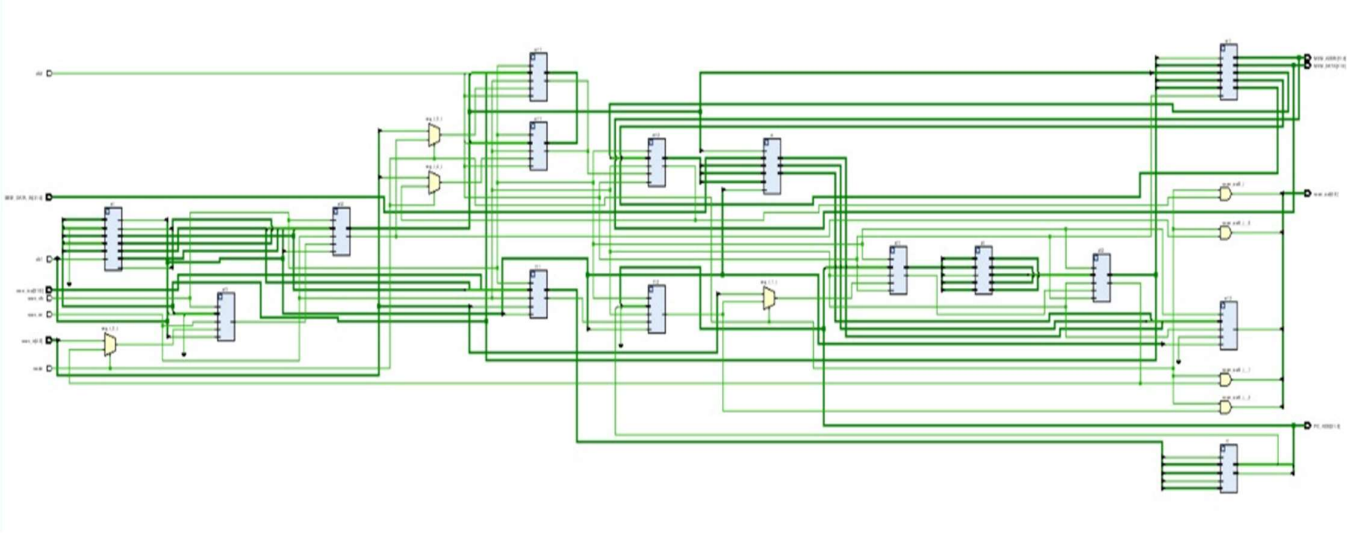


Fig 5.8: RTL schematic showing upgraded RISC processor

- **Design Netlist**

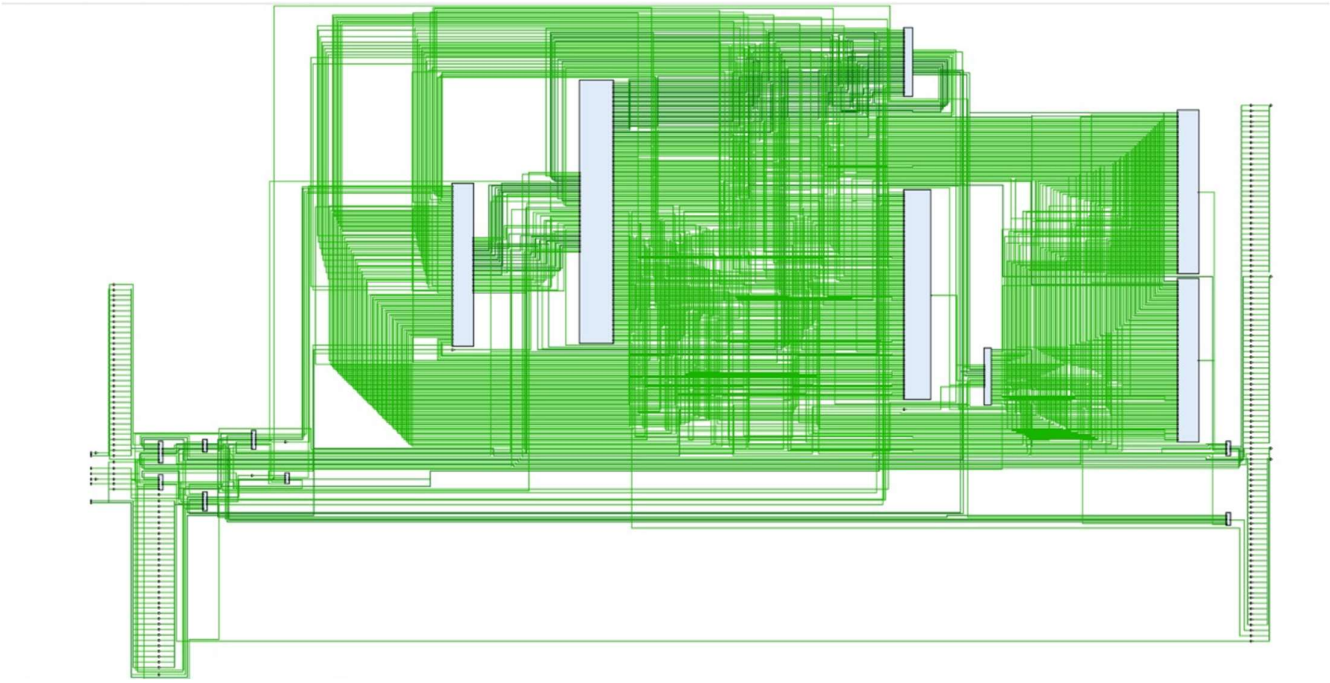


Fig 5.9: Netlist schematic showing overall design

- **Power Analysis**

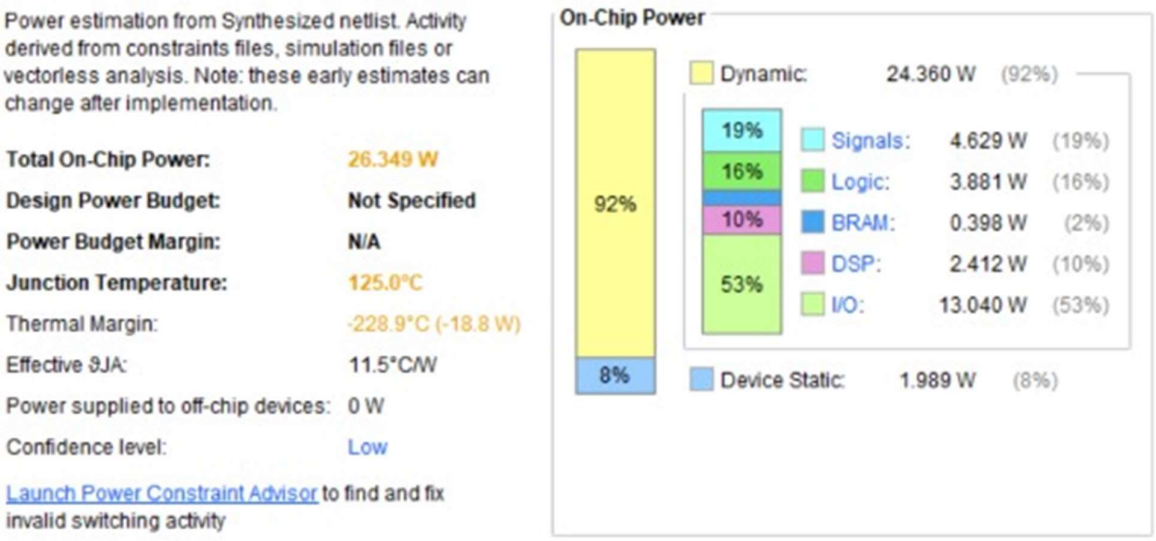


Fig 5.10: Power report of overall design

The power consumption generated in self-test RISC processor and RISC processor are **26.349 W** and **25.002 W**. So, the percentage increase in power is **5.875%** as compare to the feature added to the design. The same is shown in the graph also.

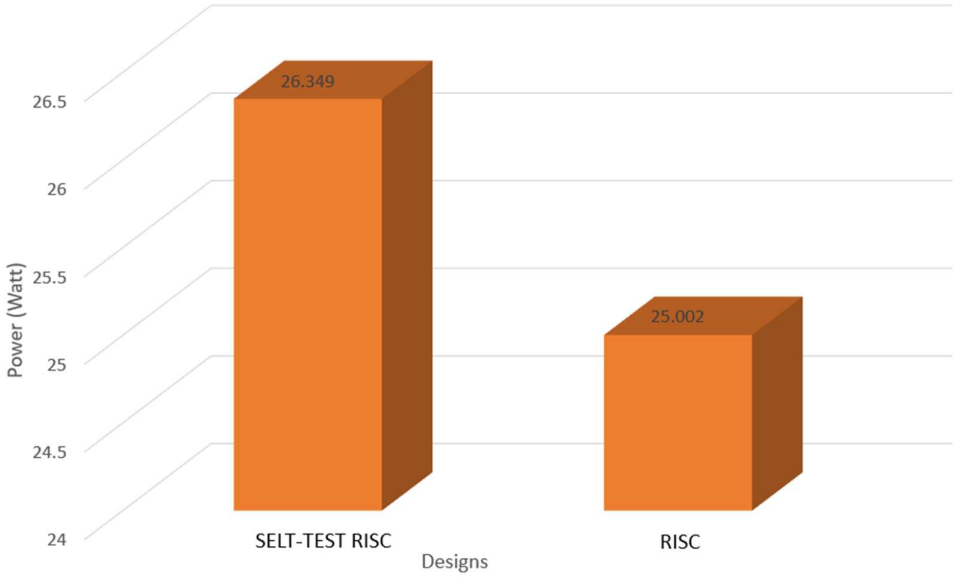


Fig 5.11: Power Comparison of both Designs

CONCLUSION AND FUTURE SCOPE

The Self-Testing RISC Processor is designed and implemented using AMD Xilinx VIVADO 2020.2. According the power results, it is clear that the testing logic added in order to make the normal RISC processor – A self-testing processor leads to increase in power consumption of all over design. Also, the area required to implementation is increased to a very good extent. As we observed that the power increased is 5.387 % which is a moderate increase in power so we can conclude that the increase in power is acceptable because the testing it adds reduced the testing timing drastically which will result in the cost cutting of the device. But the power reducing will remain an alarming issue in the VLSI industry so in order to decrease the power consumption during testing we can add Low power gating CODEC instead of normal CODEC. This will reduce the power exponentially at a particular instance of time. Then, the trade-off will be between area and power only.

REFRENCES

[1] S. Furber, Arm System-on-chip Architecture, 2nd edition, UK: Addison-Wesley ,2000.

[2] Sivarama P. Dandamudi, Guide to RISC Processors, 1st edition, NY: Springer-Verlag New York, 2005.

[3] M. Zaid, P. Mustajab, "DESIGN AND APPLICATION OF RISC PROCESSOR" 2017 International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT), pp. 242-246, November 2017.

[4] Prof. James CM Lei, VLSI Testing, Design for testability. Part II: External scan JTAG, Lab of dependable system, National Taiwan university.

[5] P. Srivastva, P. K. Yadav, P. K. Misra, "Design of 32 bit Asynchronous RISC CPU Using Micropipeline" 2018 Conference on Information and Communication Technology (CICT'18), pp. 1-5, October 2018.

[6] P. Pfeifer, H.T.Vierhaus, "On Implementation and Usage of Muller C-element in FPGA-based Dependable Systems" 2017 International Conference on Applied Electronics (AE), pp. 1-4, September 2017.

[7] Gokulan T, A. Muraleedharan, K. Varghese, "Design of a 32-bit, dual pipeline superscalar RISC-V processor on FPGA" 2020 23rd Euromicro Conference on Digital System Design (DSD), pp. 340-343, Aug 2020.

[8] S. Qureshi, K. R. Sanjeev, "Power and Performance Optimization using MultiVoltage, Multi-Threshold and Clock Gating for LowEnd Microprocessors" TENCON 2009 - 2009 IEEE Region 10 Conference, pp. 1-6, Jan 2009.

[9] Kuen Jong Lee, Zheng Yao Lu, Shih Chun Yeh, "A Secure JTAG Wrapper for SoC Testing and

Debugging", IEEE Access, pp. 37603-37612, 2022

[10] S. Palekar, N. Narkhede, "32-bit RISC Processor withFloating Point Unit for DSP Applications" IEEE International Conference On Recent Trends In Electronics Information Communication Technology, pp. 2062-2066, May 2016.

[11] D. Babayan, E. Babayan, S. Antonyan, A. Salmasyan, E. Kagramanyan, A. Avetisyan, "A new Approach of Multi Voltage and Adaptive Voltage Scaling Techniques for 16 nm FinFET RISC Processor" 2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO), pp. 128-131, April 2017 .

[12] C. Vinoth, V. S. K. Bhaaskaran2, B. Brindha, S. Sakthikumaran, V. Kavinilavu, B. Bhaskar, M. Kanagasabapathy, B. Sharath, "A Novel Low Power and HighSpeed Wallace Tree Multiplier for RISC Processor" 2011 3rd International Conference on Electronics Computer Technology, pp. 330-334, April 2011.

[13] H. Najjar, R. Bourguiba, J. Mouine, "A reusable hybrid RISC processor with programmable instruction set" 2018 15th International Multi-Conference on Systems, Signals & Devices (SSD), pp. 1028-1031, March 2018.

[14] S. Pandit, P. Sikka, "DESIGN AND IMPLEMENTATION OF POWER OPTIMIZED DUAL CORE AND SINGLE CORE DLX PROCESSOR ON FPGA" 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), July 2018.

[15] S. M. Bhagat, S. U. Bhandari, "Design and Analysis of 16-bit RISC Processor" 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), pp. 1-4, August 2018.

[16] R. Samanth, A. Amin, S. G. Nayak, "Design and Implementation of 32-bit Functional Unit for RISC architecture applications" 2020 5th International Conference on Devices, Circuits and Systems (ICDCS), pp. 46-48, March 2020.

[17] C. Venkatesan, T. Sulthana M, Sumithra M.G, "Design of a 16-Bit Harvard Structure RISC Processor in Cadence 45nm Technology", 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS), pp. 173-178, March 2019.

[18] J. Jeemon, "Low Power Pipelined 8-bit RISC Processor Design and Implementation on FPGA" 2015 International Conference on Control,Instrumentation, Communication and Computational Technologies (lCCICCT), pp. 476-481, December 2015.

[19] J. Andersson, "Development of a NOEL-V RISC-V SoC Targeting Space Applications" 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 66-67, July 2020.

[20] J. Lee, H. Chen, J. Young, H. Kim, "RISC-V FPGA Platform toward ROS-based Robotics Application", 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), pp. 370, September2020.

[21] M. Numan Ince, J. Ledet, M. Gunay, "Building An Open Source Linux Computing System On RISC-V" 2019 1st International Informatics and Software Engineering Conference (UBMYK), pp. 1-4, November 2019**.**

[22] S. Murthi, U. Verma, "FPGA based Implementation of Power Optimization of 32-Bit RISC Core using DLX Architecture" 2015 International Conference on Computing Communication Control and Automation, pp. 964-968, Feb 2015.

[23] Cecil Accetti R. A. Melo, E. Barros, "Oolong: A Baseband processor extension to the RISC-VISA" 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 241-242, July 2016.

[24] Hyunyoung Oh, J. Park, M. Yang, D. Hwang, Y. Paek, "Design of a Generic Security Interface for RISC-V Processors and its Applications" 2018 International SoC Design Conference (ISOCC), pp. 40-41, November 2018.

[25] Be Van Ngo, Peter Law, "Use of JTAG Boundary-Scan for Testing Electronic Circuit Boards and Systems", IEEE AUTOTESTCON, pp. 1-5, September 2008.

[26] Shashidhara HB, Vasant Goudanavar, Dr. Siva yellampalii, **"**Board Level JTAG/Boundary Scan Test Solution**",** Proceedings of International Conference on Circuits, Communication, Control and Computing (I4C 2014), pp. 73-76, November 2014**.**

[27] Huiguo Zhang, Yulan Tang, Zongguang Yu, "An FPGA Configuration Circuit Based on JTAG", IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 2588-2590, June 2009.

[28] Yassine Fkih, Pascal Vivet, "A JTAG based 3D DfT architecture using automatic die detection", Ph.D. Research in Microelectronics and Electronics (PRIME), pp. 341-344, September 2013.

[29] Perumalla Giridhar, Dr Priyanka Choudhury, "Design and Verification of AMBA AHB", Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE), pp. 310-315, March 2019.

[30] POORANI.P, Ms. VIJAYASHREE.B M.E, "IMPLEMENTATION OF AHB BUS PROTOCOL FOR SYSTEM ON CHIP SECURITY", 2'ND INTERNATIONAL CONFERENCE ON ELECTRONICS AND COMMUNICATION SYSTEMS(ICECS), pp. 1407-1413, February 2015 .

[31] Ashutosh Kumar Singh, Anurag Shrivastava, G.S. Tomar, "Design and Implementation of High Performance AHB Reconfigurable Arbiter for Onchip Bus Architecture", International Conference on Communication Systems and Network Technologies, pp. 455-459, July 2011.

[32] WANG Zhonghai, YE Yizheng, WANG Jinxiang, W Mingyan, "Designing AHB/PCI bridge", nternational Conference on ASIC, pp. 578-580, August 2002.

[33] EE Times DFT Tutorial 2011, March 27th Edition

[34] Mahmut Yilmaz, 2009 Dessertation, Automated Test Grading and Pattern Selection for SmallDelay Defects, Duke University

[35] L.T. Wang, C.W. Wu, and X. Wen, "VLSI Test Principles and Architectures", Morgan Kaufmann, 2006abrir ecommerce