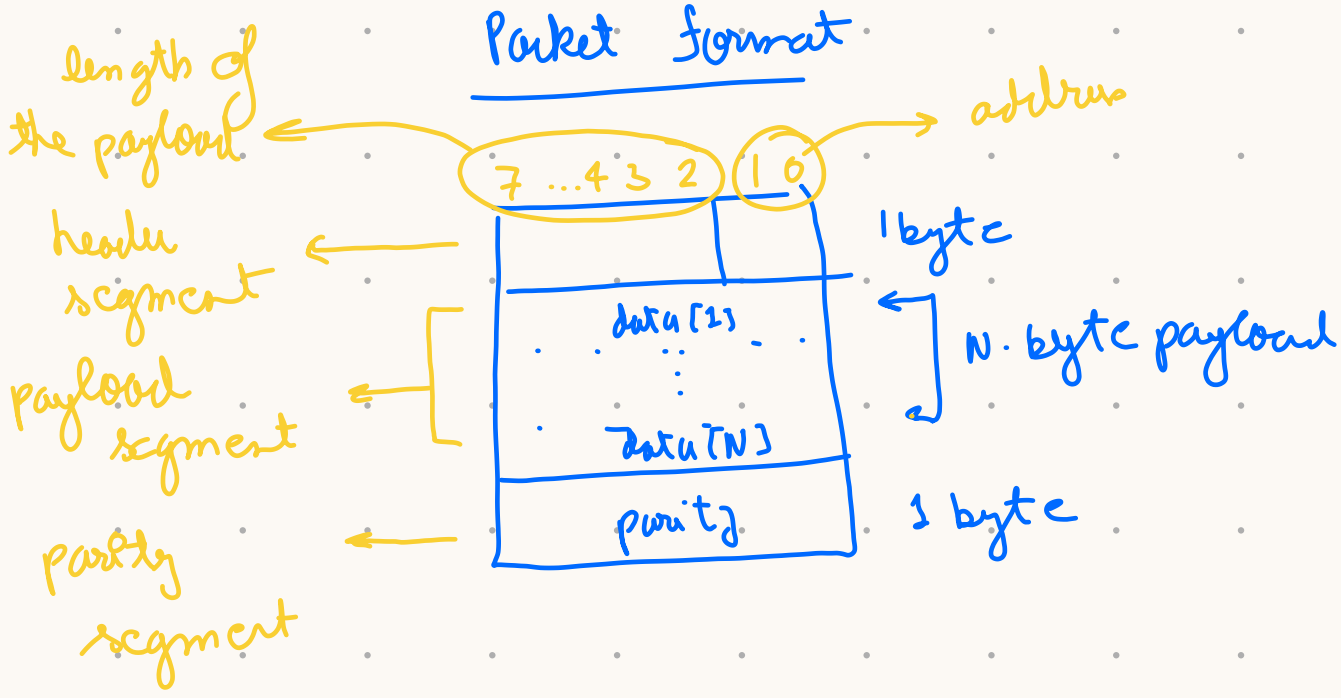
`timescale 1ns/1ps

module router_fifo #(

parameter Depth =
width =)

(input clk, resetn, soft_reset, lfd_state,
[width-1:0] data_in, wr_en, rd_en,

output reg [width-1:0] data_out,
empty, full);



reg [width:0] memory [0: Depth-1]

we added 1 extra bit while receiving to distinguish header segment from payload & parity. 9th bit : 1 → header

0 → payload / parity.

```
localparam addr_width = $clog2(Depth);
reg [addr_width:0] count;
```

- to track total no of items in FIFO
- it's (addr width + 1) bit, so we can actually count till Depth

i-e to count 64, we need 7 bit using 6 bits we can atmax count 63.

```
reg [addr_width-1:0] rd_ptr, wr_ptr;
reg [5:0] pkt_count;
```

- our header bit from bit 2 to 7 (6 bits) contain the count for length of packet so taken 6 bit pkt count size.

```
reg temp_ltd;
```

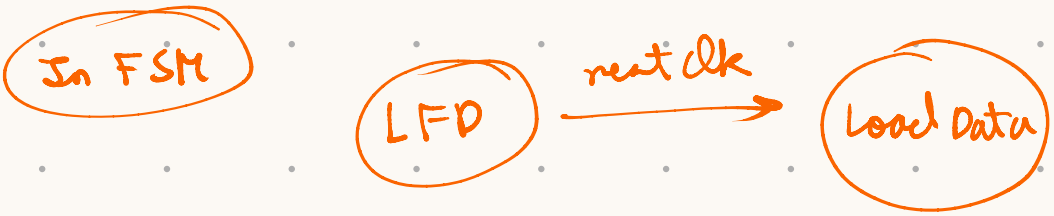
we are registering ltd (load first data) because its coming from another module (FSM); so need to be synchronized.

```
assign empty = (count == 0);
assign full = (count == Depth);
```

```
wire do_write = wr_en && !full;
wire do_read = rd_en && !empty;

always @(posedge clk)
    reset : temp_lfd → 0
    else : temp_lfd <= lfd_state
```

lfd state is high when first data byte of a packet (header) is being loaded, so that header can be handled specially.




```
always @(posedge clk)
do_write : memory[wr_ptr] <= {temp_lfd, data_in}
```

So in our memory 9th bit is to represent wether it's header or payload/parity.

```
always @(posedge clk or negedge reset_n)
```

Standard pointer (rd, wr, & count) increment logic

↓

case 2 : (!resetn || soft_rst)

case 2 : (do_write && !do_read)

case 3 : (!do_write && do_read)

case 4 : (do_write && do_read).

always @ (posedge clk)

if !resetn : data_out <= 8'd0 ;
pkt_count <= 0;

else if soft_reset : data_out <= 8'dz ;
pkt_count <= 0 ;

else if do_read : data_out <= memory [rd_ptr][7:0]

output only 8 bit data

if (memory[rd_ptr][8]) :

looking for header

pkt_count <= memory[rd_ptr][7:2] + 1'b1

1 extra for parity bit

if (pkt_count != 0)

pkt_count <= pkt_count - 1;

if (pkt_count == 1 && do_read)

data_out <= 8'bz

no need to output parity bit.

endmodule