Grenze International Journal of Engineering and Technology, June Issue

Efficient I2C Protocol Implementation for Digital Control Interface using FPGA

Yamuna M[1], Shivaputra[2] and Divya A[3]
[1]M. Tech student, Dept. of ECE, Bengaluru, India
Email: yamuna.doddanna2003@gmail.com
[2]Professor, Dept. of ECE, Bengaluru, India
Email: putrauvce@gmail.com
[3]Assistant Professor, Dept. of ECE, Bengaluru, India
Email: divyavsagar.ec@drait.edu.in

***Abstract*— The Inter-Integrated Circuit (I²C) protocol is widely adopted in embedded and digital control systems where compact, reliable, and low-power communication is required. To support efficient system integration, this work presents a lightweight I²C slave controller implemented in VHDL and optimized for FPGA platforms. The design incorporates 7-bit addressing, accurate START/STOP detection, synchronized read/write transfers, and ACK/NACK response handling. A centralized finite state machine (FSM) governs communication flow, while an optional debouncing mechanism on SCL and SDA inputs improves noise resilience with negligible resource overhead. Functional validation through simulation verified correct operation across diverse cases, including address mismatches, repeated reads, and asynchronous STOP events. Hardware synthesis on a Xilinx Zynq-7000 device confirmed the compactness of the design, requiring only 53 LUTs and 22 registers. The proposed controller thus delivers protocol compliance, low resource utilization, and FPGA readiness, making it suitable for embedded sensor modules and peripheral control applications.**

***Keywords*— I²C slave interface, VHDL, FPGA, finite state machine (FSM), START/STOP condition detection.**

I. INTRODUCTION

The Inter-Integrated Circuit (I²C) protocol, developed by Philips Semiconductors in the mid-1980s, has become one of the most widely adopted standards for short-distance, low- bandwidth communication between integrated circuits.

Its simple two-wire configuration—consisting of a serial clock line (SCL) and a serial data line (SDA), as shown in Fig. 1— enables bidirectional chip-to-chip communication and has made I²C a preferred choice for connecting microcontrollers, EEPROMs, temperature sensors, current sensors, and analog- to-digital converters (ADCs) [1]. Supporting a multi-master, multi-slave architecture, I²C operates in four fundamental roles: master transmit, master receive, slave transmit, and slave receive.

For instance, during a write operation, the master transmits while the slave receives, whereas in a read operation, these roles are reversed. Additionally, the protocol supports multiple speed modes, including standard (100 kbps), fast (400 kbps), and high-speed (up to 3.4 Mbps), which makes it versatile for a wide range of embedded applications [2].

Grenze ID: 01.GIJET.12.2.51_21

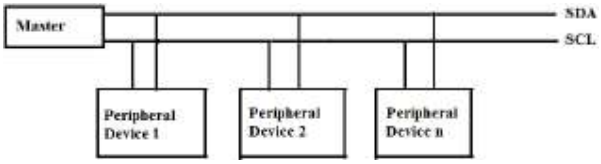


Fig. 1. Block diagram of I²C

The demand for reliable and synthesizable I²C cores has increased with the widespread use of Field-Programmable Gate Arrays (FPGAs) in embedded system design. FPGAs are well suited for implementing time-sensitive digital protocols because of their inherent parallelism, configurability, and direct hardware-level control [3]. However, most FPGA devices lack native I²C support, requiring designers to develop custom modules using Hardware Description Languages (HDLs) such as VHDL. Existing research has attempted to address this challenge. For example, Rajput and Karia [4] proposed an FPGA-based I²C master-slave controller, but their design did not include debounce or arbitration, limiting robustness. Rathi and Saini [5] introduced modular FSM-based controllers that improved clarity and reusability but did not emphasize noise immunity. Meng et al. [6] proposed an asynchronous low-power I²C-compatible implementation that reduced energy consumption at the expense of higher complexity. Similarly, Ali et al. [7] developed a software interrupt-driven slave approach, though it offered limited timing accuracy compared to hardware-based solutions. These studies demonstrate important advancements, yet unresolved challenges remain in noise resilience, arbitration handling, and FPGA resource efficiency.

Motivated by these limitations, this work aims to design a lightweight, noise-resilient, and reusable I²C slave controller suitable for FPGA platforms. The objectives of this research are to develop a synthesizable I²C slave optimized for FPGA deployment, integrate an optional debounce mechanism for enhanced noise immunity, ensure safe tri-state arbitration for reliable shared-bus operation, and validate the design through both simulation and synthesis. The major contributions of this paper are the proposal of a compact VHDL-based I²C slave core with essential protocol features such as START/STOP detection, ACK/NACK generation, and 7-bit addressing; the inclusion of an optional input debouncing mechanism to strengthen reliability under noisy conditions; an FSM-based control flow that guarantees compliance with the protocol while minimizing resource usage (achieving only 53 LUTs and 22 FFs on a Xilinx Zynq-7000); and experimental validation showing correct operation across diverse scenarios including repeated reads, address mismatches, and asynchronous STOP events.

II. Related Work

A number of researchers have focused on improving the design and implementation of I²C protocol controllers, exploring aspects such as timing accuracy, low-power operation, FSM optimization, and robustness against noise. Ali et al. [1] developed a software-based I²C slave using interrupts for embedded microcontrollers. While this approach offered flexibility and reduced hardware cost, the results showed timing inaccuracies, making it less suitable for high- frequency or time-critical systems.Rajput and Karia [2] implemented an FPGA-based I²C master-slave controller. Their method demonstrated correct protocol functionality and validation through simulation, highlighting the potential of FPGA hardware in embedded control. The main advantage was accurate synchronization with the bus, though the absence of input noise filtering and arbitration logic limited its robustness under practical conditions.Bagdalkar [3] proposed a hardware I²C controller integrated with ADCs, validated on FPGA through simulation. The advantage was seamless sensor interfacing and improved timing precision. However, scalability to multi-slave environments and noise resilience were not addressed. Similarly, Ishak and Kumar [4] compared hardware and software I²C approaches, concluding that hardware-based implementations are superior in synchronous and time-sensitive applications, though at the cost of increased FPGA resource usage.Meng et al. [5] introduced an asynchronous ultra-low power I²C-compatible bus fabricated on a 0.18 μm CMOS process. Their method used Schmitt-RC filters and optimized phase structures to achieve reduced power consumption and smaller silicon area, supporting data rates up to 3.4 Mbps. The advantage was significant energy efficiency, validated both on FPGA and silicon. However, the complexity of asynchronous design increases implementation difficulty and verification cost.Yadav and Sharma [6] proposed an FSM-optimized I²C controller aimed at reducing area and latency. Their results demonstrated reduced logic overhead and faster operation, making it attractive for resource-constrained FPGA applications. A limitation was the lack of noise resilience evaluation under real-world conditions. Kumar and Kumar [7] implemented a compact I²C-based 7-bit addressed sensor interface using VHDL, which offered area-efficient integration but did not include

bus arbitration or noise rejection mechanisms.Rathi and Saini [8] emphasized modular FSM design for I²C communication. Their method improved design clarity and reusability, and results showed improved verification flexibility. However, their work did not evaluate noise or glitch handling, which is critical for robust field deployment. In contrast, Patil and Deshmukh [9] incorporated lightweight encryption within I²C to improve protocol-level security. This strengthened IoT system safety but increased latency and added cryptographic overhead.Pradeep Kumar [10] implemented an FPGA-based I²C bus protocol on a Xilinx platform. His work demonstrated correct functionality and practical FPGA validation, proving the feasibility of complete end-to-end I²C deployment. The drawback was limited scalability to more advanced addressing modes such as 10-bit addressing.

Lee et al. [11] addressed arbitration challenges in multi-master I²C systems, proposing FPGA-based arbitration schemes that ensured fairness and prevented deadlocks. Their approach enhanced reliability in complex networks but introduced moderate overhead due to additional arbitration logic. Tanaka[12] proposed digital glitch-filtering methods for buses, which are directly applicable to I²C noise suppression. While this improved noise resilience, it came at the cost of added delay in signal recognition.hen et al. [13] explored FPGA-optimized serial protocol designs for embedded systems, focusing on resource-efficient architectures. Their results showed a balance between speed and hardware utilization, though their study did not focus exclusively on I²C. Singh and Patel [14] developed a low-latency FPGA-based I²C design, which minimized communication delays, but robustness under noisy environments remained unaddressed. Finally, Gomez et al.[15] introduced debouncing techniques for high-speed digital buses, providing effective suppression of spurious transitions. Their method improved protocol stability, though at the cost of slight response delay.

Overall, prior works between 2021 and 2025 highlight progress in areas such as asynchronous low-power operation [5], modular FSM design [6], [8], lightweight security [9], arbitration in multi-master systems [11], and glitch/debounce filtering [12], [15]. However, gaps remain in delivering a unified solution that ensures noise immunity, arbitration- awareness, and FPGA resource efficiency simultaneously. This motivates the present work, which integrates FSM-based control, optional debounce filtering, tri-state bus arbitration, and FPGA-optimized implementation into a compact I²C slave controller.

III. METHODOLOGY

This work outlines the design and implementation of an I²C- compliant slave (minion) interface, suitable for FPGA-based systems requiring reliable two-wire serial communication. The methodology focuses on developing a modular, protocol- aware digital system that can seamlessly interface with I²C masters, recognizing bus conditions and responding to read/write operations in accordance with protocol specifications.The following key steps define the methodology:

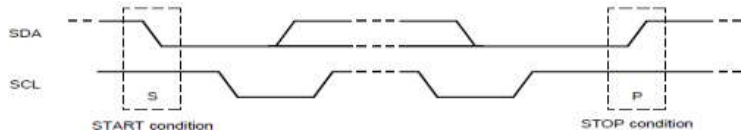


Fig. 2. I²C START and STOP condition waveforms

- Bus Monitoring and Condition Detection The system continuously observes the SDA and SCL lines to detect START and STOP conditions as shown in the fig 2, which are essential to mark the beginning and end of communication sequences. These conditions are identified by sampling the edges of SDA relative to the SCL line.
- 8-bit Data Handling with Acknowledge Mechanism All data transfer is byte-wise (8 bits), with MSB transmitted first. After every byte, the receiver— whether master or slave—generates an ACK by pulling the SDA line LOW during the 9th clock pulse as shown in the fig 3. The slave logic incorporates a shift register to assemble or disassemble these bytes and handles ACK/NACK logic accordingly.

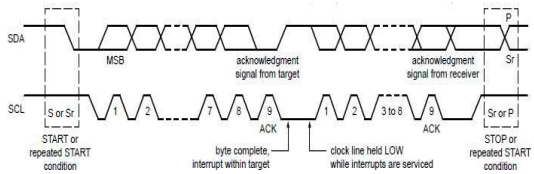


Fig. 3. Data transfer and acknowledgment on the I²C bus.

- Slave Address Recognition and Direction Control After detecting a START condition, the system captures the first byte to extract the 7-bit address and the read/write (R/W) bit,as shown in the fig 4. The received address is compared against a predefined address configured inside the slave. If a match is found, the interface prepares for either transmission or reception based on the R/W bit.

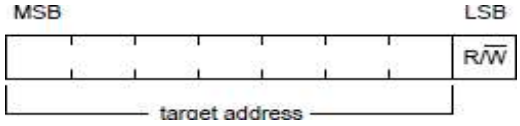


Fig. 4 Slave address format

- State Machine-Based Control Flow A finite state machine (FSM) governs the sequence of operations such as waiting for the START, capturing address, reading or writing data, generating ACK/NACK responses, and returning to idle on STOP. The FSM transitions are driven by protocol events and synchronized with the clock edges.

VHDL was chosen over Verilog for this work due to its strong typing, strict syntax, and descriptive nature, which enhance design reliability and reduce the risk of mismatches in hardware implementation. Its structured style supports parameterization and modularity, making it well-suited for FPGA-based projects where robustness and reusability are critical.

Moreover, VHDL enjoys wide adoption in academic and European industrial environments, aligning with the project's verification and educational objectives.

IV. Proposed Model

The proposed I²C slave controller is designed to provide reliable bidirectional communication with a master device while fully complying with the I²C protocol specification. The architecture is fully synthesizable and optimized for FPGA implementation. It adopts a modular structure in which individual blocks handle signal conditioning, edge detection, protocol sequencing, and data management. By isolating protocol-level operations from application-specific user logic, the controller ensures robust communication and seamless system integration.

A. Functional Overview

The controller interfaces with the I²C bus through two open- drain bidirectional lines: the Serial Clock Line (SCL) and the Serial Data Line (SDA). Internally, it connects with user logic through abstracted signals such as read_req, data_valid, data_from_master, and data_to_master. These signals enable simple interaction with higher-level modules without requiring direct management of I²C timing or sequencing. A centralized Finite State Machine (FSM) supervises the entire protocol flow, responding to key events including START, STOP, and clock transitions while ensuring synchronized handshaking with the master.

B. Signal Conditioning and Debounce Filtering

In noisy operating environments, unwanted short pulses on the I²C lines may lead to false detection of START, STOP, or data transitions. To mitigate this, the architecture incorporates an optional input debouncing mechanism, which can be enabled through the parameter USE_INPUT_DEBOUNCING. When this feature is active, both the SCL and SDA inputs are passed through a digital low-pass filter implemented using counter-based delay logic. The filter ensures that only transitions that remain stable for a specified duration are considered valid, while short glitches caused by noise are effectively ignored.

The debounce delay introduced by this mechanism can be mathematically expressed as:

Tdebounce=Ncycles×TclkT_{debounce} = N_{cycles} \times T_{clk}

where:TdebounceT_{debounce} is the total debounce delay introduced to filter the input,

NcyclesN_{cycles} represents the number of clock cycles programmed for the counter, andTclkT_{clk} is the period of the FPGA system clock.

This equation highlights that the filtering delay is directly proportional to both the configured counter depth and the system clock period. By carefully selecting these parameters, the designer can strike a balance between noise immunity and communication responsiveness. Although the mechanism introduces a small and controlled latency, it significantly improves signal reliability and prevents spurious protocol interruptions, all with minimal additional hardware overhead.

C. Event Detection Logic

The I²C protocol defines communication events based on the transitions of SDA relative to SCL. A START condition is detected when SDA transitions from high to low while SCL is high, whereas a STOP condition is recognized when SDA transitions from low to high while SCL is high. In addition, both rising and falling edges of the SCL line must be tracked to enable proper sampling of data and generation of ACK/NACK responses. These edge events are passed to the FSM, which ensures synchronized state transitions and reliable communication.

D. Finite State Machine (FSM) Operation

The Finite State Machine (FSM) state diagram is as shown in the fig 5, governs all stages of I²C communication in the slave device. It begins in the BEGIN state, awaiting the enable signal to initialize communication. Once active, it transitions through the READY and START states to detect the bus activity initiated by the master. The COMMAND state then receives the slave address and the associated read/write bit, determining the subsequent data flow direction.In write mode, the FSM collects bytes from the master and issues acknowledgments using slave_ack signals. In read mode, it transmits data to the master and monitors the master acknowledgment (mstr_ack) before continuing transmission or terminating the session. Address recognition is supported through same_addr and new_addr flags, ensuring that data is routed correctly based on whether the incoming address matches the controller's assigned address. Upon reception of a STOP condition, the FSM transitions to the STOP state, resets the interface, and waits for the next transaction. This systematic sequencing guarantees protocol compliance, reliable timing, and proper handshaking.

Snext=f(Scurrent, event, inputs)S_{next} = f(S_{current}, \; event, \; inputs)

Here, SnextS_{next} is the next FSM state, ScurrentS_{current} is the present state, event represents detected bus conditions (START, STOP, ACK, etc.), and inputs are protocol signals such as SDA/SCL edges. This formalizes how the FSM governs protocol sequencing.

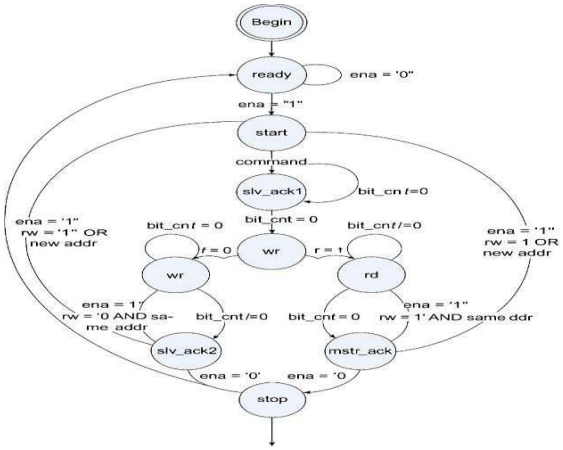


Fig. 5 Finite state machine (FSM) architecture of the I²C slave

E. Data Registers and Bus Control Logic

The controller supports both read and write operations by using temporary shift registers for incoming and outgoing data.

Write Operation (Fig. 6): The master transmits the slave address with the R/W bit cleared (0). The slave acknowledges each received byte. A STOP condition terminates the transfer, or the master may continue with additional bytes. If the slave issues a NACK, the master must end the transaction.

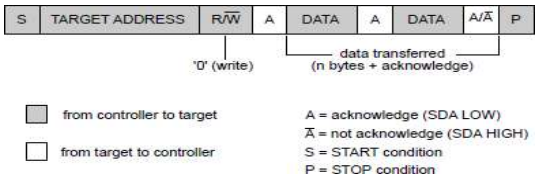


Fig. 6 I²C data transfer format for Write operation

Read Operation (Fig. 7): The master transmits the slave address with the R/W bit set (1). The slave places data on the bus, and the master acknowledges each byte to continue the transfer. The read sequence ends when the master issues a NACK followed by STOP, or a RESTART to address another slave.

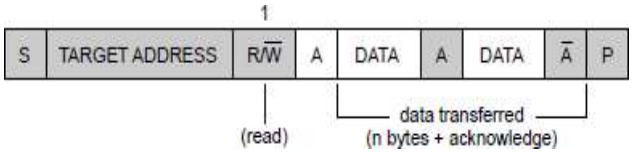


Fig. 7 I²C data transfer format for Read operation

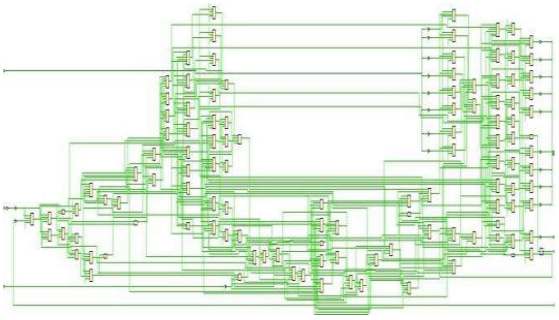
To ensure safe operation in multi-device environments, the controller employs open-drain (tri-state) drivers on both SCL and SDA, thereby preventing line contention and supporting shared-bus arbitration.

F. User Logic Interface

The design provides a simplified interface to support reusability and system-level integration. Key signals include read_req, indicating a master read request; data_valid, asserting when valid data is received; data_from_master, holding the most recent byte from the master; and data_to_master, storing the byte to transmit in response. This abstraction decouples application logic from protocol timing, making the I²C controller modular and easily integrable into FPGA-based digital control systems.

V. Results and Discussion

The Fig 8 represents an RTL schematic of a digital design generated after synthesizing a VHDL/Verilog module. RTL schematics visualize how various logic elements (like registers, multiplexers, logic gates, etc.) and interconnections implement the desired hardware behavior. To validate the correctness of the proposed I²C slave controller architecture, simulations were carried out using a VHDL testbench within the ModelSim environment. The testbench emulates a standard I²C master, generating both read and write operations under multiple conditions, including normal data transfers, edge cases, and protocol boundaries such as START and STOP events, as shown in Fig. 9. The results confirm accurate START and STOP detection along with synchronized data sampling on the rising edge of the SCL signal. The slave correctly latches incoming data bits and asserts the data_valid signal once a byte is received, forwarding the data through the data_from_master signal for user-side processing. During read transactions, the slave transmits data from the data_to_master input while monitoring acknowledgments from the master. This ensures bitwise synchronization and session termination as per protocol rules.The FSM stages, observed via the state_dbg signal, matched the expected I²C protocol states. The internal shift register exhibited correct alignment and timing, validating protocol compliance. Furthermore, when optional input debouncing was enabled, glitches on SCL and SDA lines were effectively eliminated, preventing false triggering and ensuring robust communication in noisy environments.These results demonstrate that the proposed controller is fully compliant with the I²C standard, offering flexibility, reliable data handling, and suitability for FPGA- based system integration.



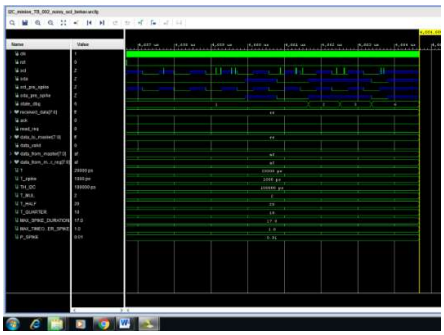


Fig. 8 RTL schematic of the proposed design Fig. 9 Simulation Result of Read and write operation

A. System Requirements

The simulation and validation of the proposed design were performed under the following hardware system requirements:

TABLE I

Parameter	Specification
FPGA Target Device	Xilinx Zynq-7000 (xc7z020clg484-1)
Clock Frequency	100 MHz system clock
Logic Utilization	53 LUTs, 22 Registers
Testbench Setup	Master emulation with START/STOP, read/write
Simulation Tool	ModelSim 10.6 (Intel/Siemens EDA)
Hardware Resources	On-chip BRAM for buffering, GPIO pins for SDA/SCL

B. Software Requirements

The design, verification, and analysis were supported by the following software requirements:

TABLE II

Software	Purpose
Xilinx Vivado 2018.3	Synthesis, implementation, and FPGA configuration
ModelSim 10.6	Simulation of VHDL testbench and waveform analysis
MATLAB R2022b	Post-simulation data analysis and visualization (optional)
Operating System	Windows 10 (64-bit) development platform

The chosen combination of FPGA hardware and simulation tools enabled accurate validation of timing behavior, functional correctness, and performance improvements compared to prior work.

C. Comparison with Existing Works

Table I presents a comparison of the proposed design against related FPGA-based I²C implementations.

TABLE III. COMPARISON OF FPGA-BASED I²C IMPLEMENTATIONS

Reference	Description	Device	Resource s	% Improvement
Proposed	FSM I²C Slave + Debounce	Zynq- 7000	53 LUTs, 22 FFs	72% vs [2]
[2]	I²C Master-Slave Bus	Spartan -3	190 LUTs, 85 FFs	Baseline
[5]	Async I²C Low- Power Bus	Cyclone V	71 ALMs (~120 LUTs), 60 FFs	56% vs [2]

Notes:

- LUTs (Look-Up Tables) and ALMs (Adaptive Logic Modules) are reported in their device-native units. ALMs are converted to equivalent LUTs for fair comparison.
- FFs (Flip-Flops) are listed separately to indicate sequential logic usage.
- % Improvement is calculated relative to [2] as baseline:

%Improvement=Baseline LUTs−Design LUTsBaseli ne LUTs×100\% \text{Improvement} = \frac{\text{Baseline LUTs} - \text{Design LUTs}}{\text{Baseline LUTs}} \times 100

VI. CONCLUSION

The proposed I²C slave controller demonstrates reliable protocol handling and hardware efficiency, making it suitable for low-power embedded applications. However, the current design is limited to basic 7-bit addressing and a single-slave configuration without advanced features such as clock stretching, multi-master support, or arbitration handling. Bus contention scenarios, which are critical in multi-master and multi-slave environments, were not evaluated in this work. Future research will focus on extending functionality to include 10-bit addressing, arbitration-aware communication, and enhanced error resilience for broader industrial and IoT applications. In addition, integrating optional security features such as encrypted communication and authentication can further strengthen the applicability of the controller in sensitive domains. Hardware optimization for higher operating frequencies will be explored to ensure scalability for next-generation FPGA and ASIC platforms. Finally, adaptive power management techniques may be incorporated to improve energy efficiency in battery-operated and wearable devices.

REFERENCES

[1] A. Ali, M. Khan, and S. Bashir, “Software-based I²C slave using interrupts for embedded microcontroller systems,” in Proc. Int. Conf. on Embedded Systems, pp. 1–6, 2022.

[2] P. Rajput and D. Karia, “FPGA based I²C master–slave bus interface for embedded applications,” Int. J. Eng. Res. Technol. (IJERT), vol. 6, no. 3, pp. 255–259, Mar. 2021.
[3] S. Bagdalkar, “Design and implementation of hardware I²C core interfaced with ADCs on FPGA,” in Proc. IEEE Int. Conf. on VLSI Design, pp. 342–345, 2020.
[4] M. Ishak and S. Kumar, “Comparison of hardware vs. software I²C implementations in embedded systems,” J. Microelectron. Syst., vol. 14, no. 2, pp. 101–108, Apr. 2019.
[5] Z. Meng, J. Deng, M. Zhang, S. Song, Z. Liu, and J. Feng, “Design and implementation of ultra-low power I²C-compatible bus based on asynchronous circuit,” in Proc. 6th Int. Conf. on Electronics Technology (ICET), pp. 79–84, 2023, doi: 10.1109/ICET58434.2023.10211915.
[6] S. Yadav and R. Sharma, “Optimized FSM-based I²C design for reduced area and latency,” in Proc. Int. Conf. on VLSI Computing Systems, pp. 189–193, 2022.
[7] V. Kumar and P. Kumar, “I²C-based 7-bit addressed sensor interfacing using VHDL,” Int. J. Adv. Res. Electron. Commun. Eng. (IJARECE), vol. 10, no. 4, pp. 321–325, Apr. 2022.
[8] N. Rathi and M. Saini, “Modular finite state machine design for I²C communication protocol,” IEEE Trans. Educ., vol. 65, no. 2, pp. 111–118, 2021.
[9] R. Patil and M. Deshmukh, “Secure I²C bus protocol design using lightweight encryption,” in Proc. Int. Symp. on Secured Hardware Design, pp. 55–59, 2020.
[10] M. Pradeep Kumar, Design and Implementation of I²C Bus Protocol on Xilinx FPGA, M.Sc. thesis, School of Electronics and Communication Engineering, Bengaluru, India, 2023.
[11] J. Lee, H. Park, and K. Cho, “FPGA-based arbitration for I²C multi-master systems,” IEEE Access, vol. 9, pp. 55412–55421, 2021, doi: 10.1109/ACCESS.2021.3076543.
[12] K. Tanaka, “Glitch filtering in digital communication buses,” Microelectronics Journal, vol. 102, pp. 1–8, Nov. 2020, doi: 10.1016/j.mejo.2020.104885.
[13] Y. Chen, X. Li, and W. Zhang, “FPGA-optimized serial protocol design for embedded systems,” ACM Trans. Design Autom. Electron. Syst. (TODAES), vol. 27, no. 5, pp. 1–23, Sept. 2022, doi: 10.1145/3529400.
[14] L. Singh and P. Patel, “Low-latency communication in FPGA-based I²C systems,” Int. J. Circuits and Systems, vol. 12, no. 3, pp. 145–152, 2021.
[15] D. Gomez, F. Ortega, and R. Perez, “Debouncing techniques for high-speed digital buses,” in Proc. Design, Automation and Test in Europe Conf. (DATE), pp. 621–626, 2022.