INNOVENTURES

Design and Implementation of an RTL FIFO Queue with Push/Pop Operations, Error Detection

KARNATI AMULYA
NAUREEN NAAZ
PULAK MISHRA
PURVI P

Project Objective and FIFO Overview

Objectives :

To design a synchronous FIFO queue with:

- Push/Pop operations
- Overflow/Underflow error detection
- Status flag management (full, empty)

FIFO (First-In First-Out):

- A type of queue where data exits in the order it was entered.
- Widely used in buffering, streaming, and inter-module communication

RTL FIFO Design

01

Modules Used:

• fifo_mem: memory array
• write_ptr, read_ptr: pointers for tracking position
• fifo_count: keeps track of number of stored items

02

Control Signals:

• push for writing
• pop for reading

03

Status Flags:

• full: high when FIFO is full
• empty: high when FIFO is empty

04

Error Flags:

• overflow_error: raised if push attempted when full
• underflow_error: raised if pop attempted when empty

VERILOG CODE HIGHLIGHTS :

MEMORY DECLARATION

```
reg [7:0] fifo_mem [0:FIFO_DEPTH-1];
```

PUSH LOGIC

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        write_ptr <= 0;
        overflow_error <= 0;
    end else if (push) begin
        if (full) begin
            overflow_error <= 1;
        end else begin
            fifo_mem[write_ptr] <= data_in;
            write_ptr <= write_ptr + 1;
            overflow_error <= 0;
        end
    end
end
```

POP LOGIC

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        read_ptr <= 0;
        underflow_error <= 0;
        data_out <= 0;
    end else if (pop) begin
        if (empty) begin
            underflow_error <= 1;
        end else begin
            data_out <= fifo_mem[read_ptr];
            read_ptr <= read_ptr + 1;
            underflow_error <= 0;
        end
    end
end
```

STATUS FLAG LOGIC

```
full  = (fifo_count == FIFO_DEPTH);
empty = (fifo_count == 0);
```

TESTBENCH

Clock and Reset Generation

```
initial clk = 0;
always #5 clk = ~clk;
```

• Clock is generated with a period of 10 ns.
• Reset initializes all the FIFO registers and pointers.

Push Task

```
task do_push(input [7:0] value);
    begin
        @(posedge clk);
        push <= 1;
        data_in <= value;
        @(posedge clk);
        push <= 0;
    end
endtask
```

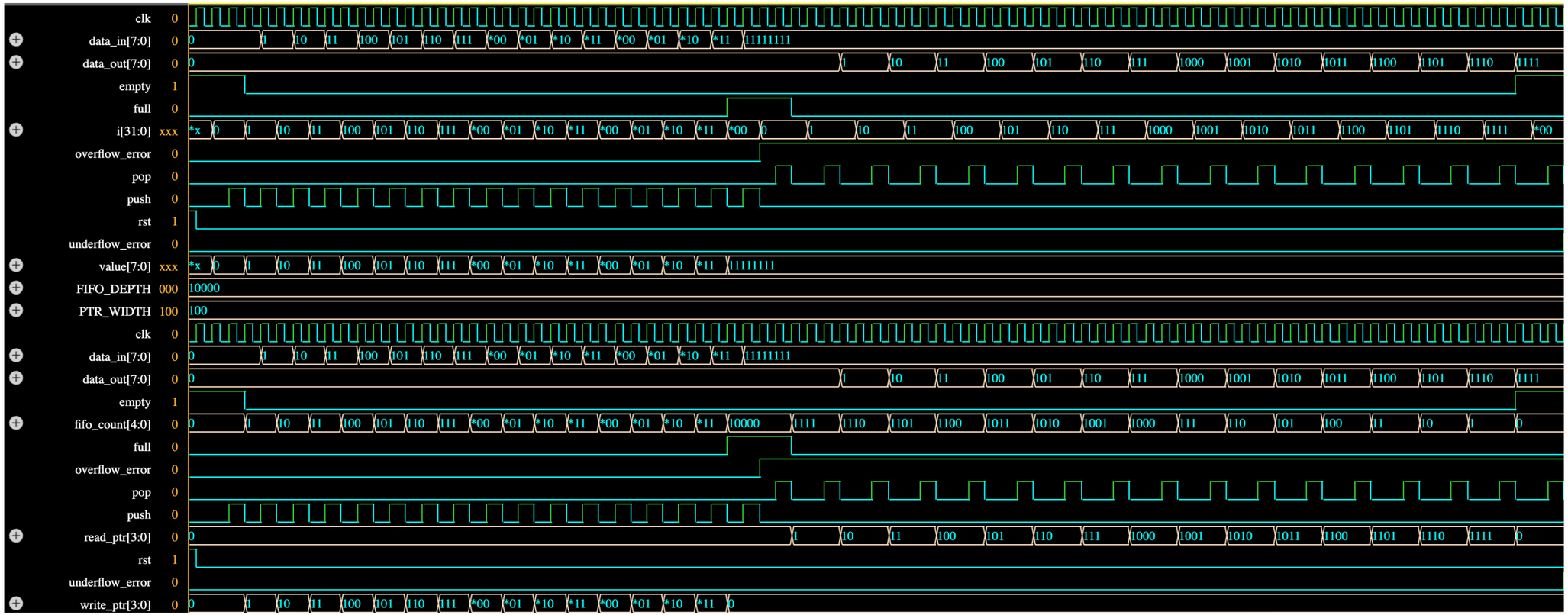
• Simplifies the testbench by allowing you to call do_push(value) to insert a byte into the FIFO.
• Push operation is synchronized to the clock edges.

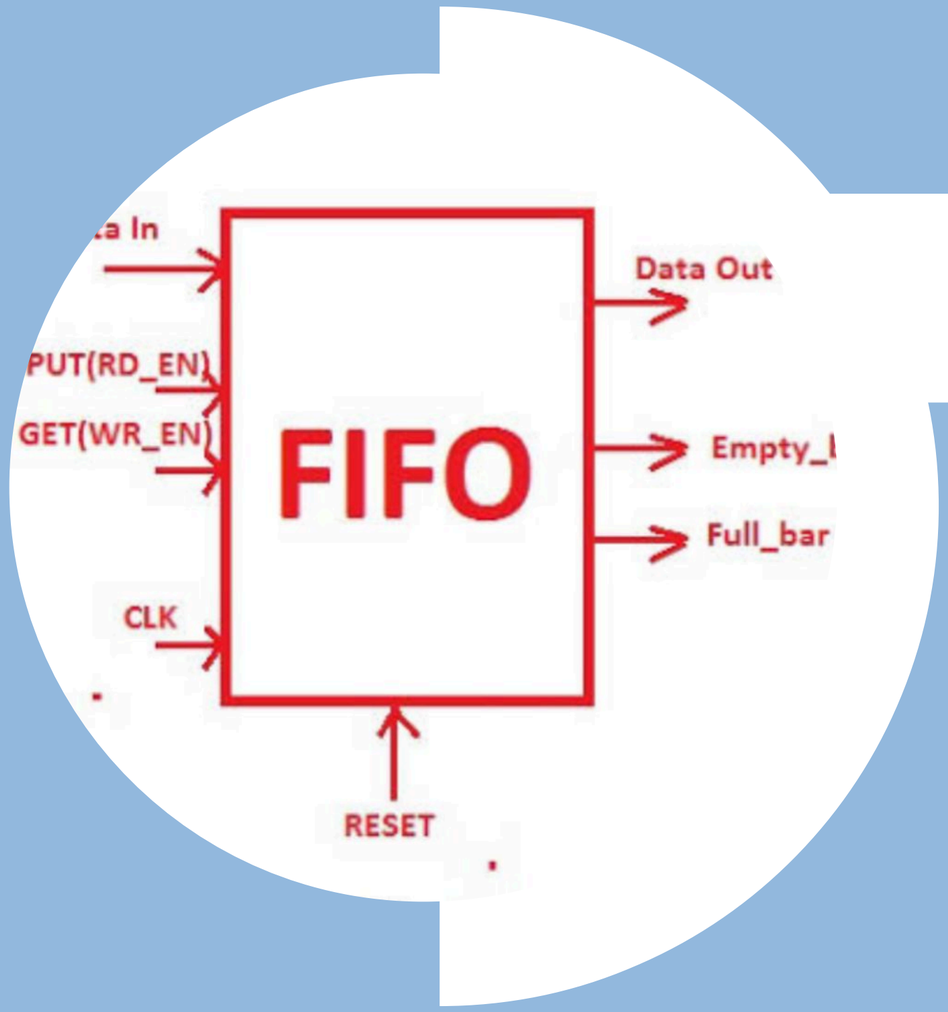
Pop Task

```
task do_pop;
    begin
        @(posedge clk);
        pop <= 1;
        @(posedge clk);
        pop <= 0;
    end
endtask
```

• Reads data from the FIFO one clock cycle at a time.

SIMULATION :


clk 0
data_in[7:0] 0
data_out[7:0] 0
empty 1
full 0
i[31:0] xxx
overflow_error 0
pop 0
push 0
rst 1
underflow_error 0
value[7:0] xxx
FIFO_DEPTH 000
PTR_WIDTH 100
clk 0
data_in[7:0] 0
data_out[7:0] 0
empty 1
fifo_count[4:0] 0
full 0
overflow_error 0
pop 0
push 0
read_ptr[3:0] 0
rst 1
underflow_error 0
write_ptr[3:0] 0



APPLICATIONS

1. Data Buffers in Communication Systems
2. DMA (Direct Memory Access) Controllers
3. Processor Pipelining
4. Real-Time Data Logging
5. Audio/Video Streaming
6.Asynchronous Clock Domain Crossing (CDC