Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Asynchronous FIFO Design and Integration with the I2C Protocol for Efficient Data Transfer in Complex Digital Systems

G. Munirathnam[1], Y. Murali mohan babu[2]

[1]*Research Scholar, Department of Electronics and Communication Engineering, JNTUA, Ananthapuramu, A.P, India, munirathnam.ece@jntua.ac.in*
[2]*Professor, Department of Electronics and Communication Engineering, NBKRIST, Vidyanagar, Affiliated to JNTUA, Ananthapuramu, A.P, India*

This paper presents the design and simulation of an integrated asynchronous FIFO interfaced with the I2C protocol. The asynchronous FIFO efficiently manages data transfer between asynchronous clock domains, while the I2C protocol ensures reliable communication with peripheral devices. Verilog code is used to implement the design, focusing on seamless interfacing and accurate timing constraints. The simulation results validate the correctness of data transfer, demonstrating error-free operation under various scenarios. This work highlights the advantages of using asynchronous FIFOs for enhanced performance and reduced latency in I2C-based systems, making it a viable solution for complex digital applications requiring robust and efficient data handling across different clock domains.

Keywords: Functional verification, Timing errors, 12C Protocol, Asynchronous FIFO.

1. Introduction

The Inter-Integrated Circuit (I2C) protocol is a widely used, simple, and efficient protocol designed for short-distance, intra-board communication between microcontrollers and peripheral devices. Developed by Philips Semiconductor in the early 1980s, I2C uses only two bidirectional open-drain lines: Serial Data Line (SDA) and Serial Clock Line (SCL), facilitating communication between multiple devices on the same bus. I2C supports multiple masters and slaves, enabling a master device to control various peripherals like sensors, displays, EEPROMs, and ADCs. Each device on the I2C bus is uniquely identified by a 7-bit or 10-bit address, allowing for up to 128 or 1024 devices, respectively [1]. The protocol operates in three standard speed modes: Standard Mode (up to 100 kbps), Fast Mode (up to 400 kbps), and High-Speed Mode (up to 3.4 Mbps). Communication is initiated by the master

device, which generates a start condition followed by the target device's address and a read/write bit. The addressed slave responds with an acknowledgment (ACK), and data transfer continues in bytes, each followed by an ACK(or)NACK (negative ack).

I2C's simplicity, low pin count, and ability to manage multiple devices on a single bus make it ideal for embedded systems, particularly in applications requiring communication between a central controller and multiple peripherals. Its widespread adoption ensures compatibility and ease of integration across various platforms and devices.

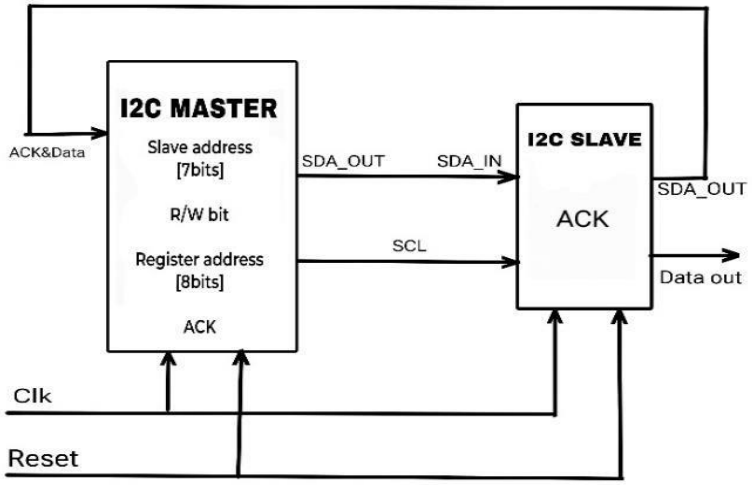


Fig 1: I2C System Architecture

An Asynchronous First-In-First-Out (FIFO) is a critical component in digital systems for managing data between asynchronous clock domains. Unlike synchronous FIFOs that synchronize data transfers using common clock signals, asynchronous FIFOs handle data transfers between independent clock domains without requiring a shared clock. This makes them essential in designs where different modules operate at distinct clock frequencies or have asynchronous timing characteristics.

The key feature of an asynchronous FIFO is its ability to handle data transfers asynchronously, ensuring that data integrity is maintained without causing issues like data loss or corruption due to clock domain mismatches. This is achieved through careful design with dual-clock synchronizers at the input and output interfaces, allowing data to be safely transferred across clock domains.

Typically, an asynchronous FIFO comprises read and write pointers, data storage elements, and control logic. The control logic manages data movement, addressing, and flow control to ensure proper operation under asynchronous conditions. When data is written into the FIFO, it is stored in a buffer until the read side is ready to retrieve it, following the FIFO principle of first-in-first-out data access. Asynchronous FIFOs find extensive use in various applications, including communication interfaces, multi-core processors, digital signal processing, and asynchronous circuit designs. Their ability to bridge different clock domains while maintaining data integrity makes them indispensable in modern digital system designs.

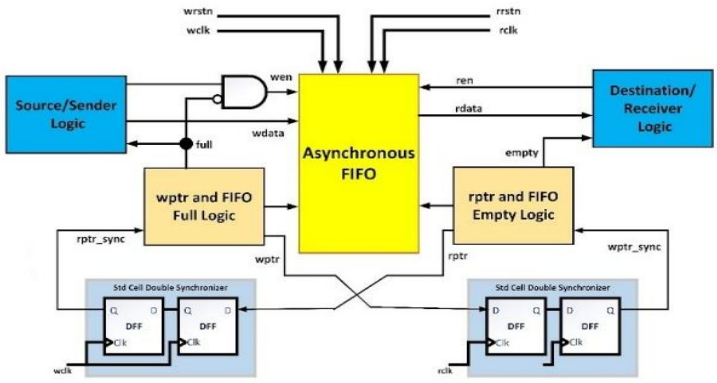


Fig 2: Asynchronous FIFO

Integrating an asynchronous FIFO with the I2C protocol is essential to manage data transfers between devices operating in different clock domains. Since I2C communication is typically asynchronous and can involve devices with varying clock frequencies, an asynchronous FIFO ensures smooth and reliable data handling. It acts as a buffer, allowing data to be transferred between the I2C bus and the internal system at a pace that suits both the sender and receiver, preventing data loss or corruption due to timing mismatches. This integration ensures efficient and synchronized data flow in complex digital systems utilizing the I2C communication protocol [2] [13].

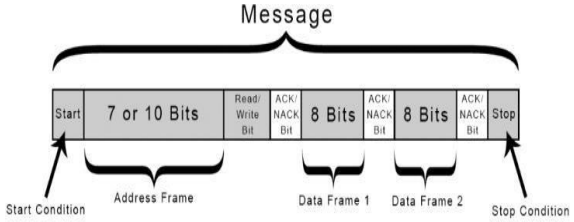


Fig 3: I2C data frame

Integrating asynchronous FIFO with the I2C protocol offers several advantages, including decoupling clock domains for seamless data transfer, ensuring data integrity, enhancing overall system performance, and providing scalability and flexibility for varying data rates. This integration simplifies the design of complex systems by handling asynchronous data transfer internally, making it ideal for embedded systems, consumer electronics, automotive systems, industrial automation, medical devices, communication systems, and digital signal processing applications. By enabling reliable communication between components operating at different clock speeds, it supports robust and efficient data handling across diverse fields.

2. PROPOSED SYSTEM FOR I2C PROTOCOL:

In the existing I2C protocol, the transmission speed is relatively slow, and it is primarily utilized for connecting various devices. If an error occurs during I2C operations, identifying

the root cause is crucial.

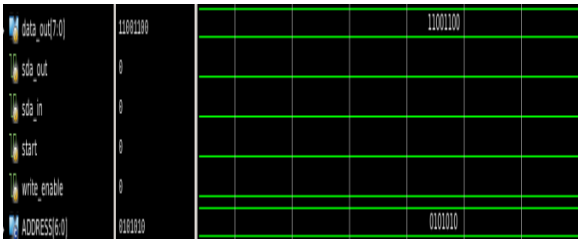


Fig 4: Simulation of Write and Read operation

The current approach is focused on verifying various timing parameter constraints during I2C operations using a Verilog module. This verification process ensures that the timing requirements specified by the I2C protocol are met, which is essential for reliable communication between devices. By rigorously checking these timing parameters, potential issues can be detected and addressed early, ensuring robust and error-free I2C communication.

The fig 3 shows that the data frame in I2C is always 8 bits long and the most significant bit will be sent first. To verify the data transmission each data frame will be followed by ACK/NACK bit. To verify the I2C protocol operations by simulation, we focus on reading and writing values between the Master and Slave devices. Initially, the in out signals for both SDA and SCL lines must be assigned. The 7-bit address of the slave is then specified by writing the address onto the SDA line. In the simulation, a value of 7'b0101010 is used as the input to be stored in a register. The simulation assigns an 8-bit data value of 8'b11001100 for testing purposes. To read from the slave, the desired address is written with the last bit set to 1 instead of 0, maintaining the same address value on the SDA line [3] [12]. The direction bit, which indicates read or write, must follow the slave address per the I2C 7-bit address format. Successfully, the data 8'b11001100 appears on the SDA line during reading, confirming that both read and write operations were executed correctly.

Write/Read operation:

Write operation: Master writes START, address, data, and the slave send ACKs only in write operation.

Read operation: Master writes START, address. Slave sends data, and master sends ACK/NACK.

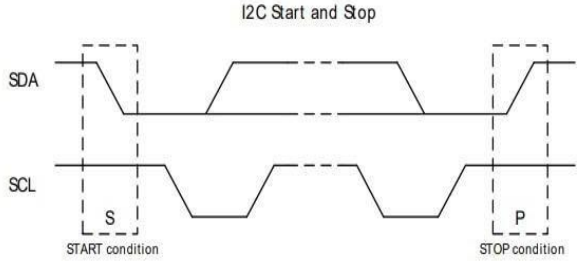
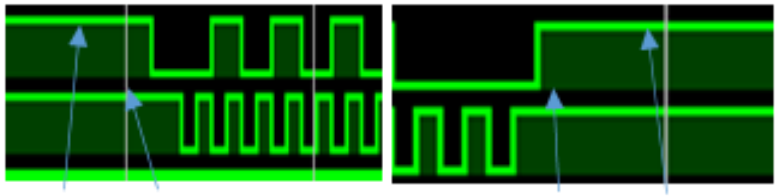


Fig 5: Start and Stop Condition

Start/Stop condition:



SDA SCL SCL SDA

Fig 6.1 Start condition 6.2 Stop condition

Each I2C command initiated by the master device involves two key conditions: a START condition and a STOP condition. Both conditions require the SCL (serial clock) line to be high. A START condition occurs when the SDA (serial data) line transitions from high to low, while a STOP condition occurs when the SDA line transitions from low to high. The simulation depicted in Figures 6.1 and 6.2 verifies these START and STOP conditions as outlined in Figure 6.

The timing characteristics of the design are defined using timing constraints. These constraints fall into two main categories. The first category is setup (long-path) constraints, which specify the minimum duration for which a data input signal must remain stable before the clock edge of each storage element, such as a flip-flop or latch. The second category is hold-time (short-path) constraints, which define the minimum duration for which a data input signal must remain stable after the clock edge at each storage element [4] [11].

Table 1: simulation statistics of Timing constraint

Parameters	Timing constraint	Functional verification	Timing with functional verification
REAL time	10s	13s	25s
CPU time	2.52s	2.51s	3.41s
Memory	6021.96 Mb	602.192 Mb	602.220 Mb

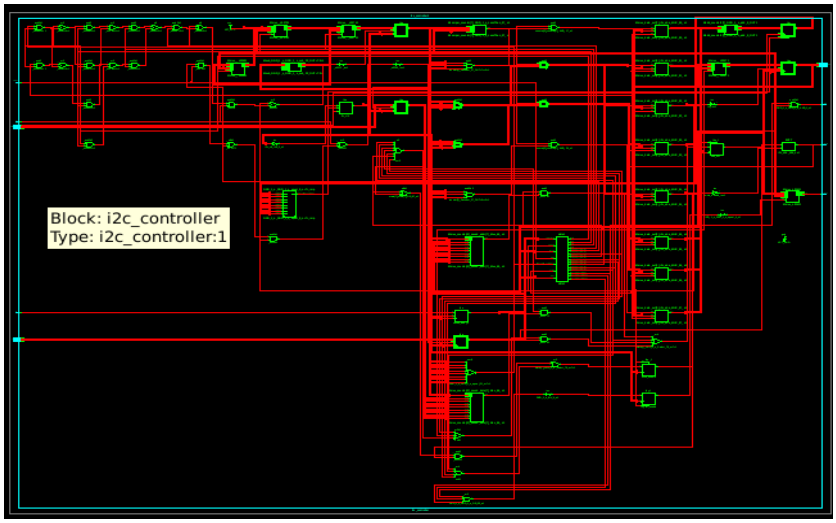


Fig 7. RTL schematic

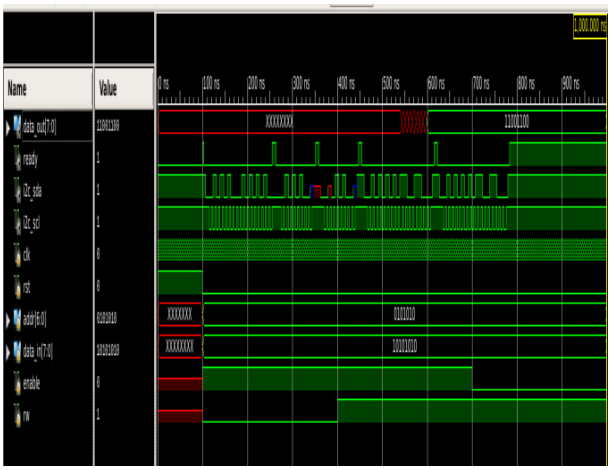
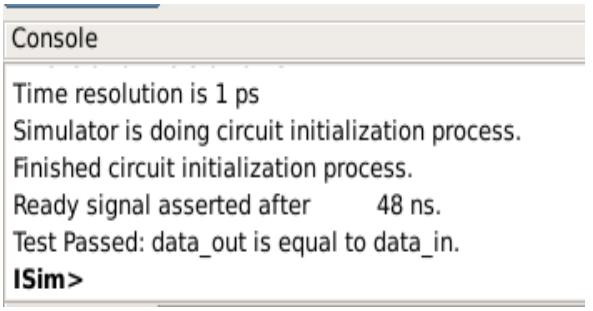


Fig 8: simulation results

I2C communication protocol is designed, and its function has verified using Verilog HDL using the Xilinx ISE tool. The timing constraint along with functional verification enhances the overall quality of the design and minimizes the risk of system failures.



Elapsed time is 48 ns data_out is equal to data_in refers to the data input (assumed value) given at slave device is transferred successfully in I2C device.

3. INTEGRATION OF ASYNCHRONOUS FIFO WITH I2C PROTOCOL:

Efficient data transfer is crucial in complex digital systems, especially when dealing with multiple clock domains [5] [9]. Asynchronous FIFO (First-In-First-Out) buffers and the I2C (Inter-Integrated Circuit) protocol are powerful tools that can be integrated to achieve this goal. This article delves into the design and integration of asynchronous FIFOs with the I2C protocol, outlining key concepts, design considerations, and practical implementation steps. In a sensor hub system, where multiple sensors operating at different speeds feed data to a central processor, an asynchronous FIFO can buffer this data and transfer it efficiently over the I2C bus. This setup ensures smooth data flow, avoiding bottlenecks and preserving data integrity despite the asynchronous nature of sensor data generation and processing.

Efficient and reliable data transfer in complex digital systems, especially those with multiple clock domains, presents significant challenges. Asynchronous FIFOs (First-In-First-Out

buffers) offer a potential solution by decoupling read and write operations, allowing data to flow smoothly between different clock domains. However, integrating asynchronous FIFOs with widely used communication protocols like I2C (Inter-Integrated Circuit) introduces additional complexities that must be addressed to achieve optimal performance. This research aims to develop a robust methodology for designing and integrating asynchronous FIFOs with the I2C protocol, ensuring efficient, reliable, and high-performance data transfer in complex digital systems.

System Bus Represents the data and control signals of your system. Asynchronous FIFO Acts as a buffer [6] between the system bus and the I2C interface. Receives data from the system bus for transmission over I2C. Manages read and write operations based on control signals. Control Logic Inside the asynchronous FIFO, manages the flow of data and control signals. I2C Interface Implements the I2C protocol for communication with external devices. Receives formatted data from the asynchronous FIFO [7] and transmits the formatted data over the I2C bus.

The below diagram illustrates the flow of data from the system bus into the asynchronous FIFO, where it is buffered and managed by the control logic. When data is ready for transmission over I2C, it is read from the FIFO and passed to the I2C interface, which handles the protocol-specific communication with external devices [8].

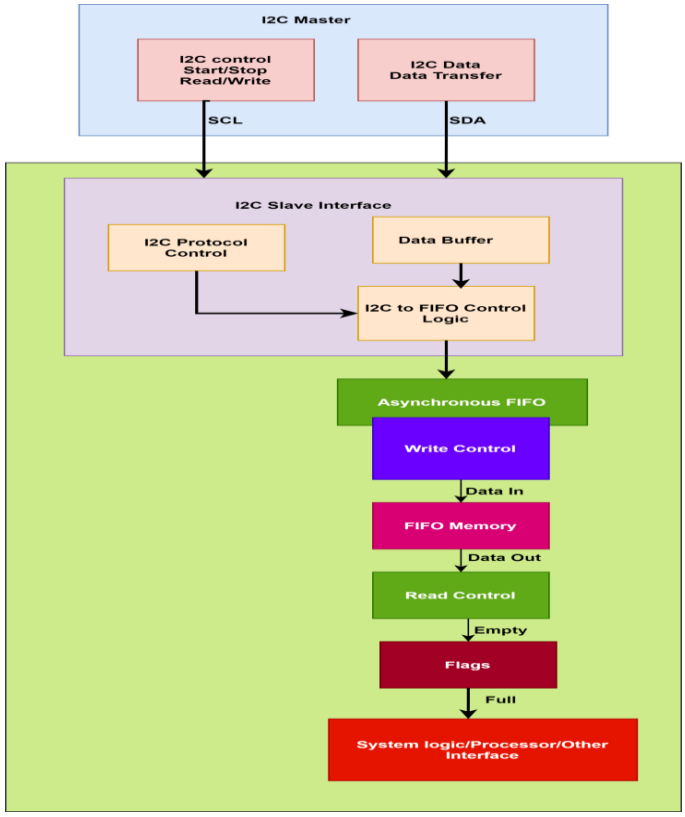


Fig 9: Asynchronous FIFO and I2C Interfacing

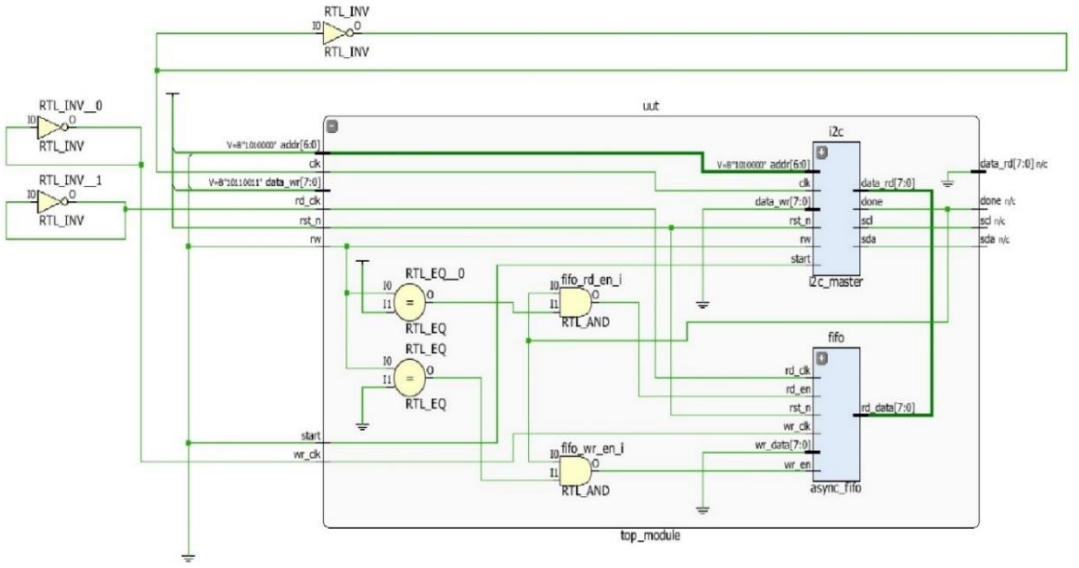


Fig 10: RTL Schematic of Asynchronous FIFO and I2C Interfacing

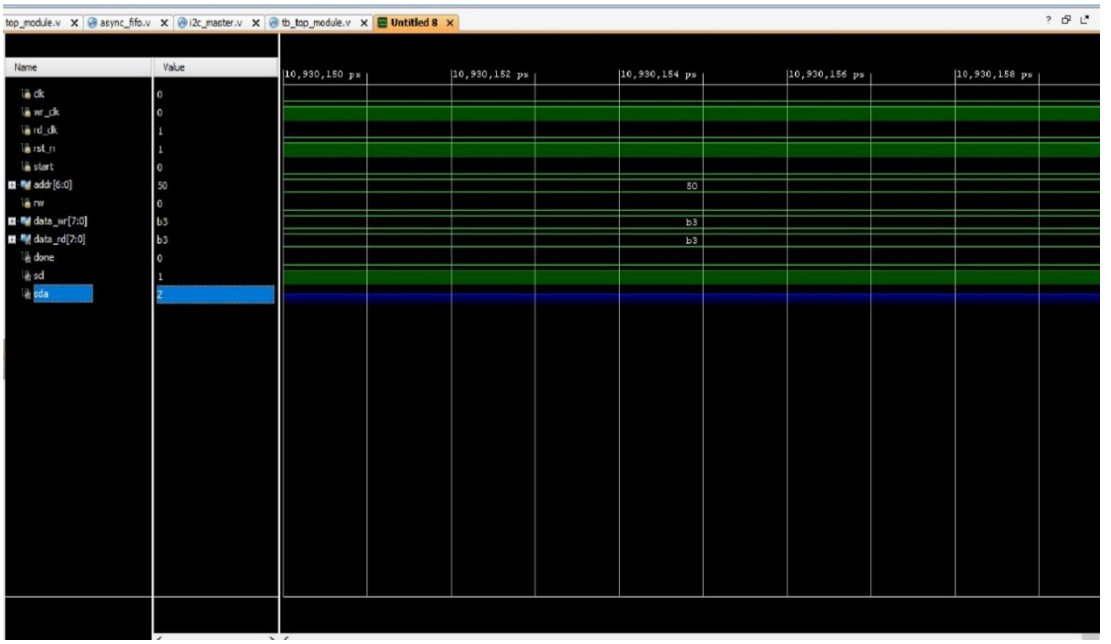


Fig 11: Simulation of Asynchronous FIFO and I2C Interfacing

The successful interfacing of an asynchronous FIFO with the I2C protocol using Verilog demonstrates robust data handling across different clock domains. The simulation results verify that data is correctly buffered and transferred between the master and slave devices without data loss or corruption. The asynchronous FIFO ensures smooth data flow despite differing clock speeds, while the I2C protocol manages device communication effectively [14]

[10]. This integration enhances the system's reliability and performance, proving the feasibility of asynchronous data transfer in complex digital systems and highlighting the advantages of combining these technologies for efficient and error-free communication.

4. CONCLUSION:

The integration of an asynchronous FIFO with the I2C protocol using Verilog proves to be an effective solution for managing data transfers across different clock domains. The successful simulation results demonstrate the system's capability to maintain data integrity and ensure reliable communication between master and slave devices. This setup allows for efficient data buffering and smooth data flow, even with varying clock speeds. The combination of asynchronous FIFO and I2C protocol enhances overall system performance and reliability, making it suitable for complex digital systems requiring robust and error-free communication.

References

1. VLSI implementation of SPI and I2C communication protocols. Available :https: //www.ijariit.com/manuscripts/v6i4/V6I4- 1312.pdf
2. F.Leens, "An Introduction to I2C and SPI Protocols", IEEE Instrumentation & Measurement Magazine, pp. 8- 13,February 2009.
3. Philips Semiconductors, "The I2C-Bus Specifications", version 2.1, January 2000.
4. Basics of I2C: The I2C protocol – Texas Instruments India.
5. Clifford E. Cummings and Peter Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers, March 2002, Section TB2, 3rd paper. Also available at www.sunburst-design.com/papers.
6. S. S. Appleton, S. V. Morton, and M. J. Liebelt, "Two phase asynchronous pipeline control," in Proc. 3rd Int. Symp. Advanced Research in Asynchronous Circuits and Systems, Apr. 1997, pp. 12–21.
7. J. Sparsø and S. Furber, Principles of Asynchronous Circuit Design—A Systems Perspective. Boston, MA, USA: Kluwer Acad. Publ., 2001.
8. S. M. Nowick and M. Singh, "Asynchronous design—Part 1: Overview and recent advances," IEEE Design Test, vol. 32, no. 3, pp. 5–18, Jun. 2015.
9. NXP.I2C bus specification and user manual. (2021). Accessed: Oct. 01, 2021.[Online]. Available: I2C-bus
10. specification and user manual (nxp.com) https://www.researchgate.net/publication/ 275771333 _Design_of_I2C_Single_Master_Using_Verilog. https://www.researchgate.net/publication/ 271323622_ Functional_verification_of_I2C_core_using_System Verilog.
11. An Implementation of I2C Slave Interface using Verilog HDL Available: https://www.ijmer.com/papers/Vol5_Issue3/Versi on-3/H0503_03-5560.pd
12. Module Implementation and Simulation of Timing Constraint Check Function of I2C Protocol Using Verilog.
13. Frank Vahid (2010). Digital Design with RTL Design, Verilog and VHDL (2nd ed.). John Wiley and Sons. p. 247. ISBN 978-0-470-5310822
14. https://www.researchgate.net/publication/332142672_Design_of_I2C_Protocol
15. UM10204, "I2C-bus specification and user manual", Rev.6, 4th April 2014.