Design Specifications for ASYNCHRONOUS FIFO

Sandeep Goud Abbagouni
Department of Elect & Computer Eng.
Portland State University

Abhishek Kumar Reddy Musku
Department of Elect & Computer Eng.
Portland State University

Saiteja Gali
Department of Elect & Computer Eng.
Portland State University

***Abstract*—People often use FIFOs to safely pass data from one asynchronous clock domain to another. Using a FIFO to transfer data from one clock domain to another necessitates multi-asynchronous clock design techniques. This paper will describe a method for designing, synthesizing, and analyzing a safe FIFO between different clocks. Domains synchronize gray code pointers into a distinct clock domain before evaluating ”FIFO full” or ”FIFO empty” conditions.**

I. INTRODUCTION

A. Purpose

The objective of the paper is to establish the specifications for an Asynchronous FIFO (First-In-First-Out) memory queue. The purpose of this Asynchronous FIFO is to securely transfer data between clock domains in digital systems that include components operating at separate clock frequencies.

B. Product Scope

The document’s scope includes the asynchronous FIFO’s design considerations, functionality, and implementation details. This entails handling full and empty scenarios using gray code counters for managing pointers and ensuring data integrity in the asynchronous clock domain transfers.

II. OVERALL DESCRIPTION

A. Product prospect

An asynchronous FIFO buffer is a specialized data buffer that follows the First In, First Out (FIFO) principle.The system enables the storage and retrieval of data between two subsystems that operate on distinct time domains. It serves as a crucial element within a broader system that manages timing inconsistencies and data movement among these subsystems. This product is essential in systems that involve both sender and receiver clock frequencies.The function operates independently, guaranteeing data integrity without requiring synchronization with an operational clock.

In the design of a synchronous FIFO (a FIFO where writes to and reads from the buffer happen in the same clock domain), one approach is to keep track of the number of write and read operations on the FIFO buffer. When there is a write operation without a corresponding read operation, we use this count to increase the buffer’s fill value, decrease it when a read operation occurs without a write operation, and maintain the fill value when there are no read or write operations or simultaneous read and write operations. When the FIFO counter reaches a predefined maximum value, it becomes full, and when the FIFO counter is at zero, it is considered empty.

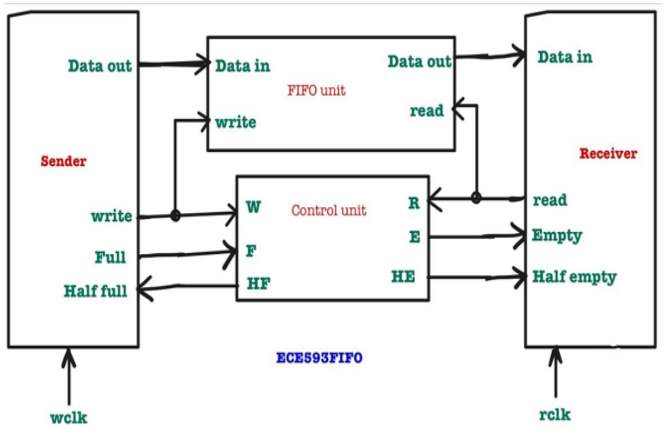

Fig. 1. Level Block Diagram of the Design System

The asynchronous FIFO serves as a crucial element that connects two separate subsystems, namely the sender and the receiver.Every subsystem operates independently on its own clock domain, while the FIFO acts as a mediator for transmitting data. Typically, the Level Block Diagram of the Design System illustrates the FIFO’s connection to both subsystems, with arrows indicating the direction of data flow. The FIFO manages the queuing and retrieval processes of data by depicting the control signals for write, read, full, half-full, empty, and half-empty operations.This component is crucial for applications such as telecommunications, computer architecture, and digital signal processing, where there are different operational speeds amongst system components. The asynchronous FIFO is an essential component for ensuring efficient and reliable data transport without any errors.

B. Product Functions

The functions of the Asynchronous FIFO are as follows:-

- **Transferring Data Between Clock Domains:** The FIFO is used to safely transfer data between asynchronous clock domains.
- **Handling Full and Empty States:** Developing logic to detect and manage FIFO full and empty states.
- **Implementation of Gray Code Counters:** Using Gray code counters for pointer creation to assure safe data synchronization.

- **Managing FIFO Pointers in Synchronous and Asynchronous contexts:** Managing FIFO pointers, including the complexity involved in asynchronous contexts.
- **Safe Data Synchronization:** Maintaining data integrity when moving across clock domains.
- **Scalability and Flexibility:** Ensuring that the FIFO architecture can be scaled and configured to multiple sizes and configurations.

III. Signal Description

The signals used in the design are as follows:

A. *Inputs*

- **wdata**: Data input for writing.
- **winc**: Write increment signal.
- **wclk**: Write clock signal.
- **wrst_n**: Active-low write reset signal.
- **rinc**: Read increment signal.
- **rclk**: Read clock signal.
- **rrst_n**: Active-low read reset signal.

B. *Outputs*

- **rdata**: Data output for reading.
- **wfull**: Write full indicator.
- **rempty**: Read empty indicator.

C. *Internal Signals*

- **waddr**: Address signal for writing.
- **raddr**: Address signal for reading.
- **wptr**: Pointer signal for writing.
- **rptr**: Pointer signal for reading.
- **wq2_rptr**: Signal for the write queue to read pointer.
- **rq2_wptr**: Signal for the read queue to write pointer.

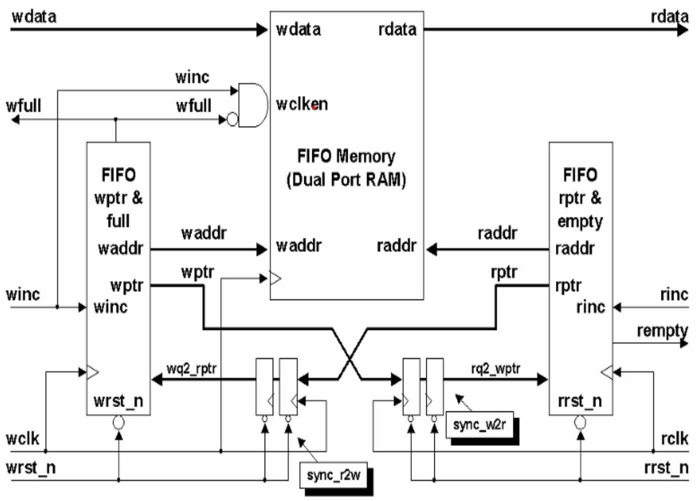

Fig. 2. The block Diagram of the FIFO

IV. Design and Implementation Constraints

The FIFO was built with the assumption that the sender operates at a frequency of 250 MHz and the receiver operates at a frequency of 100 MHz, ensuring that data is produced and consumed at different rates. In addition, a maximum write burst size of 150 is set, with two idle cycles inserted between successive reads and no idle cycles between writes. This has an impact on the overall flow of data and the management of buffers.

- Time required to write one data item : = **1/250 MHz = 4 nsec.**
- So, for every **4nsec**, the sender is going to write one data in the burst.
- Time required to write all the data in the burst = **150 * 4 nsec. = 600 nsec.**
- Time required to read one data item = **3 * 1/100MHz = 30 nsec.**
- So, for every **30 nsec,** the receiver is going to read one data in the burst.
- So, in a period of **600 nsec, 150** number of data items can be written.
- The no. of data items can be read in a period of 600 nsec = **600ns/30ns = 20**
- The remaining no. of bytes to be stored in the FIFO = **150 – 20 = 130.**
- So, the FIFO which has to be in this scenario must be capable of storing **130** data items.

A. *Assumptions and Dependencies*

The following assumptions are made in design considerations for the FIFO. The depth of the FIFO is calculated based on the following parameters.

- Sender Clock Frequency = 250 Mhz
- Rceiver Clock Frequency = 100 Mhz
- Maximum write Burst size = 150
- Number of idle cycles between successive reads = 2
- Number. of idle cycles between successive writes = 0

V. Product Features

A. *FIFO memory*

- Computers use FIFO memory hardware, a type of data buffering device, to manage and temporarily store data. The system functions based on the principle of processing or outputting the initial data that enters the queue first. Imagine a pipe where objects enter at one end and exit at the other end in the exact same sequence.
- This approach guarantees a seamless and organized transmission of information, which is particularly beneficial in situations where data needs to be processed in a sequential manner, such as streaming audio or video, telecommunications, and some real-time computer scenarios.
- Several methods, such as hardware registers within CPUs, dedicated memory chips, or as components of more comprehensive memory management systems, can implement

FIFO memory. One of FIFO memory's key benefits is its simplicity, allowing for fast and efficient data handling without the need for complex algorithms to control the order of data processing.

- However, FIFO memory's simplicity makes it less suitable for non-sequential data retrieval or processing scenarios. In such circumstances, alternative memory buffers such as stacks or random access memory (RAM) are more suitable.
- In systems where the sequence of data processing is critical, FIFO memory is essential for maintaining data integrity and data order.

B. Gray Counter

- Gray code counters are particularly advantageous in First-In-First-Out (FIFO) memory systems due to their distinct characteristics. FIFO systems are data structures in which the initial element added to the queue is the first to be removed. Having dependable and effective methods for monitoring the read and write positions within the memory buffer is crucial in these systems. In this situation, gray code counters are useful.
- A gray code is a binary numeral system where two successive values differ by only one bit. Digital circuits require the inclusion of the single-bit change functionality to reduce errors, especially when many bits are changing simultaneously. Gray code counters are employed to monitor the read and write pointers in FIFO systems due to their significant reduction in the likelihood of erroneous data read and write operations resulting from the asynchronous update of multiple bits.
- The user's text consists of a single bullet point. In a typical binary counter, transitioning from '0111' (equivalent to 7 in decimal) to '1000' (equivalent to 8 in decimal) changes all four bits. This simultaneous change of bits might potentially lead to timing issues, as each bit does not transition at the exact same moment. In contrast, a Gray code counter modifies only a single bit at each step, leading to a more reliable and resilient operation. This particular change from one bit to another is particularly advantageous in high-speed First-In, First-Out (FIFO) systems, where mistakes in timing could lead to the distortion or loss of data.
- Gray code counters enhance the integrity and dependability of FIFO systems by ensuring smoother and more accurate transitions between states.

C. FIFO full & Empty Condition

- The full and empty states of a FIFO are crucial for its efficient functioning. The 'full' state indicates that the FIFO buffer has reached its maximum storage capacity and cannot accommodate additional data unless it reads or removes some of the current data. It is imperative to do this in order to prevent overwriting or loss of data. In order to monitor the positions for reading and writing data in the FIFO, hardware often utilizes a collection of pointers or counters. The FIFO is considered full when the write pointer reaches the read pointer.
- The 'empty' state indicates that the FIFO buffer does not contain any data elements available for reading. This phenomenon occurs when the reading position indicator exceeds the writing position indicator. In this situation, attempting to read from the FIFO could result in underflow, potentially leading to the reading of invalid or outdated data.
- To handle these situations, FIFOs often incorporate control logic or status flags. The FIFO (First-In-First-Out) raises a complete flag when it's fully filled, and an empty flag when it's completely empty. Other hardware components can use these flags to determine when to stop writing to or reading from the FIFO. Timing and data flow control ensure data integrity and smooth operation in critical systems like network routers, digital signal processing data buffers, and inter-processor communication in multi-core systems.

D. Synchronizers

- Synchronizers are crucial in the design of asynchronous FIFOs, which are memory queues that operate in several time domains. An asynchronous First-In-First-Out (FIFO) buffer stores data using one clock domain and retrieves it from the buffer using another clock domain. Due to the difference in clock domains, timing issues such as metastability might arise when the data signal becomes locked in an uncertain state during the transition period.
- We employ synchronizers to alleviate this issue. A common synchronizer consists of interconnected flip-flops. The initial flip-flop captures an asynchronous signal at the receiving clock domain's edge. However, it is possible that this signal may become metastable. Passing the signal through multiple flip-flop stages significantly reduces the possibility of metastability affecting the system. Each flip-flop stage allows more time for the signal to stabilize into a steady state.
- The selection and design of synchronizers are crucial, as they must find a balance between reducing the risks of metastability and the delay caused by the synchronizer chain. The number of stages in a synchronizer is often determined by a balance between the frequency of the clocks used and the desired level of system performance and dependability.

E. Summary

The specification of the asynchronous FIFO fully defines its structure and operation. This encompasses specific information such as the width of the data bus, the capacity of the FIFO, and the control signals employed for read, write, full, half full, empty, and half empty operations. The specification provides detailed information and the process of writing and reading data from the FIFO, and the mechanism for updating the empty and full flags.