HIGH LEVEL DESIGN SPECIFICATION (HLDS)

for

ASYNCHRONOUS FIFO

Prepared by

Kumar Durga Manohar Karna, Mohammeed Abbas Shaik, Nikhitha Vadnala and Nivedita Boyina

ECE-593 Fundamentals of Pre-Silicon Validation

Venkatesh Patil

PORTLAND STATE UNIVERSITY

Contents

1. Introduction……3

1.1 Purpose……3

1.2 Intended Audience……3

1.3 Product Scope……3

1.4 References……4

2. Description……4

2.1 Product Perspective……4

2.2 Product Functions……5

2.3 Key Features of Asynchronous FIFO……6

2.4 User Classes and Characteristics……6

2.5 Tools and Software……7

2.6 Design and Implementation and Constraints……7

2.7 Assumptions and Dependencies……8

2.7.1 Assumptions……8

2.7.2 Dependencies……8

3. Dataflow and Working……8

3.1 Dataflow and Clock edge triggering……8

3.2 Clock Domain Synchronization……9

3.3 Status Indicators……9

3.4 Counter Design……9

3.5 Reset Mechanism ……9

4. Design Implementation……10

5. Timing Diagram of Asynchronous FIFO……12

5.1 Basic Timing Diagram……12

6. External Interface Requirements……13

6.1 Software Interfaces……13

6.2 Hardware Interfaces……13

7. FIFO Depth Calculations……14

8. Features……15

8.1 FIFO Memory……15

8.2 Synchronizers……15

8.3 Binary to Gray Converter…… 15

8.4 Write and Read Blocks……16

8.5 Status Indicator……16

9. Logic Design……17

10. Signal Description……..18

1. Introduction

1.1 Purpose

This document offers an in-depth specification for the Asynchronous First-In-First-Out (FIFO) memory module, detailing its design features, functionality, interface specifications, and verification aspects. It is designed to be a guide for design and verification engineers, as well as system architects, helping them in the development, integration, and application of the FIFO in digital systems. The document's goal is to describe how the FIFO manages asynchronous data transfer and synchronization between different clock domains, thereby improving data efficiency and reliability. It aims to facilitate data communication and storage within electrical systems and acts as a key component for handling data flow between various subsystems, peripherals, or external devices, ensuring smooth and low-latency communication. The scope includes detailed design aspects of the Asynchronous FIFO.

1.2 Intended Audience

The primary target audience for this Asynchronous FIFO memory module High-Level Design Specification (HLDS) is technical project managers, system architects, and design and verification engineers. It offers comprehensive technical insights for development and integration.

1.3 Product Scope

The goal of the Asynchronous FIFO memory module is to improve data flow in complicated digital systems by offering buffering and smooth data synchronization over several clock domains. In bigger systems like SoCs, CPUs, network chips, and communication interfaces, its integration is essential for enhancing data integrity and lowering latency. Enhancing efficiency and dependability in high-speed digital environments is aligned with the robust performance and adaptability of the module to varying data rates. Because of this, it plays a crucial role in sophisticated digital systems, greatly enhancing their overall efficacy and scalability.

Flexibility: The Asynchronous FIFO is designed to adapt to diverse digital system architectures, accommodating various data transfer scenarios.

Robustness: Emphasis is placed on reliability and error-free operation, ensuring data integrity and system stability.

Low Latency: The design aims to minimize data transfer delays, enhancing overall system responsiveness and efficiency.

Clock Domain Crossing: It efficiently manages data transfer between different clock domains, crucial in systems where components operate at different clock speeds.

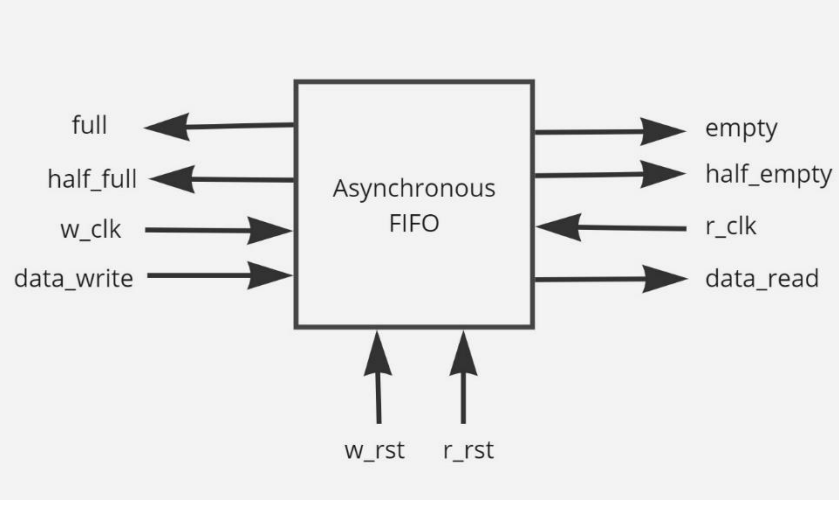
1.4 References

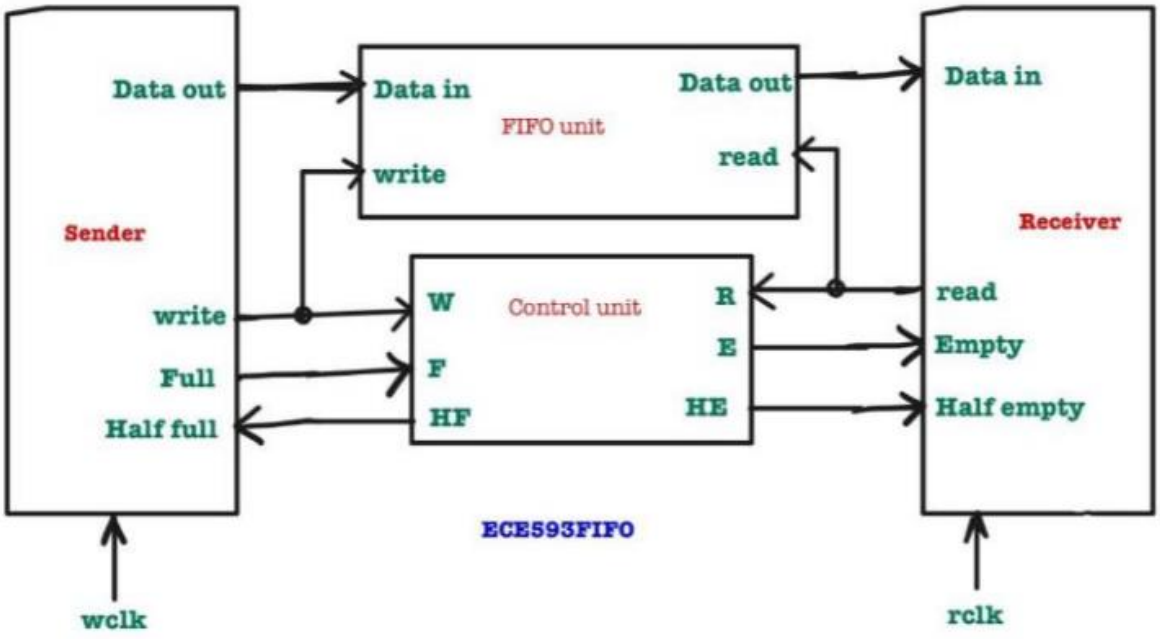
1. VLSI Verify https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/
2. S. Cummings, "FIFOs: Fast, predictable, and deep," in Proceedings of SNUG, 2002. http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf
3. S. Cummings, "FIFOs: Fast, predictable, and deep (Part II)," in Proceedings of SNUG, 2002. http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdf
4. A. Author et al., "Title of the Paper," in Proceedings of the Conference, 2011, pp. 789-012. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6041338
5. Asynchronous FIFO- Clock Domain Synchronizers https://www.youtube.com/playlist?list=PLdcY8Cf-O1Zon-8c9NDhgY2F8r_O7oeC4
6. Open AI, Chat GPT

2. Description

2.1 Product Perspective

The Asynchronous FIFO design arises from the need for a dependable data buffering solution that can be adjusted for digital systems to enhance their data storage and communication abilities. It has been created to handle the demand of managing the flow of information between different parts such as peripherals or external devices within an environment where everything is done digitally. What makes it different is its ability to carry out read and write operations at different clock frequencies hence bearing the name Asynchronous FIFO. Asynchronous FIFO is used when transferring data from one clock domain to another which means changing the points of reference for time known as clock domain crossing. Therefore, this type is necessary for synchronizing between systems operating on dissimilar clocks.





2.2 Product Functions

Generally, Asynchronous FIFO is used when the write operation is faster than the read operation. Therefore, they need to be synchronized. Otherwise, it may lead to metastability conditions. This will affect the operation of the chip. To overcome this problem Asynchronous FIFOs are used. Essential functions of Asynchronous FIFO include-

1. **Data Buffering:** The FIFO serves as a buffer for storing the data temporarily, balancing fast producers and slow consumers.
2. **Clock Domain Crossing:** It facilitates data transfer between subsystems or components operating at different clock frequencies, ensuring integrity of transferred data.
3. **Binary to Gray Code conversion:** It accepts binary input and converts to Gray code ensuring efficiency and error correction, offering configurability, ensuring validation.
4. **Data Integrity Maintenance**: The FIFO ensures that data integrity is maintained throughout the transfer process, preventing inconsistencies in the transmitted data.
5. **Flexibility and Adaptability**: The FIFO offers flexibility in terms of configuration options and adaptability to different system requirements, making it suitable for a wide range of applications and environments.
6. **Flow Control Management:** Regulates data flow to prevent overflow or underflow, ensuring smooth and reliable transmission within the system.
7. **Status Indication:** Generates signals (FIFO full, FIFO empty, etc.) to indicate system conditions and facilitate appropriate actions.
8. **Interface Compatibility:** Integrates seamlessly with SoCs, CPUs, and network chips through standard interfaces.

2.3 Key Features of Asynchronous FIFO

1. **Clock Domain Crossing:** Asynchronous FIFOs are specifically designed to handle situations where data needs to be transferred between components operating in different clock domains allowing seamless communication between different sections of a system without the need for synchronized clocks.
2. **Handshaking Signals:** Handshaking signals are used to coordinate the data transfer between the write and read sides of the FIFO. These signals include indicators such as write enable, read enable, FIFO full and FIFO empty. It involves a bidirectional communication protocol where the write side and the read side exchange control signals to manage the flow of data.
3. **Gray Code Pointers:** Gray code is a binary numeral system where two successive values differ in only one bit. In the context of asynchronous FIFOs, Gray code pointers are used for the write and read pointers. Because Gray code changes only one bit at a time, it reduces the chance of multiple bit errors during the synchronization process, thereby minimizing the risk of metastability.
4. **FIFO Depth:** FIFO depth refers to the number of data words that the FIFO can hold at any one time. It is an essential parameter that can often be configured to suit specific application needs. The depth determines the storage capacity of the FIFO, allowing it to buffer a certain amount of data between the write and read operations.
5. **Independent Clock:** The Asynchronous FIFO operates with separate, independent clocks for the write and read operations. The write side is driven by a write clock, while the read side is driven by a read clock, and these clocks can run at different frequencies and phases.
6. **Reset Signal:** The FIFO includes a reset input that can initialize the internal state of the FIFO, including the write and read pointers, status flags, and data storage. Ensures that the FIFO starts from a known state, which is crucial during power-up or system initialization.

2.4 User Classes and Characteristics

1. **Design Engineers:** They make use of Asynchronous FIFO while developing digital systems, like DSPs, memory controllers and communication interfaces.
2. **Verification Engineers:** Digital designs should be tested and validated for functionality, time, and corner cases by verification engineers using asynchronous FIFOs in verification environments.
3. **System Architects:** They Assess the use of asynchronous FIFOs in data transfer, buffering, and synchronization by integrating them into system-on-chip (SoC) architectures.
4. **Application Developers**: These people Use asynchronous FIFOs in software applications for signal processing, embedded systems, or communication protocols that require synchronization, data buffering, or communication.

5. **System Integrators:** Ensure the functioning and compatibility of asynchronous FIFOs by integrating them seamlessly into larger systems, including hardware platforms, firmware, and software stacks.
6. **End Users:** Achieve enhanced system performance and dependability while interacting with systems or applications that make use of asynchronous FIFOs for data buffering, communication, or synchronization.

2.5 Tools and Software

The environment in which the Asynchronous FIFO is designed is using QuestaSim.

Operating System and Versions: QuestaSim is compatible with multiple operating systems, including Windows and Linux. The Asynchronous FIFO design, when simulated or tested using QuestaSim, is expected to operate seamlessly on supported versions of these operating systems.

Verification Tools: Verification of asynchronous FIFOs uses simulation tools like QuestaSim, and methodologies like UVM for comprehensive testing.

Hardware Platform: The Asynchronous FIFO design is compatible with QuestaSim and can run on x86-based systems, accommodating diverse hardware configurations.

2.6 Design and Implementation Constraints

Clock Frequency Compatibility: The module is designed to operate under the conditions of a producer clock (clk1) frequency of 120 MHz and a consumer clock (clk2) frequency of 50 MHz, both with 50% duty cycle.

Data Width Compatibility: We need to modify the module's data width according to the application's needs. Ensure it matches the requirements to prevent potential problems like data truncation or overflow, considering the specified maximum write burst size of 1024.

Clock Domain Synchronization: While the module facilitates clock domain synchronization, ensure that proper clock domain crossings are implemented in your designs to prevent data synchronization issues. The ideal cycle between successive writes and reads should be considered (4 and 2 cycles, respectively) as per the design specification**.**

Binary-to-Gray Code Conversion: If the module is used for binary-to-gray code conversion, ensure that your application requires this feature and configure the module, taking into consideration the design specifications.

Design Limitations: The design imposes no specific constraints on the hardware platform, ensuring flexibility in deployment and compatibility with the specified QuestaSim version ensures access to the latest simulation features and optimizations.

2.7 Assumptions and Dependencies

2.7.1 Assumptions

Clock Signals: Users are expected to provide stable clock signals (clk1 and clk2) at specified frequencies and duty cycles in accordance with design specifications.

Configuration: Users must set the FIFO module's data width, clock frequencies, and operating parameters in accordance with design specifications.

Flow Control: The user is responsible for implementing proper flow control mechanisms to prevent data overflow and underflow.

Binary-to-Gray Code Conversion: If this feature is enabled, users assume alignment with their application requirements and perform any necessary configurations.

2.7.2 Dependencies

Clock Sources: Proper clock sources that meet specified requirements are critical for module operation.

Tools: Users may need digital design tools and Verification tools like Questa sim and EDA software for integration.

Hardware: Users may need FPGAs for integrating and testing the design.

Clock Domain Crossing: Proper clock domain crossing techniques are required for data synchronization.

3. Dataflow and Working

3.1 Data flow and Clock Edge Triggering

Write Operation: Data is written to the FIFO from the input side, which operates on a specific clock (posedge). A write pointer keeps track of where the next piece of data should be written in the FIFO memory. As data is written, the write pointer is updated and synchronized across the boundary between the write clock domain and the read clock domain.

Read Operation: : Data is read from the FIFO on the positive edge (posedge) of the consumer clock (clk2). A read pointer keeps track of where the next piece of data should be read from in the FIFO memory. This alignment with the rising edge of the clock ensures data integrity and minimizes read latency.

3.2 Clock Domain Synchronization

Clock Domains: The FIFO operates across two different clock domains, clk1 for writing and clk2 for reading, both operating at 120MHz and 50MHz.

Synchronizers: Synchronizers are essential to safely pass the read and write pointers between clk1 and clk2 domains. This prevents issues like metastability which can occur when signals are transferred between asynchronous clock domains.

Number of Synchronizers: Typically, a chain of two or three flip-flops is used as a synchronizer for each signal crossing the clock domains, ensuring reliable signal transfer.

3.3 Status Indicators

Full and Empty Flags: The FIFO provides status signals such as "FIFO Full" and "FIFO Empty" to indicate when the FIFO cannot accept more data or when there is no data to read, respectively.

Half Full/Half Flags: Additional status signals like "FIFO Half Full" and "FIFO Half Empty" provide early warnings about the FIFO's capacity status, allowing the system to take preventive actions.

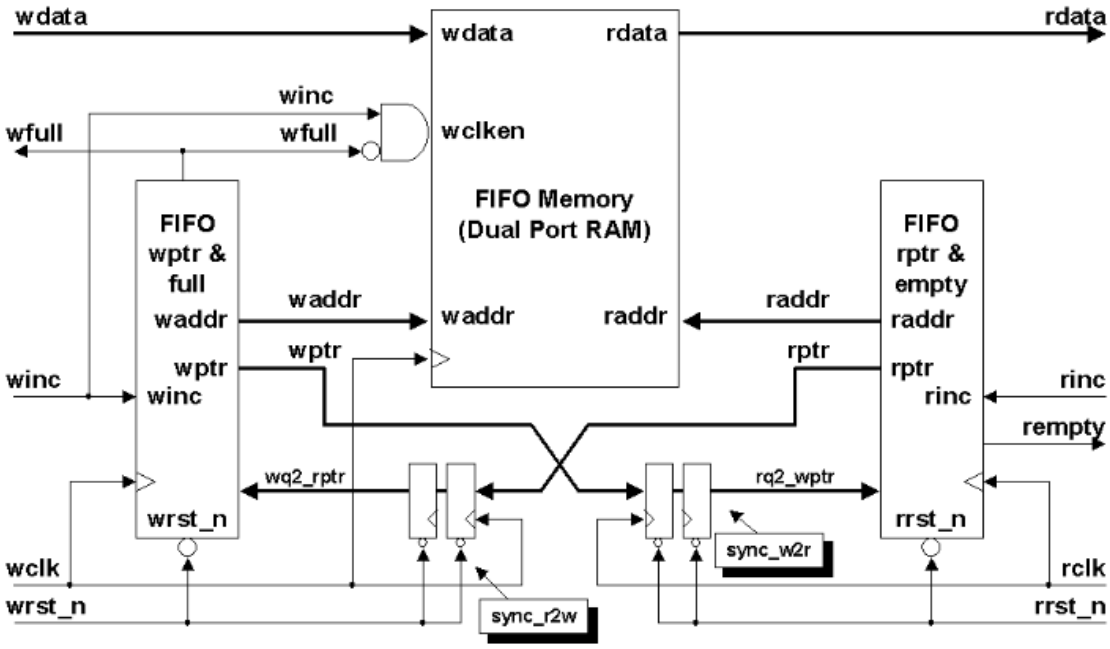
3.4 Counter Design

The design utilizes a dual Binary-Gray counter system. The binary counter manages the read and write pointers internally, while the gray code representation is used for pointer comparisons and status flag generation to ensure one-bit change stability across clock domains. Gray code is a binary numeral system where two successive values differ in only one bit. In the context of asynchronous FIFOs, Gray code pointers are used for the write and read pointers. Because Gray code changes only one bit at a time, it reduces the chance of multiple bit errors during the synchronization process, thereby minimizing the risk of metastability.

3.5 Reset Mechanism

The reset feature initializes the module's state, ensuring stability during startup or error recovery. It restores internal variables and state machines, promoting reliable and consistent operation. : The reset is designed to be asynchronous to allow immediate response, independent of clock cycles. : The FIFO includes a reset input that can initialize the internal state of the FIFO, including the write and read pointers, status flags, and data storage. Ensures that the FIFO starts from a known state, which is crucial during power-up or system initialization.

4. Design Implementation



To understand the FIFO design, we need to understand how the FIFO pointers work. The write pointer always points to the next word to be written; therefore, on reset, both pointers are set to zero, which also happens to be the next FIFO word location to be written. On a FIFO-write operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written.

Similarly, the read pointer always points to the current FIFO word to be read. On reset, both pointers are reset to zero, the FIFO is empty, and the read pointer is pointing to invalid data (because the FIFO is empty, and the empty flag is asserted). As soon as the first data word is written to the FIFO, the write pointer increments, the empty flag is cleared, and the read pointer that is still addressing the contents of the first FIFO memory word, immediately drives that first valid word onto the FIFO data output port, to be read by the receiver logic. The fact that the read pointer is always pointing to the next FIFO word to be read means that the receiver logic does not have to use two clock periods to read the data word. If the receiver first had to increment the read pointer before reading FIFO data word, the receiver would clock once to output the data word from the FIFO and clock a second time to capture the data word into the receiver. That would be needlessly inefficient.

The FIFO is empty when the read and write pointers are both equal. This condition happens when both pointers are reset to zero during a reset operation, or when the read pointer catches up to the write pointer, having read the last word from the FIFO. FIFO is full when the pointers are again

equal, that is, when the write pointer has wrapped around and caught up to the read pointer. This is a problem. The FIFO is either empty or full when the pointers are equal.

One design technique used to distinguish between full and empty is to add an extra bit to each pointer. When the write pointer increments past the final FIFO address, the write pointer will increment the unused MSB while setting the rest of the bits back to zero as shown in Figure 1 (the FIFO has wrapped and toggled the pointer MSB). The same is done with the read pointer. If the MSBs of the two pointers are different, it means that the write pointer has wrapped one more time than the read pointer. If the MSBs of the two pointers are the same, it means that both pointers have wrapped the same number of times.

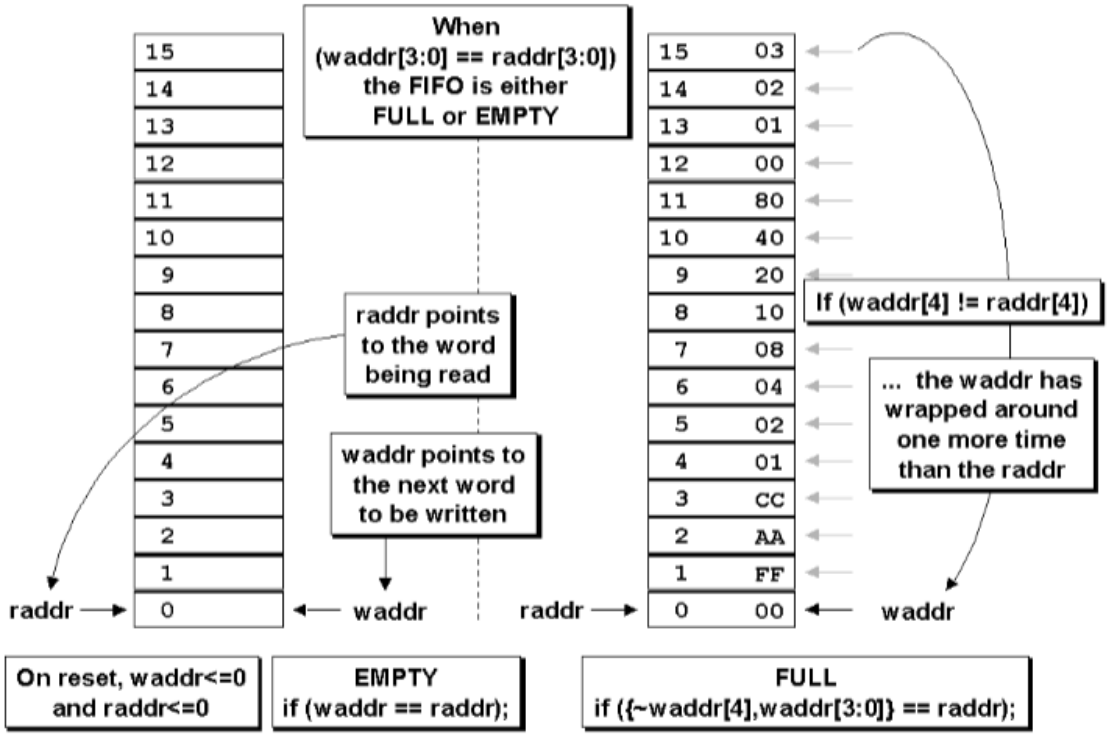
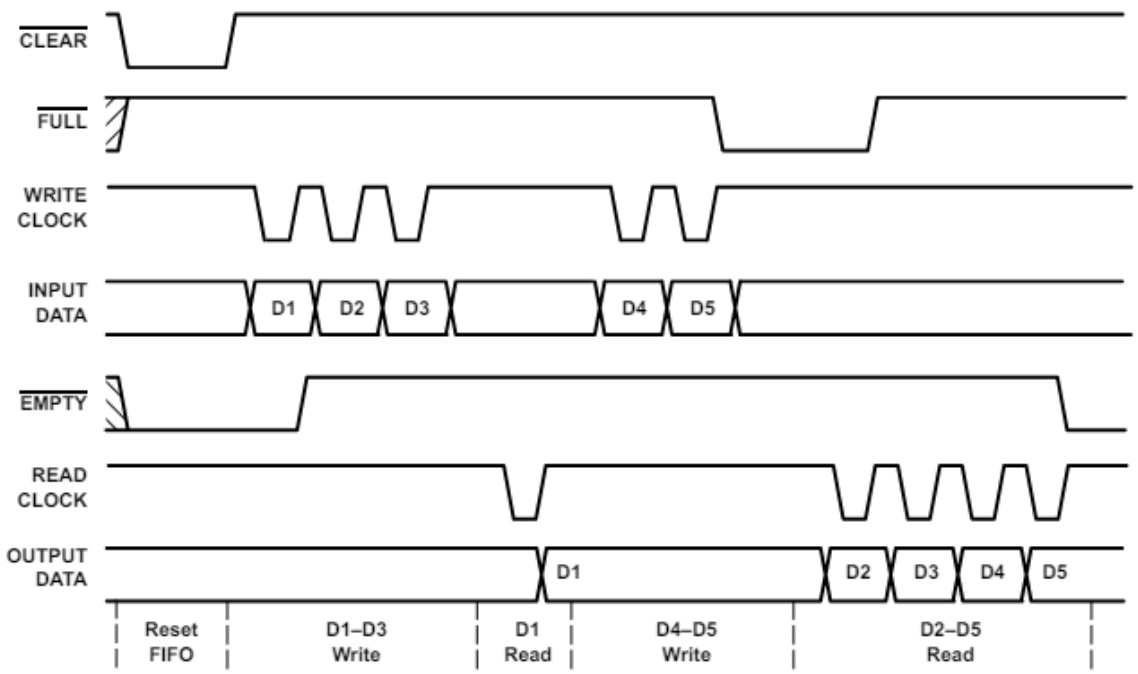


Figure 1 - FIFO full and empty conditions

To implement a FIFO memory module with "half-full" and "half-empty" flags, several steps are involved. Firstly, the fill level of the FIFO needs to be tracked, typically using counters to monitor the number of elements stored relative to its capacity. This includes determining thresholds for when the FIFO is considered half full or half empty, usually calculated as half of the total capacity. Then, logic is implemented to assert the corresponding flags based on the fill level: when the number of stored elements reaches or exceeds the half-full threshold, the "half-full" flag is set; conversely, when the number of stored elements drops below the half-empty threshold, the "half-empty" flag is asserted. These flags are updated during FIFO read and write operations, ensuring accurate indication of the FIFO's fill level. Additionally, proper synchronization and timing considerations are necessary to prevent incorrect flag states or race conditions.

5. Timing Diagrams of Asynchronous FIFO

5.1 Basic Timing Diagram



The above timing diagram shows the resetting of the FIFO that is always necessary at the beginning. Then, three data words are written in. The data words D1 through D3 appear one after the other on the INPUT DATA inputs and clock edges are applied to WRITE CLOCK for transfer of the data. Once the first data word has been written into the FIFO, the EMPTY signal changes from low level to high level. Another two data words are written into the FIFO before the first read cycle. The subsequent reading out of the first data word with the aid of a clock edge on READ CLOCK does not alter the status signals. With the writing of another two data words, the FIFO is full. This is indicated by the FULL signal. Finally, the four data words D2 through D5 remaining in the FIFO are read out. Thus, the FIFO is empty again, so the EMPTY status line shows this by low level.

The control lines WRITE CLOCK and $\overline{\text{FULL}}$ are used to write data. When a data word is to be written into an asynchronous FIFO, it is first necessary to check whether there is space available in the FIFO. This is done by querying the $\overline{\text{FULL}}$ status line. If free space is indicated, the data word is applied to the data inputs and written into the FIFO by a clock edge on the WRITE CLOCK input. In analogous fashion, the control lines READ CLOCK and $\overline{\text{EMPTY}}$ are used to read data. In this case, the $\overline{\text{EMPTY}}$ status output has to be queried before reading, because data can be read

out only if it is stored in the FIFO. Then, a clock edge is applied to the READ CLOCK input, causing the first word in the data queue to appear on the data output.

6. External Interface Requirements

6.1 Software Interfaces

The Asynchronous FIFO memory module primarily operates within digital system environments and interfaces with various software components during its integration and verification processes. Here are the key software interfaces and dependencies.

Digital Design and Verification Tools

QuestaSim-Users can employ QuestaSim, a powerful simulation and verification tool, for designing, compiling, and verifying the FIFO module. QuestaSim facilitates hardware description language (HDL) simulation, ensuring the correctness and functionality of the design.

Operating Systems

The FIFO module itself does not have dependencies on specific operating systems. It is generally platform-agnostic and can operate in various environments.

6.2 Hardware Interfaces

Data Input and Output

Data In : Allows external modules to write data into the FIFO, adhering to the specified data width.

Data Out : Permits the retrieval of data from the FIFO by external modules, with the data width matching the configured setting.

Clock Signals

w_clk : A clock signal operating at 120 MHz with a 50% duty cycle, driving FIFO write operations.

r_clk : Another clock signal operating at 50 MHz with a 50% duty cycle, controlling FIFO read operations.

Control Signals

Write Enable (WE): Used by external modules to enable or disable write operations to the FIFO.

Read Enable (RE): Controls the reading of data from the FIFO.

Binary-to-Gray Code Conversion

If binary-to-Gray code conversion is enabled, the FIFO will incorporate extra logic to support this feature. External modules can interact with this functionality as needed.

Status Signals

FIFO Full : Indicates when the FIFO is full and unable to accept further write operations.

FIFO Empty: Signifies when the FIFO is empty and cannot be read until new data is written.

7. FIFO Depth Calculation

$f_A > f_B$ with idle cycles in both write and read.

Writing frequency = f_A = 120MHz.

Reading Frequency = f_B = 50MHz.

Burst Length = No. of data items to be transferred = 1024.

No. of idle cycles between two successive writes is = 4.

No. of idle cycles between two successive reads is = 2.

The no. of idle cycles between two successive writes is 4 clock cycle. It means that, after writing one data, module A is waiting for four clock cycle, to initiate the next write. So, it can be understood that for every five clock cycles, one data is written.

The no. of idle cycles between two successive reads is 2 clock cycles. It means that, after reading one data, module B is waiting for 2 clock cycles, to initiate the next read. So, it can be understood that for every three clock cycles, one data is read.

Time required to write one data item = $5 * 1/120$ MHz =41.67 ns = 42 ns

Time required to write all the data in the burst = 1024 * 42 ns. = 43008 ns.

Time required to read one data item = $3 * (1/50)$ MHz =60 ns.

So, for every 60 ns, module B is going to read one data in the burst.

So, in a period of 43008 ns, 1024 no. of data items can be written.

The no. of data items can be read in a period of 43008 ns = 43008 ns / 60 ns= 717

The remaining no. of bytes to be stored in the FIFO = 1024 – 717 = 307.

So, the FIFO which must be in this scenario must be capable of storing 307 data items.

So, the minimum depth of the FIFO should be 307.

Duty cycle is around 50%.

8. Features

8.1 FIFO Memory

A first-in first-out (FIFO) memory is one where data written first is also the first to be read. In a concurrent read/write FIFO, which can store a variable number of data words, the read and write operations can be asynchronous. This means there is no dependency between reading and writing data, allowing for simultaneous or successive read and write actions. Consequently, two systems operating at different frequencies can be connected to the FIFO without the designer needing to handle synchronization, as the FIFO manages this internally.

8.2 Synchronizers

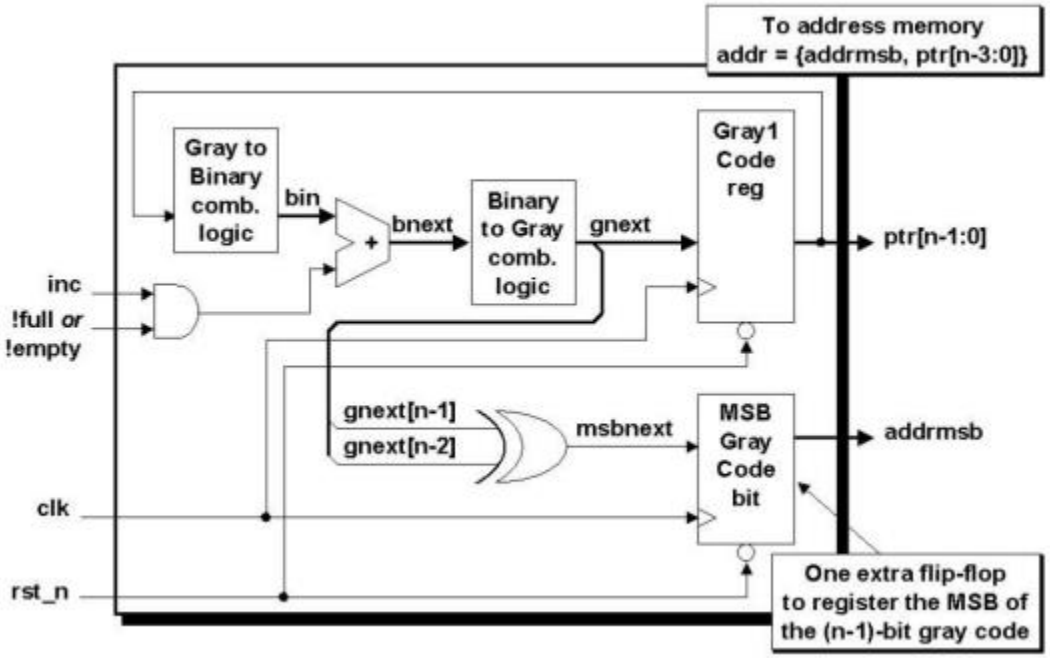
In a FIFO memory module, a synchronizer is essential for managing data transfer between different clock domains, ensuring data integrity and preventing metastability. This is particularly important in asynchronous FIFOs where read and write operations are driven by separate, unsynchronized clock signals. The synchronizer typically employs Gray code pointers and two-stage flip-flop synchronization to stabilize signals before they are used in the receiving clock domain. This process helps in accurately generating status flags (such as empty, full, half-full, half-empty) and allows independent and concurrent read/write operations. By safely handling clock domain crossing, the synchronizer ensures reliable communication and data integrity between systems operating at different frequencies.

. In Asynchronous FIFO the write pointer is aligned to the write clock domain whereas the read pointer is aligned to the read clock domain. Hence, it requires domain crossing to calculate FIFO full and empty conditions. This causes metastability in the actual design. To resolve the metastability, 2 flip flops or 3 flip flops synchronizer is used to pass write and read pointers. For explanation, we will go with 2 flip-flop synchronizers. Please note that a single “2 FF synchronizer” can resolve metastability for only one bit. Hence, depending on write and read pointers multiple 2FF synchronizers are required.

8.3 Binary to Gray Converter

The design utilizes a dual Binary-Gray counter system. The binary counter manages the read and write pointers internally, while the gray code representation is used for pointer comparisons and status flag generation to ensure one-bit change stability across clock domains. Gray code is a binary numeral system where two successive values differ in only one bit. In the context of asynchronous FIFOs, Gray code pointers are used for the write and read pointers. Because Gray code changes only one bit at a time, it reduces the chance of multiple bit errors during the synchronization process, thereby minimizing the risk of metastability. The Gray code counter operates by using the register outputs as the Gray code value itself. These Gray code outputs are then converted to binary using a Gray-to-binary converter. The binary output is fed into a conditional binary-value incrementor to produce the next binary count value. This binary value is

then converted back to Gray code using a binary-to-Gray converter, generating the next Gray code count value, which is input back into the register. The upper part of the block diagram illustrates this logic flow, while the lower part shows the logic for a second Gray code counter, which will be detailed in the following section.



8.4 Write and Read Blocks

In an asynchronous FIFO, the write and read operations are managed by separate clock domains, and synchronization mechanisms ensure reliable data transfer. The write block handles data writing operations into the FIFO, updates the write pointer, and manages the write enable signal. The read block handles data reading operations from the FIFO, updates the read pointer, and manages the read enable signal.

8.5 Status Indicator

Full and Empty Flags: The FIFO provides status signals such as "FIFO Full" and "FIFO Empty" to indicate when the FIFO cannot accept more data or when there is no data to read, respectively.

Half Full/Half Flags: Additional status signals like "FIFO Half Full" and "FIFO Half Empty" provide early warnings about the FIFO's capacity status, allowing the system to take preventive actions.

9. Logic Design

ASYNC_FIFO module

The ASYNC_FIFO module is the top-level module for an asynchronous FIFO with parameterizable data width and pointer width. This module orchestrates the entire FIFO operation, managing data flow between asynchronous write and read domains. It includes submodules for synchronizing pointers (synchronizer_r2w and synchronizer_w2r), handling read and write operations (rptr_handler and wptr_handler) and storing data (fifo_mem). The design uses write and read enable signals to control data writing and reading, while the full, half-full, empty, and half-empty status signals provide information about the FIFO's state.

fifo_mem module

The fifo_mem module implements the memory storage for the FIFO. It uses a parameterized depth based on the pointer width to define the memory array size. Data is written to the FIFO memory on the positive edge of the write clock (wclk) when the write enable signal is active and the FIFO is not full. Similarly, data is read from the FIFO memory on the positive edge of the read clock (rclk) when the read enable signal is active and the FIFO is not empty. This module ensures that data is correctly stored and retrieved based on the provided write and read addresses.

Synchronizer module

The synchronizer_r2w and synchronizer_w2r modules are responsible for synchronizing the read pointer to the write clock domain and the write pointer to the read clock domain, respectively. Each synchronizer module uses a two-stage flip-flop approach to prevent metastability issues. The synchronizer_r2w module takes the read pointer (rptr) and synchronizes it to the write clock domain (wclk), outputting the synchronized pointer (rptr_s). Similarly, the synchronizer_w2r module takes the write pointer (wptr) and synchronizes it to the read clock domain (rclk), outputting the synchronized pointer (wptr_s). These synchronized pointers are crucial for managing asynchronous read and write operations safely.

Pointer Handlers

The rptr_handler and wptr_handler modules manage the read and write pointers and their respective statuses. The rptr_handler module updates the read pointer (rptr), read address (raddr), and the empty status (rempty). It uses a Gray code for the pointer to minimize synchronization issues. The module calculates the next binary and Gray code values, ensuring correct pointer incrementation and empty status determination. Additionally, it generates a half-empty signal (wHalf_empty). The wptr_handler module performs similar functions for the write operations, updating the write pointer (wptr), write address (waddr), and the full status (wfull). It also uses Gray code and calculates the next pointer values, ensuring correct handling of the

write operations. The module also generates a half-full signal (wHalf_full). These pointer handler modules are essential for maintaining the integrity and efficiency of the FIFO operations.

10. Signal Description

Signal	Description
write_enable	This signal activates the writing operation to the FIFO. When write_enable is high and the FIFO is not full, data from data_write is written into the FIFO memory at the location pointed to by the write pointer.
read_enable	This signal activates the reading operation from the FIFO. When read_enable is high and the FIFO is not empty, data is read from the FIFO memory at the location pointed to by the read pointer and output to data_read.
wclk	The write clock signal that drives the write operations in the FIFO. It is used to synchronize the writing of data and the updating of the write pointer.
rclk	The read clock signal that drives the read operations in the FIFO. It is used to synchronize the reading of data and the updating of the read pointer.
wrst_n	An active-low reset signal for the write domain. When wrst_n is low, it resets the write pointer and related write logic to ensure the FIFO is in a known state before normal operations resume.
rrst_n	An active-low reset signal for the read domain. When rrst_n is low, it resets the read pointer and related read logic to ensure the FIFO is in a known state before normal operations resume.
data_write	The input data signal to be written into the FIFO. It holds the data that will be stored in the FIFO memory when write_enable is active and the FIFO is not full.
data_read	The output data signal that reads data from the FIFO. It holds the data retrieved from the FIFO memory when read_enable is active and the FIFO is not empty.
wfull	A status signal that indicates the FIFO is full. When wfull is high, it prevents further write operations to avoid data overflow, signaling that no more data can be written until some data is read out.
wHalf_full	A status signal that indicates the FIFO is at least half full. This signal helps in monitoring the usage of the FIFO and can be used for flow control to manage data writing rates.
rempty	A status signal that indicates the FIFO is empty. When rempty is high, it prevents further read operations to avoid underflow, signaling that no data is available to read until new data is written into the FIFO.

wHalf_empty	A status signal that indicates the FIFO is at least half empty. This signal helps in monitoring the usage of the FIFO and can be used for flow control to manage data reading rates.
waddr	The write address pointer for the FIFO memory. It points to the location in the FIFO memory where the next data word will be written.
raddr	The read address pointer for the FIFO memory. It points to the location in the FIFO memory where the next data word will be read.
wptr	The write pointer in Gray code format used for synchronization. It keeps track of the write position in the FIFO and is used to avoid synchronization issues between different clock domains.
rptr	The read pointer in Gray code format used for synchronization. It keeps track of the read position in the FIFO and is used to avoid synchronization issues between different clock domains.
wptr_s	The synchronized write pointer in the read clock domain. It is the write pointer converted and synchronized to the read clock domain to ensure correct operation of the read logic.
rptr_s	The synchronized read pointer in the write clock domain. It is the read pointer converted and synchronized to the write clock domain to ensure correct operation of the write logic.