IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Functional Verification Of Spi Master-Slave Communication In An Open Power Soc Design

[1] Kothamiddi Harini, [2]Dr. G. Mamatha

[1]PG Scholar, [2]Associate Professor

Department of Electronics and Communication Engineering JNTUA College of Engineering Anantapur, Ananthapuramu, India.

Abstract: This paper presents functional verification of master-slave communication via the Serial Peripheral Interface (SPI) within a System-on-Chip (SoC) architecture based on the A20 Open POWER processor. The verification process targeted essential SPI functions, including data writing, reading, and handling of error conditions. A custom-designed SPI protocol was incorporated into the SoC, and a verification setup was built using System Verilog along with the Universal Verification Methodology (UVM). Verification utilized industry-standard tools such as Mentor Graphics Questa Sim for simulation, Xilinx Vivado for design synthesis and integration, and associated tools for waveform analysis. The testbench was structured to validate proper data transfer and protocol operation under both standard and error scenarios. The outcomes verify dependable SPI performance, highlighting the verification approach's role in enhancing communication reliability within open-source SoC designs.

Index Terms – Verification, SPI, A2O core, AXI soc, system Verilog.

I. INTRODUCTION

Modern System-on-Chip (SoC) designs rely heavily on efficient communication protocols to manage data transfer between processors and peripherals. Among these, the Serial Peripheral Interface (SPI) stands out for its straightforward implementation, full-duplex capability, and minimal hardware requirements. It is widely used in embedded systems to connect micro protocols or processors to devices like sensors, memory, and other protocols. However, as SoCs grow more complex, verifying the correct behavior of such communication protocols becomes essential to ensure reliable operation. This work focuses on verifying SPI master-slave communication within a custom SoC built around the A20 processor, part of the Open POWER ecosystem. The A20 is a 64- bit dual-core processor based on the PowerPC instruction set, known for its open-source accessibility and suitability for low- power, high-performance applications. Its flexible architecture makes it a strong candidate for customizable SoC development and academic research. To connect the SPI protocol to the A20 core, the design employs the AXI extensible Interface) protocol. AXI is a widely adopted bus interface standard from ARM's AMBA family, offering high data throughput, low latency, and support for burst transactions. These features make AXI well-suited for interconnecting IP cores in complex SoC designs. In this project, a custom SPI protocol was integrated into the SoC over the AXI bus. The functional verification was carried out using System Verilog and UVM, focusing on validating core operations such as write, read, and error conditions. Tools like Mentor Graphics Questa Sim and Xilinx Vivado were used for simulation and synthesis. The goal was to ensure reliable data communication between the master and slave, and to confirm that the design met expected behavior under different test scenarios.

II. SPI PROTOCOL DESIGN

The SPI protocol is a key component for enabling reliable communication between the A20 Open POWER processor and peripherals in the SOC. Designed to handle both master and slave operations, the protocol supports full-duplex data transfer via the four primary SPI signals: MISO, MOSI, SCK, and SS. In master mode, the protocol generates the clock signal (SCK) and initiates data transfers, while in slave mode, it responds to the master's clock. Data is transmitted and received bit by bit through shift registers, synchronized with the SPI clock. Data is transmitted and received bit by bit through shift registers, synchronized with the SPI clock. The protocol is designed to support adjustable clock polarity CPOL) and clock phase (CPHA) settings to accommodate various peripheral devices. Error handling mechanisms are implemented to detect overrun, framing, and chip select mis- matches, ensuring reliable operation. Additionally, flow control is incorporated to manage data transfers and prevent buffer overflows. Integrated with the A20 processor via the AXI protocol, the SPI protocol leverages high-speed, low-latency data transfer capabilities, ensuring seamless communication between the processor and connected peripherals. The design was implemented in Verilog, supporting single-byte and multi- byte transfers with customizable clock speeds for flexibility in peripheral integration.

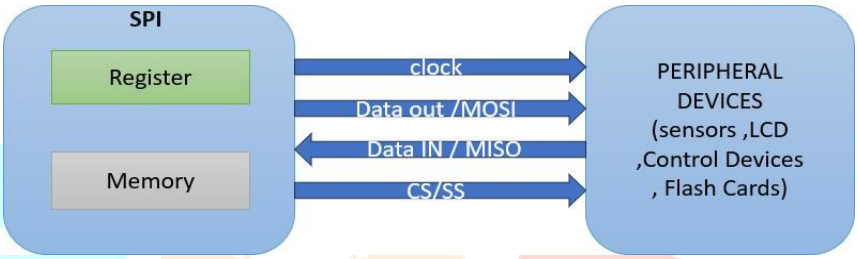
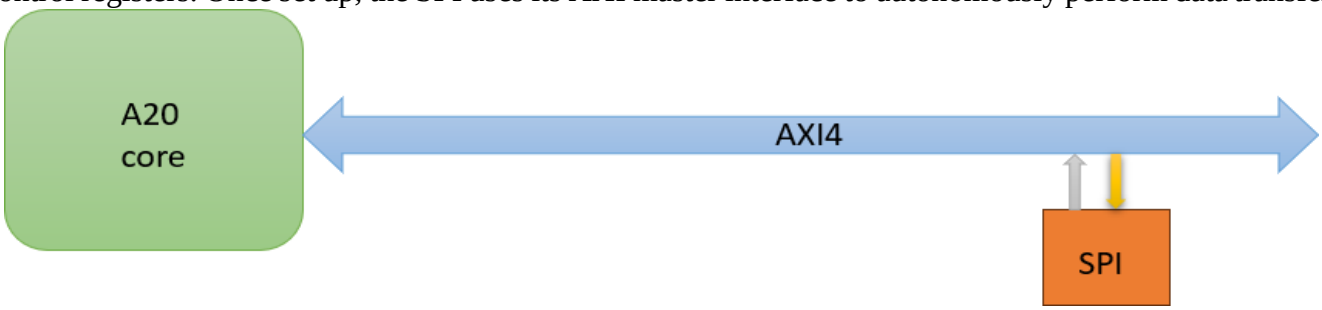


Fig.1. General block diagram of SPI

III. INTEGRATION OF A20 WITH SPI WITH AXI4

The AXI Interconnect handles the routing and arbitration of AXI transactions between multiple masters and slaves. It connects the SPI master interface to one or more AXI slave peripherals or memory modules on the right side of the diagram. Additionally, the interconnect also links to slave ports (likely for control and status operations), which could be part of CPU or configuration logic. This architecture allows the SPI to autonomously read from or write to memory or peripherals, supporting high-speed data transfers within the SoC while reducing CPU load. The dense wiring reflects multiple AXI channels: address, write, read, control, and handshake signals, each critical to proper AXI4 protocol operation. Integrating a SPI protocol with the A2O core enhances the connectivity. In this setup, the SPI's AXI slave interface is connected to the A2O core's AXI master port, allowing the processor to configure the SPI by writing to its control registers. Once set up, the SPI uses its AXI master interface to autonomously perform data transfers



between memory and peripherals via the AXI interconnect. This architecture enables concurrent CPU processing and data movement, improving overall throughput. Proper handshake and clock synchronization are maintained across components for reliable data flow.

Fig.2.Integration of A20 And SPI with Axi4

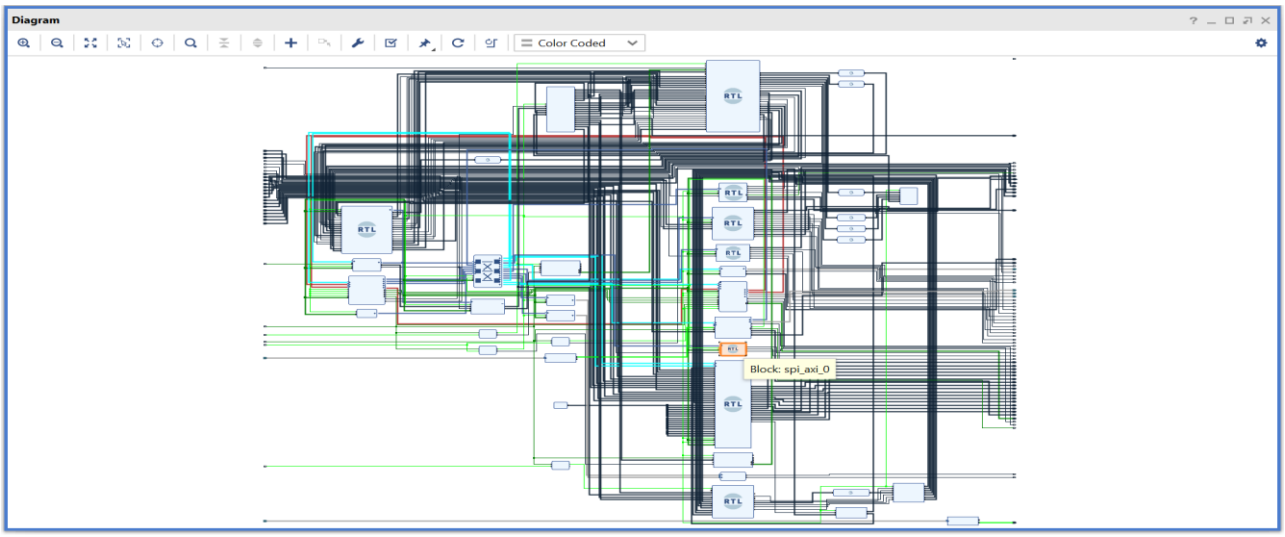


Fig.3.Block diagram of soc design After Integration of A20 and SPI with AXI4 using Xilinx vivado.

Figure 3 shows the SoC-level integration of the SPI protocol with the Open POWER A20 processor through the AXI4 interconnect. The A20 core accesses SPI using memory-mapped I/O, and transactions are routed via the AXI4 bus. The verification environment ensures that the SPI responds correctly to these AXI transactions, using testbenches written in System Verilog. Loopback and read-write testcases verify the end-to-end functionality of the SPI in this integrated setup.

IV. VERIFICATION METHODOLOGY

The verification of the SPI protocol integrated with the A20 Open POWER processor and interfaced via the AXI protocol was carried out using a comprehensive testbench. The primary goal was to validate the SPI communication, focusing on write, read and error-handling scenarios, in both master and slave configurations. A System Verilog- based testbench was developed, following the Universal Verification Methodology (UVM), to model the interaction between the A20 processor and the SPI protocol through the AXI interface. This setup enabled the simulation of high-speed data transfers between SPI master and slave devices, allowing for thorough validation of the SPI protocol, including data integrity, clock synchronization, and chip select signal management. For error handling, a loop back test was employed to simulate various fault conditions. In this test, data sent from the master via MOSI was looped back to the master via MISO, allowing the detection of transmission errors, misalignment, and data corruption. This approach ensured that errors in the SPI communication, such as overrun or framing errors, were properly identified and handled by the protocol. The AXI interface was used for efficient data transfer between the SPI protocol and the A20 processor, ensuring minimal latency and reliable control of read/write operations. The testbench simulated a range of normal and faulty conditions, validating that the SPI protocol correctly managed the data flow and responded to errors as expected. Simulation was conducted using Mentor Graphics Questa Sim for functional verification, while Vivado was utilized for design synthesis and integration. Waveform analysis further validated the transmission, reception, and error detection processes. The results confirmed that the SPI protocol performed correctly under all tested conditions, ensuring reliable data communication and robust error handling through the loopback mechanism.

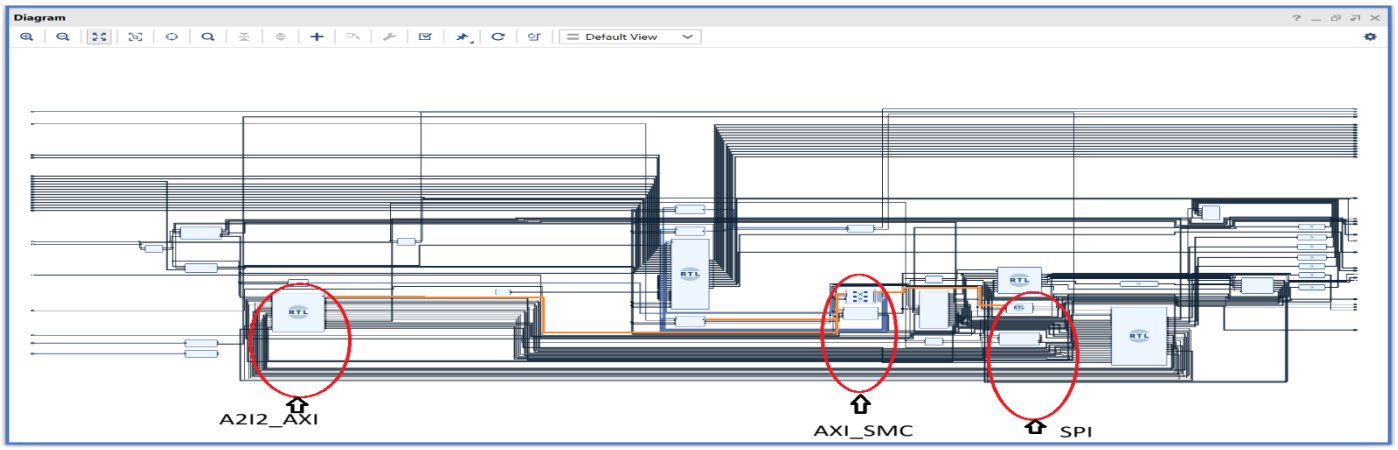
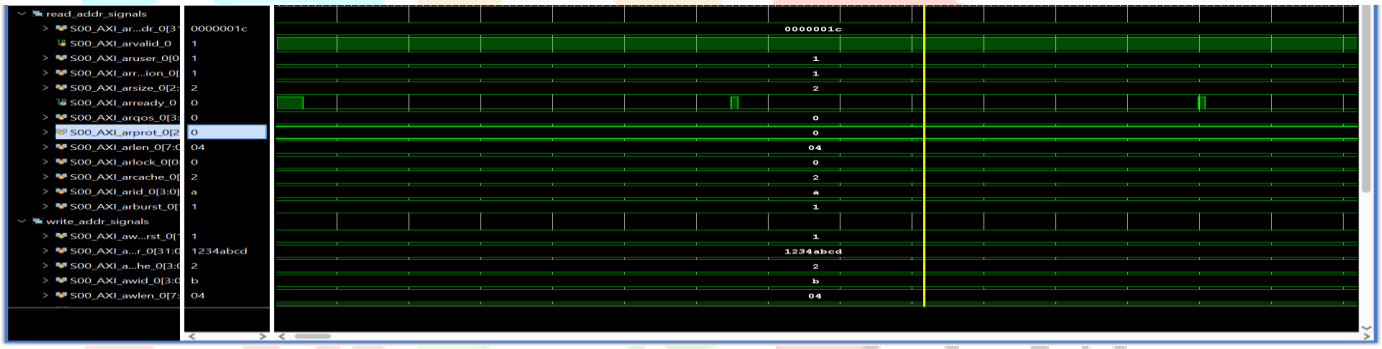


Fig. 4. Verification environment.

V. RESULTS AND DISCUSSION

The verification of the SPI protocol integrated with the A20 Open POWER processor through the AXI interface was carried out using the read, write, and loopback testcases. These testcases were implemented in the System Verilog-based testbench, ensuring that the interaction between the A20 processor and the SPI protocol via the AXI protocol was correctly validated. Write Testcase Results: The write testcase validated the SPI protocol's ability to transmit data from the A20 processor (master) to the SPI slave. In this scenario, the A20 processor initiated Axi4 write transactions, where the AXI_AWADDR signal addressed the memory-mapped location of the SPI protocol's data register. The AXI_WDATA signal carried the data to be



written to the SPI protocol, and the AXI_WVALID signal asserted the validity of the data being transferred. The SPI protocol successfully received the data, and the AXI_BVALID signal confirmed the successful completion of the write transaction, indicating that the data was written correctly.

Fig 5. read and write addresses at A20.

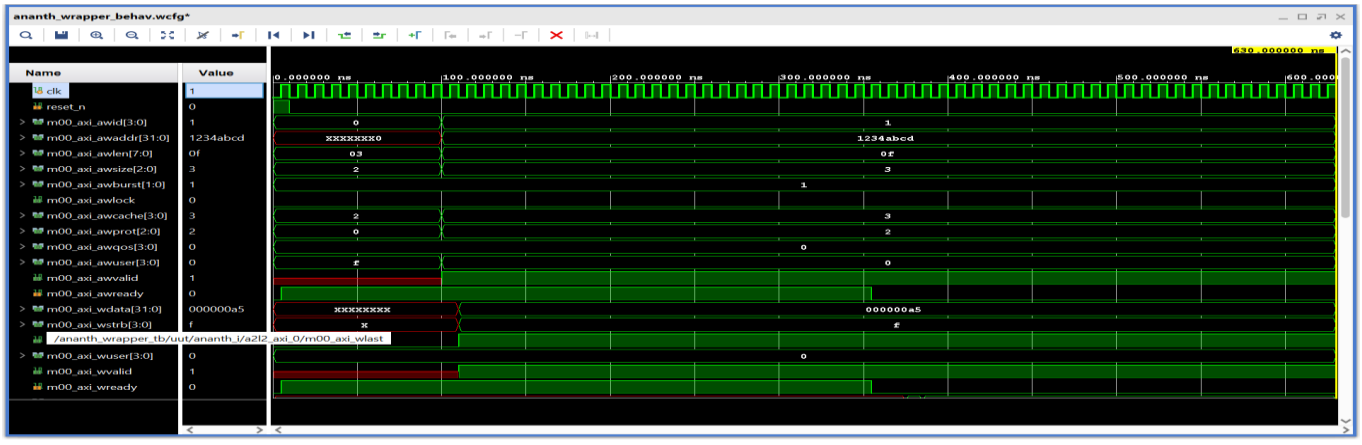


Fig. 6. write data of A20-AXI (master).

The entire write operation occurred without any timing violations or data mismatches between the AXI interface and SPI communication.

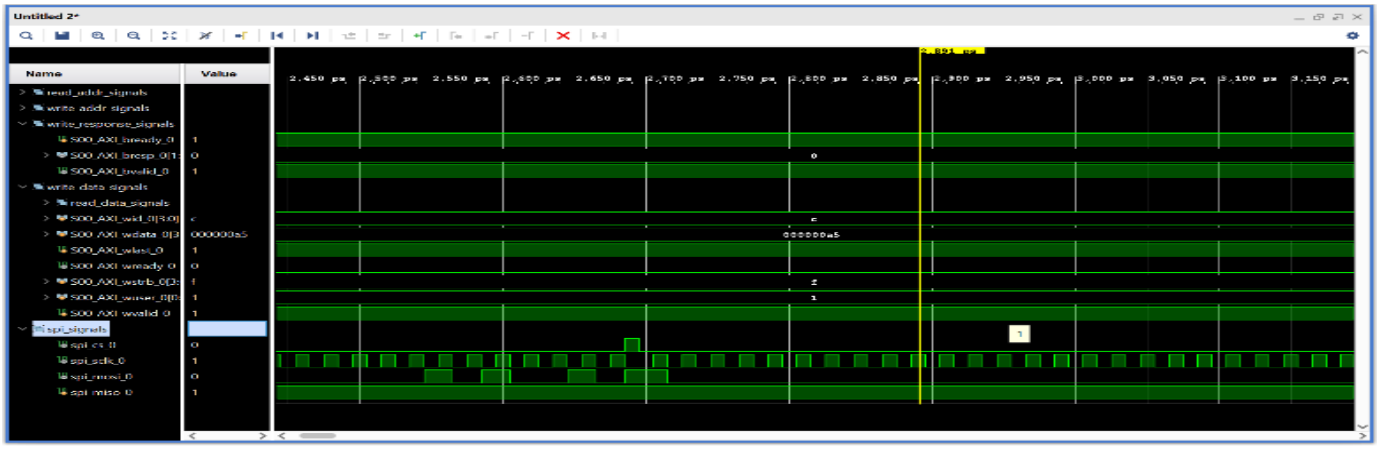
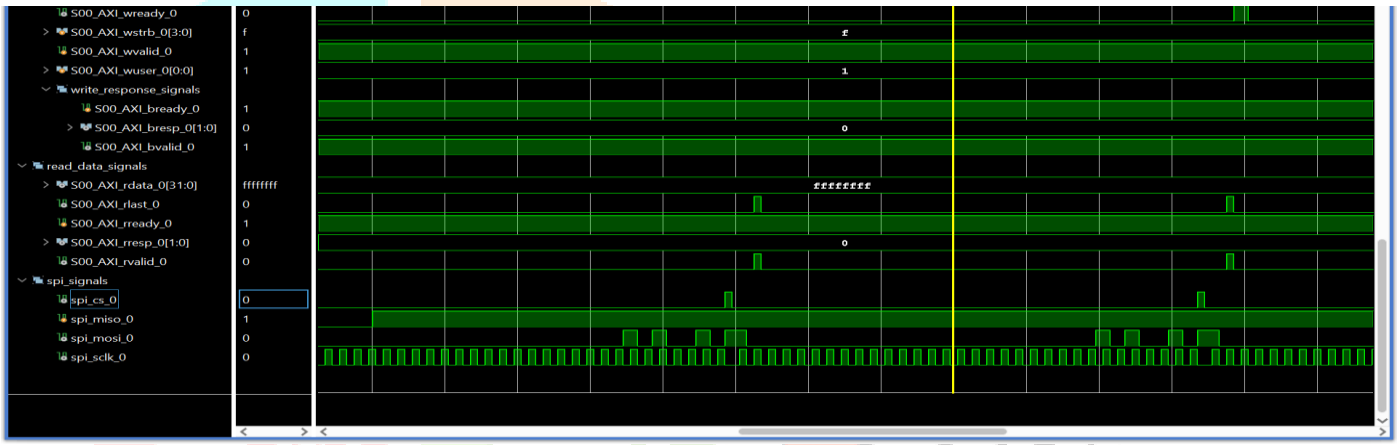


Fig. 7. MOSI carrying write data (SPI).

Read Testcase Results: The read testcase ensured the SPI protocol's ability to correctly receive data from the slave and communicate this back to the A20 processor. In this case, the A20 processor initiated AXI4_ READ transactions, where the AXI_ARADDR signal addressed the read operation to the appropriate memory-mapped register in the SPI protocol. Upon receiving the read request, the SPI protocol responded by placing the data on the AXI_RDATA signal, and the AXI_RREADY signal indicated that the master was ready to accept the data. The data received via AXI_RDATA was compared against the expected values to



confirm that the SPI protocol was correctly receiving data from the slave device. This operation completed successfully with no data corruption or timing errors, ensuring reliable data transfer from the SPI slave to the master via the AXI interface.

Fig. 8. miso reading data from the slave

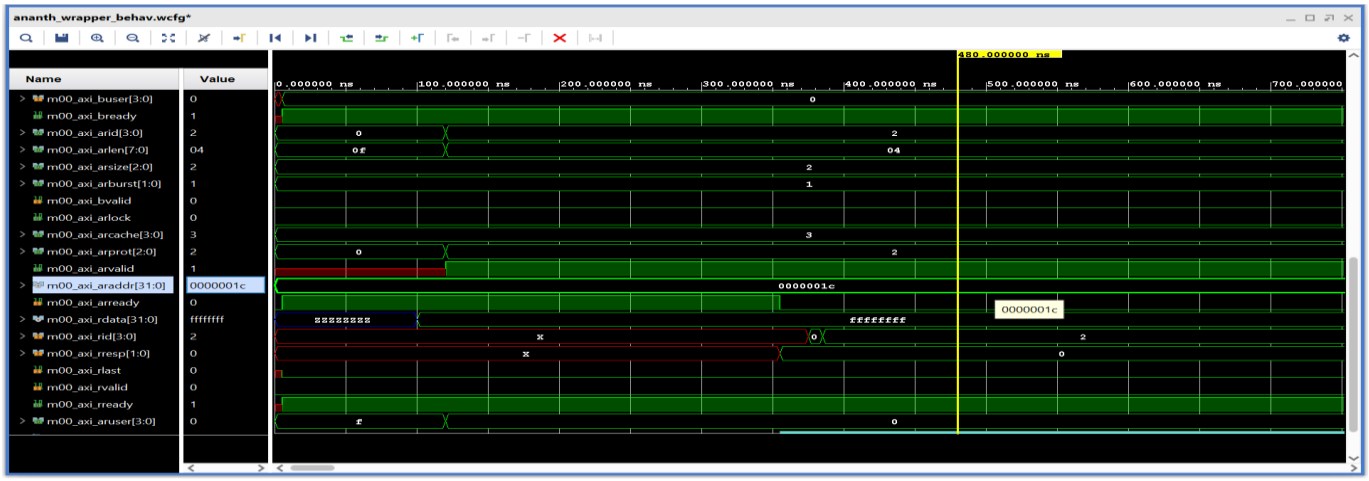


Fig. 9. read data from the slave (SPI) at A20.

Error Handling: The loopback test, executed on the MOSI and MISO lines, validated the SPI protocol's ability to detect and handle transmission errors. Data sent from the master via MOSI was looped back via MISO, and the test confirmed that the transmitted and received data matched, indicating correct data transmission and reception. The loopback mechanism was effective in detecting issues such as misalignment and data corruption. Additionally, the AXI–BVALID and AXI–RREADY signals confirmed proper completion of the transactions, and error conditions such as framing errors were detected, with the SPI protocol responding by handling these errors appropriately.

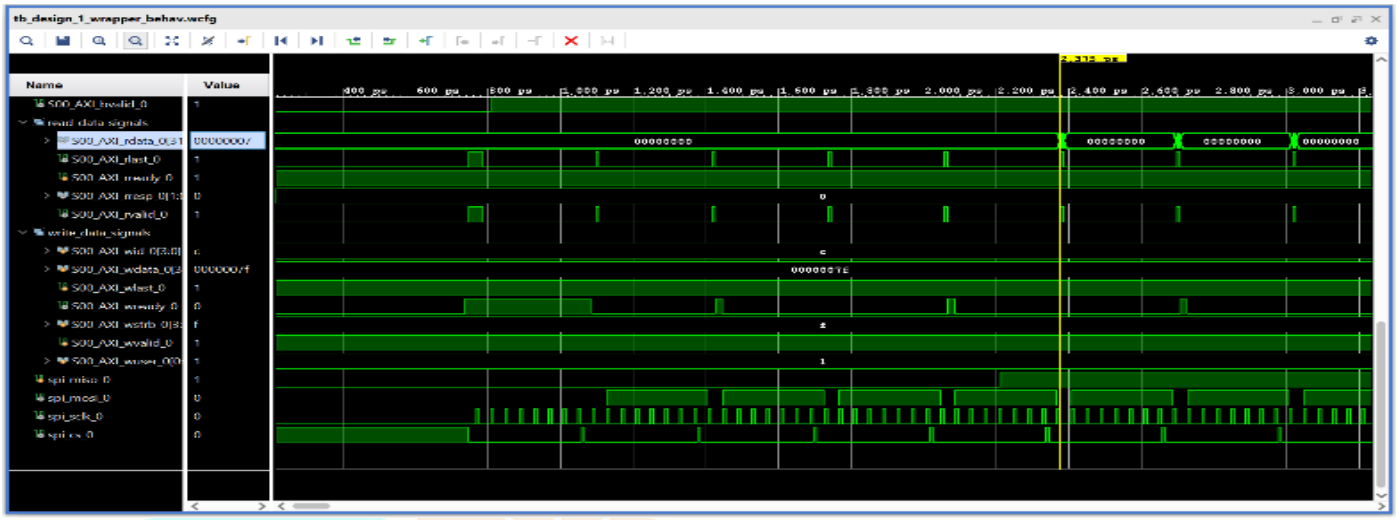


Fig. 10. Mosi, miso carrying the data.

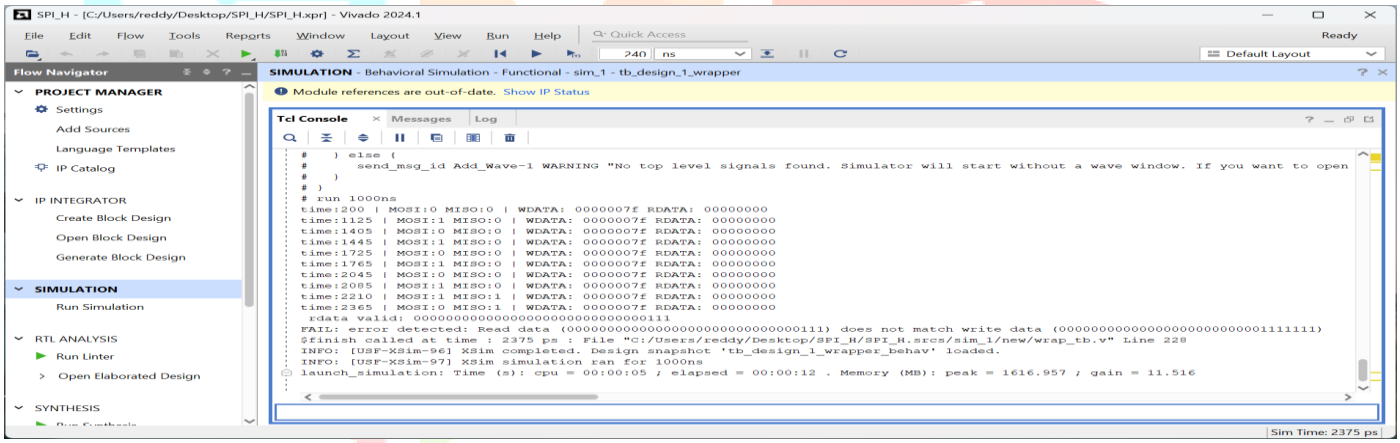


Fig 11. TCL Console window showing error detection

REFERENCES

1] J. Axelson, Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems, 2nd ed. Madison, WI, USA: Lakeview Research, 2007.
[2] ARM Ltd., "AMBA AXI and ACE Protocol Specification," ARM IHI 0022E, 2010. \[Online]. Available: [https://developer.arm.com] (https://developer.arm.com)
[3] Open POWER Foundation, "A2O Core GitHub Repository." \[Online]. Available: [https://github.com/openpower-cores/a2o] (https://github.com/openpower-cores/a2o)
[4] Mentor Graphics, "Questa Advanced Simulator." \[Online]. Available: [https://www.mentor.com/products/fv/questa/] (https://www.mentor.com/products/fv/questa/)
[5] Xilinx Inc., "Vivado Design Suite User Guide." UG910, 2021. \[Online]. Available: [https://www.xilinx.com] (https://www.xilinx.com)
[6] C. Spear and G. Tumbush, SystemVerilog for Verification: A Guide to Learning the Testbench Language Features, 3rd ed. Springer, 2012.