

Implementation and Verification of an Asynchronous FIFO

Bansari Mistry

Department of Electronics & Communication Engineering

Faculty of Technology, Dharmsinh Desai University

Nadiad, Gujarat, India

Email: bansarimistry9859@gmail.com

Abstract—This paper presents the implementation and verification of an asynchronous FIFO memory for reliable data transfer between independent clock domains. The design utilizes dual-port RAM, Gray-coded pointers, and two-stage synchronizers to eliminate metastability and ensure correct flag generation. A parameterized Verilog HDL implementation allows configurable data width and depth. Comprehensive testbench simulations verify proper operation across reset conditions, boundary cases, concurrent read/write operations, and multi-frequency scenarios. Results demonstrate robust data integrity, accurate status flags, and effective overflow/underflow protection. The architecture is synthesizable for FPGA/ASIC deployment in high-speed communication systems, SoCs, and multimedia applications.

Index Terms—Asynchronous FIFO, Clock Domain Crossing, Metastability, Verilog HDL, Gray Code, Synchronizers

I. INTRODUCTION

Contemporary digital systems increasingly incorporate heterogeneous processing components running on isolated clock domains to balance power, performance, and modularity. Direct data transfer across asynchronous clock domains presents severe challenges, mainly metastability—a situation where flip-flops go into an indeterminate state when setup and hold time requirements are not met. Metastability propagates erratically throughout downstream logic, leading to data corruption, functional failure, and system instability.

Asynchronous First-In-First-Out (FIFO) buffers offer a strong solution by eliminating timing dependencies between producer and consumer interfaces while maintaining strict data integrity and ordering. They are an essential component in high-speed communication protocols (UART, SPI, Ethernet, USB), Network-on-Chip (NoC) designs, system-on-chip (SoC) sub-systems, and real-time multimedia processing pipelines where clock frequency disparities and phase delays are unavoidable.

The basic problem of asynchronous FIFO design is to securely synchronize read and write pointers between clock domains to produce correct full and empty status flags without causing race conditions or metastability. Traditional binary pointer encoding is inappropriate for clock domain crossing because of multi-bit changes that can cause transient invalid states upon synchronization. Gray code addressing—where only a single bit changes between two successive values—removes this danger, allowing glitch-free pointer comparison following synchronization.

Multi-stage synchronizer chains, which commonly consist of two or more cascaded flip-flops, ensure sufficient Mean Time Between Failures (MTBF) by permitting metastable states to settle prior to progressing to control logic. Dual-port RAM architecture also allows simultaneous, independent read and write without access conflicts, optimizing throughput efficiency.

This work demonstrates the implementation and validation of a parameterized asynchronous FIFO design in Verilog HDL. The design includes Gray-coded pointer synchronization, metastability mitigation using dual-stage synchronizers, and dual-port memory for simultaneous access. Parameterization facilitates scalable data width and FIFO depth configuration to suit varied application needs. An exhaustive verification methodology using Verilog testbenches confirms functional correctness on initialization sequences, boundary values (full/empty states), concurrent read/write accesses, and stress testing with frequency variations. Simulation outputs verify strong data integrity, deterministic flagging, and proper overflow/underflow protection, illustrating the architecture's applicability to FPGA and ASIC implementations in high-reliability applications.

II. DESIGN METHODOLOGY

The asynchronous FIFO architecture uses a modular design methodology for efficient and robust clock domain crossing, metastability elimination, and error-free data transfer. The design is divided into four basic subsystems: dual-port memory for storing data, pointer control logic for generating addresses, synchronization logic for secure clock domain crossing, and status flag generation logic for controlling flow.

A. Dual-Port RAM Architecture

The memory subsystem employs a true dual-port RAM with separate read and write ports, allowing for concurrent access by asynchronous clock domains without arbitration overhead. The write port is synchronous to the write clock (`wclk`), while the read port is autonomous on the read clock (`rclk`). Both ports possess dedicated address, data, and control interfaces that eliminate structural hazards and achieve maximum throughput efficiency.

The memory is realized as a 2^N -depth parameterized register array, where N is the width of the address pointer (usually

$N \geq 4$ for a depth of at least 16 locations). The data width is set using the `DATA_WIDTH` parameter (default 8 bits, scalable to application needs). Write operations are performed when the write enable signal (`w_en`) is high and the FIFO is not full, writing data at the address indexed by the binary write pointer. Read operations retrieve data from the location indexed by the binary read pointer when the read enable signal (`r_en`) is high and the FIFO is not empty. The two-port design provides zero-cycle latency for each of the operations, essential to preserve throughput in high-speed applications.

B. Pointer Management and Gray Code Conversion

Address generation uses independent binary counters for read and write pointers, each incremented modulo 2^N on valid access cycles. Binary pointers address the dual-port RAM directly in a deterministic manner. However, binary-to-binary pointer comparison across clock domains is dangerous because multi-bit transitions during incrementation can create transient glitches and incorrect flag generation when sampled by the destination clock domain.

To eliminate this hazard, both write and read pointers are converted to Gray code representation before clock domain crossing. Gray code is a reflected binary encoding where consecutive values differ by exactly one bit, ensuring glitch-free transitions during synchronization. The binary-to-Gray conversion follows the standard algorithm:

$$\text{Gray}[i] = \text{Binary}[i] \oplus \text{Binary}[i + 1],$$

where the most significant bit remains unchanged, i.e., $\text{Gray}[N] = \text{Binary}[N]$. This coding ensures that even when a pointer is read while in transition, at worst only one bit could be metastable, and the synchronized value corresponds to either the last or next valid count—both safe for the comparison logic.

Pointer width is chosen as $(N + 1)$ bits instead of N bits to distinguish between full and empty states. If only N bits are employed, both situations arise when read and write pointers are equal, creating ambiguity. The additional MSB resolves this: the FIFO is empty when pointers are precisely equal, and full when pointers differ only in the MSB, meaning the write pointer wrapped around and caught up with the read pointer.

C. Synchronization Circuits

Gray-coded pointers are transferred across clock domains using multi-stage synchronizer chains to suppress metastability propagation. Each synchronizer is composed of two cascaded D flip-flops clocked by the destination domain, forming the classic two-flop synchronizer topology. The first flip-flop captures the asynchronous input and may enter metastability if setup/hold requirements are violated. The second flip-flop provides additional settling time, statistically minimizing the likelihood of metastability propagation (ensuring MTBF much larger than the system operating lifetime at practical clock frequencies).

The Gray-encoded write pointer is synchronized to the read clock domain (`rclk`) using a two-stage synchronizer prior to empty flag generation. Similarly, the read pointer (Gray-encoded) is synchronized to the write clock domain (`wclk`) for full flag generation. Synchronization introduces a two-cycle latency in the destination clock domain, which creates a conservative boundary for flag assertion while ensuring reliable metastability protection.

The synchronized Gray-coded pointers are compared directly without conversion to binary, since Gray code comparison suffices for equality detection—the only requirement for flag generation. This avoids additional combinational logic delays and potential timing violations in the critical flag generation path.

D. Status Flag Generation Logic

The FIFO supplies two essential status flags for flow control: `empty` (read domain) and `full` (write domain). These flags prevent underflow and overflow situations, respectively, and must be generated with absolute reliability to guarantee data consistency.

1) *Empty Flag Logic:* The `empty` flag is asserted in the read clock domain when no unread data remains in the FIFO. This is detected by comparing the read pointer (Gray-encoded) with the synchronized write pointer (Gray-encoded, received from the write domain). Equality indicates that the read pointer has caught up with the write pointer:

$$\text{empty} = (\text{rptr_gray} == \text{wptr_gray_sync}).$$

The comparison is performed entirely within the read clock domain using synchronized values, ensuring metastability-free operation. The `empty` flag is registered to eliminate glitches and provide a stable control signal for downstream logic.

2) *Full Flag Logic:* The `full` flag is asserted in the write clock domain when the FIFO is full, preventing new writes from overwriting unread data. The FIFO is full when the write pointer is exactly one cycle behind the read pointer after wrapping around the circular buffer. In Gray code notation with $(N + 1)$ bits, this condition is detected by comparing the next write pointer with the synchronized read pointer, where the two MSBs are inverted:

$$\begin{aligned} \text{full} = & (\text{wptr_gray}[N : N - 1] == \sim \text{rptr_gray_sync}[N : N - 1]) \\ & \wedge (\text{wptr_gray}[N - 2 : 0] == \text{rptr_gray_sync}[N - 2 : 0]). \end{aligned} \quad (1)$$

This condition accounts for the wrap-around scenario: when the MSB bits of the write pointer are inverted and the remaining bits match, the write pointer has advanced through the entire buffer and caught up from behind. Like the `empty` flag, the `full` flag is registered to avoid glitches. Due to synchronization latency, the FIFO may conservatively report full slightly earlier (one or two cycles), trading minimal buffer utilization for guaranteed overflow protection.

E. Reset Strategy and Initialization

Separate resets for each domain allow independent initialization sequences during power-up, which is crucial in asynchronous designs where clocks may start at different times or frequencies. Synchronizer flip-flops are also reset to eliminate undefined states during the first synchronization cycles.

F. Parameterization and Scalability

The FIFO architecture is fully parameterized to accommodate diverse application needs without redesign. The primary configurable parameters are:

- **DATA_WIDTH:** Width of each FIFO entry (default 8 bits, expandable to 16, 32, 64, or user-defined widths).
- **ADDR_WIDTH:** Address pointer width defining FIFO depth as $2^{\text{ADDR_WIDTH}}$ (at least 4 for 16-entry depth, typically 4–10 for 16 to 1024 entries).

Parameterization enables multiple FIFO instances with different configurations within the same design, supporting heterogeneous buffering needs in SoC architectures. Synthesis tools automatically optimize unused logic, resulting in area-efficient implementations for both FPGA and ASIC targets.

III. VERIFICATION AND TIMING ANALYSIS

A. Verification Methodology

A systematic verification strategy was implemented using Verilog HDL testbenches to validate functional correctness across critical operating scenarios. The verification environment employed independent clock generators with `wclk` operating at 10 ns period (100 MHz) and `rcclk` at 14 ns period (71.4 MHz), creating a realistic asynchronous frequency ratio of 1.4:1. Three primary test cases were executed: (1) Reset and initialization testing verified that asynchronous assertion of `wrst_n` and `rrst_n` correctly initializes all pointers to zero, asserts the `empty` flag, and deasserts the `full` flag; (2) Full condition testing systematically filled the FIFO with 16 consecutive writes (for 4-bit addressing), confirming precise full flag assertion at maximum capacity and successful overflow prevention when write operations were attempted on a full buffer; (3) Simultaneous read/write operations validated dual-port RAM functionality and pointer synchronization under concurrent access, with data integrity checks confirming correct FIFO ordering and stable flag operation during overlapping transactions. All test scenarios executed without data corruption, pointer aliasing, or flag glitches, validating robust clock domain crossing and metastability mitigation.

B. Timing Analysis

Detailed timing characterization quantified synchronization delays across clock domains through waveform analysis. Table I summarizes measured delays for critical paths. The empty flag synchronization delay of 46 ns accounts for write pointer increment, binary-to-Gray conversion, two-stage synchronization through the read clock domain (minimum 28 ns for two 14 ns cycles), and comparison logic propagation. The

full flag synchronization delay ranges 17–32 ns, reflecting read pointer synchronization through two write clock cycles (minimum 20 ns) with variability due to asynchronous phase relationships. Write-to-read data path latency spans 2–3 read clock cycles (28–42 ns), representing the time for write pointer updates to propagate through synchronizers before the empty flag deasserts and permits read operations. These conservative delays ensure metastability-free operation while trading minimal latency for guaranteed data integrity. Simulation results confirmed glitch-free Gray code synchronization, clean flag transitions on clock edges, and absence of metastable states across all tested conditions.

TABLE I
SYNCHRONIZATION DELAYS AND TIMING CHARACTERISTICS

Parameter	Value	Description
Empty Sync Delay	46 ns	Time from write to empty flag update in read domain
Full Sync Delay	17–32 ns	Time from read to full flag update in write domain
Write-to-Read Path	2–3 rclk cycles	Gray code pointer synchronization

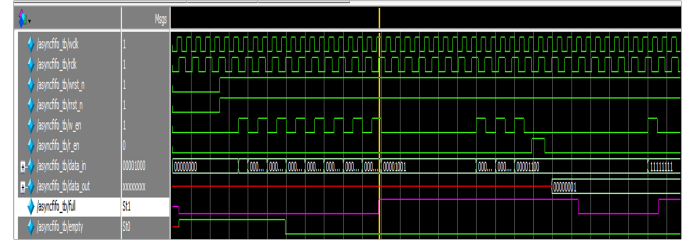


Fig. 1. Full Flag is Set

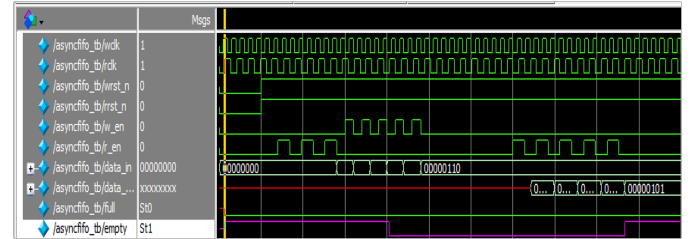


Fig. 2. Empty Flag is Set

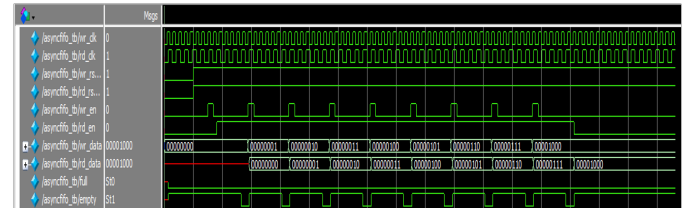


Fig. 3. Simultaneous Write-Read Operation

IV. APPLICATIONS

Asynchronous FIFOs are important building blocks in contemporary digital systems that need trustworthy data trans-

fer between autonomous clock domains. In high-speed communication interfaces like UART, USB, SPI, and Ethernet controllers, async FIFOs buffer incoming and outgoing data streams between protocol-specific clock domains and system bus speeds to absorb instantaneous rate differences and avoid data loss during burst transfers. In system-on-chip (SoC) designs, they act as crucial bridge elements between heterogeneous subsystems—separating processors running at gigahertz frequencies from peripheral controllers, hardware accelerators, and memory interfaces operating at distinct clock rates, facilitating modular IP integration without overheads of global clock distribution. Asynchronous FIFOs in multimedia processing pipelines remove artifacts due to audio/video codec-to-display controller-to-processing engine clock domain crossings, providing glitch-free media playback through frequency drift compensation and phase misalignment correction between independent timing sources. Also, they are widely used in network-on-chip (NoC) router buffers, FPGA clock domain crossing circuits, and mixed-signal systems in which analog-to-digital converters and digital signal processors run asynchronously, achieving deterministic latency bounds and assured throughput with data ordering and integrity.

V. CONCLUSION

This paper presented the implementation and verification of an asynchronous FIFO architecture. The key outcomes are:

- The design utilized Gray code pointer encoding to eliminate multi-bit transition hazards during clock domain crossing and dual-stage synchronizers to mitigate metastability risks.
- Dual-port RAM architecture enabled independent, simultaneous read and write operations across asynchronous write clock (100 MHz, 10 ns period) and read clock (71.4 MHz, 14 ns period) domains.
- Status flag generation logic implemented Gray code comparison for empty detection and MSB-inverted comparison for full detection, providing overflow and underflow protection with synchronization-induced latency.
- Verification through three test scenarios—reset initialization, full condition testing, and simultaneous read/write operations—validated functional correctness with measured synchronization delays of 46 ns for empty flag updates and 17–32 ns for full flag updates.
- The parameterized Verilog HDL implementation supports configurable data width and FIFO depth, enabling flexible instantiation for different buffering requirements.
- Simulation results demonstrated correct FIFO ordering, stable flag transitions, and no data corruption during tested conditions, confirming the design’s suitability for FPGA/ASIC deployment in communication systems, SoC interconnects, and multimedia applications requiring asynchronous data buffering.

REFERENCES

- [1] C. Cummings, “Simulation and Synthesis Techniques for Asynchronous FIFO Design,” SNUG Synopsys User Group, 2002.
- [2] P. R. Rao and M. Nanda, “Implementation and Verification of Asynchronous FIFO under Boundary Condition,” *IJERT*, vol. 6, no. 13, 2018.
- [3] E. Xie and J. Zhou, “Analysis and Comparison of Asynchronous FIFO and Synchronous FIFO,” *Proc. IEEE EEBDA Conf.*, pp. 260–264, 2023.