Lab Oriented Project Report

On

Final Specification

Submitted in partial fulfilment

of

System-on-Chip

Under the Guidance of :
Prof. Dr. A. Siggelkow

Submitted By :
Jayant Patil (18041745)
Aditya Grewal (29541041)
Avanindra Kumar Mishra (20241804)
Sagar Shreeshailappa Hosmani (27741741)

MASTER OF ELECTRICAL ENGINEERING AND EMEBDDED SYSTEMS

HOCHSCHULE RAVENSBURG-WEINGARTEN

January 15, 2024

Contents

List of Figures

List of Tables

1. Requirements

Requirement	ID	Importance	Verifiable	Description
Register file (peripheral)	G01	High	VHDL-TB	An IP with 32x 32-bit registers is designed to be interfaced as a peripheral with the ARM Cortex-A9 core. This register file acts as an slave.
Write access	G02	High	VHDL-TB	Register file (peripheral) provides write access to the other surrounding peripherals.
Read access	G03	High	VHDL-TB	Register file (peripheral) provides read access to the other surrounding peripherals.
Communication between ARM Cortex-A9 core and Register file	G04	High	VHDL-TB	AXI4-Lite bus is utilized to communicate between ARM Cortex-A9 Core and memory mapped Register file (peripheral).
AXI4-Lite channels	G05	High	VHDL-TB	All the five channels of AXI4-Lite bus are essential for communication.
Memory mapping of register file (peripheral)	G06	High	VHDL-TB/ Application program	The register peripheral must be mapped to certain addresses among the memory map of the ARM Cortex-A9 core.
RISC V compliance	G07	High	Application program	The register file (peripheral) must be RISC V compliant, i.e., all RISC V instructions should be able to access it (Ex: R-Type instruction, I-Type instruction etc).
Max registers in Register file (peripheral)	G08	High		There can be only 32 maximum registers within the register file, as RISC V instruction layout only supports 5 bits for addressing both source and destination registers .
Word size	G09	High	VHDL-TB	Word size of the ARM Cortex-A9 core is 32 bit, hence the each register within the register file (peripheral) is of 32 bits wide.
Register file access test	G10	High	Application Program	Two registers of register file (peripheral) are interfaced with Z-Board LEDs and switch from a cross-compiled assembly/C program.
Scan Test	G11	Medium	JTAG	TAP controller is designed to perform scan tests on the register file (peripheral).
Clock access to the TAP controller	G12	Medium	JTAG	TAP controller needs a clock signal to operate the TAP controller state machine. TCLK pin of the TAP controller provides this clock input to the state machine from the JTAG network.

Control signal for the TAP controller	G13	Medium	JTAG	TAP controller needs a control signal to control the TAP controller state machine. TMS pin of the TAP controller provides this control signal to the state machine from the JTAG network.
Instruction data to TAP controller	G14	Medium	JTAG	TDI pin of TAP controller forwards the instruction data from the JTAG network to the state machine.
Input test data to TAP controller	G15	Medium	JTAG	TDI pin of TAP controller forwards the input test data from the JTAG network to the register file.
Collect output test data from the TAP controller	G16	Medium	JTAG	TDO pin of TAP controller collects the output test data from the register peripheral and then sends it to the JTAG network for observation.
Transfer test data from TAP controller to register file (peripheral)	G17	Medium	JTAG	Test data from the TDI of TAP controller can be sequentially cascaded into each flipflop of the register file (peripheral).
Collect test data from register file (peripheral) into TAP Controller	G18	Medium	JTAG	Cascaded test data from the register file (peripheral) will be cascaded back into the JTAG network via TDO of TAP controller.
Processor core	G19	High	Z-Board	ARM Cortex-A9 core present on the Z-Board is used as the main CPU to interface the designed peripheral. This processor core acts as an master.

2. History

Rev. No.	Rev. Date	Description of change in Current Version	Author
V 1.0	15.10.2023	Functional Requirements	Jayant Patil Aditya Grewal Avanindra Mishra Sagar Shreeshailappa Hosmani
V 2.0	30.10.2023	Specifications	Jayant Patil Aditya Grewal Avanindra Mishra Sagar Shreeshailappa Hosmani
V 3.0	15.01.2024	Final Specifications, Design Descriptions and Simulation	Jayant Patil Aditya Grewal Avanindra Mishra Sagar Shreeshailappa Hosmani

3. Document Overview

3.1 Glossary

Type	Description
Integrated Circuit (IC)	An assembly of transistors over a chip.
FPGA	Field Programmable Gate Array (FPGA) An IC that can be programmed after manufacture.
AMBA	Advanced Microcontroller Bus Architecture (AMBA) An architecture for system-on-chip (SoC)
AXI	Advanced eXtensible Interface (AXI) A high-performance protocol by ARM.
APU	Application Processing Unit (APU) A section of Zynq Processing System that includes ARM processor cores, and their associated cache memories and NEON engines.
Processing System(PS)	Section that includes the processor and associated functionalities.
Programmable Logic (PL)	IP package that handles tasks in the CPU.
ARM	Processor architecture of the Zynq board used is ARM Cortex A9
Bus	Pathway for communications between CPU and peripherals
Debug	Process to protect the system from manufacturing failures and errors.
GPIO	General Purpose Input Output (GPIO) digital pins that can be used as input, ouput or both
HDL	Specialized high-level description language (HDL) that is used to program ASICs and FPGAs.
System-on-Chip (SoC)	An IC that has all components in a computer integrated into a single chip.
JTAG	Joint Test Access Group (JTAG) A method to test electronic products.
Intellectual Property (IP)	A hardware specification that helps set up the logic components of an FPGA.
BSP (Board Support Package)	Set of files that provides software support for a specific hardware platform.
Clock Management Unit (CMU)	Function is to control clocks and oscillators.
ISA	Instruction Set Architecture (ISA) Set of instructions inside a CPU.

Table 3.1: Glossary Type Descriptions

3.2 Document List

Number	Document Type	Version	Release Date
1	The Zynq® Reference Manual	1st Edition	August 2015

Table 3.2: List of documents

4. Product Overview

4.1 Introduction

The main aim of this project is to design a custom peripheral interfaced with the ARM microprocessor and deploy it on the Z-Board/Zybo FPGA. The peripheral is a 32 x 32 bit register having read and write access. A synthesised netlist of the custom SoC is deployed onto the FPGA and then further interfaced with the ARM core present on the zynq-7000 SoC (PS+PL) with the help of custom firmware. As a part of a test, one of the registers among the peripherals is used to send data to the LEDs present on the FPGA and another register is used to receive data from the switches on the FPGA. The design is scan-testable with the help of a TAP-controller and scan chain included to perform several test cases. The top-level system overview can be seen in the below figure.

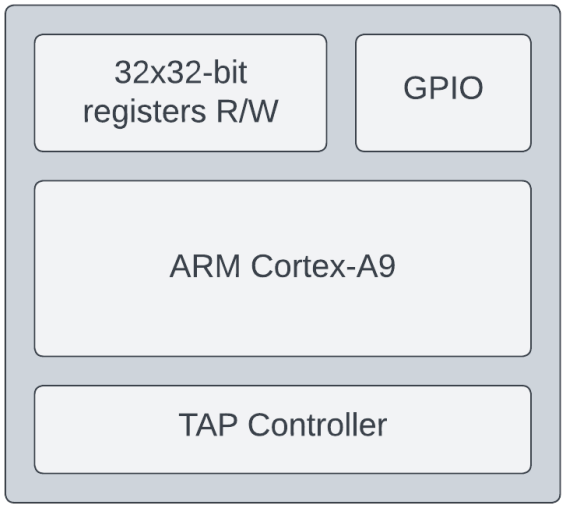

Figure 4.1: System view

4.2 Key Features

Type	Description
Processing Core	ARM Cortex-A9
Peripherals	32x32-bit Registers
Bus System	AXI4-Lite
Debugger	Scan-Chain and TAP Controller

Table 4.1: Key Features

4.3 Functional Block Diagram

The custom IP with 32x 32-bit registers is interfaced as a peripheral with the ARM Cortex-A9 core. The register file designed hereby acts as a slave and the ARM core acts as a master. Both master and slave are interfaced with the AXI4 lite bus. This bus ensures high-speed and reliable data communication between the register peripheral. The registers are RISC-V compliant hence they should be able to be used to execute any of the instructions from the RISC-V instruction set. Hence, any firmware running on the Processing System (PS) of the SoC can use the registers on the Programmable Logic (PL) of the SoC as memory-mapped registers with the help of the AXI4 lite bus. Since this SoC will be deployed into the Zybo development board based on Zynq-7000 SoC, the LED's and Switches present on the board can be accessed and interfaced with the help of custom register peripheral, ARM core and Custom firmware. In order to make the SoC scan testable, a TAP controller is implemented as per IEEE 1149.1 JTAG specification and interfaced with the register peripheral over a scan chain. This TAP controller receives instructions and test data from the JTAG network via standard JTAG serial pins.

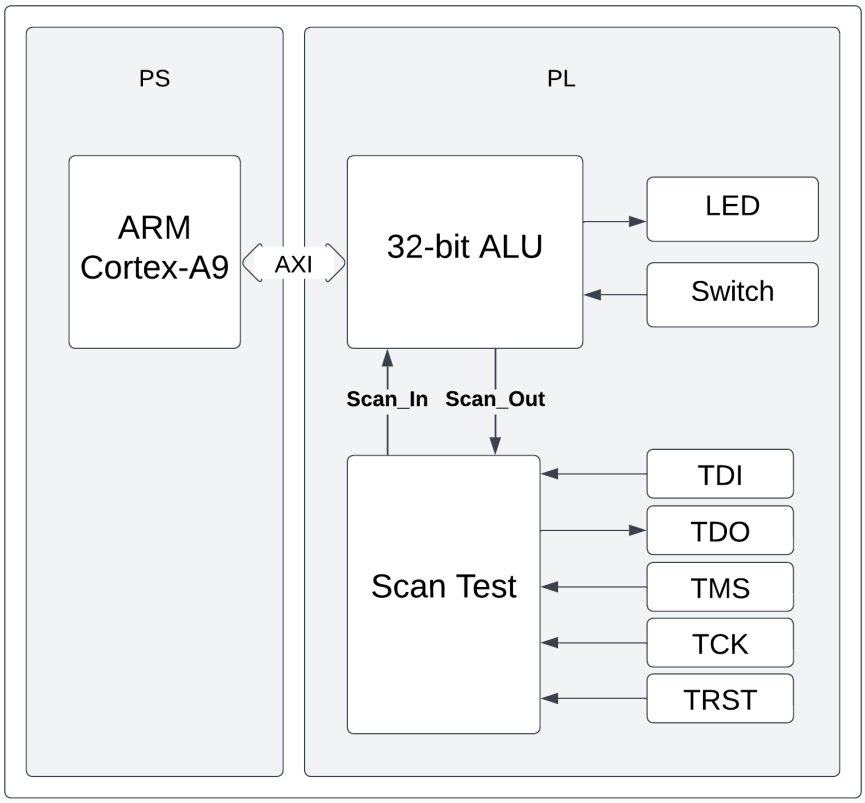

Figure 4.2: Functional Block Diagram

5. Architecture Concepts

5.1 Bus concept

5.1.1 Introduction to AXI4-Lite Bus

The AXI4-Lite interface facilitates communication between the programmable logic (PL) peripheral device and the software running on the processing system (PS). Unlike the standard AXI bus, AXI4-Lite is tailored for simplicity, mainly focusing on the control and monitoring of IP blocks in the ZYNQ device, establishing a connection between ARM and FPGA components.

5.1.2 AXI4-Lite Functionality

AXI4-Lite is designed to handle 32-bit read-and-write data transfers. However, it differs from the standard AXI as it does not supporting variable bus width or cache operations. Each read or write operation allows only a single 32-bit data transfer. Its primary purpose lies in controlling and monitoring IP blocks, serving as an effective communication bridge between the ARM and FPGA elements.

5.1.3 AXI4-Lite Channels

The AXI bus structure incorporates distinct channels for read and write transactions, enabling semi-independent progression. These channels include Read Address, Read Data, Write Address, Write Data, and Write Response. Each channel is equipped with handshaking signals to manage operations.

5.1.4 Read and Write Channels

- Read Address and Read Data Channels: Responsible for moving data from the slave to the master.
- Write Address, Write Data, and Write Response Channels: Transfer data from the master to the slave.
- Concurrent read and write transactions are supported without interference.
- Handshaking signals, such as READY and VALID, are unique to each channel, allowing for efficient coordination.

5.1.5 Concurrent Transactions

Figure 4.2 illustrates the overall flow of AXI4-Lite bus transactions, highlighting the concurrent nature of read and write operations. This flexibility ensures that the Write Address and Write Data channels can operate simultaneously or sequentially, with additional considerations for a second write address (A2) before the completion of the previous write cycle (A0).

5.1.6 AXI4-Lite IP Interface (IPIF)

The AXI4-Lite IP Interface (IPIF) is an integral part of the Xilinx family, providing a bidirectional interface between a user Intellectual Property (IP) core and the AXI interconnect. Its purpose is to simplify the connection between ARM AXI and user IP cores, streamlining the management of different addresses.

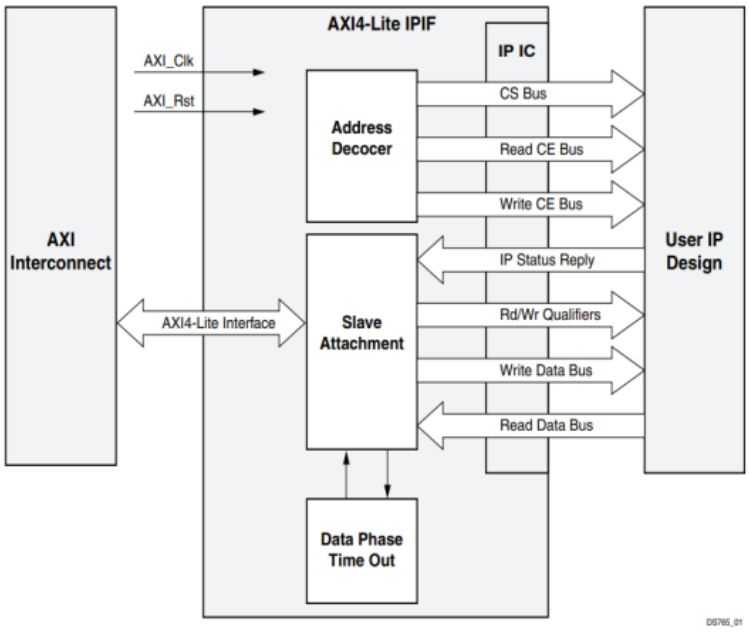

Figure 3: AXI4-Lite IPIF Block Diagram

Figure 5.1: AXI transaction

5.1.7 Features of AXI4-Lite IPIF

- 32-bit Slave Configuration: Supports a 32-bit slave configuration for effective data handling.
- Read and Write Data Transfers: Facilitates read and write data transfers of 32-bit width.
- Multiple Address Ranges: Supports multiple address ranges for versatile IP core integration.
- Read Priority Over Write: Prioritizes read operations over write operations.
- Address Space Handling: Reads to holes in the address space returns 0x00000000, and writes to holes after the register map is ignored and responds with an OKAY response.

5.2 System control concept

The entire system is controlled by a processing system (PS). In this case, the chosen PS is an ARM Cortex A-9 processor. The main significance of the processor is to control the functionalities of the peripherals interfaced with it. In this case, the PS controls the peripherals deployed on the programmable Logic (PL) of SoC over the AXI-lite bus after establishing a successful handshake mechanism.

The custom IP has 32x 32-bit registers. This IP is memory-mapped with the ARM core through an AXI-lite bus. The PS can read and write data into a register file via AXI bus using the base address of the peripheral. The Xilinx design platform is vast and provides an IDE to develop custom drives and firmware for our custom IPs. Vitis is one such IDE where the custom IP which is designed in the Vivado tool can be exported as an HDL wrapper for further application development and testing.

5.3 Clock system

The Zynq-7000 SoC (System on Chip) from Xilinx integrates both a Processing System (PS) and programmable logic (PL). The PS includes ARM Cortex-A9 processors, and the clocking architecture involves several clock sources. Here are some of the key clock sources in the Zynq-7000:

- **PS Clocks:**

  ARM Cortex-A9 Clocks (CPU_CLK): The ARM Cortex-A9 processors in the PS have their own clocks. Maximum possible clock frequency of ARM Cortex-A9 is 866MHz.

- **PL Clocks:**

  FCLKs (Fabric Clocks): These clocks are generated by the Clock Management Unit (CMU) in the PS and are typically used to clock logic within the programmable fabric (PL). FCLK_CLK0 is an example, but there can be multiple FCLKs available. The clock frequency of FCLK_CLK0 is 100MHz. This is the configured PL clock frequency in our design, hence all the custom logic elements within the PL section will receive this clock frequency.

5.4 Debug Concept

The Test Access Port (TAP) controller is a fundamental component of the Joint Test Action Group (JTAG) interface, which is used for testing and debugging digital electronic devices, such as integrated circuits (ICs), printed circuit boards (PCBs), and other hardware systems. The TAP controller manages the communication and control between a JTAG-compliant device and an external test or debugging tool.

The designed system can be further tested in detail with the help of a TAP controller based on the IEEE 1149.1 standard. The implementation of the TAP controller is explained in a further section in great detail.

5.5 RISC-V Instruction Set Architecture

The custom register peripheral needs to be compatible with the RISC-V ISA. Based on the ideas of reduced instruction set computing (RISC), RISC-V is an open standard instruction set architecture (ISA). Since RISC-V is an open standard, anyone can implement their own RISC-V processors without having to pay license costs because the architecture's specifications are made available to the public. Because RISC-V instructions have a set length (usually 32 bits), pipelining and decoding in processor designs are made easier. Because RISC-V has a load-store architecture, load and store instructions are the only ways to access memory and all data processing instructions function on registers.

32-bit RISC-V Instruction Formats

Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper Immediate	imm[31:12]																				rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						
Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

- **opcode (7 bit):** partially specifies which of the 6 types of *instruction formats*
- **funct7 + funct3 (10 bit):** combined with **opcode**, these two fields describe what operation to perform
- **rs1 (5 bit):** specifies register containing first operand
- **rs2 (5 bit):** specifies second register operand
- **rd (5 bit):**: Destination register specifies register which will receive result of computation

Figure 5.2: RISC-V instruction formats

A variety of instruction forms are available for RISC-V, including I-type (immediate value operations), S-type (store instructions), B-type (branch instructions), J-type (jump instructions), R-type (register-register operations), the figure 7 depicts the same. Multiple integer and floating-point register sets are provided by RISC-V and can be utilized for a variety of applications. This adaptability helps in managing various application domains and data kinds.

6. Description of the Design Elements

6.1 Register file design

In a RISC-V processor architecture, the register file typically consists of a set of general-purpose registers (GPRs) used for temporary storage of data during program execution.

6.1.1 RISC-V compatibility

For the register file to be RISC-V compatible the typical IO ports of the register file would be A1, A2, A3, WD3, RD1, RD2, RST, REGWR and CLK as shown in the below figure 6.1. Here A1, A2 and A3 are 5-bit register addresses. A1 and A2 are the addresses of the read registers and A3 is the address of the write register. RD1, RD2 and WD3 are 32-bit registers. REGWR is 1-bit enable pins which enables write operation. CLK is the clock signal as the register file is a synchronous block. RST is a reset 1-bit reset signal.

6.1.2 Write Operation

The data that needs to be written into the register file is written into the 32-bit WD3 register and the location of the write operation is determined by the 5-bit A3 register. Upon every rising edge of the clock cycle when the REGWR is high the data present in the WD3 register is written into the corresponding register determined by the address present in A3 of the register file.

6.1.3 Read Operation

RD1 and RD2 registers hold the data that is read from the registers of the register file based on the addresses present in A1 and A2 respectively. Read operation is an asynchronous operation.

6.1.4 Scan Flip Flop

To make the register file scan testable every register of the register file is designed with scan flip-flops. The architecture of the scan chain-enabled register file will be further described in section 7.1. The RISC-V instruction set architecture uses a CPU register file with 32x 32-bit registers named R0 through R31. Instructions encode register addresses as 5-bit fields within the instruction binary number using names such as Rs1, Rs2 and Rd. These address bitfields are connected to three address inputs of the register file. Address A1 selects the register to output as RD1 and become input A of the ALU. Address A2 selects the register to output as RD2 and potentially becomes input B of the ALU. Address A3 selects which register should store the data calculated by the circuit if the controller commands a store using the REGWR control signal. Figure 6.1 shows the register file which can be interfaced with the Arithmetic Logic Unit.

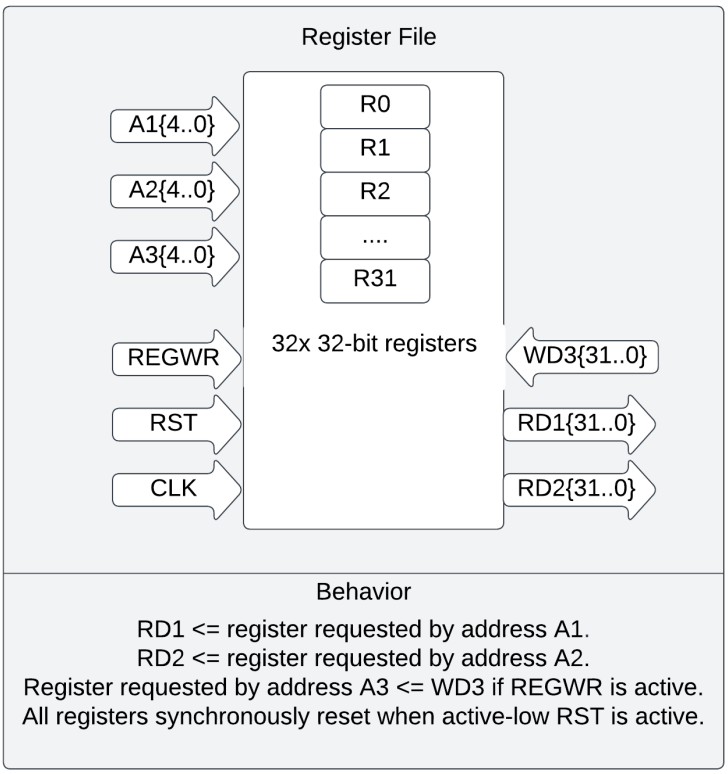

Figure 6.1: Register File

6.2 TAP Controller

The Test Access Port (TAP) controller, as defined by the IEEE Std. 1149.1 standard, plays a crucial role in enhancing the efficiency of testing procedures for integrated circuits. The IEEE Std. 1149.1, commonly known as Boundary-Scan Testing, establishes a standardized approach for testing and debugging digital circuits. This standard introduces the TAP controller, a fundamental component in the implementation of the Boundary-Scan architecture.

TAP Controller Overview: The TAP controller serves as the primary interface between the external testing equipment and the internal components of an integrated circuit. It facilitates communication and control during the testing process, enabling the execution of various test patterns and configurations.

Key features of the TAP controller include:

Test Access Port (TAP): The TAP comprises a series of standardized instructions and registers that allow external devices, such as Automatic Test Equipment (ATE), to communicate with and control the internal components of a digital circuit.

Shift Register Mechanism: The TAP controller utilizes a shift register mechanism to capture, shift, and update data within the integrated circuit. This mechanism enables the testing of individual components and the observation of their responses.

Instruction Register: The TAP controller incorporates an instruction register that holds a set of instructions for testing specific components or performing specific operations within the circuit. These instructions are standardized by the IEEE Std. 1149.1.

6.2.1 IEEE Std. 1149.1 Scan Testing:

Key components of IEEE Std. 1149.1 Scan Testing include:

Scan Register : This register surrounds each digital component and provides a means to observe and control the signals at the component's boundary. This allows for the serial testing of interconnected components.

Scan Cells: These cells, integrated into the design of digital components, enable the capture and observation of data at the component's input and output pins. These cells facilitate the testing of connections and components on the PCB.

Standardized Instructions: IEEE Std. 1149.1 defines a set of standardized instructions that the TAP controller uses to perform various operations, such as entering and exiting test modes, shifting data in and out of the Scan register, and executing test patterns.

TAP Controller Architecture:
The boundary-scan circuitry can be divided into four main hardware components:

- A test access port (TAP), which consists of four mandatory terminals test data input (TDI), test data output (TDO), test mode select (TMS), and test clock (TCK) and one optional terminal, test reset (TRST).
- A TAP controller (TAPC).
- An instruction register (IR) and its associated decoder.
- Several test data registers, including the mandatory boundary-scan register and bypass register, and some optional miscellaneous registers, such as the device-ID register, and some design-specific test data registers.

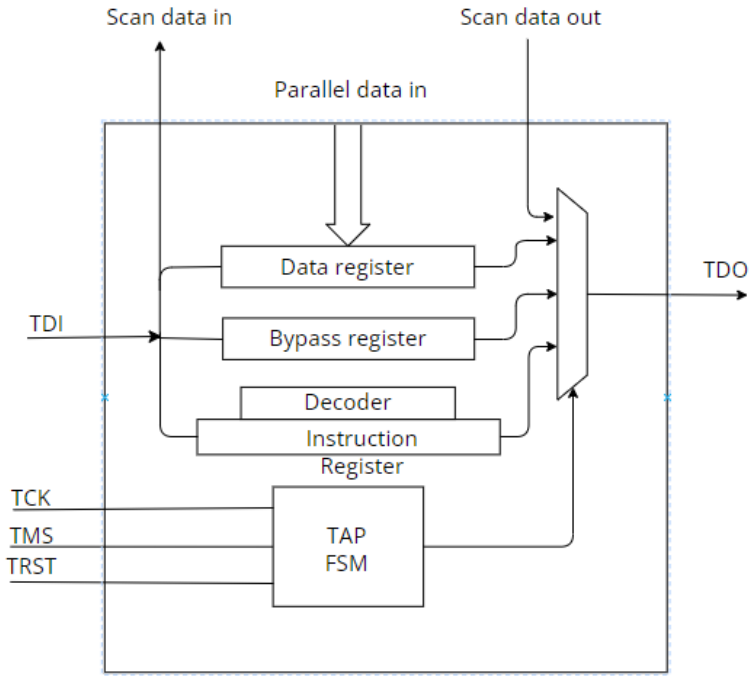

Figure 6.2: TAP Controller Architecture

TAP Controller architecture in IEEE Std. 1149.1 requires serial loading of instructions into the instruction register via the TDI pin. The associated decoder generates test signals for configuring hardware. Test data registers store data or system-related info. IEEE Std. 1149.1 defines instructions like BYPASS, SAMPLE, PRELOAD. The typical test procedure involves shifting a test instruction into the IR, decoding it to configure logic, applying a test pattern, capturing the response in a data register, observing via TDO, and repeating for all test patterns. As per

the implementation in project the TapController_e entity defines the input and output ports of the TAP controller. These include inputs such as TDK (Test Clock), TMS (Test Mode Select), TDI (Test Data In), TRST (Test Reset), and parallel_data_in (Parallel Data In). Outputs include TDO (Test Data Out), scan_in_data (Scan In Data), and scan_out_data (Scan Out Data). The first process handles the state transitions of the TAP controller based on the rising edge of the test clock (TCK) and the test mode select (TMS) signals. The second process is responsible for the actual implementation of the TAP controller's functionality based on its current state.

6.2.2 TAP FSM:

The TAP controller operates based on a Finite State Machine (FSM) that defines different states for the testing process. These states include Test_Logic_Reset, Run_Test_Idle, Select_DR_Scan, Capture_DR, Shift_DR, Exit1_DR, Pause_DR, Exit2_DR, Update_DR, Select_IR_Scan, Capture_IR, Shift_IR, Exit1_IR, Pause_IR, Exit2_IR, and Update_IR. Transitions between states are based on the values of the TMS signal.

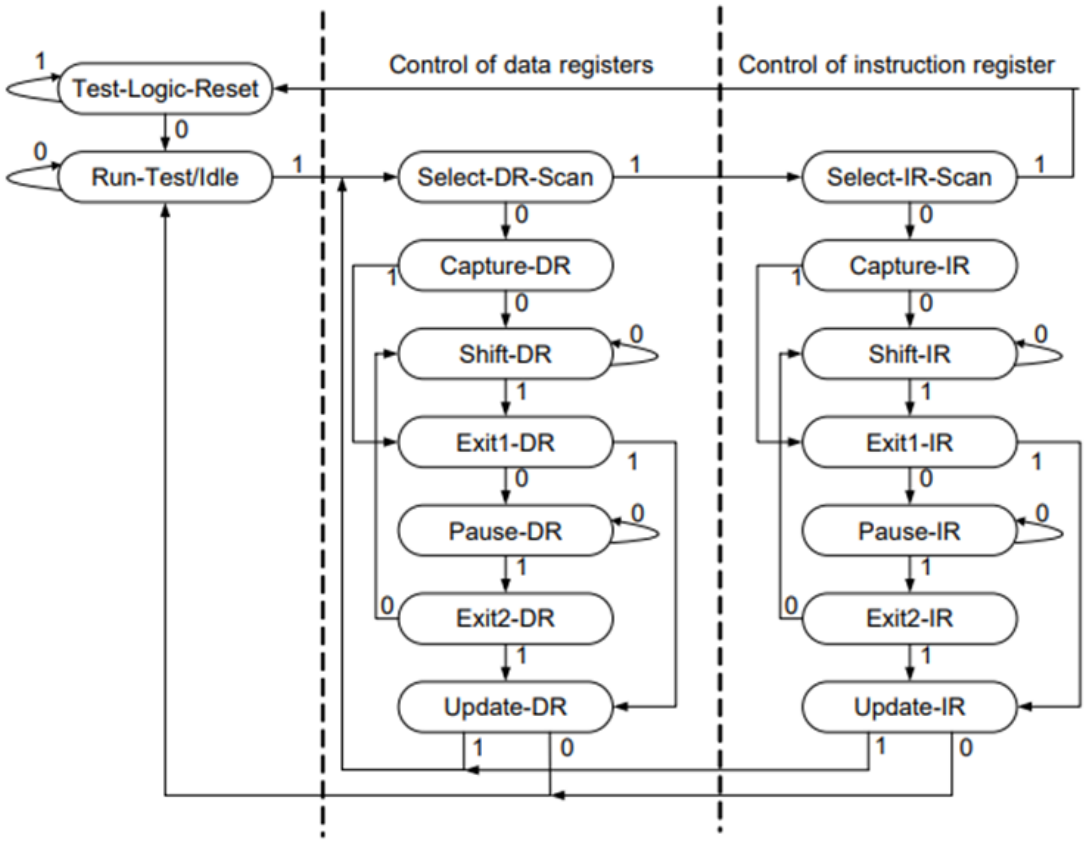

Figure 6.3: Finite State Machine

The FSM guides the TAP controller through a sequence of steps, ensuring the proper execution of test instructions and the capture of test responses. It plays a crucial role in coordinating the testing process, from loading instructions into registers to shifting data and updating internal states. Together, the TAP controller and FSM form a standardized approach to boundary-scan testing, enhancing the efficiency and reliability of digital circuit testing procedures.

6.2.3 Data Register:

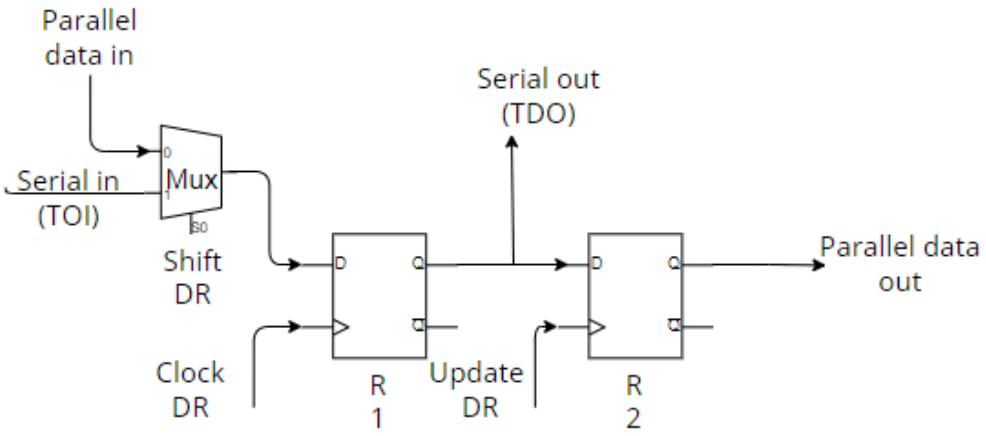

Figure 6.4: Data Register

In normal mode (Mode = 0), data flows directly from IN to OUT, and the cell is transparent to functional logic. In Test mode (Mode = 1), test data from the dr2 flip-flop passes through a multiplexer to the OUT signal. TAP controller signals (ClockDR, ShiftDR, UpdateDR) control three main test operations: Capture, Shift, and Update. During Capture, data at IN is captured into D-FF dr1. In Shift, test data is shifted in from SI, and the test response is scanned out through SO. Update propagates dr1 data to dr2. If Mode is 1, dr2 output connects to OUT. Capture and Shift operations can also occur in normal mode. Test data can be latched in dr2 (and OUT if Mode = 1) while other data is shifted in/out.

6.2.4 Instruction Registration:

The Instruction Register (IR) stores the instruction to be executed in a two-stage(ir1 and ir2) design same as data register, preventing indeterminate states during instruction shifting. Four mandatory boundary-scan test instructions (SAMPLE, PRELOAD, BYPASS, EXTEST) are defined in IEEE Std. 1149.1.

6.2.5 Parallel and Serial Data Handling:

The code includes the handling of parallel data (parallel_data_in, parallel_data_out) and serial data (TDI, TDO) based on the current state of the TAP controller. Shift and capture operations are performed on the dr1 and ir1 signals.

[Note : As per the implementation, Parallel input is taken from registers and Parallel output is not being used, it kept as open to use in future.]

6.2.6 Update Operations:

Update operations are handled for both the instruction register (ir2) and the data register (dr2). The TAP controller shifts the updated data into the respective registers.

6.2.7 TAP Controller Tests:

Scan Chain: The code includes handling for the Scan Chain operation, allowing data to be shifted in and out through the scan_in_data and scan_out_data signals.

Bypass Handling: Bypass functionality is also implemented, allowing the TAP controller to bypass the internal circuitry and directly pass the data through.

Sample: During the rising edge of the test clock (TCK), if the test reset is not active, the parallel data from parallel_data_in is loaded into dr1. This captures the input data at the parallel input into the data register for subsequent testing operations.

Preload: The "Preload" operation shifts and updates data, sending the MSB to TDO and preparing the register for the next bit with TDI. Test Reset (TRST) Handling: The code includes logic to reset the internal registers (dr1, dr2, ir1, ir2) when the test reset signal (TRST) is asserted.

6.3 Target SoC Architecture

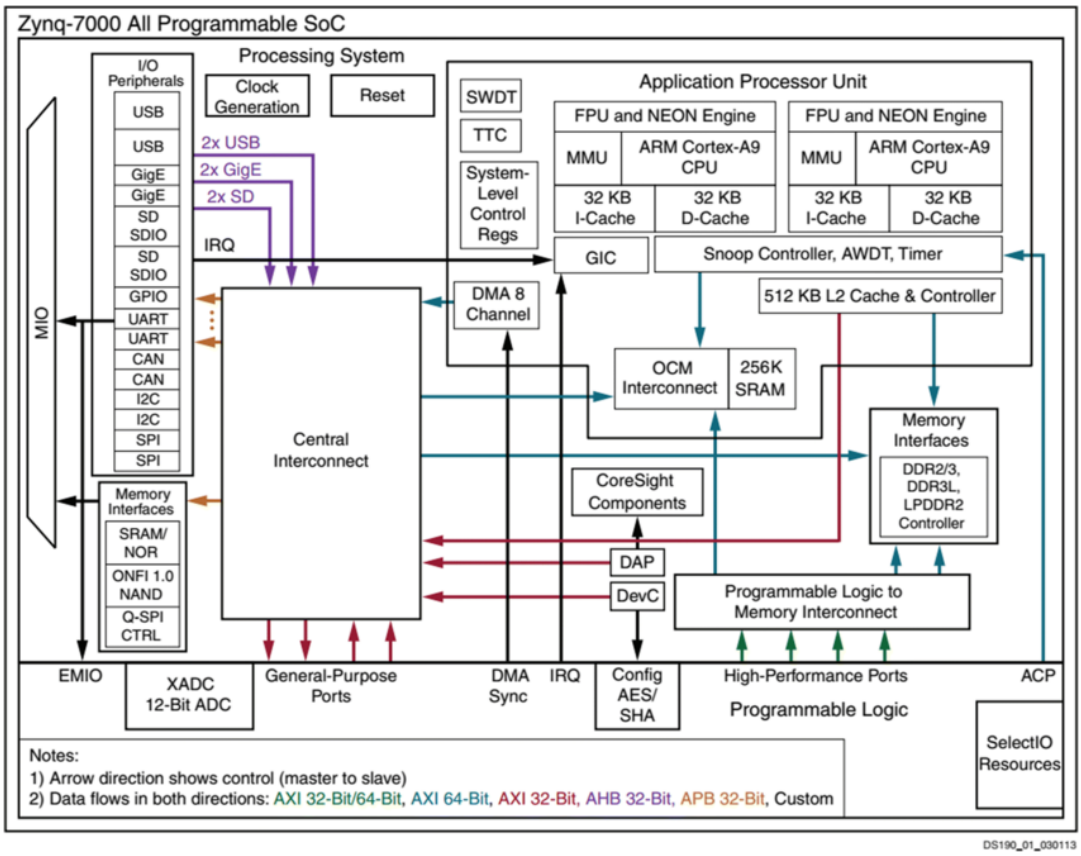

Figure 6.5: Architecture overview

Xilinx Zynq-7000 SoC allows a flexible platform to launch solutions while providing traditional SoC users with a fully programmable alternative. The Arm Cortex-A9 processor present provides an integrated processing system (PS). The XA Zynq-7000 architecture enables the implementation of custom logic in the PL and custom software in the PS.

The PS comprises four major blocks Application processor unit (Two dual-core ARM Cortex-A9 processors), Interconnect, I/O peripherals and Memory interfaces. The APU, memory interface unit and I/O peripherals are all connected to each other and to the PL through an ARM AMBA AXI interconnect. The interconnect is non-blocking and supports multiple simultaneous master-slave transactions.

In the scope of our SoC, the custom synthesised IP's (Register file, TAPC) will be deployed into the PL section of Xilinx Zynq-7000 SoC and these IP's will be interfaced with the PS (ARM Cortex APU) via an AXI interconnect.

6.4 ARM Core

The Cortex-A9 is a multi-core microprocessor that uses a dual-issue, in-order pipeline design. It features various architectural enhancements over its predecessor, the Cortex-A8, resulting in improved performance and energy efficiency. The Cortex-A9 uses the ARMv7-A instruction set architecture, which supports both 32-bit and 64-bit instruction sets. This architecture is widely used in modern ARM-based processors.

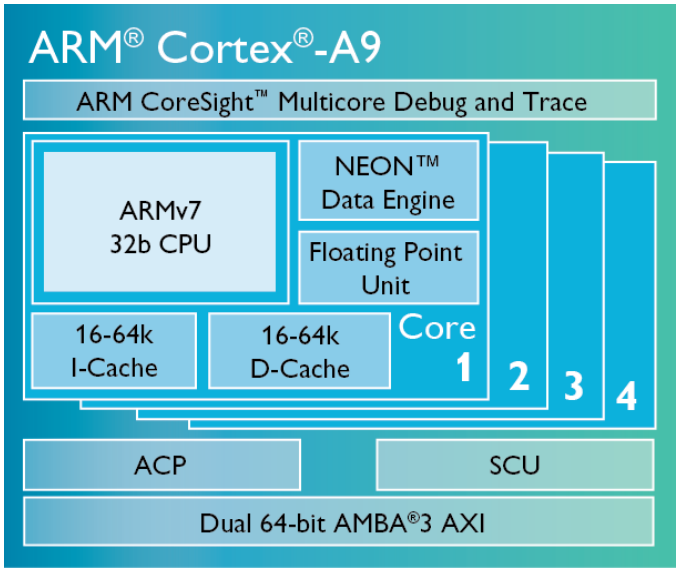

Figure 6.6: ARM Cortex-A9

The Cortex-A9 can be configured as a single-core or multi-core processor. It is often used in multi-core configurations, which can provide increased processing power for more demanding applications. The Cortex-A9 is capable of delivering strong performance for a wide range of applications, including smartphones, tablets, networking equipment, automotive infotainment systems, and more. Its performance can be further enhanced in multi-core configurations. The Cortex-A9 is available in a form that can be implemented in Field-Programmable Gate Arrays (FPGAs). This allows for hardware acceleration and customization in embedded systems and specialized applications.

7. Test and Debug

7.1 Scan Chain

In VLSI design, a scan chain is a technique used for testing and debugging digital circuits. It is employed to facilitate the efficient testing of integrated circuits by enabling the observation and control of internal signals.

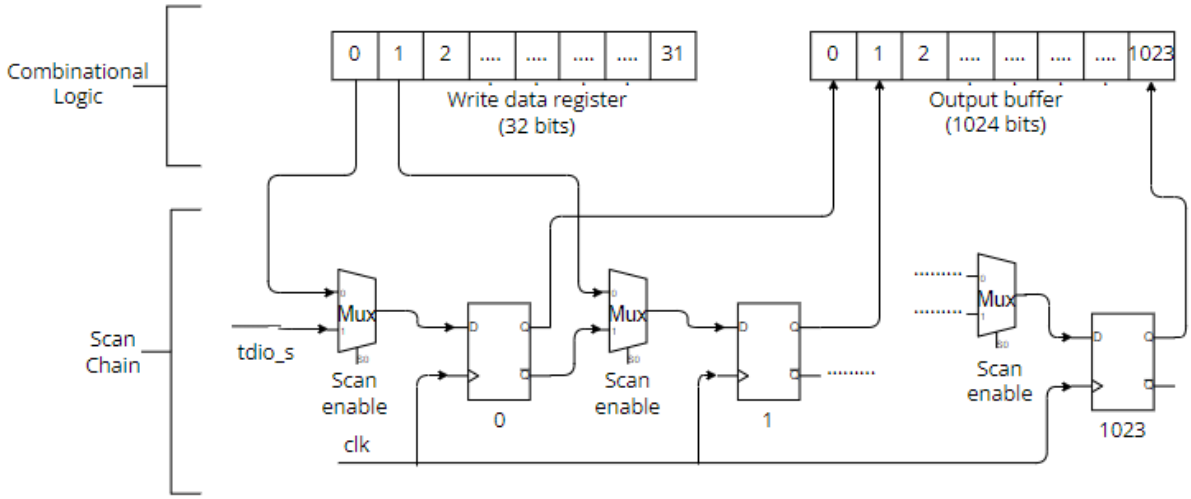

Figure 7.1: Scan Chain Implementation

The basic idea behind a scan chain is to create a serial path that connects all the scan flip-flops in the design. This serial path forms a shift register that allows you to shift in test vectors and observe the corresponding outputs. This capability is particularly useful for performing scan-based testing, which involves serially scanning in test patterns and capturing the responses.

This scan chain is implemented in the register file to store the data sent from other peripherals into the register file. Since the register file is designed to have 32x 32bit- registers a scan chain of 1024 scan flip-flops has been designed in the register file as mentioned in section 5.1.4.

7.2 Design

In the context of a scan chain, each flip-flop is specifically designed with supplementary input and output features to streamline the scan testing process. These additional elements typically consist of a serial scan input (SI) and a scan output (SO).

The serial scan input, sourced from the TDI port of the TAP controller, facilitates the introduction of test data into the flip-flop, where it is subsequently shifted into the following flip-flops. Another input for the scan flip-flop is the parallel data input, which originates from the internal register (WD3) of the register file.

The scan flip-flop is capable of storing data from either of these input sources at any given moment. To achieve this capability, a multiplexer (mux) is integrated into the input port of the scan flip-flop. The control of this mux is governed by a scan enable signal, determining which data is to be stored in the scan flip-flop.

Illustrated in the accompanying figure 7.1, each scan flip-flop features two output ports, both yielding identical output data – the information stored in the flip-flop. One output port is linked to the output buffer of the register file, while the other connects to the serial data input port of the subsequent flip-flop. This configuration establishes a scan chain, enabling diverse tests to be conducted on the designed hardware.

7.3 Operation

The scan has two operating modes as mentioned below:

- **Normal mode:** In regular operation, the scan chain remains inactive, and the flip-flops function conventionally, operating as part of a standard sequential circuit. The designated scan flip-flops serve as registers within the register file, capturing parallel data from the 32-bit WD3 internal register of the register file. WD3 holds data intended for writing into the register file. The design of the scan flip-flops ensures that the data from the WD3 register is stored sequentially in the corresponding 32 scan chain flip-flops based on the 5-bit address stored in the A3 internal register of the register file.

  Subsequently, the data stored in the scan flip-flops is written into a 1024-bit wide output buffer during the falling edge of the clock cycle. In read operations of the register file, the data from the output buffer is transferred to the RD1 and RD2 internal 32-bit registers of the register file, contingent on the addresses present in the A1 and A2 ports. All these operation happen when the scan enable signal is low and allows parallel data to be stored into the scan flipflops.

- **Scan mode:** This mode activates when the scan enable signal is set high, permitting the storage of data from the tdio_s signal into the initial scan flip-flop. In this state, data is sequentially shifted into the scan flip-flops via the TDI port of the TAP controller.

  Upon each rising clock edge, data undergoes a rightward shift from one scan flip-flop to the next. Conversely, during the falling edge of each clock cycle, the data existing in the scan flip-flops becomes accessible at the scan input port of the succeeding flip-flop. Consequently, with each clock cycle, data undergoes a right shift, and the information present in the 1024th flip-flop is shifted out into the TDO signal of the TAP controller.

The use of scan chains makes it easier to apply test patterns and observe responses during the testing phase of semiconductor manufacturing. This helps identify faults and defects in the integrated circuit, contributing to improved quality and reliability of the final product. The scan chain is also valuable for debugging and analyzing the behavior of a circuit during development.

8. Memory Maps and Register Lists

The custom IP is instantiated in the AXI slave interface and then the ports of the register file are binded with the internal registers of the AXI slave interface as described in section 8.1.1. This allows further memory mapping of the custom IP with ARM core. As a result, all the ports of the register file can be accessed in the ARM core applications with the help of the address.

The base address of the custom IP is 0x43C0_0000 and the high address is 0x43C0_007F. The addresses of all the register file ports are as follows.

Port Address (base address + offset)	**Port name**
0x43C0_0000	Reset
0x43C0_0004	Register write enable
0x43C0_0008	Register read address 1
0x43C0_0012	Register read address 2
0x43C0_0016	Register write address
0x43C0_0020	Write data register
0x43C0_0024	Scan enable
0x43C0_0024	Read data register 1
0x43C0_0032	Read data register 2

Table 8.1: Port Addresses

9. System simulation and applications

9.1 Block diagram

The below figure 9.1, depicts the custom SoC which has been designed including our custom IP with register file and TAP controller. The zynq7 processing system is interfaced with the IP using an AXI interconnect block. This is possible because the custom IP is an AXI slave.

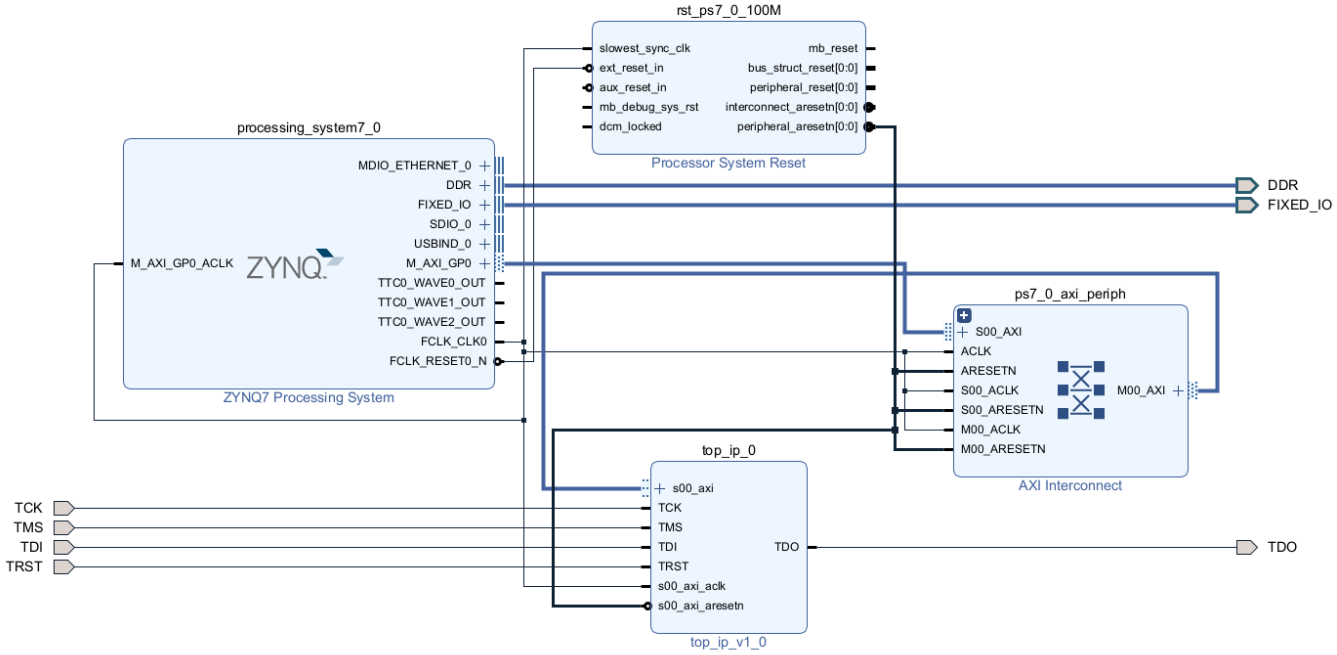

Figure 9.1: SoC Block diagram

AXI interconnect block plays a crucial role in facilitating communication and data transfer between different AXI masters and slaves. In this case, the AXI master is a processor that initiates data transfer with the custom IP which is an AXI slave. The interconnect block performs address decoding to route transactions from masters to the appropriate slaves based on the address information in the AXI transactions. The AXI interconnect may include an arbitration mechanism to resolve contention for the bus when multiple masters are trying to access the same slave simultaneously. In this custom SoC the AXI interconnect block is configured to have one master and one slave.

AXI interconnect signal description:

S00_AXI	INPUT	Master Interfacing with Slave
M00_AXI	OUTPUT	Slave Interfacing with Master
ACLK	INPUT	Clock to AXI Interconnect
ARESETN	INPUT	Reset Signal to AXI Interconnect
S00_ACLK	INPUT	Clock to AXI Slave
S00_ARESETN	INPUT	Reset Signal to AXI Slave
M00_ACLK	INPUT	Clock to AXI Master
M00_ARESETN	INPUT	Reset Signal to Master

Table 9.1: AXI Interconnect

Custom IP signal description:

TCK	INPUT	Clock signal to control Test Access Port
TMS	INPUT	Signal which decides shiting of Data in and out through data/Instruction register
TDI	INPUT	To serially inputting test data
TDO	OUTPUT	To serially outputting test data
TRST	INPUT	To reset the tets logic
s00_axi_aclk	INPUT	clock input for a specific AXI interface
s00_axi_aresetn	INPUT	reset signal for a specific AXI interface

Table 9.2: Custom IP

9.1.1 Custom IP Hierarchy

The provided diagram illustrates the structure of the customized IP. It is evident from the diagram that the Register File and Scan Chain modules are instantiated within the AXI environment. Consequently, all ports of the register file are associated with AXI registers, establishing a memory-mapped connection with the ARM core. This configuration enables the seamless transfer of data from the ARM core into the register file by utilizing designated memory locations.

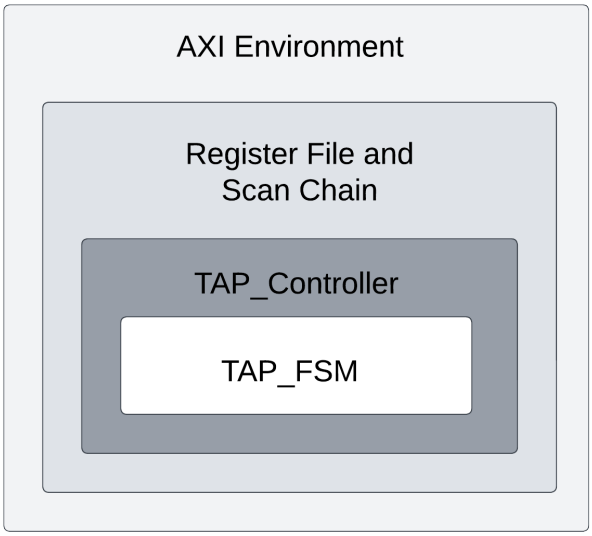

Figure 9.2: Custom IP

Certain ports of the TAP controller, including TDI, TDO, TMS, TCK, and TRST, are not linked to the AXI registers. These ports are essential for receiving data from a JTAG network and are exposed at the top level of the files, serving as IO ports for the custom IP. The data received through these IO ports from the JTAG network is then directed to the TAP controller within the design.

9.2 Custom IP simulation

To validate the functionality of the custom IP and incorporate any required modifications prior to the physical construction of the system, we employed VHDL simulation. VHDL simulation is the practice of utilizing a software tool to simulate and assess the performance of a digital circuit based on VHDL language descriptions. In this process, the simulation tool interprets the VHDL code, particularly the testbench component that outlines the expected behaviour of the circuit, and produces a simulated model of the circuit. Subsequently, we examined the simulation outcomes using waveform viewers available in the Vivado platform.

9.2.1 Test case 1

Test the functionality of the register file.

- Write Operation:

  Input: write x"ffffffff", x"0f0f0f0f" and x"0000ffff" data into the registers R31, R30 and R29 respectively when regwr signal is LOW.

  Expected output: Storing the above-mentioned data into the registers R31, R30 and R29 of the register file.

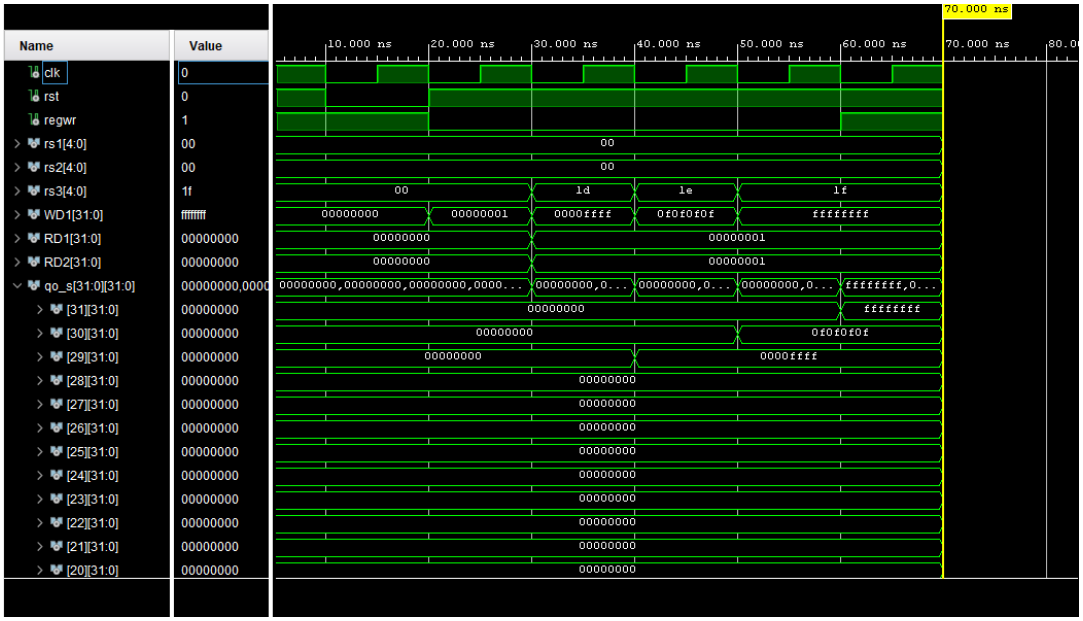

Figure 9.3: Write Operation

Observations: From the above simulation output figure 9.2, it can be observed that the rs3 signal holds the address of the registers R29, R30 and R31 and it can be seen that during the falling edge of every clock cycle data present in the WD3 register is being written into respective registers of register file qo_s.

- Read Operation:

  Input: Read the data present in the registers R31 and R30 of the register file into the internal registers WD1 and WD2.

  Expected output: To fetch the data present in R31 and R30 registers of the register file and then store it into the WD1 and WD2 registers respectively.

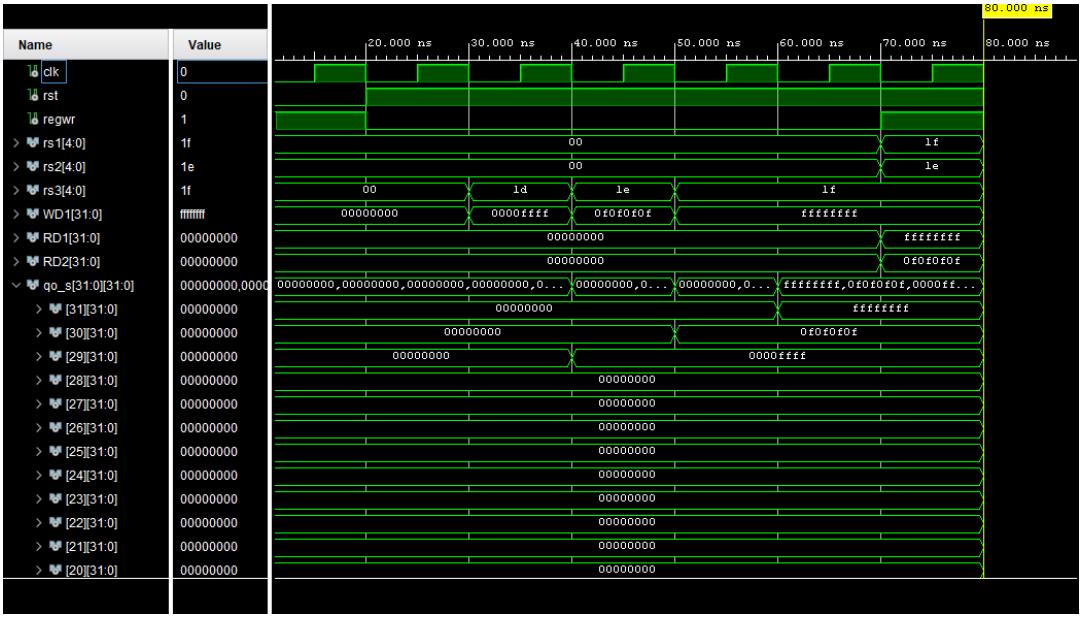

Figure 9.4: Read Operation

9.2.2 Test case 2

Test the functionality of the scan chain.

Scan chain:

From the below simulation figure 9.4, it can be observed that at the simulation period 240ns the current state of the TAP controller is Shift_DR and scan functionality is enabled as scan_enable signal is HIGH. The Instruction register ir2_s has a value of 2, this implies that the TAP controller is currently in scan chain mode. Also, it can be observed that the scan_in_signal is also made high for one clock cycle. The states of the above-mentioned signals enable the TAP controller and register file to be in scan mode.

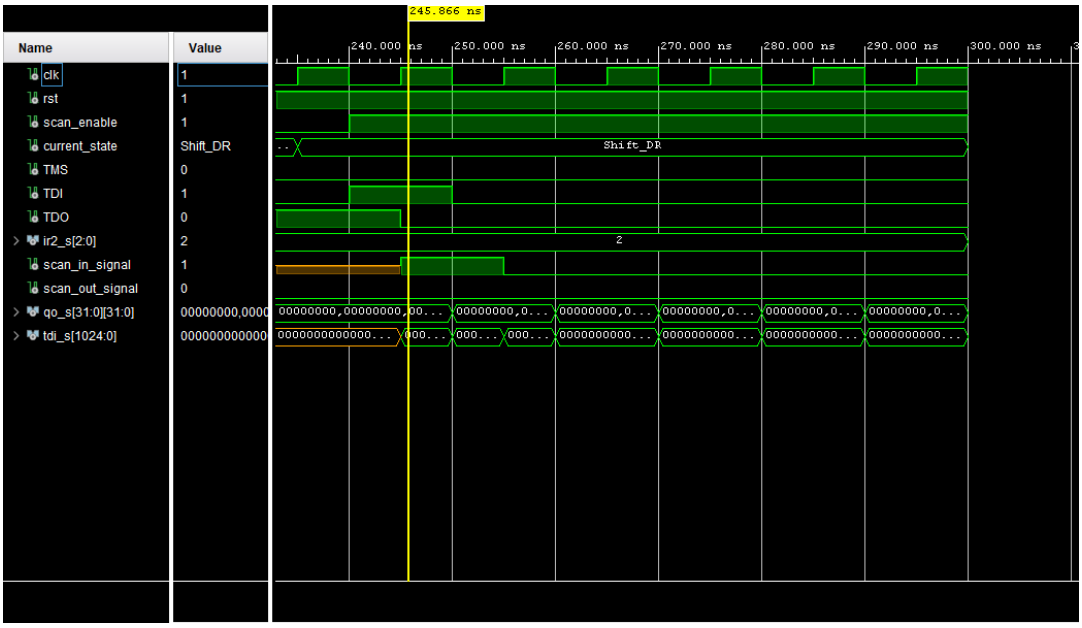

Figure 9.5: Scan chain 1

Scan_in_data signal will now be shifted from 1st scan flipflop to the 1024th scan flipflop after 1023 clock cycles and will eventually reflect in the scan_out_data signal and will pass through TDO port of the TAP controller. This can be observered in the below figure 8.5.

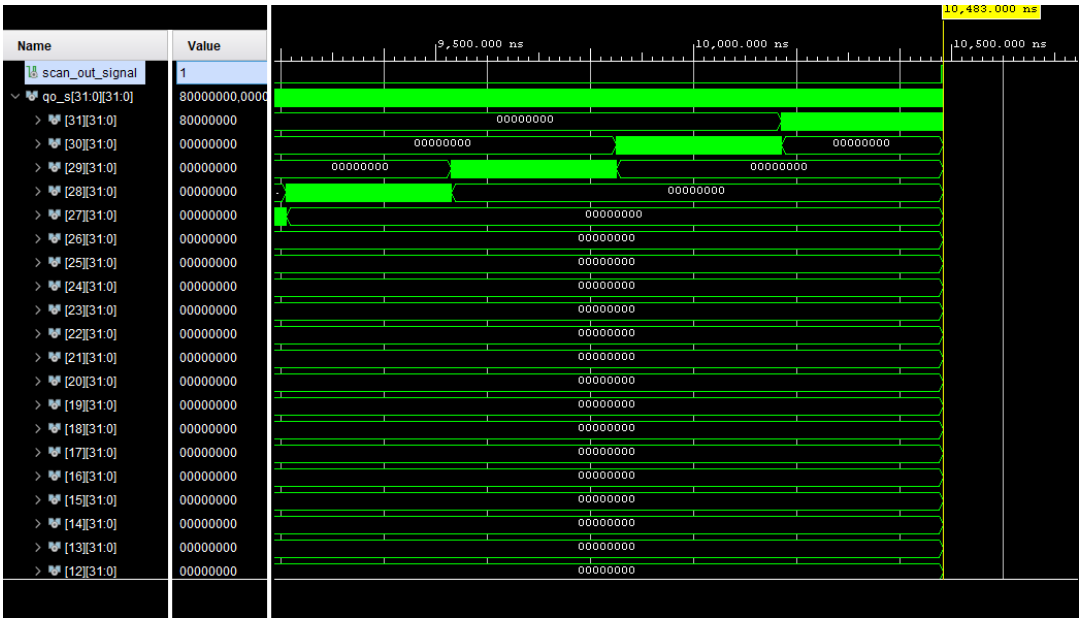

Figure 9.6: Scan chain 2

9.2.3 Test case 3

Test the functionality of the Bypass.

Bypass:

In the Bypass scenario, test data is directly passed to the output as data from TDI(Input) passed to TDO(Output) on rising edge in ShiftDR state.

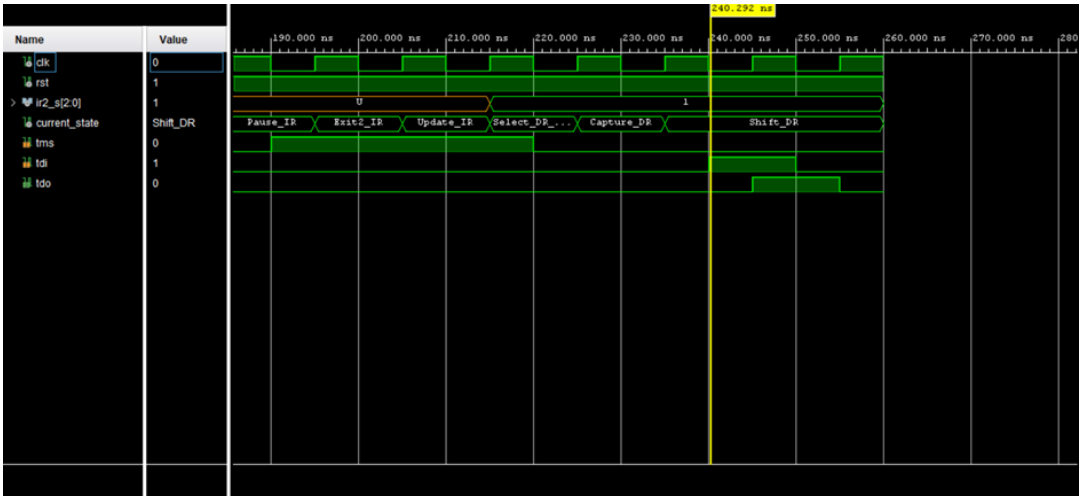

Figure 9.7: Bypass

9.3 Synthesis

The synthesis process entails examining the design, optimizing logic, and translating a high-level hardware description, written in a hardware description language like VHDL, into a lower-level representation, typically in the form of a gate-level netlist. This gate-level netlist, generated as output, serves as input for the subsequent place-and-route implementation of the design on either an FPGA or ASIC. Upon completion of the block design,

a HDL Wrapper is generated in the Vivado platform, automating the VHDL code generation for the entire block design. In the final step of the Vivado design process, a Bitstream, along with a constraint file, can be produced and downloaded into the target board.

9.4 Application development

After the hardware synthesis process has been completed, the focus shifts to developing the software application that will run on the synthesized hardware. Vitis platform is a collection of hardware and software components. After synthesis, we need to create a platform that includes the synthesized hardware design along with the necessary software components. HDL wrapper XSA file is imported into the Vitis IDE for further BSP (Board support packages) generation for the custom hardware.

For our custom IP which includes a register file, we have written some basic drivers which will enable application development for the custom hardware.

Driver	Description
initRegFile	Initialize base address depending on the memory mapping of custom IP
writeRegFile	Writes data into the desired register of the register file
readRegFileR1	Reads data from the RD1 of the register file
readRegFileR2	Reads data from the RD2 of the register file
resetRegFile	Reset signal for the register file
writeDisable	To disable the write signal in the register file
scanEnable	To enable scan chain functionality of the register file

Table 9.3: Basic Drivers

10. Bibliography

Number	Document Type	Author	Version	Release Date
1	VLSI Test Principles and architectures	Laung-Terng-Wang	Revised	February 27, 2017
2	IEEE 1149.1 (JTAG) Boundary-Scan Testing Testing for MAX II Devices	ALTERA	1.7	October 2008
3	Zybo FPGA Reference Manual	DIGILENT	Revised	2007
4	Computer Organisation and Design	David A. Patterson	Revised	2018

Table 10.1: List of documents