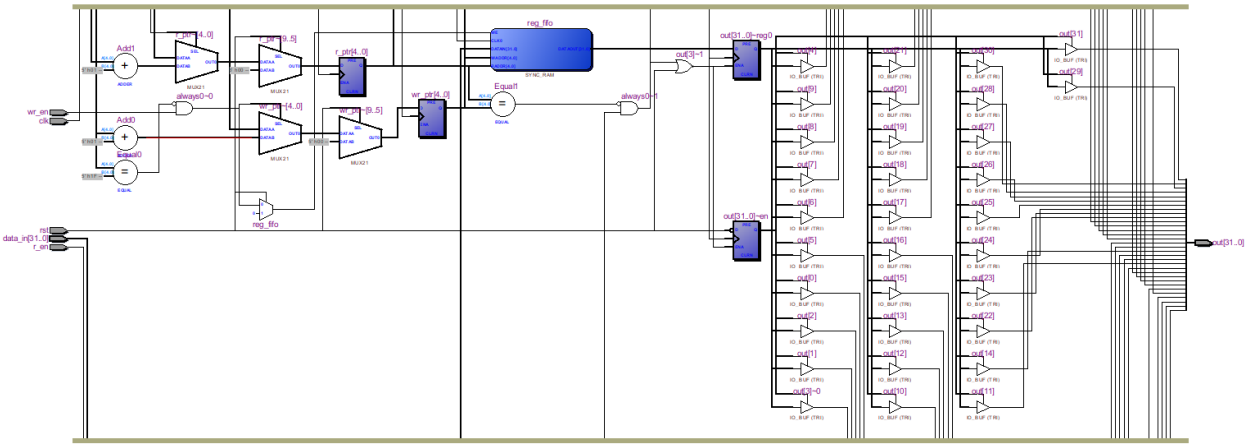
FIFO
Code

```
1  module register(clk, wr_en, r_en, rst, out, data_in);
2  input clk, rst, wr_en, r_en;
3  input [31:0] data_in;
4  reg [31:0] reg_fifo [31:0];
5  output reg [31:0] out;
6  reg [4:0] wr_ptr, r_ptr;
7  assign full = (wr_ptr == 5'b11111);
8  assign empty = (wr_ptr == r_ptr);
9  always @(posedge clk)
10 begin
11    if(rst) begin
12       wr_ptr = 0;
13       r_ptr = 0;
14       out = 32'bZ;
15    end
16    else begin
17       if(wr_en && !full) begin
18          reg_fifo[wr_ptr] = data_in;
19          wr_ptr = wr_ptr + 5'b00001;
20          end
21       if(r_en && !empty) begin
22          out = reg_fifo[r_ptr];
23          r_ptr = r_ptr + 5'b00001;
24          end
25    end
26 end
27 endmodule
28
29
```

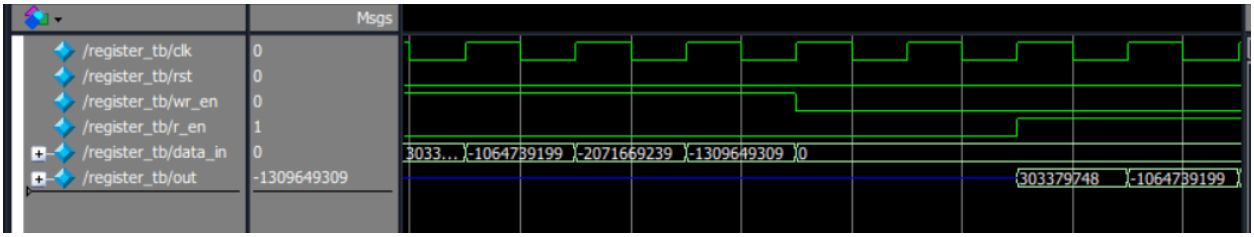
RTL


reg_fifo
Add1
Add0
Equal1
Equal0
always0=0
always0=1
out[3]=1
out[31..0]=reg0
out[31..0]=en
out[31..0]
wr_en
clk
rst
data_in[31..0]
r_en

Testbench

```
1  `timescale 1ns / 1ps
2
3  module register_tb;
4
5      reg clk;
6      reg rst;
7      reg wr_en;
8      reg r_en;
9      reg [31:0] data_in;
10
11     wire [31:0] out;
12
13     register uut (
14         .clk(clk),
15         .rst(rst),
16         .wr_en(wr_en),
17         .r_en(r_en),
18         .data_in(data_in),
19         .out(out)
20     );
21
22
23     always #5 clk = ~clk;
24
25     initial begin
26         clk = 0;
27         rst = 1;
28         wr_en = 0;
29         r_en = 0;
30         data_in = 0;
31
32         #10;
33         rst = 0;
```

```
34
35          repeat (4) begin
36              @(posedge clk);
37              wr_en = 1;
38              data_in = $random;
39          end
40
41          @(posedge clk);
42          wr_en = 0;
43          data_in = 0;
44
45          #20;
46           r_en = 1;
47          #100;
48
49          $finish;
50      end
51
52  endmodule
```

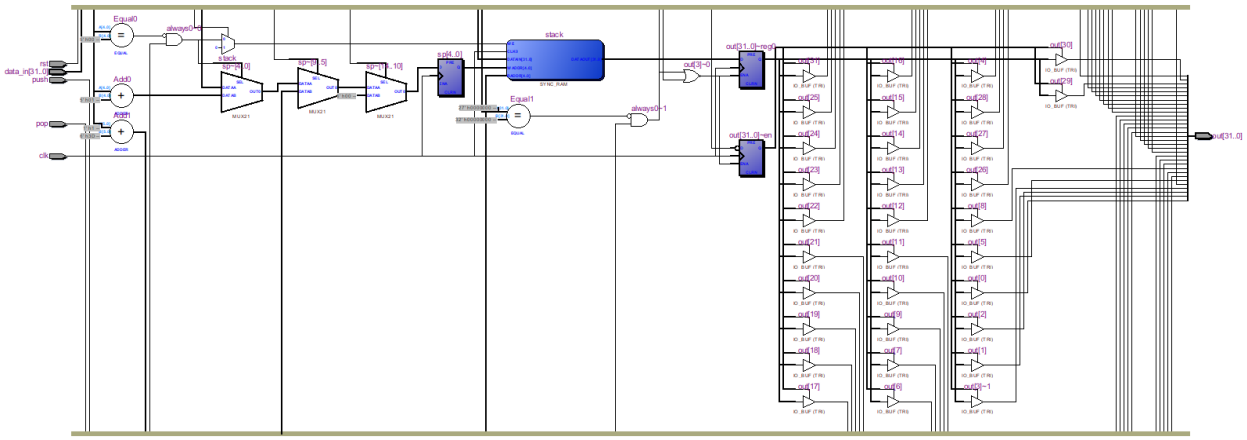


LIFO

Code

```
1  module lifo_stack (
2      input clk,
3      input rst,
4      input push,
5      input pop,
6      input [31:0] data_in,
7      output reg [31:0] out
8  );
9
10     reg [31:0] stack [31:0];
11     reg [4:0] sp;
12
13     wire full = (sp == 5'd32);
14     wire empty = (sp == 0);
15
16     always @(posedge clk) begin
17         if (rst) begin
18             sp <= 0;
19             out <= 32'bz;
20         end else begin
21             if (push && !full) begin
22                 stack[sp] <= data_in;
23                 sp <= sp + 1;
24             end
25             if (pop && !empty) begin
26                 sp <= sp - 1;
27                 out <= stack[sp];
28             end
29         end
30     end
31
32 endmodule
```

RTL


Equal0
always0=0
stack
sp[4..0]
stack
Equal1
always0=1
out[31..0]=sp0
out[4..0]=0
out[31..0]=en
out[31..0]

Testbench

```
1  `timescale 1ns / 1ps
2
3  module lifo_stack_tb;
4
5
6      reg clk;
7      reg rst;
8      reg push;
9      reg pop;
10     reg [31:0] data_in;
11
12
13     wire [31:0] out;
14
15     lifo_stack uut (
16         .clk(clk),
17         .rst(rst),
18         .push(push),
19         .pop(pop),
20         .data_in(data_in),
21         .out(out)
22     );
23
24     always #5 clk = ~clk;
25
26     initial begin
27
28         clk = 0;
29         rst = 1;
30         push = 0;
31         pop = 0;
32         data_in = 0;
```

```
33
34
35          #10;
36          rst = 0;
37
38
39          repeat (4) begin
40              @(posedge clk);
41              push = 1;
42              data_in = $random;
43          end
44
45
46          @(posedge clk);
47          push = 0;
48          data_in = 0;
49
50
51          #20;
52
53
54          repeat (4) begin
55              @(posedge clk);
56              pop = 1;
57          end
58
59
60          @(posedge clk);
61          pop = 0;
```

```
62
63
64          #20;
65          $finish;
66      end
67
68      initial begin
69          $monitor("Time=%0t | push=%b, pop=%b, data_in=%h, out=%h", $time, push, pop, data_in, out);
70      end
71
72  endmodule
73
```