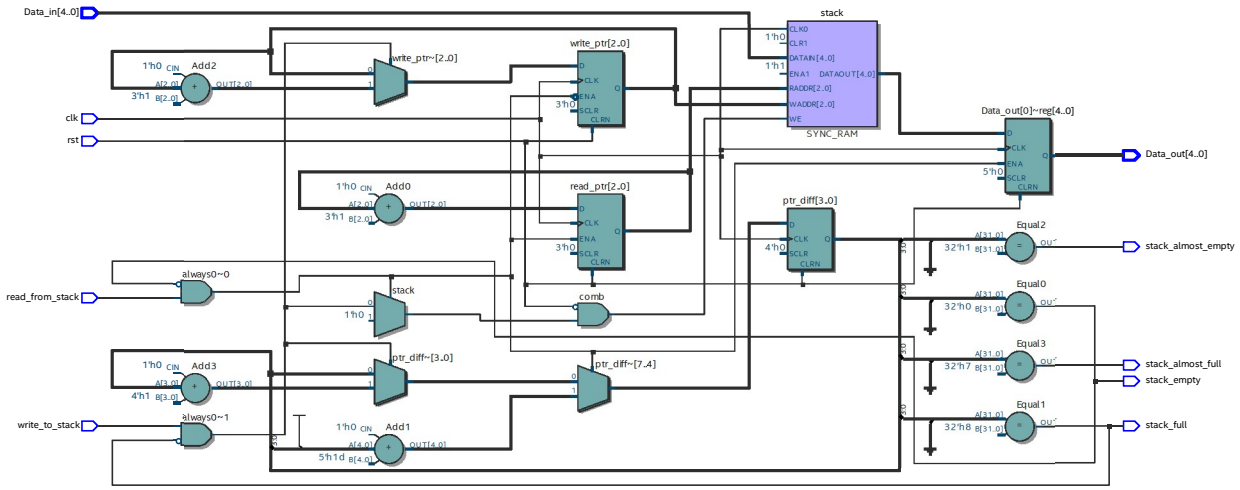
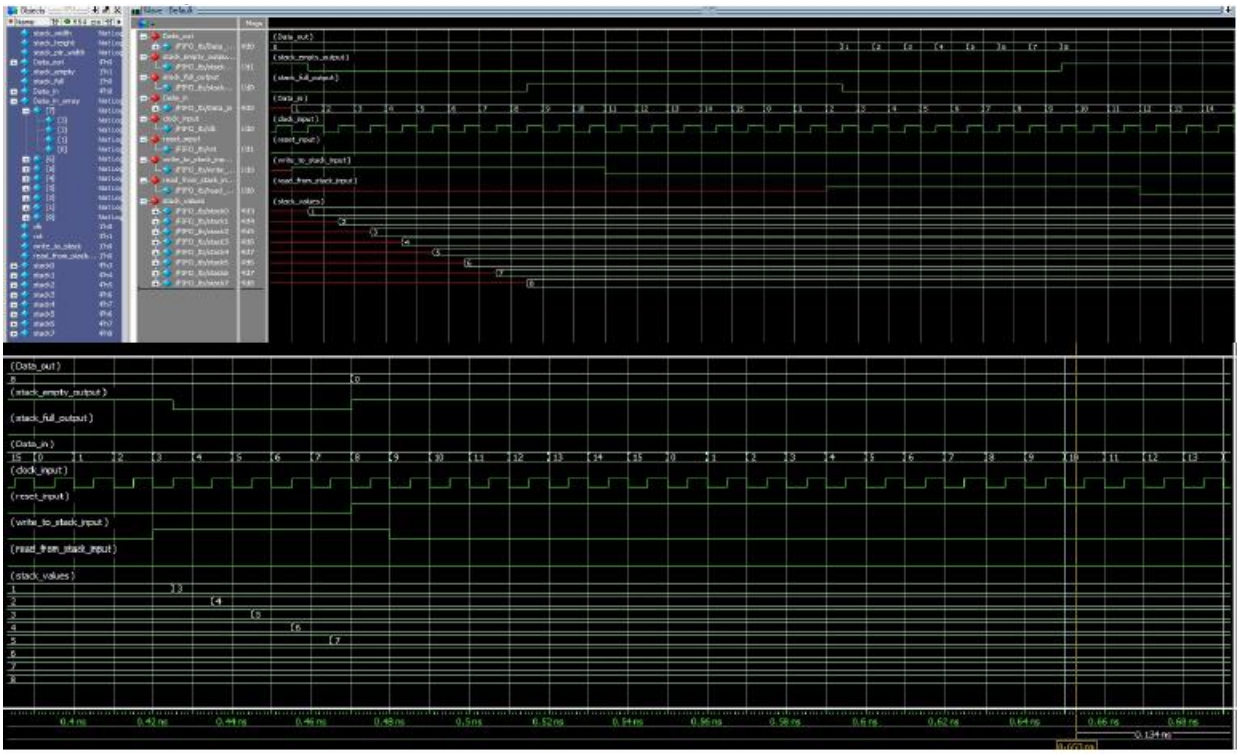
```
1  // ee417 lesson 11 Assignment 1 L11A1
2  // Name: Ron Kalin, Date: 07-25-24  Group: Kalin/Jammeh
3  // Design: FIFO using common clock for reading and writing, with reset.
4  // FIFO has input & output data ports, & flags denoting status of the stack (full or empty).
5  // FIFO module should not 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
```


Data_in[4..0]
clk
rst
read_from_stack
write_to_stack
Add2
Add0
Add3
Add1
write_ptr~[2..0]
write_ptr[2..0]
read_ptr[2..0]
always0~0
always0~1
stack
comb
ptr_diff~[3..0]
ptr_diff~[7..4]
ptr_diff[3..0]
SYNC_RAM
Data_out[0]~reg[4..0]
Data_out[4..0]
Equal2
Equal0
Equal3
Equal1
stack_almost_empty
stack_almost_full
stack_empty
stack_full

```
1  //Name: Ron Kalin  EE417 Summer 2024, Date: 07/25/24
2  //Group: Ron Kalin/ Lamin Jammeh
3  //Lesson 11 HW Question 01. Project: test-bench for FIFO
4  module FIFO_tb ();
5  parameter   stack_width =  5;   // width of stack and data paths (word length)
6  parameter   stack_height = 8;   // height of stack (# of words)
7  parameter stack_ptr_width = $clog2(stack_height); //3;// width of pointer to address stack
8  //declare outputs and inputs
9  wire  [stack_width -1 : 0]  Data_out;
10 wire                        stack_empty, stack_full;
11 wire                        stack_almost_empty, stack_almost_full;
12 reg   [stack_width -1 : 0]  Data_in;
13 reg   [stack_width -1 : 0]  Data_in_array [stack_height-1 : 0]; //internal register array
   of data
14 reg                         clk, rst;
15 reg                         write_to_stack, read_from_stack;
16
17 wire  [stack_width-1: 0]    stack0, stack1, stack2, stack3,
18                             stack4, stack5, stack6, stack7;
19 wire [stack_ptr_width-1: 0] read_ptr, write_ptr;
20 wire [stack_ptr_width  : 0] ptr_diff; //pointer difference
21 integer                     j;
22
23 //initialize the unit under test M1
24 FIFO  M1 ( Data_out, stack_empty, stack_almost_empty, stack_almost_full, stack_full,
   Data_in,
25            write_to_stack, read_from_stack, clk, rst );
26 // assign probe wires for troubleshooting and visibility
27 assign   stack0  = M1.stack [0];
28 assign   stack1  = M1.stack [1];
29 assign   stack2  = M1.stack [2];
30 assign   stack3  = M1.stack [3];
31 assign   stack4  = M1.stack [4];
32 assign   stack5  = M1.stack [5];
33 assign   stack6  = M1.stack [6];
34 assign   stack7  = M1.stack [7];
35 assign  read_ptr = M1.read_ptr;
36 assign write_ptr = M1.write_ptr;
37 assign  ptr_diff = M1.ptr_diff;
38
39 always begin clk = 0; forever #5 clk = ~clk; end  // create clock
40 initial  #1500 $stop;
41
42 initial begin
43    #10   rst = 1;   #40 rst = 0;    #420  rst = 1;   # 460 rst = 0;  end  // cycle reset
44
45 initial begin
46    $readmemb ("FIFO_Input_Data.txt", Data_in_array); // read data to Data_in_array from
   text file saved in simulation path
47    end
48
49 initial begin
50 //#80 Data_in = 1; forever #10 Data_in = Data_in + 1;  //manual method for creating Data_in
51    #80 Data_in = 1;
52    for (j=0; j<stack_height; j=j+1)
53      begin
54        Data_in = Data_in_array[j]; // data in coming from text file
55        $display("Data_out: %u, Data_in: %u", Data_out, Data_in );
56      end //for
57 end
58
59 initial fork  // cycling read/write inputs
60    # 80 write_to_stack  = 1;
61    #180 write_to_stack  = 0;
62    #250 read_from_stack = 1;
63    #350 read_from_stack = 0;
64    #420 write_to_stack  = 1;
65    #480 write_to_stack  = 0;
66   join
```

```
67
68    //display results
69    always @(posedge clk)
70    begin
71      $display("Data_out: %u, Data_in: %u, Data_in_array: %u, write_to_stack: %b,
      read_from_stack: %b, stack_empty: %b, stack_full: %b",
72                Data_out, Data_in, Data_in_array, write_to_stack, read_from_stack, stack_empty,
      stack_full );
73    end
74
75    endmodule
```


{Data_out}
{stack_empty_output}
{stack_full_output}
{Data_in}
{clock_input}
{reset_input}
{write_to_stack_input}
{read_from_stack_input}
{stack_values}
0.4 ns
0.42 ns
0.44 ns
0.46 ns
0.48 ns
0.5 ns
0.52 ns
0.54 ns
0.56 ns
0.58 ns
0.6 ns
0.62 ns
0.64 ns
0.66 ns
0.68 ns
0.134 ns