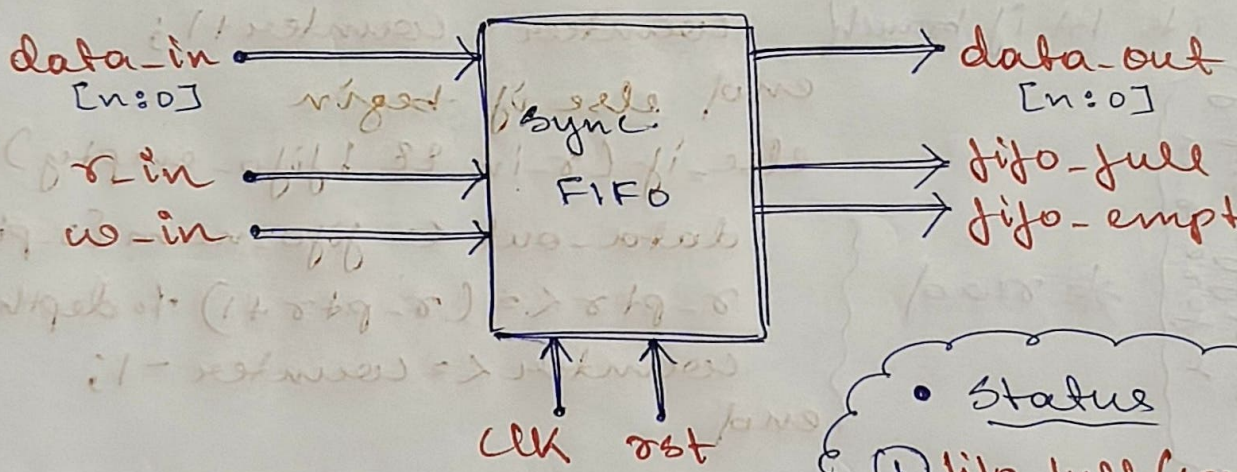
Synchronous FIFO (parameterizable width & depth)

- A single-clk synchronous FIFO (circular buffer) that accepts write (w_in) and reads (r_in) on same clk
- It tracks occupancy with a counter, uses w_ptr / r_ptr for circular addressing, then exposes fifo_full / fifo_empty and presents read data on data_out
- Reset (rst, active low) ⇒ clr pointers, counter and otp.



- Status
  1. fifo_full (counter == depth)
  2. fifo_empty (counter == 0)

- Code explanation

⇒ parameters & I/O,

- param - width, depth
- data_in and data_out
- control - w_in (write request), r_in (read request), clk, rst (active low)

⇒ internal state

- fifo_mem [0:depth-1] - storage array
- w_ptr, r_ptr - $clog2 depth - bit write / read pointer (wrap by 4: depth)
- counter - occupancy; width = $clog2(depth)+1

⇒ reset - always @(posedge clk or negedge rst)

```
if (!rst) begin
    counter=0; r_ptr=0; w_ptr=0;
    data_out=0;
end
```

⇒ Normal op^n -

```
if (w_in && !fifo_full) begin
    fifo_mem [w_ptr] <= data_in;
    w_ptr <= (w_ptr+1) % depth;
    counter = counter+1;
end
else if (r_in && !fifo_empty) begin
    data_out <= fifo_mem [r_ptr];
    r_ptr <= (r_ptr+1) % depth;
    counter <= counter -1;
end
```

* write takes presendence bcz its 1st if branch
* read

No back pressure handshake

- check fifo_full before w_in
- and fifo_empty before r_in

* ⇒ status outputs :- fifo_empty = (counter == 0)
  fifo_full = (counter == depth)

- implementat^n uses if ... else if so a cycle cannot perform both read & write - write has priority
- wrap around - modulo depth is used by pointer to perform circular Buffer
- No-Back pressure handshake