10.48047/jocaaa.2024.33.08.61

DESIGN AND IMPLEMENTATION OF ASYNCHRONOUS FIFO WITH DUAL CLOCK DOMAIN IN VERILOG

Mr.B.AJANTHA REDDY [(1)] , Dr.A.RANGANAYAKULU[(2)] , ATCHUTA RAKESH[(3),] DUGGI RAMI REDDY[(4),] SREERAMULA SURESH[(5),] CHALLA UDAY KIRAN [(6)]

[1,2] *Faculty-ECE Department Krishna Chaithanya Institute of Technology & Sciences,Markapur, AP, India.*

[3,4,5,6] *Student ECE Department,Krishna Chaithanya Institute of Technology & Sciences, Markapur, AP, India.*

Abstract: This paper presents the design and implementation of an Asynchronous FIFO (First-In-First-Out) buffer in Verilog, where the write and read operations are controlled by two independent clock domains. The FIFO buffer accepts 8-bit data inputs and provides 8-bit data outputs. It includes various features such as write enable, read enable, and reset functionality. The core of the design involves managing the read and write pointers using grey code to avoid metastability issues in asynchronous systems, ensuring reliable operation across different clock domains. The FIFO is equipped with status flags, fifo_full and fifo_empty, to indicate when the FIFO buffer is full or empty. The Verilog implementation synchronizes the write and read pointers across clock domains using flip-flops to ensure data integrity and prevent errors due to timing mismatches. The FIFO buffer's performance is verified through testbenches simulating various operating scenarios, demonstrating its reliability in a multi-clock environment

I.INTRODUCTION

.

An Asynchronous FIFO is a type of FIFO buffer where the data is transferred between two clocks or systems that operate asynchronously. This means that the reading and writing operations do not occur under the same clock domain, which is a crucial characteristic for communication between subsystems or components that run on different clock frequencies.It's called asynchronous because the FIFO operates independently of the two systems' clock frequencies. The FIFO can receive data from one system and transmit it to another system without being synchronized to either system's clock. In SPI communication, data transfer happens in a synchronous manner, meaning the data is clocked out from the master and clocked in by the slave using the same clock signal. However, in systems where the master and slave devices operate asynchronously (i.e., their clock signals are not synchronized), managing the flow of data can be problematic. Data might be produced faster than it can be consumed, or vice versa.To solve this issue, an Asynchronous FIFO is often used to temporarily store data between the master and slave. This buffer enables the system to handle differences in data production and consumption rates between the two devices, preventing data loss, underflow, or overflow. In SPI (Serial Peripheral Interface) communication, data is transferred between a master and one or more slave devices. When using **FIFO (First-In-First-Out)** buffers in SPI, it helps to manage the flow of data between the master and slave devices, especially when the devices are operating at different speeds or when there is a mismatch between the rate at which data is produced and consumed. An asynchronous FIFO ensures that data is not lost or corrupted during the transfer, even when there are differences in the data rates or timing between the master and slave devices. The FIFO allows for smooth data transfer without the need for complex handshaking mechanisms between the devices. It effectively decouples the rate at which data is produced from the rate at which data is consumed. By buffering data, the FIFO prevents situations where the master or slave might attempt to read or write data when the buffer is empty or full, reducing the risk of overflow or underflow errors. The **master** device generates the clock (SCK) and initiates communication with one or more slave devices. The master writes data to the MOSI (Master Out, Slave In) line and reads data from the MISO (Master In, Slave Out) line.

10.48047/jocaaa.2024.33.08.61

II EXISTING SYSTEM

RAM Device is used in SPI Protocol
SPI (Serial Peripheral Interface) is a popular communication protocol for transferring data between a master device (such as a microcontroller or processor) and peripheral devices, including **RAM** (Random Access Memory) modules. When using SPI with RAM, the SPI interface allows for the fast and efficient exchange of data between the microcontroller and the RAM, especially for systems that require non-volatile storage or fast data retrieval. When using **SPI with RAM**, the SPI protocol allows a microcontroller to interact with a RAM chip (either SRAM or Flash) by sending commands and data over the SPI bus. The microcontroller writes data to the SPI RAM by sending a series of commands followed by the data over the SPI bus. The microcontroller reads data from the SPI RAM. SPI is a serial communication protocol, meaning it transfers data bit-by-bit over a single line. This can be slower compared to parallel buses, which transfer multiple bits of data simultaneously. When using RAM with SPI, especially for large amounts of data, the serial nature of SPI could become a bottleneck, particularly when working with large data buffers or high-speed memory. Unlike memory systems that use more complex addressing schemes (like those used in parallel buses), SPI requires the master to send specific address information as part of the communication (in the command phase).In large systems, if the memory is significantly large (e.g., multiple megabytes or gigabytes), manually managing memory addressing can add complexity to the software stack and increase overhead.This is especially true for SPI Flash memory, where large memory areas must be addressed and managed using commands that add complexity

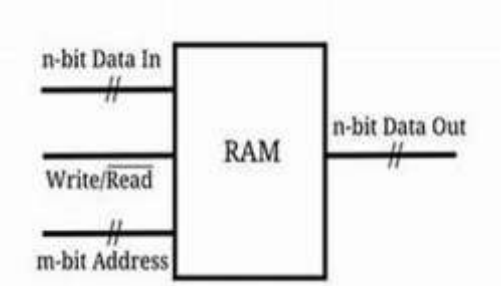


Fig 1 Pin diagram

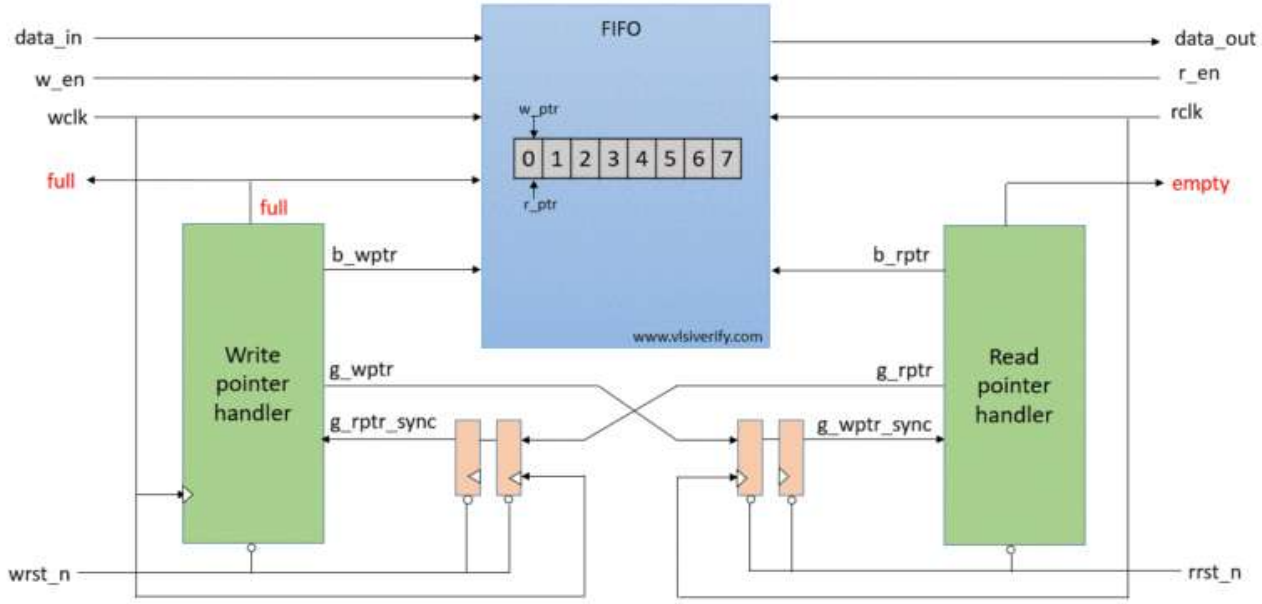
III. PROPOSED METHOD

Design and Implementation of Asynchronous FIFO with Dual Clock Domains in Verilog
An Asynchronous FIFO is a type of FIFO buffer that operates between two systems with different clock domains, meaning the writer and reader of the data do not share the same clock signal. It is commonly used in digital systems, particularly in hardware design, to transfer data safely between components that operate at different clock frequencies or phases.In simpler terms, it ensures that data is transferred from one part of a system (like a processor or sensor) to another (such as memory or a communication interface) even when the producer and consumer of the data are not synchronized.
An Asynchronous FIFO (First-In-First-Out) buffer is commonly used in SPI (Serial Peripheral Interface) systems to facilitate the safe transfer of data between the SPI master and SPI slave when the master and slave operate at different clock frequencies or when data needs to be temporarily stored due to speed mismatches in reading and writing operations.The **producer** writes data into the FIFO at a rate determined by its **write clock** (write_clk).Suppose the producer writes **data 1** at time t0. The **write pointer** will point to the first location in the FIFO, and data 1 will be stored in the first slot.The write pointer is then incremented to point to the next location (slot 2).The producer continues writing data (data 2, data 3, etc.) as long as there is space in the FIFO and the FIFO is not full.The **consumer** reads data from the FIFO at a rate determined by its **read clock** (read_clk).Suppose the consumer reads from the FIFO at time t1. The **read pointer** will point to the first location in the FIFO, and **data 1**

10.48047/jocaaa.2024.33.08.61

will be read out.The read pointer is incremented to the next location (slot 2).The consumer continues reading data at its own rate, as long as the FIFO is not empty. If the producer is writing data faster than the consumer can read, the FIFO will temporarily hold the data in its buffer. The write pointer will move faster than the read pointer.If the consumer is reading data faster than the producer can write, the read pointer will move faster, and eventually, the FIFO will become empty. In this case, the **empty flag** will indicate that no data is available for reading.

.



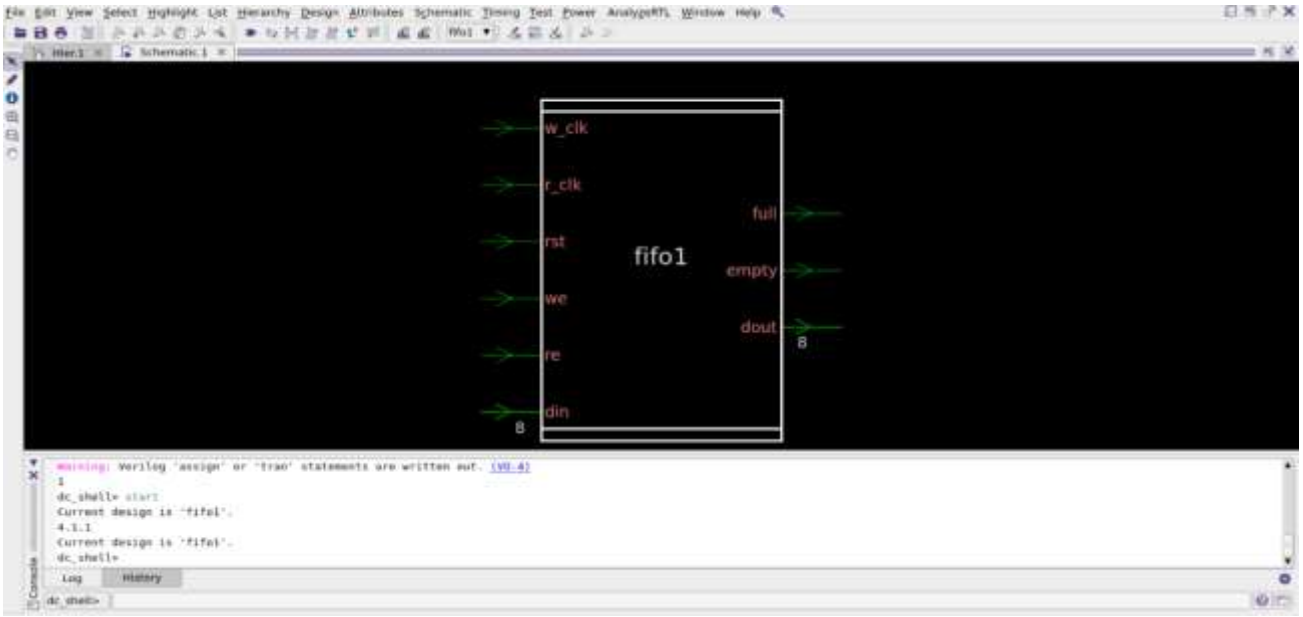
Asynchronous FIFO

10.48047/jocaaa.2024.33.08.61

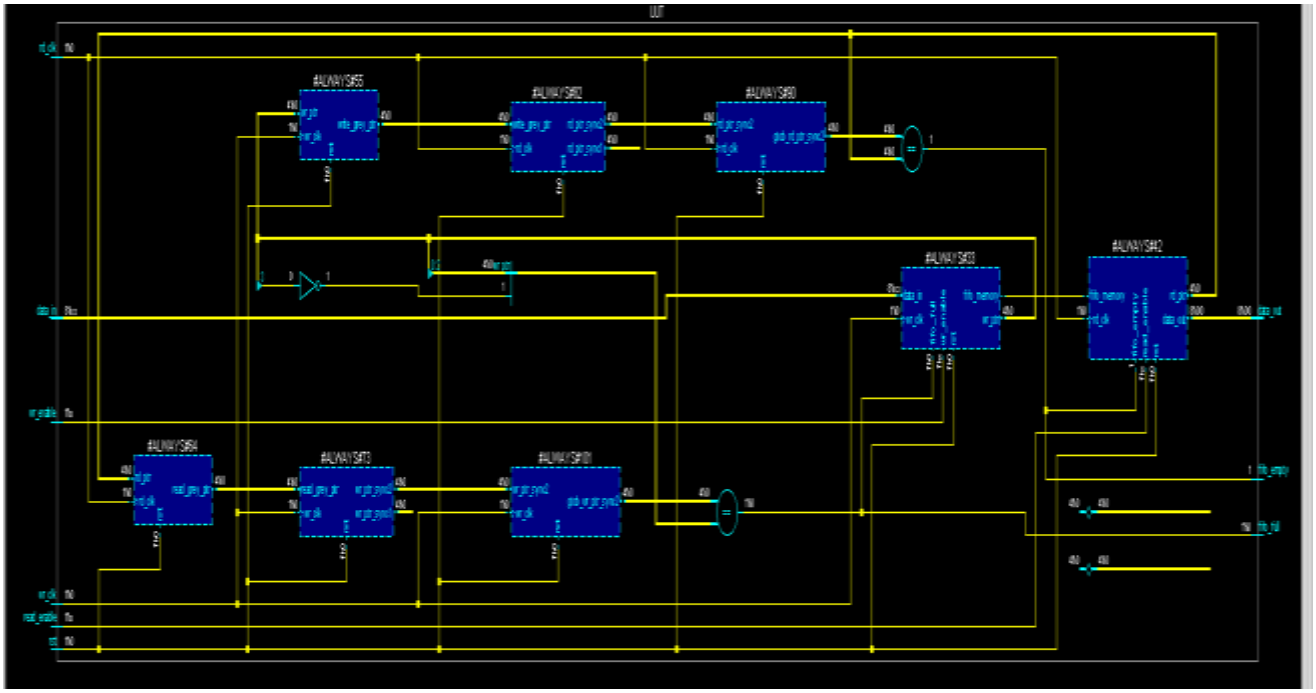
Fig 2 Asynchronous Fifo

IV. RESULTS AND ANALYSIS DISCUSSION

pin diagram



Schematic diagram

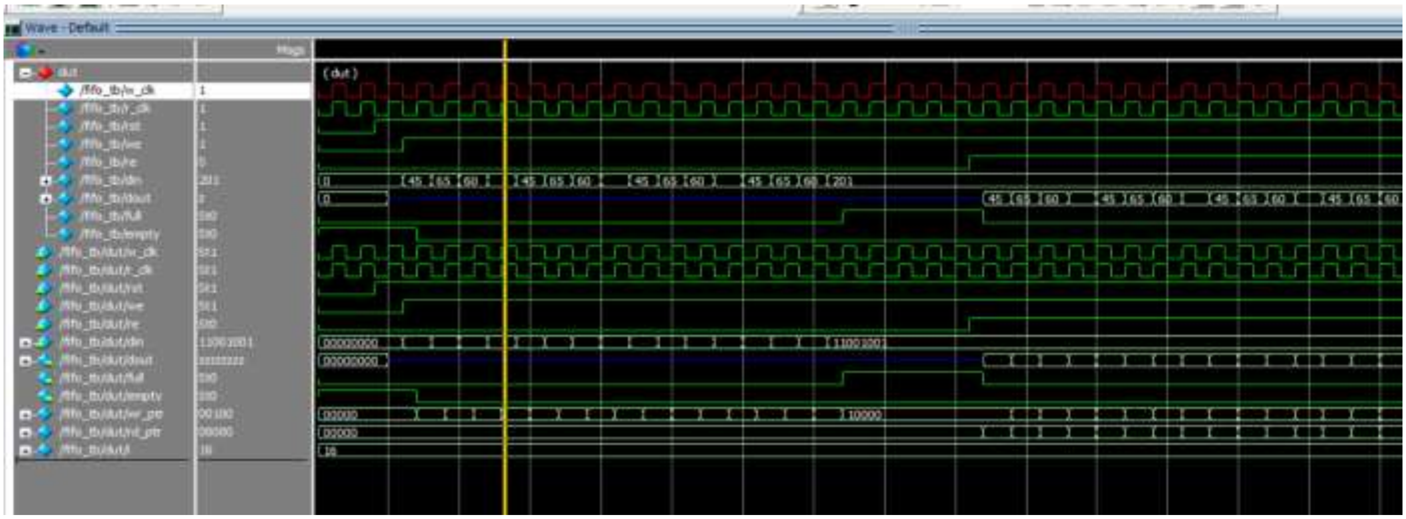


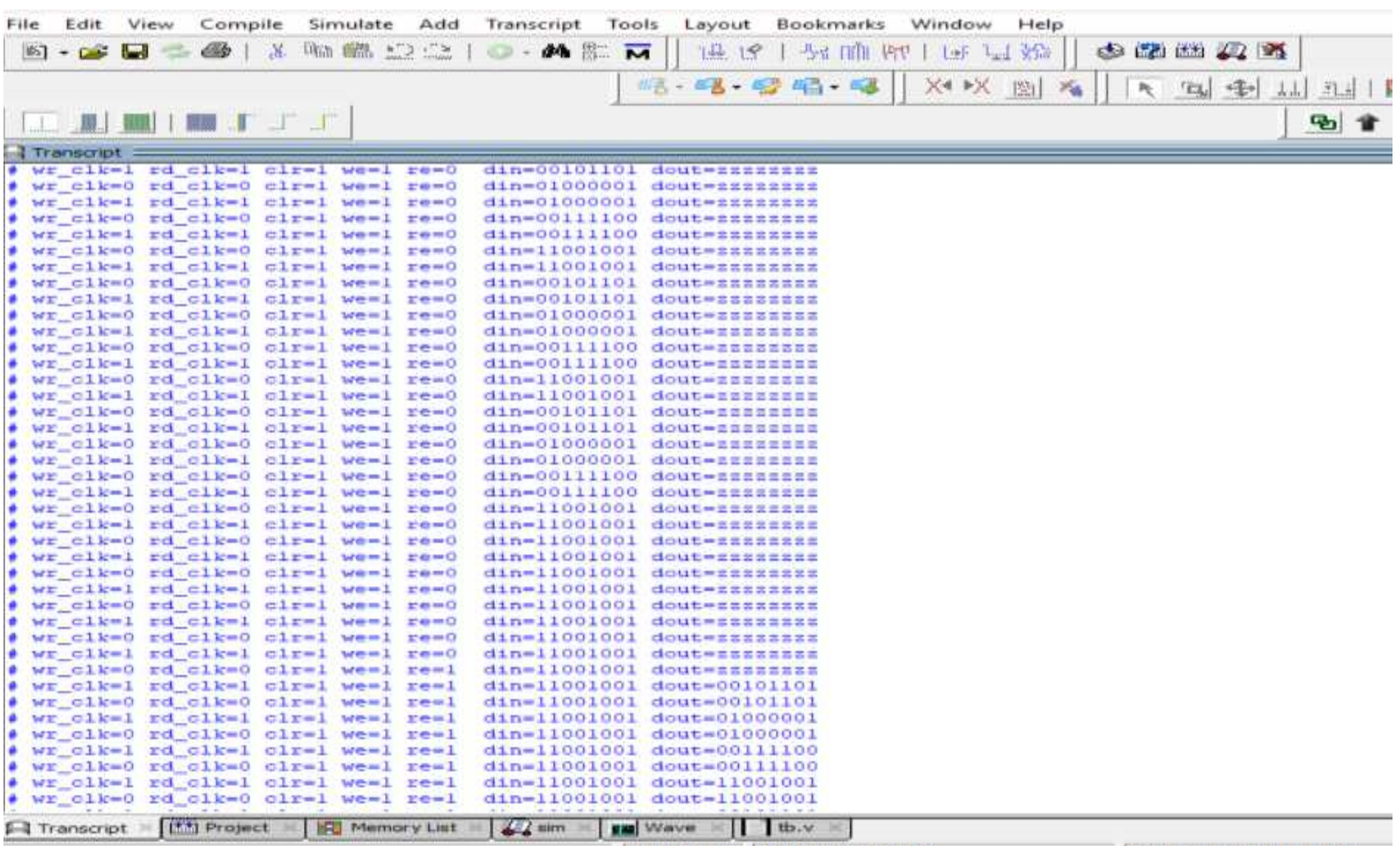
10.48047/jocaaa.2024.33.08.61

waveforms Transcript

V CONCLUSION

FIFOs play a vital role in medical imaging by ensuring efficient and reliable data transfer and processing. Their use has numerous benefits, including improved image quality, increased throughput, and enhanced patient care.





10.48047/jocaaa.2024.33.08.61

Asynchronous FIFOs are essential components for reliable data communication in systems with differing clock domains. Their proper implementation ensures the smooth transfer of data without loss or corruption, though it requires careful design considerations to handle the complexity of clock domain crossing

VI REFERENCES

1. **Digital Design and Computer Architecture** by David Money Harris and Sarah L. Harris

   This book covers various aspects of digital system design, including the concept of FIFOs and their applications in systems like SPI. It is a comprehensive resource for understanding how FIFOs work in different digital communication protocols.

   Link: Digital Design and Computer Architecture

2. **FPGA Prototyping by VHDL Examples** by Pong P. Chu

   This textbook includes practical examples on how to implement various types of FIFOs, including asynchronous FIFOs, on FPGA platforms. The SPI protocol and other communication systems are also discussed in this context.

   Link: FPGA Prototyping by VHDL Examples

3. **Advanced Digital Design with the Verilog HDL** by Michael D. Ciletti

   This book discusses the design and implementation of FIFOs in Verilog, including asynchronous FIFOs used in communication protocols such as SPI.

   Link: Advanced Digital Design with Verilog HDL

4. **Design and Implementation of a Dual Clock FIFO with Application to SPI Communication"**
   a. This research paper discusses the design of asynchronous FIFOs, particularly focusing on dual-clock FIFOs for SPI communication. It explains how to handle clock domain crossing issues using techniques like synchronizers in SPI systems.

   b. Available in IEEE Xplore: IEEE Xplore Link

5. **"Efficient Dual-Clock FIFO Design and Implementation for SPI Interface"**

   This paper delves into the implementation of dual-clock FIFOs, which are typically asynchronous, and their applications in SPI systems. It offers insights into handling data transfer between different clock domains and ensuring high-speed communication.

   Available in IEEE Xplore: IEEE Xplore Link

6. **"A Study of FIFO Implementation in Asynchronous Systems"**

10.48047/jocaaa.2024.33.08.61

a. This paper discusses the concept of asynchronous FIFOs, comparing them to synchronous FIFOs, and illustrates their use in various communication protocols, including SPI. It provides insights into how clock domain crossing and timing issues are managed in asynchronous FIFOs.

b. Available in ScienceDirect: ScienceDirect Link