Asynchronous FIFO: Design Specification

Session 1 Group 4

Nick Allmeyer, Alexander Maso, Ahliah Nordstrom

Version 3.0 - June 11, 2024

# Features

* Separate modules for READ control, WRITE control, Memory, and for each clock synchronizer
* Pointer management using binary counter, *later implementation of Gray Code to come*
* Simplified binary counter to minimize synchronization issues between clock domains initially
* Sender clock frequency of 80 Mhz for rapid data transmission
* Receiver clock frequency of 50 Mhz for optimized processing capabilities
* Sender data burst size of 120 data items for efficient bulk processing
* Zero idle cycles for both reading and writing allows for maximized throughput
* High data rate with minimal latency by achieving 12.5 ns for every WRITE operation and 20 ns for every READ operation
* Maintains a 8 : 5 write to read ratio to optimize buffer utilization and prevent bottlenecks
* Minimum FIFO depth of 45
* FIFO depth, Addr width and Data width are parameterized
* Initial implementation has depth of 64, addr width of 6 (+1 for ptrs) and data width of 8

# Functional Description

The asynchronous FIFO acts as a buffer to manage data flow between two parts of a system operating on different clock domains. The FIFO ensures data integrity and timing without the need for clock synchronization between sender and receiver blocks, operating at 80 Mhz and 50 Mhz respectively.

A memory buffer with a depth of at least 45 items is used to store data elements. Operations are consistent and predictable for both READ and WRITE, as they are fully synchronous to their respective clocks. Data is written to the FIFO at a rate of 12.5 ns per operation and is determined by the rising edge of the WRITE clock. Data is read at a rate of 20 ns per operation and is determined by the rising edge of the READ clock. The ratio of WRITEs to READs is 8:5, simplified down from 120:75. This ratio defines the operational balance between input and output operations so that the FIFO does not overflow/underflow under normal operating conditions.

Gray code is used to manage the read and write pointers, minimizing metastability issues during clock domain crossings. These pointers are initialized to zero upon system reset and increment as data is written to and read from the FIFO. The FIFO also features several status flags: the FULL flag is generated in the write-clock domain when the FIFO cannot accept more data, the EMPTY flag is generated in the read-clock domain when the FIFO is empty, and the HALF FULL and HALF EMPTY flags indicate when the FIFO is nearing these respective states. if an invalid read operation is attempted on an empty FIFO, the RD\_ERR signal is asserted. Similarly, if an invalid write operation is attempted on a full FIFO, the WR\_ERR signal is asserted.

# Synchronization and Timing Issues

Synchronization of the READ and WRITE pointers between different clock domains is a common source of error. To mitigate this and the risk of metastability, each pointer is passed through a series of dual flip-flop stages and serves as a synchronization mechanism. When data is transferred from one clock domain to another, it passes through an initial flip-flop, which captures the input data and holds it stable. The output of this flip-flop is then fed into a second flip-flop, clocked by the receiving clock domain, ensuring that the data is correctly synchronized with the receiving clock.

# Behavior of Status Signals

An asynchronous reset will render the FIFO empty with no valid data. This clearing signal instantaneously resets all synchronizing registers, WCLK, FULL, HALF\_FULL and RCLK but sets the EMPTY and HALF\_EMPTY flag status since the FIFO is indeed empty/half empty. Although this may seem abrupt, this does not affect the FIFO since all of its contents are reinitialized. Reset signals are negated in alignment with the rising edge of their respective clock domains.

The FULL signal is activated when the FIFO buffer reaches its maximum capacity that is indicated by the WRITE pointer being just one position away from the READ pointer. This FULL signal prevents any additional data writes that could lead to data overflow. The EMPTY signal indicates that the FIFO is lacking any data and the W\_PTR and R\_PTR are equal. There are edge cases where the FULL and EMPTY signals could be simultaneously asserted so an additional logic bit can be used to differentiate between these states. This FULL/EMPTY bit toggles upon each wrap around of the pointers to buffer.

Removing the FULL status is intentionally delayed afterwards so that even after the R\_PTR increments and the FIFO is no longer full, the logic retains the FULL asserted for two additional rising edges of the WCLK. This latency results in synchronization of the incremented R\_PTR into the write domain and thus giving the FIFO sufficient time to clear space before accepting new data. Similarly, when removing the EMPTY status, the FIFO will hold the EMPTY flag high for an additional two rising edges of the RCLK following a WRITE. This synchronizes the incremented W\_PTR into the read domain.

The HALF\_FULL signal is asserted when the data reaches or exceeds half of the FIFOs capacity; when the count of items in the FIFO is at least half the buffer size. The HALF\_EMPTY signal is asserted when the content of the FIFO falls below half.

# 

# FIFO Calculations

For the asynchronous FIFO, we know:

* Sender frequency = 80 MHz
* Receiver frequency = 50 MHz
* Write idle cycles = 0
* Read idle cycles = 0
* Write burst size = 120

No idle cycle in reading or writing operations, thus, all the items in the burst will be written and read in consecutive clock cycles.

The time required to write one items:

The time required to write all the data in the burst:

The time required to read one data item:

So, every 20ns the receiver sends one data in the burst and in a period of 1500ns, 120 data items can be written.

The number of data items that can be read in the time it takes to write all the data in the burst:

Remaining number of bytes stored in the FIFO (minimum FIFO depth):

However, 64 is assigned as the minimum FIFO depth to ensure proper incrementing of the gray code pointer around the max and min value (boundary); it needed to be a power of 2.

# References

[1] “Crossing clock domains with an Asynchronous FIFO,” zipcpu.com. <https://zipcpu.com/blog/2018/07/06/afifo.html>. [Accessed April 20, 2024]

[2] C. Cummings, “Simulation and Synthesis Techniques for Asynchronous FIFO Design,” *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002)* *User Papers.* March 2002, Vol. 281.

[3] R. Salemi, “The UVM Primer: An Introduction to the Universal Verification Methodology”. Boston: Boston Light Press, 2013. Accessed: Apr. 1, 2024. [Online].

[3] B. Wile and J. Gross and W. Roesner, “Comprehensive Functional Verification: The Complete Industry Cycle”. San Francisco, CA, USA: Morgan Kaufmann, 2005. Accessed: Apr. 1, 2024. [Online].

# 

# Document Revision History

| **Version** | **Date** | **Description** |
| --- | --- | --- |
| v 1.1 | 4/22/2024 | Milestone 1 deliverable |
| v 2.0 | 6/2/2024 | Milestone 5 deliverable |
| v 3.0 | 6/11/2024 | Final deliverable |

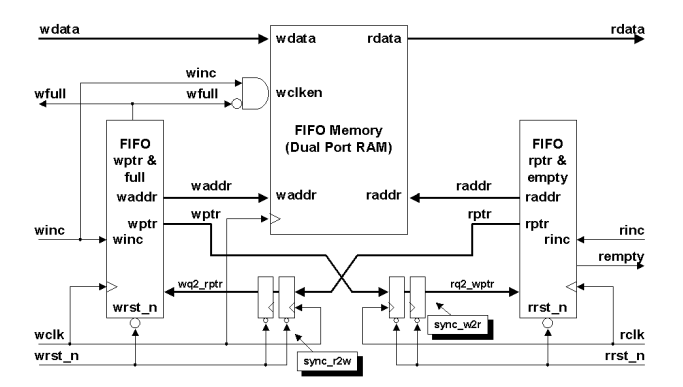
# 

# 

# 

# 

# Appendices



*Figure 2: Block diagram for asynchronous FIFO from C. Cummings FIFO1*

| **Signal** | **Direction** | **Description** |
| --- | --- | --- |
| WR\_EN | Input | Data to be written into the FIFO. |
| WINC | Input | Control signal to write data into FIFO. |
| WCLK | Input | Write domain clock signal. |
| RINC | Input | Control signal to read data from FIFO. |
| RCLK | Input | Read domain clock signal. |
| RST\_N | Input | Active-low asynchronous reset for FIFO. |
| WCLKEN | Input | Optional signal to enable writing on wclk rising edge. |
| RCLKEN | Input | Optional signal to enable reading on rclk rising edge. |
| RD\_EN | Output | Data read from the FIFO. |
| FULL | Output | Indicates FIFO is full. |
| EMPTY | Output | Indicates FIFO is empty. |
| HALF\_FULL | Output | Optional signal indicating FIFO is approaching full. |
| HALF\_EMPTY | Output | Optional signal indicating FIFO is approaching empty. |
| WR\_ACK | Output | Optional handshake signal for successful write. |
| WR\_ERR | Output | Optional handshake signal for failed write. |
| RD\_ACK | Output | Optional handshake signal for successful read. |
| RD\_ERR | Output | Optional handshake signal for failed read. |

*Table 1: Core signal pinout for asynchronous FIFO*