```
1  // ee417 lesson 11 Assignment 1 L11A1
2  // Name: Ron Kalin, Date: 07-25-24  Group: Kalin/Jammeh
3  // Design: FIFO using common clock for reading and writing, with reset.
4  // FIFO has input & output data ports, & flags denoting status of the stack (full or empty).
5  // FIFO module should not support simultaneous read & write, and gives preference to the
   read operation.
6  // parameters are used for the stack height and width. starting point: data width=5 & stack
   height=8
7  module FIFO ( Data_out,        // data path from FIFO
8                stack_empty,     // flag asserted high for empty stack
9                stack_almost_empty,
10               stack_full,      // flag asserted high for full stack
11               stack_almost_full,
12               Data_in,         // data path into FIFO
13               write_to_stack, // input controlling write to stack
14               read_from_stack,// input controlling read to stack
15               clk,             // clock
16               rst );           // reset
17 parameter stack_width  = 5;    // width of stack and data points (bit length of word)
18 parameter stack_height = 8;    // height of stack (# of words)
19 parameter stack_ptr_width = $clog2(stack_height); //3;// pointer width stack addresses,
   log2 fcn = no. bits to represent addr for all values in stack
20
21 output [stack_width -1 : 0]     Data_out;
22 output                         stack_empty, stack_full;
23 output                         stack_almost_empty, stack_almost_full;
24 input  [stack_width -1 : 0]     Data_in;
25 input                          write_to_stack, read_from_stack;
26 input                          clk, rst;
27
28 // Pointers for reading and writing
29 reg  [stack_ptr_width -1 : 0]  read_ptr, write_ptr;
30 reg  [stack_ptr_width    : 0]  ptr_diff; //pointer difference
31 reg  [stack_width -1 : 0]      Data_out; //declaring as output register
32 reg  [stack_width -1 : 0]      stack [stack_height -1 : 0]; // memory array
33
34 /*assign stack_empty = (ptr_diff == 0) ? 1 : 0; //1'b1 : 1'b0; //single bit setup
35 assign stack_full  = (ptr_diff == stack_height) ? 1 : 0; //1'b1 : 1'b0;*/
36
37 assign stack_empty = (ptr_diff == 0) ? 1 : 0; //stack fullness indicators
38 assign stack_full  = (ptr_diff == stack_height) ? 1 : 0;
39 assign stack_almost_empty = (ptr_diff == 1) ? 1 : 0;
40 assign stack_almost_full  = (ptr_diff == stack_height-1) ? 1 : 0;
41
42 always @ (posedge clk or posedge rst)
43 begin
44 if (rst) begin
45      Data_out  <= 0; // reset output and pointers to zero
46      read_ptr  <= 0;
47      write_ptr <= 0;
48      ptr_diff  <= 0;
49 end
50
51 else  begin
52       if ( (read_from_stack) && (!stack_empty) ) //prioritize reading over writing
53           begin  Data_out  <= stack [read_ptr];//send stack to data_out
54                  read_ptr  <= read_ptr + 1;    // incr read pointer
55                  ptr_diff  <= ptr_diff - 1;    // decr pointer diff
56           end
57       else if  ( (write_to_stack) && (!stack_full) )
58           begin  stack [write_ptr] <= Data_in; //send stack to data_in
59                  write_ptr         <= write_ptr + 1;// incr write pointer
60                  ptr_diff          <= ptr_diff + 1; // incr pointer diff
61           end
62    end
63 end
64 endmodule
```