PROJECT – SYNCHRONOUS FIFO

- BY RAJ KUMAR

OBJECTIVE

The goal of this project is to design, implement, and simulate a **Synchronous FIFO (First-In First-Out)** memory buffer using **Verilog HDL**. The FIFO will include programmable data width and depth, overflow/underflow detection, and support for simultaneous read and write in a synchronous clock domain.

THEORY

A **FIFO (First-In First-Out)** memory is a queue-based data structure commonly used in digital systems for **data buffering** between asynchronous or synchronous blocks.

A **Synchronous FIFO** is a First-In-First-Out queue in which there is a single clock pulse for both data write and data read. The number of rows is called depth or number of words of FIFO and number of bits in each row is called as width or word length of FIFO. This kind of FIFO is termed as Synchronous because the rate of read and write operations are same.

Basically Synchronous FIFO are used for High speed systems because of their high operating speed. Synchronous FIFO are easier to handle at high speed because they use free running clocks whereas in case of Asynchronous FIFO they uses two different clocks for read and write. Synchronous FIFO is more complex then the Asynchronous FIFO.

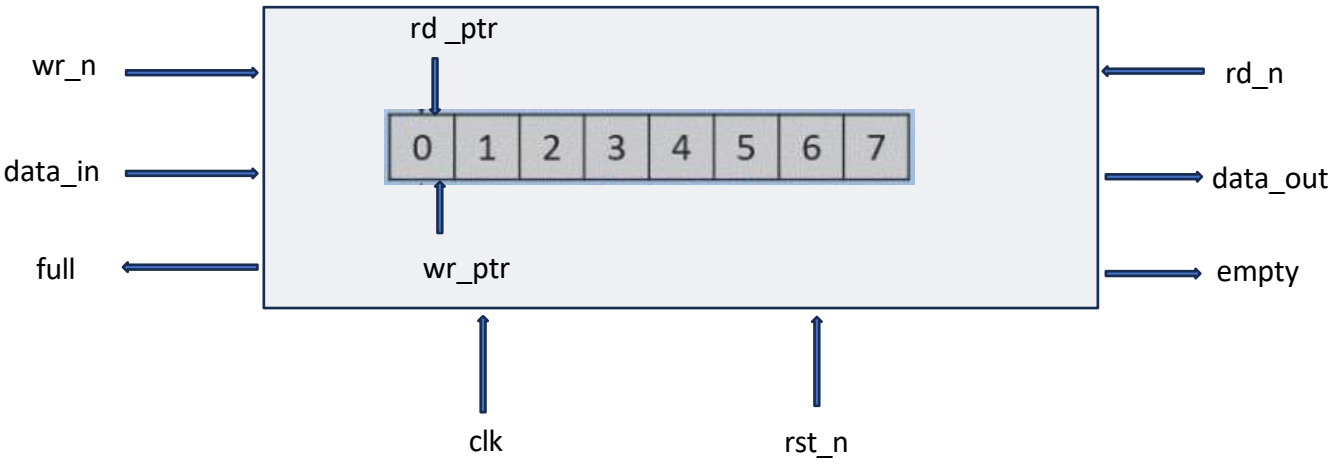
✅ **Key Concepts:**

- **Write Operation**: Data is written at the current write pointer location and the pointer is incremented.
- **Read Operation**: Data is read from the current read pointer location and the pointer is incremented.
- **Full Condition**: When the number of stored elements equals the maximum capacity.
- **Empty Condition**: When no elements are stored.
- **Overflow**: Occurs if a write is attempted when the FIFO is full.
- **Underflow**: Occurs if a read is attempted when the FIFO is empty.

✅ **Advantages:**

- Smooth communication between producer/consumer modules
- Avoids data loss with status flags
- Parameterized design enables reusability

BLOCK DIAGRAM



VERILOG CODE

```
// Synchronous FIFO


`define DATA_WIDTH 8         //size of each data
`define DEPTH 8              //no.of locations

module fifo (
   input clk,
   input rst_n,           //active low reset
   input rd_n,            //active low read enable
   input wr_n,           //active low write enable
   input [`DATA_WIDTH - 1:0] data_in,
   output reg [`DATA_WIDTH - 1:0] data_out,
   output reg underflow,
   output reg overflow
);

  //Read and write pointers and counters

  reg [$clog2(`DEPTH)-1:0] rd_ptr, wr_ptr;  // if DEPTH=8 -> $clog2(DEPTH)
gives 3,so these will hold 8 values(from 0 to 7)

  reg [$clog2(`DEPTH):0] count;   // No.of stored entries

  //Memory array declaration
  reg [`DATA_WIDTH-1:0] memory [`DEPTH-1:0];

  //Status signals
  reg full,empty;

 // Reading and writing of FIFO
always @(posedge clk or negedge rst_n) begin
```

```
    if (!rst_n) begin
        rd_ptr <= 0;
        wr_ptr <= 0;
        count <= 0;
    end
    else begin
        // WRITE
        if (!wr_n && count < `DEPTH) begin
            memory[wr_ptr] <= data_in;
            wr_ptr <= wr_ptr + 1;
            count <= count + 1;
        end

        // READ
        if (!rd_n && count > 0) begin
            rd_ptr <= rd_ptr + 1;
            count <= count - 1;
        end
    end
end

// Data output register
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        data_out <= 0;
    else if (!rd_n && count > 0)
        data_out <= memory[rd_ptr];
end

// FIFO Status (Full and Empty)
always @(*) begin
    full = (count == `DEPTH);
    empty = (count == 0);
end
```

```
// Status Flags (underflow and overflow)
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
      overflow <= 0;
      underflow <= 0;
   end
   else begin
      overflow <= (!wr_n && full);
      underflow <= (!rd_n && empty);
   end
end

endmodule
```

TESTBENCH

```
//Testbench for synchronous FIFO

`define DATA_WIDTH 8
`define DEPTH 8

module fifo_tb;

  reg clk;
  reg rst_n;
  reg rd_n;
  reg wr_n;
  reg [`DATA_WIDTH - 1 : 0] data_in;
  wire [`DATA_WIDTH - 1 : 0] data_out;
  wire underflow;
  wire overflow;

  // Instantiate the FIFO
```

```
 fifo my_dut (
   .clk(clk),
   .rst_n(rst_n),
   .rd_n(rd_n),
   .wr_n(wr_n),
   .data_in(data_in),
   .data_out(data_out),
   .underflow(underflow),
   .overflow(overflow)
 );

 // Clock generation
 always #5 clk = ~clk;

 integer i;

 initial begin


$display("\nTime\tclk\twr_n\trd_n\tdata_in\tdata_out\toverflow\tunderf
low");
   $monitor("%0t\t%b\t%b\t%b\t%h\t%h\t\t%b\t\t%b", $time, clk, wr_n,
rd_n, data_in, data_out, overflow, underflow);

   // Initial state
   clk = 0;
   rst_n = 0;
   wr_n = 1;
   rd_n = 1;
   data_in = 0;

   // Apply reset
   #12;
   rst_n = 1;
   #10;
```

```
// Write data into FIFO
$display("\n-- Writing to FIFO --");
for (i = 0; i < `DEPTH; i = i + 1) begin
@(posedge clk);
  data_in = i + 1;
  wr_n = 0; rd_n = 1;
  @(posedge clk);
  wr_n = 1;
end

// Attempt overflow
@(posedge clk);
$display("\n-- Attempting overflow --");
data_in = 8'hFF;
wr_n = 0; rd_n = 1;
@(posedge clk);
wr_n = 1;

// Read data from FIFO
$display("\n-- Reading from FIFO --");
for (i = 0; i < `DEPTH; i = i + 1) begin
@(posedge clk);
  wr_n = 1; rd_n = 0;
  @(posedge clk);
  rd_n = 1;
end

// Attempt underflow
@(posedge clk);
$display("\n-- Attempting underflow --");
wr_n = 1; rd_n = 0;
@(posedge clk);
rd_n = 1;

#20;
```

```
    $finish;
  end


 initial begin
        $fsdbDumpfile("file.fsdb");
        $fsdbDumpvars(0,fifo_tb);
 end
endmodule
```

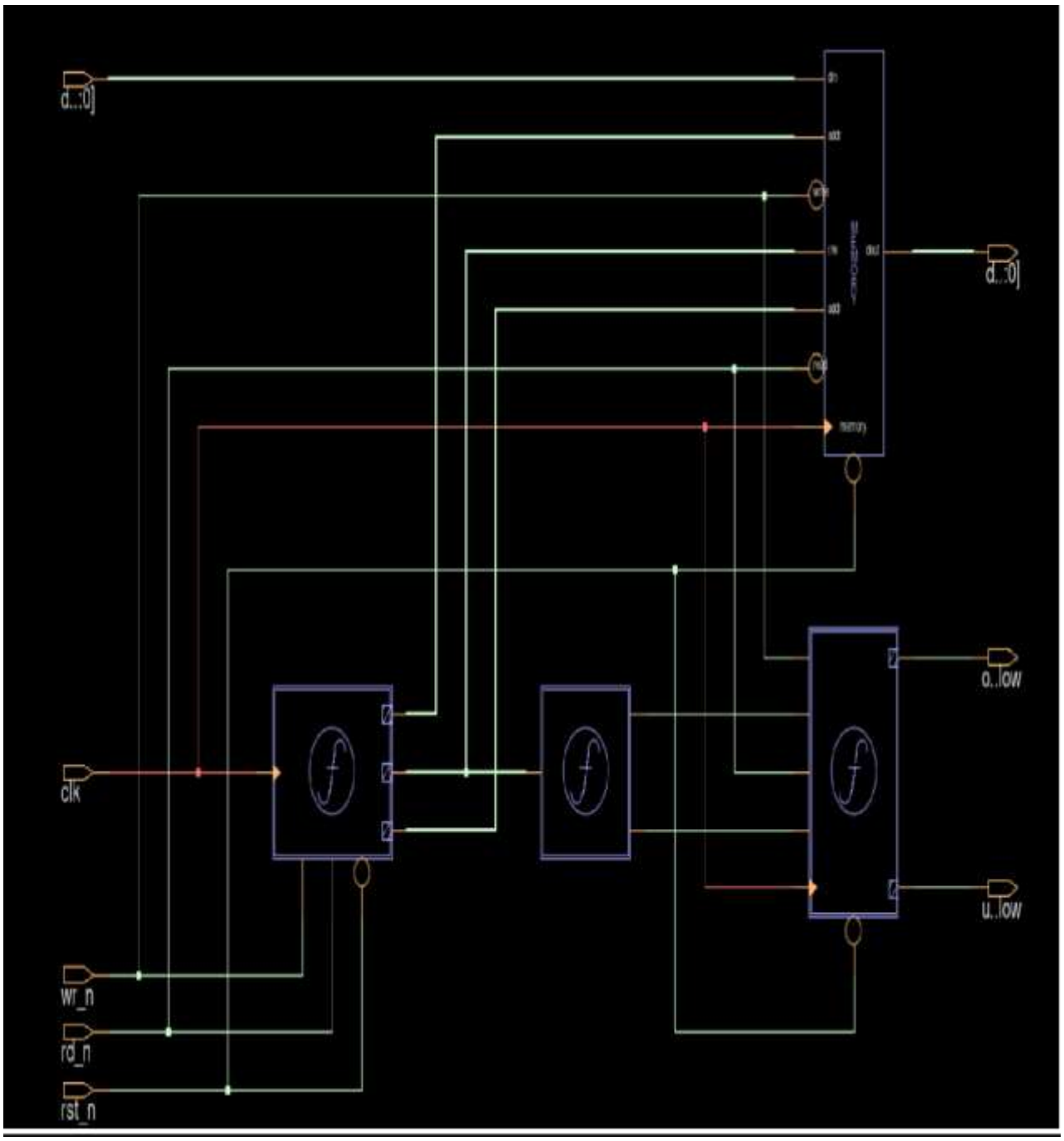
OUTPUT

Time	clk	wr_n	rd_n	data_in	data_out	overflow	underflow
0	0	1	1	00	00	0	0
5	1	1	1	00	00	0	0
10	0	1	1	00	00	0	0
15	1	1	1	00	00	0	0
20	0	1	1	00	00	0	0
-- Writing to FIFO --							
25	1	0	1	01	00	0	0
30	0	0	1	01	00	0	0
35	1	1	1	01	00	0	0
40	0	1	1	01	00	0	0
45	1	0	1	02	00	0	0
50	0	0	1	02	00	0	0
55	1	1	1	02	00	0	0
60	0	1	1	02	00	0	0
65	1	0	1	03	00	0	0
70	0	0	1	03	00	0	0
75	1	1	1	03	00	0	0
80	0	1	1	03	00	0	0
85	1	0	1	04	00	0	0
90	0	0	1	04	00	0	0
95	1	1	1	04	00	0	0
100	0	1	1	04	00	0	0
105	1	0	1	05	00	0	0
110	0	0	1	05	00	0	0
115	1	1	1	05	00	0	0
120	0	1	1	05	00	0	0
125	1	0	1	06	00	0	0
130	0	0	1	06	00	0	0
135	1	1	1	06	00	0	0
140	0	1	1	06	00	0	0
145	1	0	1	07	00	0	0
150	0	0	1	07	00	0	0
155	1	1	1	07	00	0	0
160	0	1	1	07	00	0	0
165	1	0	1	08	00	0	0
170	0	0	1	08	00	0	0
175	1	1	1	08	00	0	0
180	0	1	1	08	00	0	0

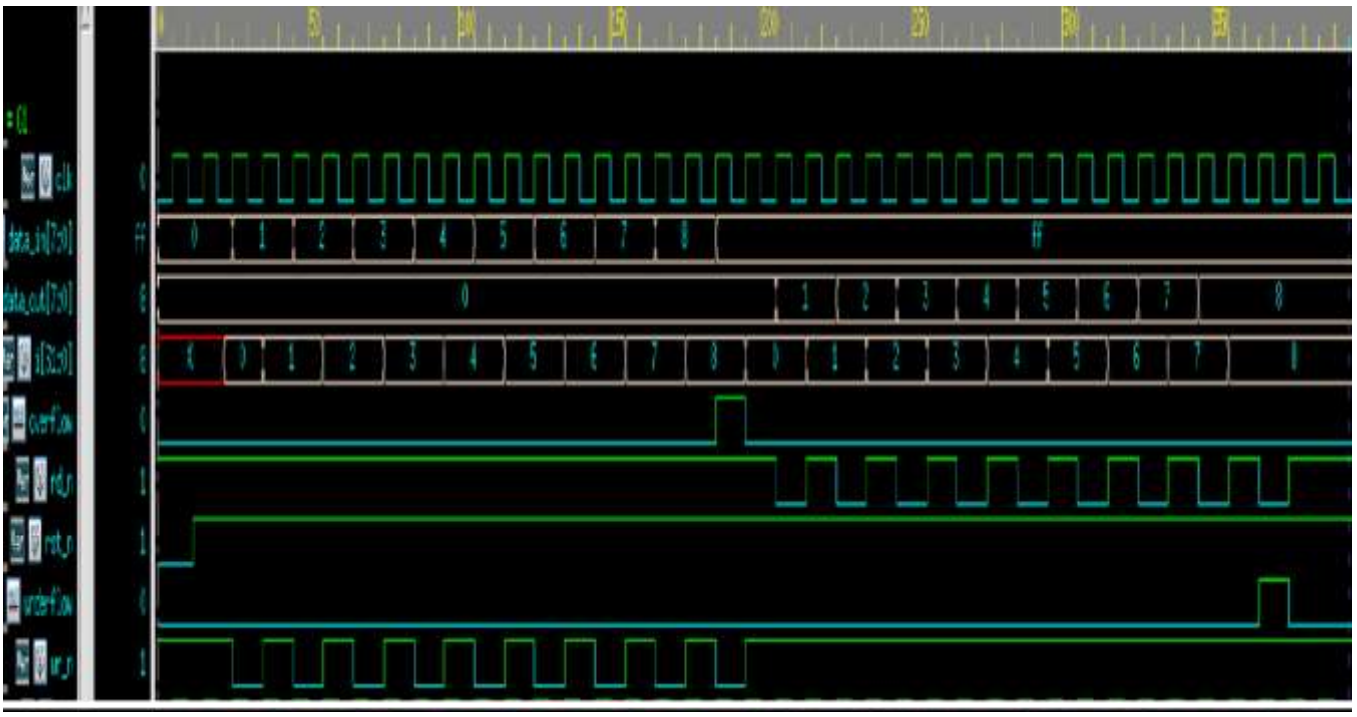
```
-- Attempting overflow --
185     1       0       1       ff      00              1               0
190     0       0       1       ff      00              1               0

-- Reading from FIFO --
195     1       1       1       ff      00              0               0
200     0       1       1       ff      00              0               0
205     1       1       0       ff      01              0               0
210     0       1       0       ff      01              0               0
215     1       1       1       ff      01              0               0
220     0       1       1       ff      01              0               0
225     1       1       0       ff      02              0               0
230     0       1       0       ff      02              0               0
235     1       1       1       ff      02              0               0
240     0       1       1       ff      02              0               0
245     1       1       0       ff      03              0               0
250     0       1       0       ff      03              0               0
255     1       1       1       ff      03              0               0
260     0       1       1       ff      03              0               0
265     1       1       0       ff      04              0               0
270     0       1       0       ff      04              0               0
275     1       1       1       ff      04              0               0
280     0       1       1       ff      04              0               0
285     1       1       0       ff      05              0               0
290     0       1       0       ff      05              0               0
295     1       1       1       ff      05              0               0
300     0       1       1       ff      05              0               0
305     1       1       0       ff      06              0               0
310     0       1       0       ff      06              0               0
315     1       1       1       ff      06              0               0
320     0       1       1       ff      06              0               0
325     1       1       0       ff      07              0               0
330     0       1       0       ff      07              0               0
335     1       1       1       ff      07              0               0
340     0       1       1       ff      07              0               0
345     1       1       0       ff      08              0               0
350     0       1       0       ff      08              0               0
355     1       1       1       ff      08              0               0
360     0       1       1       ff      08              0               0
```

SCHEMATIC



WAVEFORM



SIMULATION RESULTS

The simulation of the synchronous FIFO design was carried out using a structured testbench with the following key observations:

1. Reset and Initialization:
   - Initially, all control signals were inactive (wr_n = 1, rd_n = 1) and both data_in and data_out were 00h.
   - FIFO was empty with no flags set.
2. Write Operation:
   - Data values from 01h to 08h were sequentially written into the FIFO using wr_n = 0.
   - Write operations were done on alternate clock cycles to allow write pointer increment.
   - After 8 successful writes, FIFO reached its maximum capacity.
3. Overflow Condition:
   - A 9th write attempt (data_in = ffh) triggered the overflow flag, confirming FIFO is full and rejecting further writes.

4. Read Operation:
    - Read operations were then performed with rd_n = 0, sequentially retrieving data from the FIFO.
    - Data was correctly read in the same order it was written, confirming First-In First-Out (FIFO) behavior.
    - The data_out changed accordingly, while data_in remained unchanged (ffh) as no writes were performed during read.
5. Underflow Condition:
    - An extra read attempt after the FIFO was emptied triggered the underflow flag, indicating no data was available for reading.

CONCLUSION

We have successfully designed the Verilog code for a Synchronous FIFO which correctly handles read and write, underflow and overflow operation.
We have also observed the schematic diagram and waveform for our design.

-By RAJ KUMAR