CE-325: Digital System Design

Mid-Term Exam Q6 Take Home Part

Huzaifah Tariq Ahmed - ha07151

15th March 2024

1 FIFO (NON-FSM)

The fifo non fsm module is a Verilog implementation of a non-FSM based FIFO buffer. It lacks explicit state transitions commonly found in FSM-based designs. The module includes input/output ports for data transfer and signals indicating FIFO fullness or emptiness. Parameters define stack characteristics such as element width, maximum capacity, and pointer width. Internal registers manage read and write pointers, along with a gap register tracking their difference. Data read from the stack is stored in a register for output. Functionality is driven by clock edges or reset signals. During reset, pointers and data registers are initialized to zero. Conditional execution of write and read operations is based on stack status and requested operations.

1.1 Code

```
`timescale 1ns/1ps

// Define the module named fifo_non_fsm with
// specified input and output ports
module fifo_non_fsm(
    Data_out,
    stack_full,
    stack_empty,
    Data_in,
    write_to_stack,
    read_from_stack,
    clk,
    rst
);

    // Parameters defining the width and height of the stack,
    // and pointer width
    parameter stack_width = 8;
    parameter stack_height = 32;
    parameter stack_ptr_width = 5;

    // Output ports
```

```
23      output [stack_width-1:0] Data_out;
24      output stack_full, stack_empty;
25
26      // Input ports
27      input [stack_width-1:0] Data_in;
28      input write_to_stack, read_from_stack, clk, rst;
29
30      // Registers for read and write pointers, and pointer gap
31      reg [stack_ptr_width-1:0] read_ptr, write_ptr;
32      reg [stack_ptr_width:0] ptr_gap;
33
34      // Register to hold data read from the stack
35      reg [stack_width-1:0] Data_out;
36
37      // Memory array representing the stack
38      reg [stack_width-1:0] stack [stack_height-1:0];
39
40      // Check if stack is full and empty
41      assign stack_full = (ptr_gap == stack_height);
42      assign stack_empty = (ptr_gap == 0);
43
44      // Always block triggered on positive edge of clock or reset signal
45      always@(posedge clk or posedge rst)
46      if(rst)
47      begin
48          // Reset: Initialize data out, read pointer, write
49          // pointer, and pointer gap
50          Data_out <= 0;
51          read_ptr <= 0;
52          write_ptr <= 0;
53          ptr_gap <= 0;
54      end
55      else if (write_to_stack && (!stack_full) && (!read_from_stack))
56      begin
57          // Write data into the stack if it's not full and not being read
58          stack[write_ptr] <= Data_in;
59          write_ptr <= write_ptr + 1;
60          ptr_gap <= ptr_gap + 1;
61      end
62      else if ((!write_to_stack) && (!stack_empty) && read_from_stack)
63      begin
64          // Read data from the stack if it's not empty and being read
65          Data_out <= stack[read_ptr];
66          read_ptr <= read_ptr + 1;
67          ptr_gap <= ptr_gap - 1;
68      end
69      else if (write_to_stack && read_from_stack && stack_empty)
70      begin
71          // Write data into the stack if it's empty and being
72          // read and written simultaneously
```

```
73          stack[write_ptr] <= Data_in;
74          write_ptr <= write_ptr + 1;
75          ptr_gap <= ptr_gap + 1;
76      end
77      else if(write_to_stack && read_from_stack && stack_full)
78      begin
79          // Read data from the stack if it's full and being
80          // read and written simultaneously
81          Data_out <= stack[read_ptr];
82          read_ptr <= read_ptr + 1;
83          ptr_gap <= ptr_gap - 1;
84      end
85      else if(write_to_stack &&
86              read_from_stack &&
87              (!stack_full) &&
88              (!stack_empty))
89      begin
90          // Read data from the stack and write new data
91          // simultaneously if it's not full or empty
92          Data_out <= stack[read_ptr];
93          stack[write_ptr] <= Data_in;
94          read_ptr <= read_ptr + 1;
95          write_ptr <= write_ptr + 1;
96      end
97  endmodule
```

1.2 Testbench

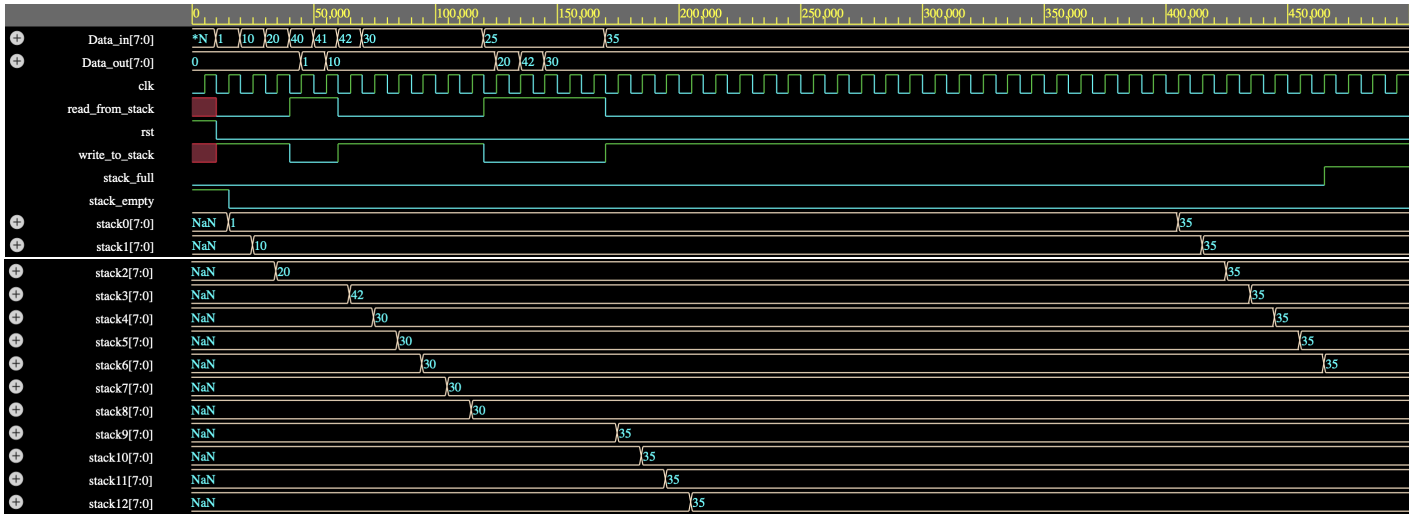
```
1  // Set the timescale for simulation
2  `timescale 1ns/1ps
3
4  // Include the non-fsm fifo module
5  `include "fifo_non_fsm.v"
6
7  // Define the testbench module
8  module tb_fifo_non_fsm();
9
10   // Parameters defining the stack width, height, and pointer width
11   parameter stack_width = 8;
12   parameter stack_height = 32;
13   parameter stack_ptr_width = 5;
14
15   // Output wires for data out and stack status
16   wire [stack_width-1:0] Data_out;
17   wire stack_full, stack_empty;
18
19   // Input registers for data in, write and read signals, clock, and reset
20   reg [stack_width-1:0] Data_in;
21   reg write_to_stack, read_from_stack;
```

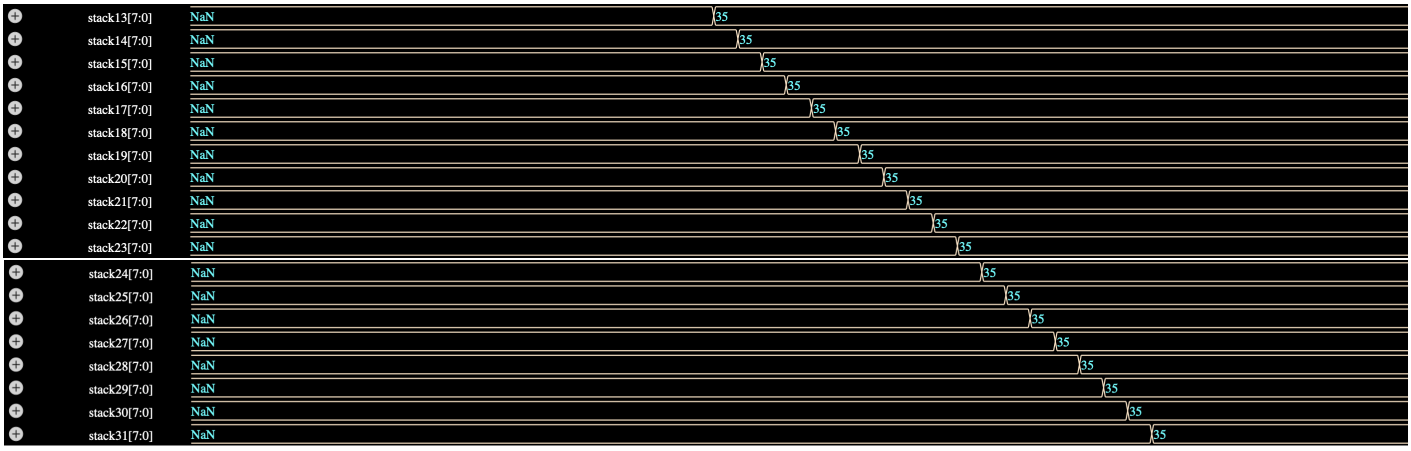
```
22    reg clk, rst;
23
24    // Wires to access individual stack elements
25    wire [stack_width-1:0] stack0,
26                           stack1,
27                           stack2,
28                           stack3,
29                           stack4,
30                           stack5,
31                           stack6,
32                           stack7,
33                           stack8,
34                           stack9,
35                           stack10,
36                           stack11,
37                           stack12,
38                           stack13,
39                           stack14,
40                           stack15,
41                           stack16,
42                           stack17,
43                           stack18,
44                           stack19,
45                           stack20,
46                           stack21,
47                           stack22,
48                           stack23,
49                           stack24,
50                           stack25,
51                           stack26,
52                           stack27,
53                           stack28,
54                           stack29,
55                           stack30,
56                           stack31;
57
58    // Assignments to access individual stack elements from the fifo module
59    assign stack0 = M1.stack[0];
60    assign stack1 = M1.stack[1];
61    assign stack2 = M1.stack[2];
62    assign stack3 = M1.stack[3];
63    assign stack4 = M1.stack[4];
64    assign stack5 = M1.stack[5];
65    assign stack6 = M1.stack[6];
66    assign stack7 = M1.stack[7];
67    assign stack8 = M1.stack[8];
68    assign stack9 = M1.stack[9];
69    assign stack10 = M1.stack[10];
70    assign stack11 = M1.stack[11];
71    assign stack12 = M1.stack[12];
```

```
72    assign stack13 = M1.stack[13];
73    assign stack14 = M1.stack[14];
74    assign stack15 = M1.stack[15];
75    assign stack16 = M1.stack[16];
76    assign stack17 = M1.stack[17];
77    assign stack18 = M1.stack[18];
78    assign stack19 = M1.stack[19];
79    assign stack20 = M1.stack[20];
80    assign stack21 = M1.stack[21];
81    assign stack22 = M1.stack[22];
82    assign stack23 = M1.stack[23];
83    assign stack24 = M1.stack[24];
84    assign stack25 = M1.stack[25];
85    assign stack26 = M1.stack[26];
86    assign stack27 = M1.stack[27];
87    assign stack28 = M1.stack[28];
88    assign stack29 = M1.stack[29];
89    assign stack30 = M1.stack[30];
90    assign stack31 = M1.stack[31];
91
92    // Instantiate the fifo_non_fsm module
93    fifo_non_fsm M1(
94      Data_out,
95      stack_full,
96      stack_empty,
97      Data_in,
98      write_to_stack,
99      read_from_stack,
100     clk,
101     rst
102   );
103
104   // Initial block for simulation setup
105   initial begin
106     // Dump waveform to a VCD file
107     $dumpfile("dump.vcd");
108     $dumpvars;
109     // Initialize clock and reset signals
110     clk = 0;
111     rst = 1;
112
113     // Wait for a few cycles before releasing reset
114     #10 rst = 0;
115     // Set read and write signals for writing data into the stack
116     read_from_stack = 0;
117     write_to_stack = 1;
118     Data_in = 8'd1;
119
120     // Set new data inputs at different time intervals
121     #10 Data_in = 8'd10;
```

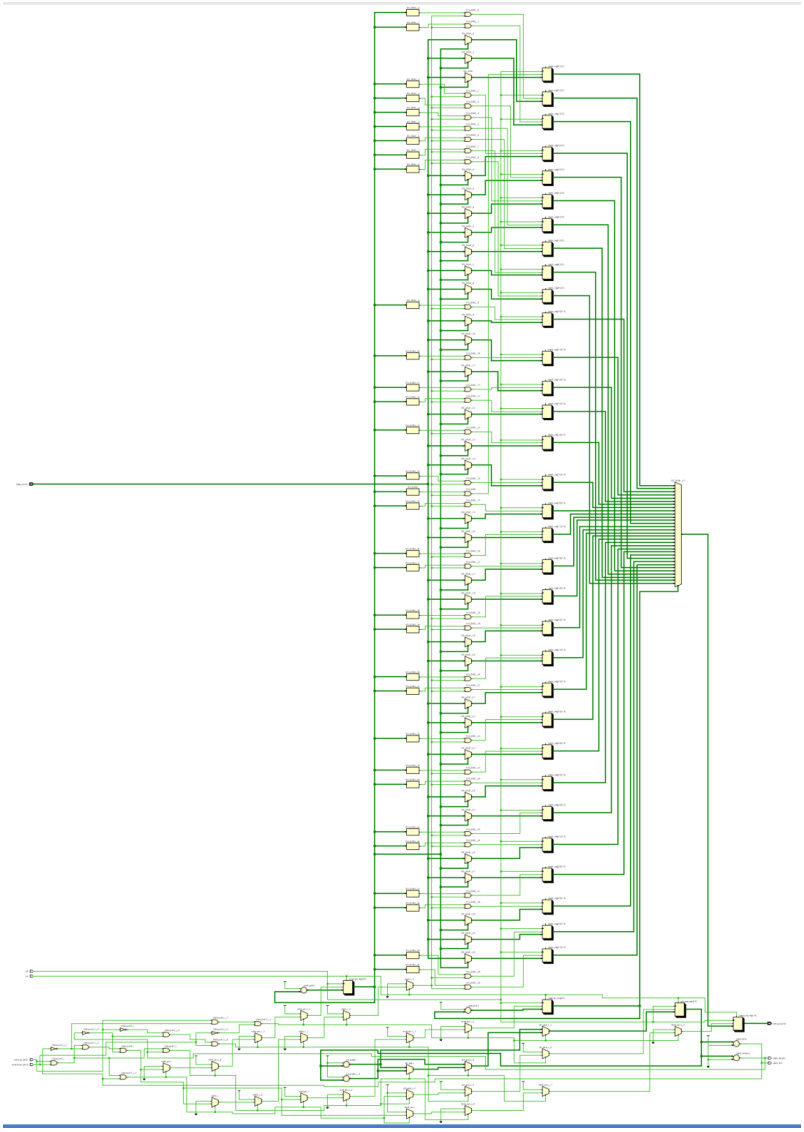
```
122      #10 Data_in = 8'd20;
123
124      // Set read and write signals for reading data from the stack
125      #10 read_from_stack = 1;
126          write_to_stack = 0;
127          Data_in = 8'd40;
128
129      // Continue setting new data inputs and read/write signals
130      #10 write_to_stack = 0;
131          Data_in = 8'd41;
132      #10 write_to_stack = 1;
133          read_from_stack = 0;
134          Data_in = 8'd42;
135      #10 Data_in = 8'd30;
136      #50 write_to_stack = 0;
137          read_from_stack = 1;
138          Data_in = 8'd25;
139      #50 write_to_stack = 1;
140          read_from_stack = 0;
141          Data_in = 8'd35;
142    end
143
144    // Toggle clock every 5 time units
145    always #5 clk = ~clk;
146
147    // Finish simulation after 500 time units
148    initial #500 $finish;
149
150  endmodule
```

1.3 Simulation Waveform





1.4 Schematic



2 FIFO (FSM)

The fifo fsm module is a Verilog implementation of a finite state machine (FSM) based FIFO buffer. It includes input and output ports for data transfer, as well as signals indicating FIFO fullness or emptiness. Parameters define stack characteristics such as element width, maximum capacity, and pointer width. Internal registers manage FSM states and transitions, read and write pointers, along with a gap register tracking their difference. The module features sequential logic for state transition triggered by clock edges and combinational logic for FSM behavior based on current state and inputs. It employs a case statement to define behavior for different states, including reset, idle, write, and read states. This design enables controlled data flow within the FIFO buffer.

2.1 Code

```
// Set the timescale for simulation
`timescale 1ns/1ps

// Define the FIFO FSM module
module fifo_fsm(Data_out,
                stack_full,
                stack_empty,
                Data_in,
                write_to_stack,
                read_from_stack,
                clk,
                rst);

    // Parameters defining the width, height, and pointer width of the stack
    parameter stack_width = 8;
    parameter stack_height = 32;
    parameter stack_ptr_width = 5;

    // Output registers for data out and stack status
    output reg [stack_width-1:0] Data_out;
    output stack_full,stack_empty;

    // Input ports for data in, write and read signals, clock, and reset
    input [stack_width-1:0] Data_in;
    input write_to_stack, read_from_stack,clk,rst;

    // Define states of the FSM
    parameter S_reset = 0;
    parameter S_idle = 1;
    parameter S_read = 2;
    parameter S_write = 3;

    // Registers for FSM state and next state
    reg [2:0] state,next_state;

    // Registers for read and write pointers, and pointer gap
```

```
37      reg [stack_ptr_width-1:0] read_ptr, write_ptr;
38      reg [stack_ptr_width:0] ptr_gap;
39
40      // Memory array representing the stack
41      reg [stack_width-1:0] stack [stack_height-1:0];
42
43      // Calculate stack_full and stack_empty signals
44      assign stack_full = (ptr_gap == stack_height);
45      assign stack_empty = (ptr_gap == 0);
46
47      // Sequential logic for FSM state transition
48      always@(posedge clk)
49        if(rst == 1) state <= S_reset;
50        else state <= next_state;
51
52      // Combinational logic for FSM behavior based on current
53      // state and inputs
54      always@(state or write_to_stack or read_from_stack or Data_in)
55        begin
56          // State machine behavior using case x statement
57          casex(state)
58            S_reset: begin
59                       Data_out = 0;
60                       read_ptr = 0;
61                       write_ptr = 0;
62                       ptr_gap = 0;
63                       if(rst)
64                         next_state = S_reset;
65                       else if((write_to_stack == 0) && (read_from_stack == 0))
66                         next_state = S_idle;
67                       else if((write_to_stack == 1) && (read_from_stack == 0))
68                         next_state = S_write;
69                       else if((write_to_stack == 0) && (read_from_stack == 1))
70                         next_state = S_read;
71                     end
72            S_idle: begin
73                       Data_out = 0;
74                       if(rst)
75                         next_state = S_reset;
76                       else if((write_to_stack == 0) && (read_from_stack == 0))
77                         next_state = S_idle;
78                       else if((write_to_stack == 1) && (read_from_stack == 0))
79                         next_state = S_write;
80                       else if((write_to_stack == 0) && (read_from_stack == 1))
81                         next_state = S_read;
82                    end
83            S_write:begin
84                       if(write_to_stack && (!stack_full))
85                         begin
86                           stack[write_ptr] = Data_in;
```

```
                        write_ptr = write_ptr + 1;
                        ptr_gap = ptr_gap + 1;
                      end

                    if(rst)
                      next_state = S_reset;
                    else if((write_to_stack == 0) && (read_from_stack == 0))
                      next_state = S_idle;
                    else if((write_to_stack == 1) && (read_from_stack == 0))
                      next_state = S_write;
                    else if((write_to_stack == 0) && (read_from_stack == 1))
                      next_state = S_read;
                  end
          S_read:begin
                    if(read_from_stack && (!stack_empty))
                      begin
                        Data_out = stack[read_ptr];
                        read_ptr = read_ptr + 1;
                        ptr_gap = ptr_gap - 1;
                      end

                    if(rst)
                      next_state = S_reset;
                    else if((write_to_stack == 0) && (read_from_stack == 0))
                      next_state = S_idle;
                    else if((write_to_stack == 1) && (read_from_stack == 0))
                      next_state = S_write;
                    else if((write_to_stack == 0) && (read_from_stack == 1))
                      next_state = S_read;
                  end
          default: next_state = S_idle;

        endcase
    end
endmodule
```

2.2 Testbench

```
// Set the timescale for simulation
`timescale 1ns/1ps

// Include the finite state machine (FSM) based FIFO module
`include "fifo_fsm.v"

// Define the testbench module
module tb_fifo_fsm();

  // Parameters defining the stack width, height, and pointer width
  parameter stack_width = 8;
```

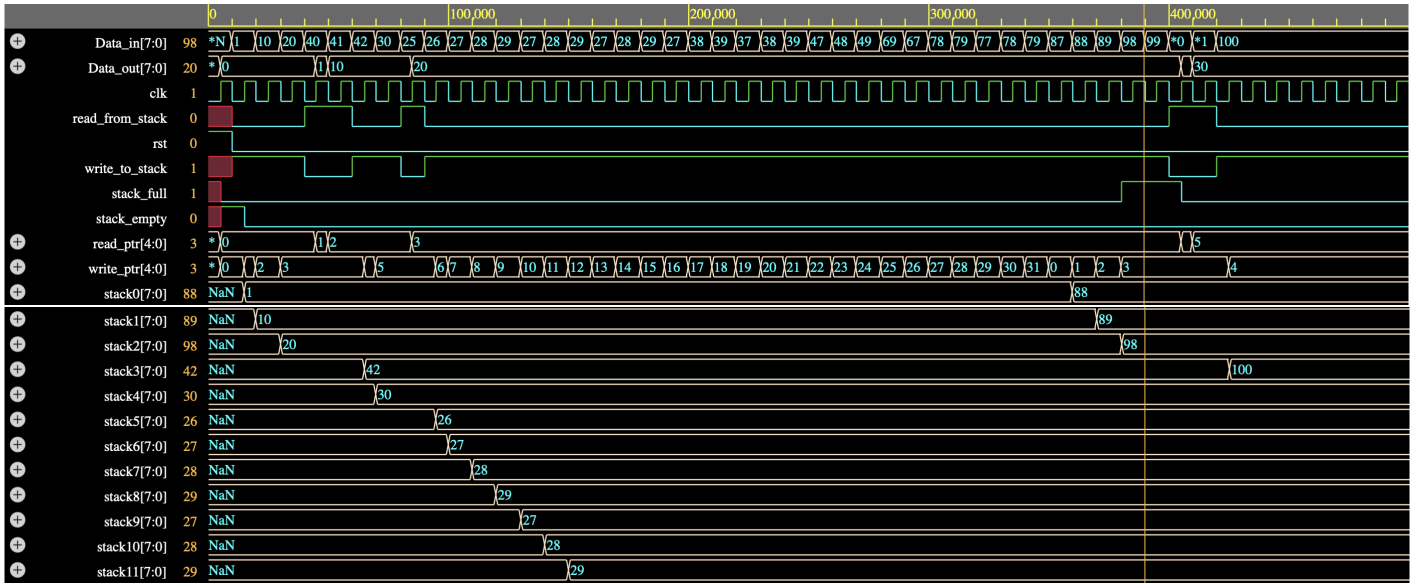
```
12    parameter stack_height = 32;
13    parameter stack_ptr_width = 5;
14
15    // Define FSM states
16    parameter S_reset = 0;
17    parameter S_idle = 1;
18    parameter S_read = 2;
19    parameter S_write = 3;
20
21    // Output wires for data out and stack status
22    wire [stack_width-1:0] Data_out;
23    wire stack_full, stack_empty;
24
25    // Input registers for data in, write and read signals, clock, and reset
26    reg [stack_width-1:0] Data_in;
27    reg write_to_stack, read_from_stack;
28    reg clk, rst;
29
30    // Wires to access individual stack elements
31    wire [stack_width-1:0] stack0,
32                           stack1,
33                           stack2,
34                           stack3,
35                           stack4,
36                           stack5,
37                           stack6,
38                           stack7,
39                           stack8,
40                           stack9,
41                           stack10,
42                           stack11,
43                           stack12,
44                           stack13,
45                           stack14,
46                           stack15,
47                           stack16,
48                           stack17,
49                           stack18,
50                           stack19,
51                           stack20,
52                           stack21,
53                           stack22,
54                           stack23,
55                           stack24,
56                           stack25,
57                           stack26,
58                           stack27,
59                           stack28,
60                           stack29,
61                           stack30,
```

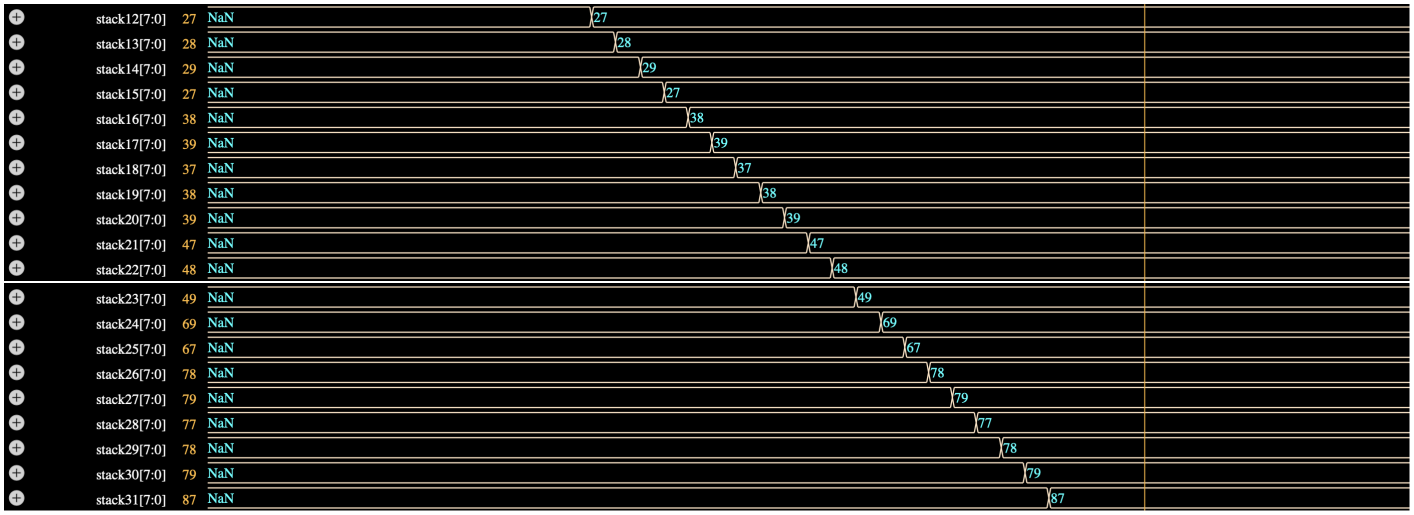
```
                        stack31;

  // Assignments to access individual stack elements from the FIFO module
  assign stack0 = M1.stack[0];
  assign stack1 = M1.stack[1];
  assign stack2 = M1.stack[2];
  assign stack3 = M1.stack[3];
  assign stack4 = M1.stack[4];
  assign stack5 = M1.stack[5];
  assign stack6 = M1.stack[6];
  assign stack7 = M1.stack[7];
  assign stack8 = M1.stack[8];
  assign stack9 = M1.stack[9];
  assign stack10 = M1.stack[10];
  assign stack11 = M1.stack[11];
  assign stack12 = M1.stack[12];
  assign stack13 = M1.stack[13];
  assign stack14 = M1.stack[14];
  assign stack15 = M1.stack[15];
  assign stack16 = M1.stack[16];
  assign stack17 = M1.stack[17];
  assign stack18 = M1.stack[18];
  assign stack19 = M1.stack[19];
  assign stack20 = M1.stack[20];
  assign stack21 = M1.stack[21];
  assign stack22 = M1.stack[22];
  assign stack23 = M1.stack[23];
  assign stack24 = M1.stack[24];
  assign stack25 = M1.stack[25];
  assign stack26 = M1.stack[26];
  assign stack27 = M1.stack[27];
  assign stack28 = M1.stack[28];
  assign stack29 = M1.stack[29];
  assign stack30 = M1.stack[30];
  assign stack31 = M1.stack[31];

  // Instantiate the FIFO module
  fifo_fsm M1(Data_out,
          stack_full,
          stack_empty,
          Data_in,
          write_to_stack,
          read_from_stack,
          clk,
          rst);

  // Initial block for simulation setup
  initial begin
    // Dump waveform to a VCD file
    $dumpfile("dump.vcd");
```

```
112     $dumpvars;
113     // Initialize clock and reset signals
114     clk = 0;
115     rst = 1;
116 
117     // Wait for a few cycles before releasing reset
118     #10 rst = 0;
119     // Set read and write signals for writing data into the stack
120     read_from_stack = 0;
121     write_to_stack = 1;
122     Data_in = 8'd1;
123 
124     // Set new data inputs at different time intervals
125     #10 Data_in = 8'd10;
126     #10 Data_in = 8'd20;
127 
128     // Set read and write signals for reading data from the stack
129     #10 write_to_stack = 0;
130         read_from_stack = 1;
131         Data_in = 8'd40;
132 
133         // Continue setting new data inputs and read/write signals
134     #10 write_to_stack = 0;
135         Data_in = 8'd41;
136     #10 write_to_stack = 1;
137         read_from_stack = 0;
138         Data_in = 8'd42;
139     #10 Data_in = 8'd30;
140     #10 write_to_stack = 0;
141         read_from_stack = 1;
142         Data_in = 8'd25;
143     #10 write_to_stack = 1;
144         read_from_stack = 0;
145         Data_in = 8'd26;
146     #10 Data_in = 8'd27;
147     #10 Data_in = 8'd28;
148     #10 Data_in = 8'd29;
149     #10 Data_in = 8'd27;
150     #10 Data_in = 8'd28;
151     #10 Data_in = 8'd29;
152     #10 Data_in = 8'd27;
153     #10 Data_in = 8'd28;
154     #10 Data_in = 8'd29;
155     #10 Data_in = 8'd27;
156     #10 Data_in = 8'd38;
157     #10 Data_in = 8'd39;
158     #10 Data_in = 8'd37;
159     #10 Data_in = 8'd38;
160     #10 Data_in = 8'd39;
161     #10 Data_in = 8'd47;
```

```
162       #10 Data_in = 8'd48;
163       #10 Data_in = 8'd49;
164       #10 Data_in = 8'd69;
165       #10 Data_in = 8'd67;
166       #10 Data_in = 8'd78;
167       #10 Data_in = 8'd79;
168       #10 Data_in = 8'd77;
169       #10 Data_in = 8'd78;
170       #10 Data_in = 8'd79;
171       #10 Data_in = 8'd87;
172       #10 Data_in = 8'd88;
173       #10 Data_in = 8'd89;
174       #10 Data_in = 8'd98;
175       #10 Data_in = 8'd99;
176       #10 write_to_stack = 0;
177           read_from_stack = 1;
178           Data_in = 8'd100;
179       #10 Data_in = 8'd101;
180       #10 write_to_stack = 1;
181           read_from_stack = 0;
182           Data_in = 8'd100;
183     end
184
185     // Toggle clock every 5 time units
186     always #5 clk = ~clk;
187
188     // Finish simulation after 500 time units
189     initial #500 $finish;
190
191 endmodule
```

2.3 Simulation Waveform





2.4 Schematic