

Design and FPGA Implementation of a 64 bit QSPI Controller

Ulidri Vijay Kumar¹, Dr.V.Sumalatha²

¹PG Scholar, Department of ECE, JNTUA College of Engineering Ananthapuramu, Anantapur, Andhra Pradesh, India, ulidrivijaykumar@gmail.com

²Sr.Professor, Department of ECE, JNTUA College of Engineering Ananthapuramu, Anantapur, Andhra Pradesh, India, vsumalatha.ece@jntua.ac.in

Abstract— This paper presents the design, verification, and FPGA implementation of a 64bit Quad Serial Peripheral Interface (QSPI) controller for high-speed communication with external flash memory. The proposed architecture incorporates a 64-bit internal data path and four-bit parallel data transfer to enhance data-transfer efficiency over conventional single-bit SPI communication. The controller is developed in Verilog HDL and employs a finite-state-machine-based control architecture to manage the command, address, dummy, and data phases of QSPI transactions. Both read and write operations are supported, with a 100 MHz system clock and a 50 MHz QSPI serial clock. Functional verification is performed through RTL simulation to validate the sequencing and timing of control and data signals across different transaction phases. Following verification, the design is synthesized and implemented on a Xilinx ZedBoard featuring the Zynq-7000 XC7Z020 FPGA. Hardware validation confirms the correct operation of the chip-select, serial clock, and quad I/O signals during QSPI read and write transactions. The proposed architecture provides an efficient FPGA-based solution for high-speed flash-memory interfacing and is suitable for data-intensive embedded and digital systems requiring improved transfer efficiency and reduced transaction latency.

Keywords—64-bit QSPI Controller, FPGA Implementation, Quad Serial Peripheral Interface, Verilog HDL, Zynq-7000.

I. INTRODUCTION

Modern embedded systems and System-on-Chip (SoC) platforms require efficient and reliable data transfer between processing units, memory

devices, and peripheral components. With the increasing use of FPGA-based systems in embedded computing, Internet of Things (IoT), industrial automation, and data-intensive applications, the performance of communication interfaces has become an important consideration. Serial interfaces are widely used because they require fewer interconnections while providing a simple and controllable method for transferring data between digital devices.

The Serial Peripheral Interface (SPI) is a widely used synchronous serial communication protocol for interfacing microcontrollers, processors, FPGAs, memories, and peripheral devices. A conventional SPI interface generally consists of Serial Clock (SCLK), Chip Select (CS), Master out Slave in (MOSI), and Master in Slave out (MISO) signals as shown in figure 1. The master controls the communication by generating the clock and selecting the required slave device through the chip-select signal. Its simple architecture, low pin count, and straightforward control mechanism make SPI suitable for a wide range of embedded applications.

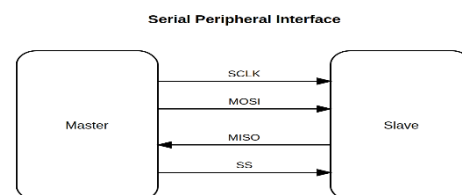


Fig. 1: Basic SPI communication interface

However, conventional SPI transfers data through a single active data path, resulting in one-bit data transfer per clock cycle. As the amount of transferred data increases, more clock cycles are required to complete a transaction, which can

increase transfer time and limit the effective throughput. Increasing the clock frequency can improve the transfer rate, but may also introduce additional timing and signal-integrity requirements. These limitations become more significant in applications involving frequent and high-volume data transfers.

Quad Serial Peripheral Interface (QSPI) extends the conventional SPI concept by using four data lines, IO0–IO3, for parallel data transfer as shown in figure 2. In quad operation, four bits can be transferred simultaneously during a clock cycle, thereby improving the effective data-transfer rate without requiring a proportional increase in clock frequency. A typical QSPI transaction includes instruction, address, dummy, and data phases, each requiring appropriate control and timing.

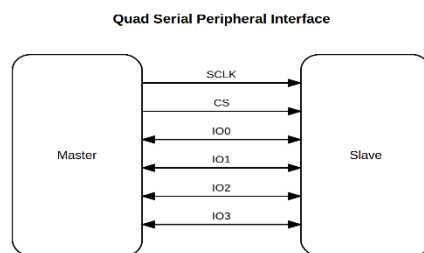


Fig. 2 : Basic QSPI communication interface

Although QSPI provides improved transfer efficiency, the use of multiple I/O lines and different transaction phases introduces additional requirements for the controller architecture. Proper sequencing of transaction phases, clock generation, chip-select control, and parallel data handling are essential for reliable operation. A wider internal datapath can further support efficient organization and processing of data during QSPI transactions.

To address these requirements, this work presents a high-performance 64-bit QSPI controller implemented using Verilog HDL. The proposed design combines a 64-bit internal datapath with four-bit QSPI I/O operation to improve data-transfer efficiency and supports both read and write transactions. The design focuses on efficient transaction control, parallel data handling, and reliable QSPI operation for high-speed data-transfer applications.

II. LITERATURE REVIEW

The increasing demand for high-speed data transfer in embedded systems and System-on-Chip (SoC) applications has encouraged research on SPI and advanced interfaces such as QSPI. Existing studies have explored controller architectures, memory access, throughput, latency, FPGA implementation, DMA-based transfer, high-frequency operation, and verification methodologies. The following studies provide representative approaches and identify areas for further improvement.

Li and Zuo [1] presented a UVM-based functional verification approach for a QSPI module. Their work developed a reusable verification environment for evaluating different QSPI operations and test scenarios. The study demonstrates the importance of systematic verification for identifying functional errors. However, the work mainly focuses on verification methodology rather than datapath optimization, throughput, or latency improvement.

C.-W. Liao et al. [2] proposed a UVM-based SPI verification framework for IoT and wearable applications. The methodology combines assertion-based verification, functional coverage, and constrained-random stimulus generation, achieving more than 95% coverage and reducing debugging time by approximately 30–40%. However, the study mainly addresses verification efficiency and does not extensively investigate controller throughput, latency, or high-speed architecture.

B. Li et al. [3] proposed a lightweight SPI-Flash controller using the AMBA AHB-Lite architecture. The design operates at approximately 50 MHz and provides 25–50 Mbps throughput, while reducing boot delay to below 5 ms. The work demonstrates efficient processor-to-flash access through bus integration. However, conventional single-line SPI limits its bandwidth compared with multi-I/O interfaces such as QSPI.

Y. Zhang et al. [4] proposed a QSPI interface incorporating a Bits-Decoding algorithm for high-speed Flash-memory access. The design uses a state-machine-based architecture and supports

multiple communication modes to improve compatibility and access efficiency. The study demonstrates the benefits of QSPI for high-speed memory communication. However, further optimization of datapath performance, latency, and internal data processing remains possible.

D. C. Costache et al. [6] developed an FPGA-based SD-card controller using SPI communication. The implementation reports data rates of approximately 10–25 Mbps at frequencies of 25–50 MHz, with latency around 100–200 ns. The work demonstrates practical FPGA-based storage interfacing. However, the achievable data rate remains constrained by the single-data-path architecture of conventional SPI communication.

S. Sneha et al. [8] proposed a DMA-based SPI communication approach for booting the Shakti C-Class processor using QSPI mode. The approach reduces processor involvement during data movement and reports approximately 40–60% reduction in boot time with data rates exceeding 150 Mbps. However, the primary focus is system-level boot optimization and DMA transfer rather than internal QSPI datapath architecture.

X. Wang and Z. Huang [11] proposed a high-speed QSPI memory controller supporting Single Data Rate (SDR), Double Data Rate (DDR), and Execute-In-Place (XIP) operation. Signal calibration techniques were included for reliable high-frequency operation. These features improve memory-access flexibility and performance. However, high-frequency SDR/DDR operation introduces additional timing challenges, implementation complexity, and energy considerations.

The reviewed studies address SPI/QSPI verification, memory access, high-frequency operation, DMA integration, and multi-I/O communication. However, limited emphasis is placed on combining a 64-bit datapath, quad-I/O transfer, and FPGA-based implementation within a compact QSPI controller architecture.

III. PROPOSED METHODOLOGY

The proposed work develops a high-performance 64-bit QSPI controller using Verilog HDL, featuring four-line Quad-I/O transfer, read/write

operations, FSM-based transaction control, and reset management. Operating at a 100 MHz system clock with a 50 MHz QSPI clock, the design is verified through RTL simulation and FPGA hardware implementation.

A. Proposed QSPI Architecture

The proposed QSPI controller is designed to provide efficient data transfer between the FPGA and an external serial flash memory by combining a 64-bit internal datapath with a four-line Quad-SPI interface. The architecture consists of a transaction-control unit, internal data registers, a clock-generation unit, and the external QSPI interface comprising SCLK, CS_n, and IO0–IO3. The 64-bit specification refers to the width of the internal data word processed by the controller, whereas the Quad-SPI interface uses four data I/O lines for parallel transfer. The overall architecture of the proposed controller is illustrated in Fig. 3.

The transaction-control unit coordinates the different phases of a memory operation and generates the required control signals. The internal datapath stores command, address, and 64-bit data information and performs the required serialization and deserialization operations. The clock-generation unit derives the QSPI clock from the system clock and provides the timing reference for data transfer.

A QSPI transaction consists of sequential instruction, address, dummy, and data phases. During the data phase, four bits are transferred simultaneously through IO0–IO3 in Quad mode. Therefore, a complete 64-bit data word is transferred as sixteen 4-bit groups over sixteen QSPI clock cycles, excluding instruction, address, and dummy cycles. At the receiving side, these groups are sampled and reconstructed to obtain the original 64-bit data word.

B. Transaction Control and 16-Bit Data path

The proposed controller employs a synchronous transaction-control mechanism to coordinate the instruction, address, dummy, and data phases of a QSPI transaction. The control unit generates the required sequencing and interface-control signals, while the 64-bit datapath performs data storage, shifting, serialization, and deserialization. The

transaction begins with assertion of the active-low chip-select signal, CS_n, followed by instruction and address transmission we can see in table 1 and 2. Where required, dummy clock cycles are inserted before

the data phase to accommodate memory access latency. The transaction-control FSM manages the sequential progression through command, address, dummy, data, and completion states.

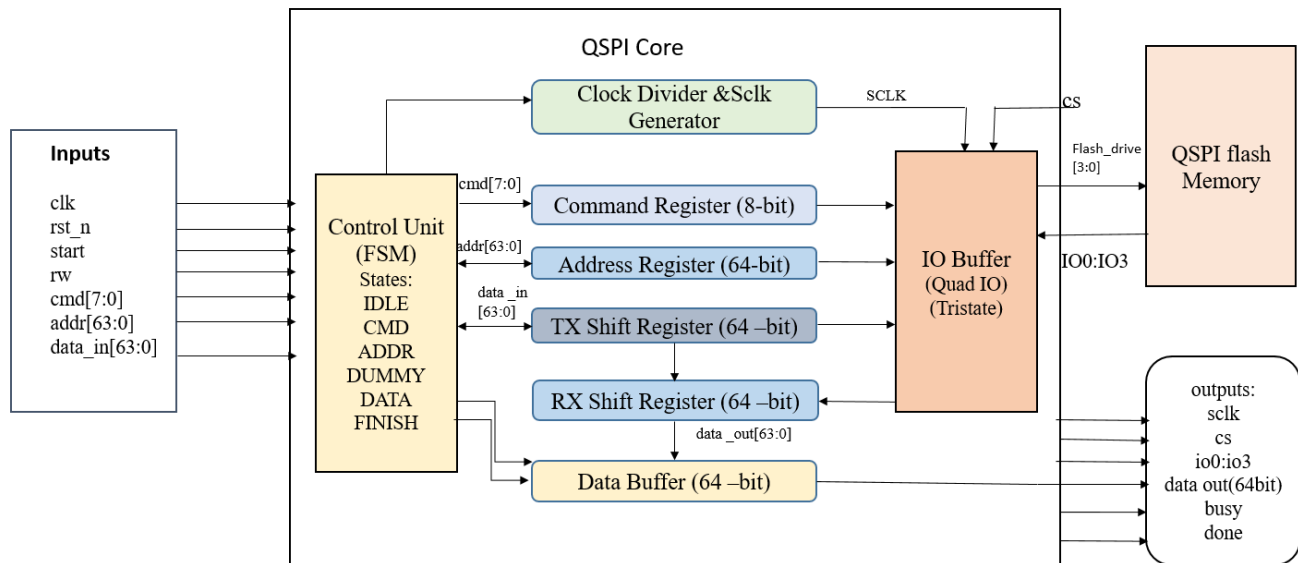


Fig.3.Proposed QSPI Controller

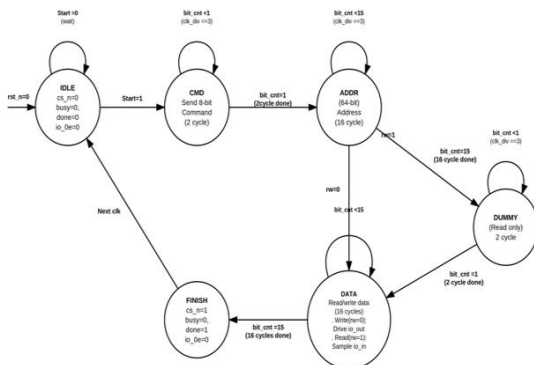


Fig. 4. FSM-based transaction-control diagram of the proposed QSPI controller.

The 64-bit internal datapath interfaces with the four Quad-I/O lines, IO0–IO3. During Quad data transfer, the 64-bit data word is divided into sixteen 4-bit groups, with each group transferred in parallel through IO0–IO3 during one QSPI clock cycle. Thus, the four I/O lines collectively transfer four bits per QSPI clock cycle. The corresponding FSM-based transaction-control sequence is illustrated in Fig. 4.

Table 1:Shows the Control registers

Width	Name	Signal Source	Description
1-bit	start	start	Initiates QSPI transaction (moves

Table 2:Shows the Status registers

Width	Name	Signal Source	Description
1-bit	busy	Busy	Indicates active transaction (1 – Busy, 0 – Idle)
1-bit	done	Done	Operation completed flag
1-bit	intr	Intr	Interrupt signal asserted after completion
64-bit	data_out	data_out	Read data output from QSPI
1-bit	cs_n	cs_n	Chip Select (Active Low)
1-bit	selk	Sclk	Serial clock output
4-bit	io	io[3:0]	QSPI bidirectional data lines

C. QSPI Protocol Sequencing and Transaction Timing

The proposed QSPI controller organizes each memory transaction into a deterministic sequence of protocol phases to ensure proper coordination of control signals and data transfer. The transaction begins with assertion of the active-low chip-select signal, CS_n, followed by the instruction and address phases. The instruction specifies the required memory operation, while the address identifies the target memory location. During read operations, dummy clock cycles may be inserted after the address phase to accommodate the memory access latency before data transfer begins.

During the data phase, the controller uses the four QSPI data lines, IO0–IO3, to perform parallel Quad-mode transfer. Four bits are transferred simultaneously during each QSPI clock cycle. Thus, the 64-bit internal data word is transferred as sixteen 4-bit groups. For write operations, these groups are transmitted through IO0–IO3, while for read operations, the received groups are reconstructed into the corresponding 64-bit data word.

After the required data cycles are completed, CS_n is deasserted and the controller returns to the idle state. The timing relationship among CS_n, SCLK, IO0–IO3, and transaction phases is shown in Fig. 5.

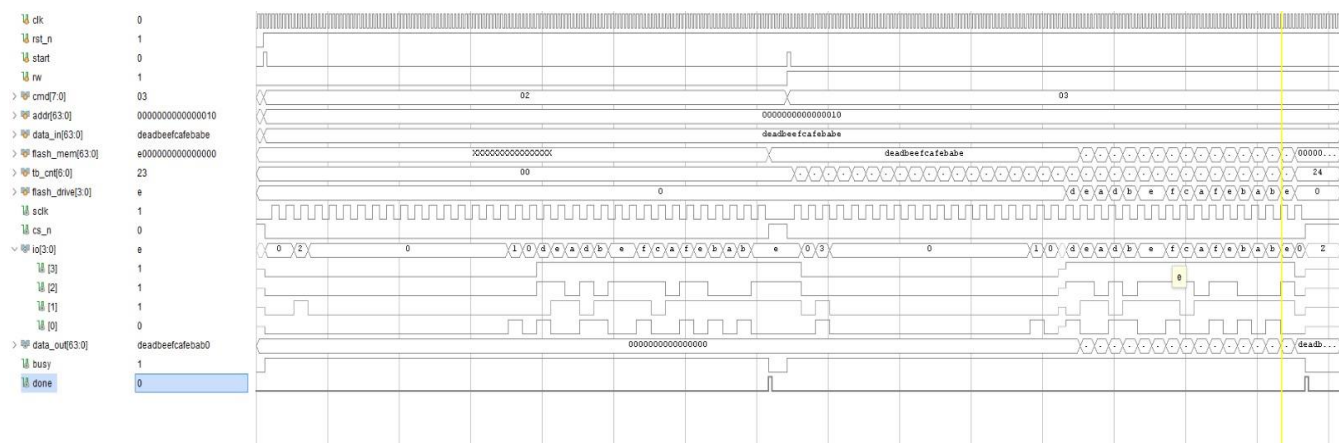


Fig. 5. Timing sequence of the proposed QSPI transaction

D. Design and Implementation Flow

The proposed QSPI controller follows an RTL-to-FPGA development flow, beginning with specification and architecture design, followed by Verilog HDL implementation and testbench development. RTL simulation verifies functional correctness, read/write operations, and timing. After successful verification, the design undergoes synthesis, FPGA implementation, bitstream generation, and hardware validation using ILA. The complete development flow is summarized as figure6. This flow provides a systematic transition from the functional design description to a physically realizable FPGA implementation

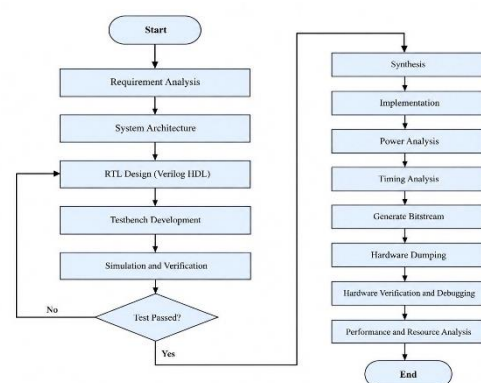


Fig.6:Design Flow of QSPI

E. FPGA Implementation and Hardware Validation

Following RTL verification, the QSPI controller undergoes synthesis, placement, routing,

timing analysis, and bitstream generation. The programmed FPGA is validated using ILA to observe CS_n, SCLK, IO0–IO3, control, and data signals. The observed hardware behavior is compared with RTL results to confirm functional correctness and implementation feasibility.

IV. RESULTS

The proposed QSPI controller is evaluated through RTL simulation, functional test cases, reset verification, FPGA implementation, ILA observation, and hardware validation. The obtained results demonstrate functional correctness, reliable transaction control, and practical FPGA implementation.

i. RTL Simulation Results

Functional verification of the proposed QSPI controller is performed at the RTL level to validate transaction sequencing, control-signal generation, Quad-I/O operation, and 64-bit data transfer. The simulated waveforms are analyzed for both read and write transactions under the specified clock and control conditions.

a. QSPI Read Operation

The QSPI read transaction is initiated by asserting the active-low chip-select signal, followed by the transmission of the read command and address. The controller subsequently generates the required dummy phase before enabling Quad-I/O data transfer. During the data phase, the four I/O lines, IO0–IO3, operate in parallel to transfer the 64-bit data word as shown in figure 7. The simulation waveform verifies the prescribed phase sequence, QSPI clock generation, chip-select timing, and Quad-I/O activity, confirming correct read-transaction behavior.

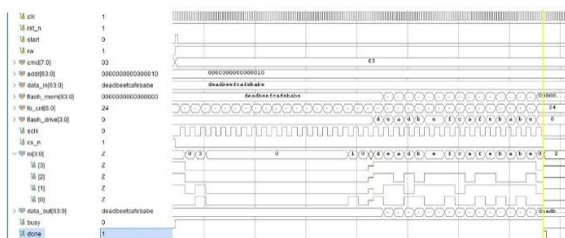


Fig. 7. RTL simulation waveform of the QSPI read transaction.

b. QSPI Write Operation

The QSPI write transaction begins with chip-select assertion and sequential transmission of the write command and target address. The controller then transfers the 64-bit input data through IO0–IO3 using Quad-mode operation. The simulation verifies the synchronization of data transfer with the generated QSPI clock and confirms proper sequencing of the command, address, and data phases. The observed waveform demonstrates correct control-signal coordination and successful execution of the write transaction as shown in figure 8.

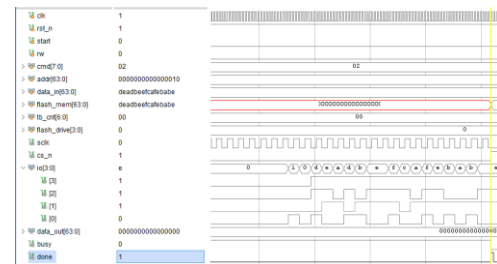


Fig. 8. RTL simulation waveform of the QSPI write transaction.

ii. Reset Verification

The reset operation is verified at the RTL level to ensure that the QSPI controller initializes all control, timing, and interface signals to their defined inactive states. An active-low reset signal, rstnrst_n, is applied to the controller, during which the chip-select signal remains deasserted, the controller status indicates an idle condition, and the QSPI clock and Quad-I/O lines remain inactive. Following reset release, the controller returns to its initial operating state and becomes ready to initiate a new transaction. The observed waveform confirms deterministic initialization and prevents unintended QSPI activity during the reset interval as shown in figure 9.

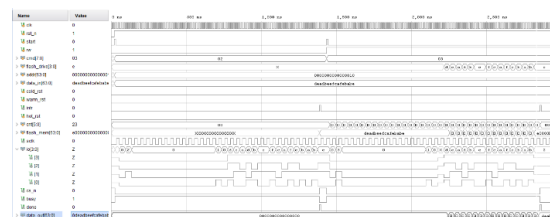


Fig. 9. RTL simulation waveform showing the reset condition of the QSPI controller

iii. FPGA Implementation and ILA Verification

Following RTL verification, the QSPI controller undergoes synthesis, placement, routing, timing analysis, and bitstream generation. The programmed FPGA is validated using ILA to monitor CS_n, SCLK, IO0–IO3, status, and data signals shown in figure 10. The captured waveforms are compared with RTL simulation results to confirm correct transaction sequencing and Quad-I/O operation.

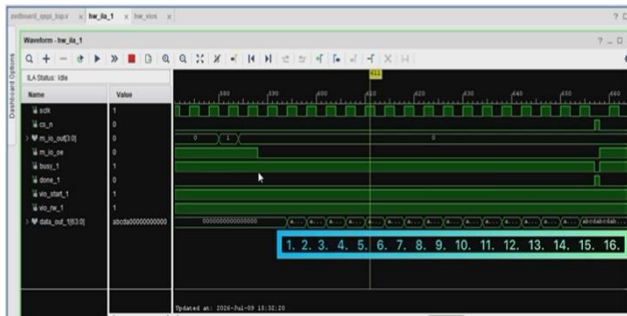


Fig. 10. ILA capture of the implemented QSPI controller during a data transaction.

iv. Hardware Validation

The implemented QSPI controller is validated on the programmed FPGA hardware to confirm its functional operation beyond RTL simulation and internal ILA observation. The controller is operated using the configured 100 MHz system clock and 50 MHz QSPI clock, with the Quad-I/O interface providing parallel data transfer through IO0–IO3.

During hardware validation, the generated QSPI transactions are observed to verify chip-select control, clock generation, command and address transmission, and 64-bit data transfer. The measured hardware behavior is consistent with the expected transaction sequence obtained from RTL simulation and ILA analysis. These observations demonstrate that the proposed controller operates correctly after FPGA implementation and is suitable for practical QSPI-based memory interfacing as shown in figure 10 and hardware validation and implemented or dumped into board as shown in figure 11.

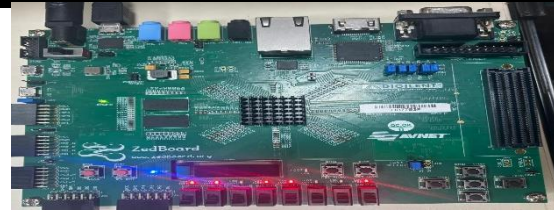


Fig. 11: Hardware validation of the implemented QSPI controller.

v. Resource Utilization and Comparative Analysis

Parameter	[6]	Proposed QSPI
FPGA Platform	Artix-7	Zynq-7000 (ZedBoard)
Communication Mode	Single-bit SPI	Quad-SPI (4-bit data)
SCLK Frequency	25 MHz	50 MHz
Maximum / Theoretical Data Rate	Up to 25 Mbps	200 Mbps
Effective Read Throughput	Not reported	86.49 Mbps
Effective Write Throughput	Not reported	91.43 Mbps
LUTs	414	226
Flip-Flops	283	243

Table 3: Comparison of SPI and QSPI of Proposed

The implemented QSPI controller is evaluated using FPGA resource utilization, operating frequency, datapath width, and Quad-I/O configuration. The obtained synthesis and implementation results are summarized in Table 3.

The proposed architecture is also compared with representative SPI and QSPI implementations based on relevant architectural and performance parameters table 6 and 3. These results provide quantitative measures of implementation efficiency and operating characteristics. Table 4 summarizes the functional verification test cases performed for the proposed QSPI controller. The tests evaluate reset initialization, transaction control, 64-bit read and write operations, burst

transfer, clock generation, chip-select behavior, data patterns, reset recovery, and back-to-back

transactions to verify functional correctness and reliable operation.

S.No.	Test Case	Objective	Input / Stimulus	Expected Result
1	Reset Functionality	Verify proper initialization after reset	rst_n = 0 for few cycles	CS_n=1, busy=0, SCLK=0, IO inactive, FSM = IDLE
2	Start Transaction	Verify FSM transition from IDLE to active state	rst_n=1, start=1 pulse	busy=1, CS_n=0, transaction begins
3	Read Operation (64-bit)	Verify complete read flow and data capture	Read request with flash response	FSM executes correctly and data_out matches expected 64-bit value
4	Write Operation (64-bit)	Verify complete 64-bit data transmission	data_in = 64-bit pattern	Data transmitted correctly through IO0–IO3
5	Burst Read/Write Operation	Validate multi-word transfer in burst mode	Burst of 64-bit data	All words are transferred and verified correctly
6	Multi-Mode SPI Operation	Validate Single, Dual, and Quad SPI modes	Mode selection	Correct I/O mapping and data transfer in each supported mode
7	Clock Generation and Scaling	Verify SCLK generation and stability	System clock = 100 MHz	Correct and stable QSPI clock generation
8	Chip-Select Control	Validate CS behavior during transactions	Read/write operation	CS_n=0 during transfer and CS_n=1 after completion
9	Reset During Operation	Verify safe recovery during an active transfer	Reset applied during transaction	FSM returns to IDLE and interface returns to inactive state
10	Data Pattern Test	Verify transfer correctness for different patterns	All 0s, all 1s, alternating patterns	Received data matches transmitted data
11	Back-to-Back Transactions	Verify multiple sequential operations	Multiple start pulses	Each transaction completes correctly without data corruption

Table 4: Different test cases

Table 5: Performance Comparison Of SPI And Proposed QSPI Architectures

V. CONCLUSION

This paper presented the design, verification, and FPGA implementation of a high-performance 64-bit QSPI controller for efficient serial flash-memory interfacing. The proposed architecture

Parameter	[6]	[3]	Proposed QSPI
Protocol Type	Single-bit SPI	Multi-bit SPI	Quad SPI
Max. SCLK Frequency	25 MHz	125 MHz	50MHz
Throughput	25 Mbps	13.62Mbps	200 Mbps
Area (LUTs)	414 LUTs	107 LUTs	21LUTs
Registers (Flip-Flops)	283 FFs	138 FFs	243 FFs
FPGA Platform	Artix-7	FPGA	Zynq 7000

integrates a 64-bit internal datapath with a four-line

Quad-SPI interface and FSM-based transaction control to support read and write operations through defined command, address, dummy, and data phases. RTL simulation was performed to verify functional sequencing, reset behavior, control-signal generation, and Quad-I/O data transfer. Following functional verification, the design was synthesized and implemented on FPGA hardware, where ILA-based observation and hardware validation were used to confirm the implemented operation. The obtained implementation results demonstrate the feasibility of the proposed architecture for high-speed FPGA-based memory interfaces. The combination of wide internal data handling and Quad-I/O transfer provides a structured approach for improving data-transfer efficiency while maintaining deterministic transaction control.

VI. REFERENCES

- [1] H. Li and S. Zuo, "Functional verification of QSPI module based on UVM implementation," *Journal of Physics: Conference Series*, vol. 2645, no. 1, Art. no. 012002, pp. 1–8, 2023, doi: 10.1088/1742-6596/2645/1/012002.
- [2] C.-W. Liao, H.-C. Yu, and Y.-C. Liao, "Verification of SPI Protocol Using Universal Verification Methodology for Modern IoT and Wearable Devices," *Electronics*, vol. 14, no. 5, Art. no. 837, 2025, doi: 10.3390/electronics14050837.



[3] B. Li, W. Wang, H. Chu, Z. Liu, and W. Wang, "A Lightweight SPI-Flash Controller Based on AMBA AHB-Lite Bus," *Micromachines*, vol. 15, no. 10, Art. no. 1246, Oct. 2024, doi: 10.3390/mi15101246.

[4] Y. Zhang et al., "A QSPI Interface Supporting 'Bits-Decoding' for High-Speed Access to Flash," in *Proc. 16th IEEE Int. Conf. Anti-counterfeiting, Security, and Identification (ASID)*, Xiamen, China, 2022, pp. 24–27, doi: 10.1109/ASID56930.2022.9995778.

[5] P. Kavitha, V. Sreelatha Reddy, N. Swapna, and C. Suneetha, "Design and Implementation of an SPI-Based Embedded System Using MicroBlaze on FPGA," in *Proc. Int. Conf. Computer Science and Communication Engineering (ICCSCE)*, *Advances in Computer Science Research*, vol. 124, pp. 2338–2347, 2025.

[6] D. C. Costache, L. A. Perișoară, and A. Florescu, "FPGA Implementation of a SD Card Controller Using SPI Communication," in *Proc. 12th Int. Conf. Electronics, Computers and Artificial Intelligence (ECAI)*, Bucharest, Romania, 2020, pp. 1–4, doi: 10.1109/ECAI50035.2020.9223238.

[7] I.-B. Brezeanu, S. Isari, A. Vasile, and G. Brezeanu, "Design and Verification of a High Frequency, SPI Control Block," in *Proc. Int. Semiconductor Conf. (CAS)*, Sinaia, Romania, 2023, pp. 175–178, doi: 10.1109/CAS59036.2023.10303721.

[8] S. S., G. G. Nath, A. V. R., and S. G., "Efficient Booting Process of Shakti C-Class Core Using UDMA SPIM in QSPI Mode," in *Proc. Int. Conf. Smart Electronics and Communication Systems (ISENSE)*, Kottayam, India, 2024, pp. 1–5, doi: 10.1109/ISENSE63713.2024.10872359.

[9] L. Quill, C. Imogen, and I. Casimir, "Design and Verification of FlexSPI Controller Using UVM for Efficient Flash Access," *Transactions on Computational and Scientific Methods*, vol. 4, no. 9, 2024.

[10] A. K., S. Sushma, and V. S. Teja, "Implementing and Verifying the SPI Communication Protocol in ASICs with Cadence EDA Tools," in *Proc. Global Conf. Communications and Information Technologies (GCCIT)*, Bangalore, India, 2024, pp. 1–5, doi: 10.1109/GCCIT63234.2024.10861947.

[11] X. Wang and Z. Huang, "Design and Implementation of High Speed QSPI Memory Controller," in *Proc. IEEE 4th Int. Conf. Electronics Information and Emergency Communication*, Beijing, China, 2013, pp. 82–85, doi: 10.1109/ICEIEC.2013.6835459.

[12] Y. Guo et al., "A SPI Interface Module Verification Method Based on UVM," in *Proc. IEEE Int. Conf. Information Technology, Big Data and Artificial Intelligence (ICIBA)*, Chongqing, China, 2020, pp. 1219–1223, doi: 10.1109/ICIBA50161.2020.9277156.