Special Assignment FPGA

(2EC202)

FIFO & LIFO Buffer using mode control switch

Vishesh Gevariya – 22BEC041

Department of Technology, Nirma University, Ahmedabad

Email : 22bec041@nirmauni.ac.in

Received 2 January 2024; Revised 22 April 2024; Accepted 24 April 2024.

INDEX

Implementation of LIFO and FIFO buffer using mode control switch in Verilog HDL

Abstract:

This Verilog-based project introduces the "lifo_fifo" buffer module, crafted to offer adaptable data storage solutions within digital systems. This module is engineered to support both FIFO (First-In-First-Out) and LIFO (Last-In-First-Out) operational modes, allowing users to choose between them using a designated mode input. Functioning under synchronous logic, it boasts inputs for data input, read and write control, buffer enablement, and reset signals, alongside corresponding outputs that convey buffer status and data output. In FIFO mode, data insertion and retrieval are managed through a counter and pointers, while LIFO mode employs a stack-based approach facilitated by a stack pointer. Buffer status is determined by combinational logic, while sequential logic governs data handling and state transitions. With its adaptable nature, this module offers efficient buffering capabilities suitable for a wide array of digital systems, catering to diverse data processing needs through configurable FIFO and LIFO operation modes.

Keywords :

Verilog, buffer module, FIFO, LIFO, digital systems, synchronous logic, data storage, configurable, mode selector, counter, pointers, stack-based approach, combinational logic, sequential logic, data handling, state transitions.

Introduction :

Efficient data management is paramount in digital system design to achieve optimal performance. Buffers serve a crucial role by providing temporary storage for data flow within a system. Introducing the Verilog-based buffer module, lifo_fifo, designed to meet the diverse data buffering needs of digital systems. This module offers support for both FIFO (First-In-First-Out) and LIFO (Last-In-First-Out) operation modes, providing flexibility to suit various applications.

The lifo_fifo module, driven by synchronous logic, integrates inputs for data input, read and write control, buffer enable, and reset signals, along with corresponding outputs indicating buffer status and data output. Employing combinational and sequential logic, the module efficiently manages data insertion, retrieval, and state transitions, ensuring robust performance across different digital system environments. This introduction sets the stage for exploring the design, functionality, and versatility of the lifo_fifo buffer module, emphasizing its role in enhancing data processing capabilities within digital systems. With the increasing complexity and versatility of digital systems, there is a rising demand for buffer modules capable of supporting multiple operating modes to accommodate diverse application needs.

Objective :

The main goal of this project is to create and implement a flexible buffer module called lifo_fifo, capable of handling both FIFO and LIFO operations in digital systems. The project aims to achieve the following:

- Develop a Verilog-based buffer module that efficiently stores and retrieves data to meet various data processing needs in digital systems.
- Implement robust data handling mechanisms within the lifo_fifo module to ensure reliable operation and seamless integration into different digital system architectures.
- Design the lifo_fifo buffer module with flexibility, allowing users to choose between FIFO and LIFO operation modes based on specific application requirements.
- Ensure compatibility and easy integration with existing Verilog-based digital system designs, facilitating smooth deployment and utilization of the lifo_fifo buffer module in practical scenarios.
- Verify the functionality and performance of the lifo_fifo buffer module through rigorous simulation and testing methods, confirming its effectiveness in enhancing data processing capabilities within digital systems.

By accomplishing these objectives, the project aims to contribute to the advancement of digital system design by offering a versatile buffer module solution capable of meeting the diverse data handling requirements of modern applications.

Literature Survey:

Buffers are indispensable components in digital system design, serving as temporary repositories for managing data flow within the system. Among the array of buffer types, two prominent ones are FIFO (First-In-First-Out) and LIFO (Last-In-First-Out), each offering unique traits and applications.

FIFO Buffer:

A FIFO buffer operates akin to a queue, where data enters one end and exits from the other in the sequence it was added. This ensures that the earliest stored data is the first to be retrieved. Such buffers find extensive usage in scenarios demanding data processing in the order of its arrival, such as communication interfaces or task scheduling systems.

LIFO Buffer:

In contrast, a LIFO buffer behaves like a stack, with data being added to and retrieved from the same end. Here, the most recently added data is the first to be accessed. LIFO buffers are preferred in situations where the most recent data holds the highest significance or when data processing requires reverse chronological order, such as managing function call stacks or implementing undo operations in software applications.

Comprehending the attributes and utility of FIFO and LIFO buffers is pivotal for devising efficient mechanisms for data storage and retrieval within digital systems.

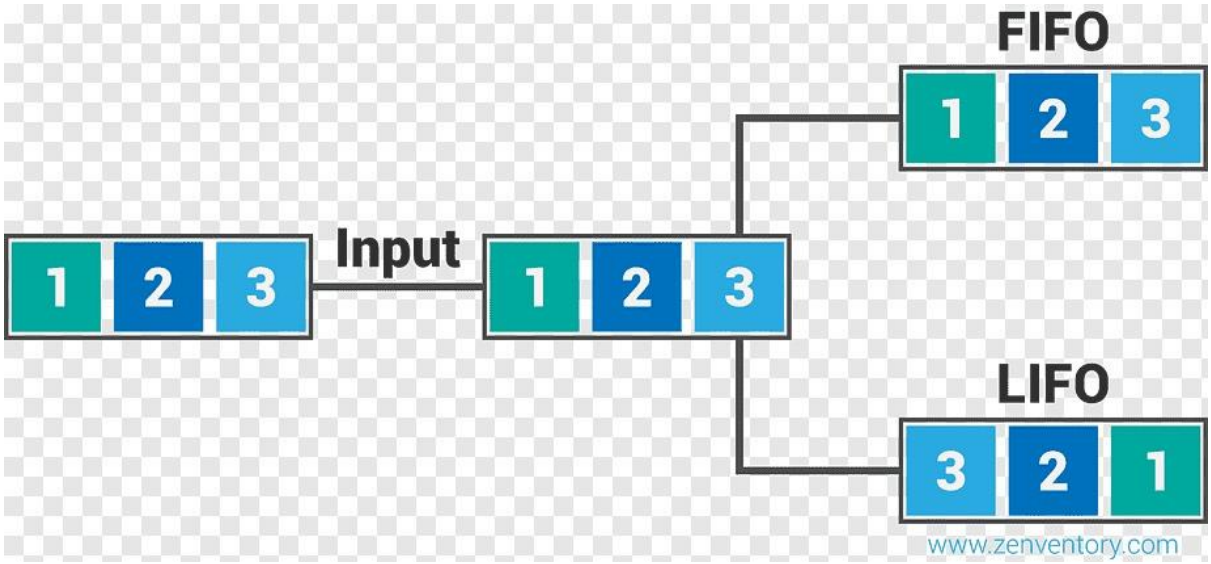


Fig. 1: Understanding FIFO & LIFO using block diagram.

Features Of FIFO & LIFO Buffer:

FIFO (First-In-First-Out) Buffer:

- Data processed in the order it was added.
- Operates like a queue: first data in, first data out.
- Ideal for scenarios needing sequential data processing, like communication interfaces and task scheduling.

LIFO (Last-In-First-Out) Buffer:

- Data processed in reverse chronological order.
- Behaves like a stack: last data in, first data out.
- Suited for situations where recent data is prioritized, like managing function call stacks and undo operations.

Flow of Code:

1. Module Declaration: The module lifo_fifo is declared with inputs and outputs, including mode (selector for FIFO or LIFO), clock (Clk), data input (dataIn), read and write enables (RD and WR), buffer enable (EN), reset (Rst), data output (dataOut), and flags for empty (EMPTY) and full (FULL) buffer conditions.

2. Buffer Signals and Variables Initialization: Registers and arrays are declared to manage the buffer data and control signals for both FIFO and LIFO modes. This includes Count for FIFO, FIFO array, readCounter and writeCounter for FIFO, stack_mem array, and SP for LIFO.

3. Combinational Logic for Empty and Full Flags: The EMPTY and FULL flags are determined based on the current buffer state and operation mode. If the buffer is empty or full, the corresponding flags are set accordingly.

4. Sequential Logic for Buffer Operations: Sequential logic determines the behaviour of buffer operations based on clock edges and control signals.

5. Reset Operation: When the reset signal (Rst) is asserted, the buffer is reset based on the selected mode. For FIFO mode, read and write counters are reset along with the buffer count. For LIFO mode, the stack pointer (SP) is reset.

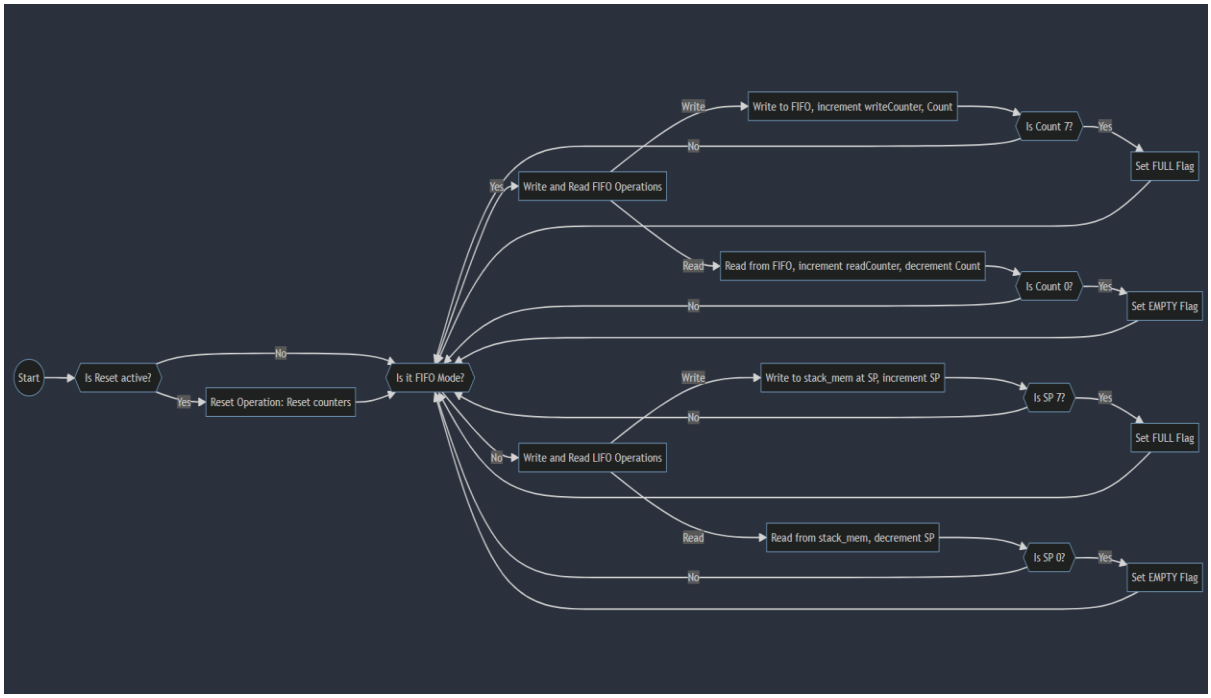
6. Buffer Operation Handling: During normal operation (when EN is asserted), the module performs buffer read and write operations based on the selected mode and control signals (RD and WR).

- FIFO Operations: If in FIFO mode, data is written to and read from the FIFO array. Write operations increment the write counter and count, while read operations increment the read counter and decrement the count.

- LIFO Operations: If in LIFO mode, data is pushed onto and popped from the stack. Write operations push data onto the top of the stack, incrementing the stack pointer. Read operations pop data from the top of the stack, decrementing the stack pointer.

This step-wise flow outlines how the Verilog code manages the buffer operations and state transitions for both FIFO and LIFO modes within the lifo_fifo module

Flow chart :



RTL View :

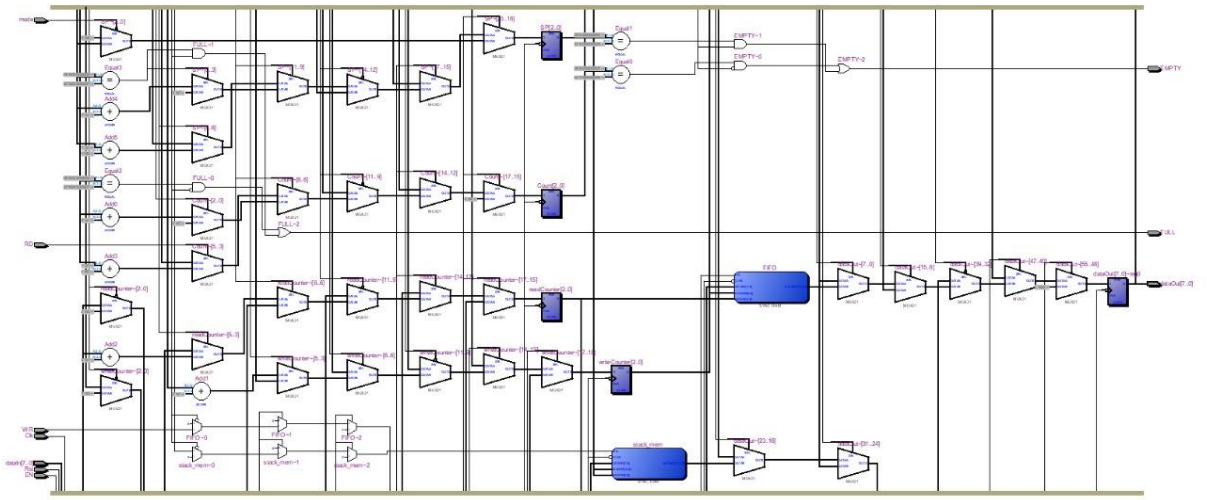


Fig. 2: RTL View

TTL View:

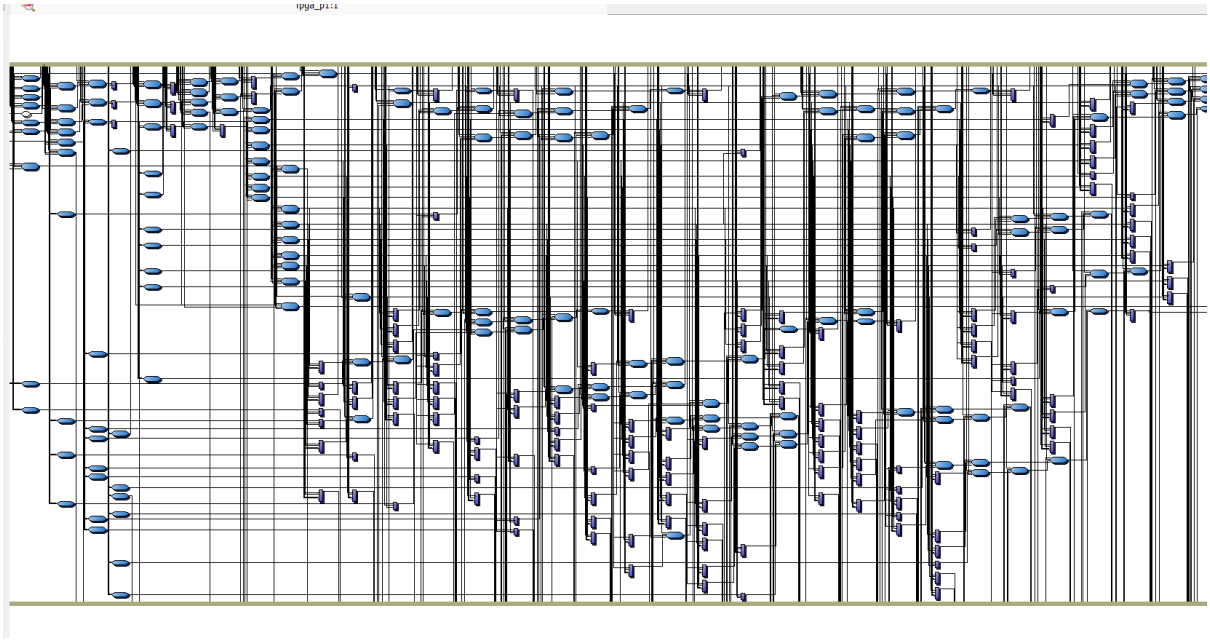


Fig. 3: TTL View

RTL Simulation:

For mode=0 [FIFO Buffer]

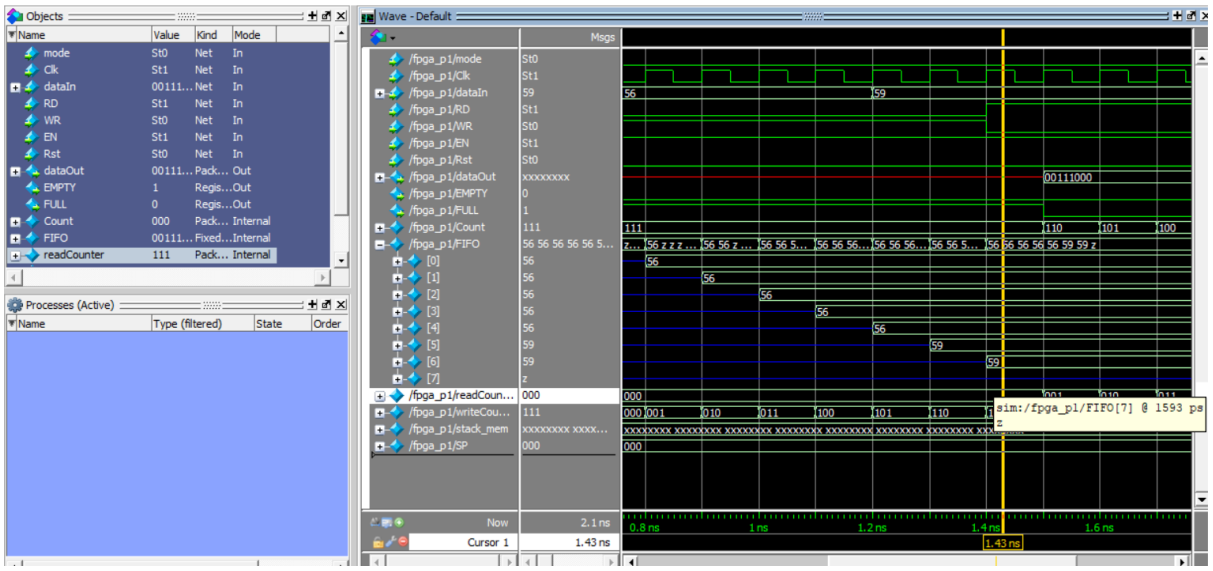


Fig.4: RTL Simulation of FIFO Operation (mode=0)

For mode=1 [LIFO Buffer]

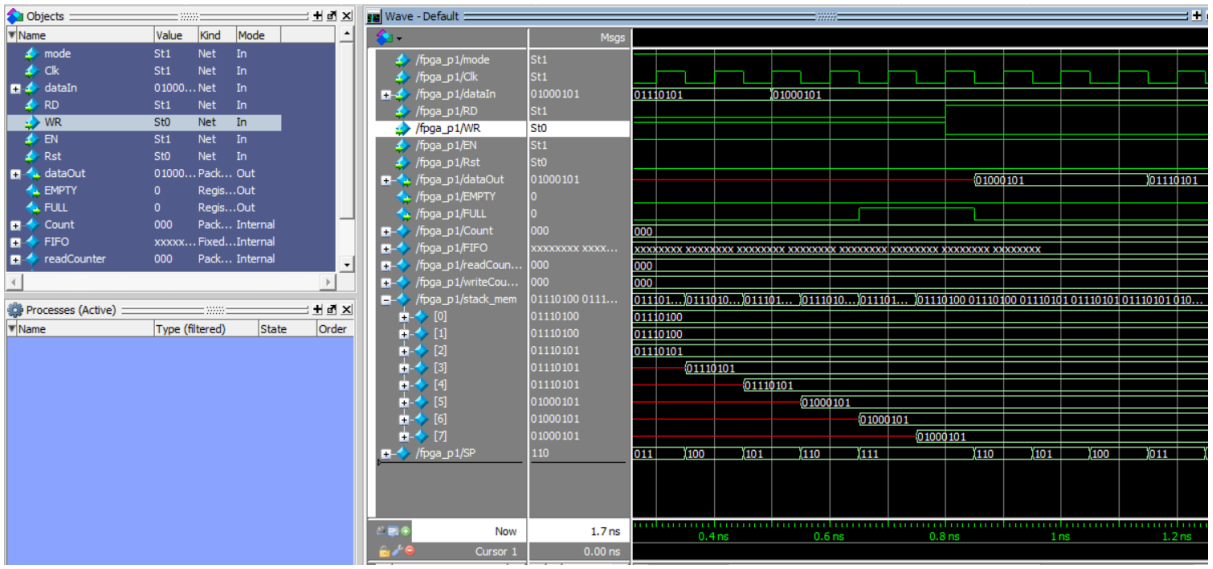


Fig.4: RTL Simulation of LIFO Operation (mode=1)

Limitations:

Despite its versatility and functionality, the lifo_fifo buffer module may exhibit certain limitations that warrant consideration in practical applications:

1. Fixed Buffer Size: The lifo_fifo buffer module has a fixed buffer size, which may limit its scalability and adaptability to applications requiring dynamic buffer sizing based on changing data processing requirements.
2. Synchronous Logic: The module relies on synchronous logic for data handling and state transitions, which may introduce timing constraints and limitations in high-speed or asynchronous systems.
3. Single-Clock Domain: The design operates within a single clock domain, which may pose challenges in systems with multiple clock domains or complex timing constraints.
4. Resource Utilization: Depending on the target FPGA device and synthesis settings, the resource utilization of the lifo_fifo module may vary, potentially leading to inefficient use of FPGA resources in some cases.

Acknowledging these limitations allows for a more informed assessment of the lifo_fifo buffer module's suitability for specific application domains and informs potential areas for further optimization or improvement in future iterations of the design.

Conclusion:

Through this project, we've gained a comprehensive understanding of buffering concepts, particularly FIFO and LIFO operations, within digital systems. Our proficiency in Verilog programming has been enhanced, allowing us to efficiently implement complex functionalities. Designing a buffer module supporting both FIFO and LIFO modes has emphasized the importance of flexibility and adaptability in meeting diverse data processing needs. We've also learned the significance of ensuring compatibility and seamless integration with existing Verilog-based designs, alongside rigorous testing methodologies to validate functionality and performance. Overall, this project has equipped us with the skills to design and implement versatile buffering solutions, contributing to advancements in digital system design practices.

References:

[1] GeeksforGeeks. FIFO vs LIFO approach in programming. Retrieved from https://www.geeksforgeeks.org/fifo-vs-lifo-approach-in-programming/

[2] Blogpost. Last-In-First-Out (LIFO) buffer. Retrieved from https://esrd2014.blogspot.com/p/last-in-first-out-buffer.html

[3] Silicon VLSI. First-In-First-Out (FIFO) buffer Verilog code. Retrieved from https://siliconvlsi.com/first-in-first-out-buffer-verilog-code/

[4] ChipVerify. Synchronous FIFO. Retrieved from https://www.chipverify.com/verilog/synchronous-fifo

Appendix:

Main Code:

```
module lifo_fifo(
    input mode,          // Mode selector (0 for FIFO, 1 for LIFO)
    input Clk,           // Clock
    input [7:0] dataIn,  // Data to be written
    input RD,            // Read enable
    input WR,            // Write enable
    input EN,            // Buffer enable
    input Rst,           // Reset
    output reg [7:0] dataOut,  // Data read from buffer
    output reg EMPTY,    // Buffer empty flag
    output reg FULL      // Buffer full flag
);

// FIFO buffer signals and variables
reg [2:0] Count = 0;
reg [7:0] FIFO [0:7];
reg [2:0] readCounter = 0, writeCounter = 0;

// LIFO buffer signals and variables
reg [7:0] stack_mem[0:7];
```

```
reg [2:0] SP = 0;


// Combinational logic for EMPTY and FULL
always @* begin
    EMPTY = (mode == 0 && Count == 0) || (mode == 1 && SP == 0);
    FULL = (mode == 0 && Count == 7) || (mode == 1 && SP == 7); // Updated comment: Buffer full condition
end


// Sequential logic for buffer operations
always @(negedge Clk) begin
    if (Rst) begin
        if (mode == 0) begin
            dataOut <= 8'b00000000;  // FIFO reset
            readCounter <= 0;
            writeCounter <= 0;
            Count <= 0;
        end else begin // LIFO reset
           dataOut <= 8'b00000000;
           SP <= 0;
                                Count <=0; // Initialize Stack Pointer to the size of the stack
        end
    end else if (EN) begin
        if (mode == 0) begin // FIFO operations
            if (WR) begin // Write operation
```

```
            FIFO[writeCounter] <= dataIn;

            if (Count == 7) begin
                Count <= 7;
            end else begin
                Count <= Count + 1;
                writeCounter <= writeCounter + 1;
            end
        end else if (RD) begin // Read operation
            dataOut <= FIFO[readCounter];
            readCounter <= readCounter + 1;
            Count <= Count - 1;
        end
    end else begin // LIFO operations
        if (WR) begin // Write operation
            stack_mem[SP] <= dataIn; // Push dataIn onto the top of the stack

            if (SP == 7) begin
                SP <= 7;
            end else begin
                SP <= SP + 1;
            end
        end else if (RD) begin // Read operation
            dataOut <= stack_mem[SP];
```

```
                SP <= SP - 1; // Increment the stack pointer after read operation
            end
        end
    end
end


endmodule
```