Programmable Digital Systems – Exercise 7

Goals:

- introduction to FIFO (First-In First-Out) buffers -queue,
- introduction to LIFO (Last-In First-Out) buffers -stack.

Exercise:

1. Run Active-HDL, create a new workspace and a new design.
2. Enter the program below into editor:

testbench.v file:

```
module mytestbenchmodule();
reg CLK;
initial CLK <= 0;
always #50 CLK <= ~CLK;
reg RST;
initial
begin
       RST <= 0;
       RST <= #100 1;
       RST <= #500 0;
end
reg fi_stb;
reg [7:0] fi_dat;
wire fi_bsy;
wire fo_stb;
wire [7:0] fo_dat;
reg fo_ack;
initial
       begin
              fi_stb <= 0;
              fo_ack <= 0;
              fi_dat <= 0;
              #1000;
              @( posedge CLK);
              fi_stb <= 1;
              fi_dat <= 17;
              @(posedge CLK);
              fi_dat <= 18;
              @( posedge CLK);
              // !!!!!!!!!!!!!!!!!
              // .... please type some additional data here ... //
              // !!!!!!!!!!!!!!!!!
              fi_dat <= 55;
              @( posedge CLK);
              fi_stb <= 0;
              #1000;
              @(posedge CLK);
              fo_ack <= 1;
       end

fifo
#(
       .WIDTH(8)
)
f1
(
       .CLK(CLK),
       .RST(RST),
       .FI_STB(fi_stb),
       .FI_DAT(fi_dat),
       .FI_BSY(fi_bsy),
       .FO_STB(fo_stb),
       .FO_ACK(fo_ack),
       .FO_DAT(fo_dat)
);
endmodule
```

fifo.v file:

```
module fifo
#(
       parameter WIDTH = 8
)
(
       input wire CLK,
       input wire RST,
       input wire FI_STB,
       input wire [WIDTH-1:0] FI_DAT,
       output wire FI_BSY,
       output wire FO_STB,
       input wire FO_ACK,
       output wire [WIDTH-1:0] FO_DAT
);
//---------------------------------------
reg [WIDTH-1:0] ff_dat [0:15];
reg [3:0] ff_sel;
reg ff_in_busy;
reg ff_out_stb;
//---------------------------------------
integer ff_state;
//---------------------------------------
assign FI_BSY = ff_in_busy;
//--------------------------------------
assign FO_STB = ff_out_stb;
//--------------------------------------
always@(posedge CLK or posedge RST) ff_in_busy <= (RST) ? 1'b0 : (ff_state==4);
//--------------------------------------
always@(posedge CLK or posedge RST)
if(RST)
begin
       ff_state <= 0;
       ff_sel <= 4'hF;
       ff_out_stb <= 1'b0;
end
else casex(ff_state)
       //..............................
       // clear state
       //..............................
       0:
              begin
                     ff_sel <= 4'hF;
                     ff_out_stb <= 1'b0;
                     ff_state <= 1;
              end
       //..............................
       // empty
       //..............................
       1:
              if(FI_STB)
                     begin
                            ff_sel <= ff_sel + 1'b1;
                            ff_out_stb <= 1'b1;
                            ff_state <= 2;
                     end
       //..............................
       // not empty but only one symbol in the buffer
       //..............................
       2:
              if(FI_STB && FO_ACK)
                     begin
                            ff_sel <= ff_sel;
                            ff_out_stb <= 1'b1;
                            ff_state <= 2;
                     end
              else if(FI_STB)
                     begin
                            ff_sel <= ff_sel + 1'b1;
                            ff_out_stb <= 1'b1;
```

```
                        ff_state <= 3;
                    end
                else if(FO_ACK)
                    begin
                        ff_sel <= ff_sel - 1'b1;
                        ff_out_stb <= 1'b0;
                        ff_state <= 1;
                    end
        //..............................
        // not empty
        //..............................
        3:
                if(FI_STB && FO_ACK)
                    begin
                        ff_sel <= ff_sel;
                        ff_out_stb <= 1'b1;
                        ff_state <= 3;
                    end
                else if(FI_STB)
                    begin
                        ff_sel <= ff_sel + 1'b1;
                        ff_out_stb <= 1'b1;
                        ff_state <= (ff_sel>=8)? 4 : 3;
                    end
                else if(FO_ACK)
                    begin
                        ff_sel <= ff_sel - 1'b1;
                        ff_out_stb <= 1'b1;
                        ff_state <= (ff_sel<=1)? 2 : 3;
                    end
        //..............................
        // full
        //..............................
        4:
                if(FI_STB && FO_ACK)
                    begin
                        ff_sel <= ff_sel;
                        ff_out_stb <= 1'b1;
                        ff_state <= 4;
                    end
                else if(FI_STB)
                    begin
                        ff_sel <= ff_sel + 1'b1;
                        ff_out_stb <= 1'b1;
                        ff_state <= 4;
                    end
                else if(FO_ACK)
                    begin
                        ff_sel <= ff_sel - 1'b1;
                        ff_out_stb <= 1'b1;
                        ff_state <= (ff_sel>=8)? 4 : 3;
                    end
        //..............................
        default: ff_state <= 0;
        //..............................
endcase
//------------------------------------------------------------------------------
always@(posedge CLK)
if(FI_STB) begin
        ff_dat[4'h0] <= FI_DAT;
        ff_dat[4'h1] <= ff_dat[4'h0];
        ff_dat[4'h2] <= ff_dat[4'h1];
        ff_dat[4'h3] <= ff_dat[4'h2];
        ff_dat[4'h4] <= ff_dat[4'h3];
        ff_dat[4'h5] <= ff_dat[4'h4];
        ff_dat[4'h6] <= ff_dat[4'h5];
        ff_dat[4'h7] <= ff_dat[4'h6];
        ff_dat[4'h8] <= ff_dat[4'h7];
        ff_dat[4'h9] <= ff_dat[4'h8];
        ff_dat[4'hA] <= ff_dat[4'h9];
        ff_dat[4'hB] <= ff_dat[4'hA];
        ff_dat[4'hC] <= ff_dat[4'hB];
```

```
       ff_dat[4'hD] <= ff_dat[4'hC];
       ff_dat[4'hE] <= ff_dat[4'hD];
       ff_dat[4'hF] <= ff_dat[4'hE];
end
//-----------------------------------------------------------------------------------------
assign FO_DAT = ff_dat [ff_sel];
//----------------------------------------------------------------------------------------
endmodule
```

3. Compile and run the simulation. Analyze waveforms of the fifo module.
4. Does the module work correctly? Check with different numbers and different diagrams of data:
- more that one data in testbench,
Does the problem lays in fifo module or within data feeding? if the program does not work correctly, fix it.
5. Type the following program:
Testbench.v file:

```
module mytestbenchmodule();
reg CLK;
initial CLK <= 0;
always #50 CLK <= ~CLK;
reg RST;
initial
  begin
    RST <= 0;
    RST <= #100 1;
    RST <= #500 0;
  end

reg push_stb;
reg [7:0] push_dat;
wire lifo_full;
reg pop_stb;
wire [7:0] pop_dat;
initial
  begin
    push_stb <= 0;
    push_dat <= 0;
    pop_stb <= 0;
    #1000;
    @( posedge CLK);
    push_stb <= 1;
    push_dat <= 17;
    @(posedge CLK);
    push_dat <= 18;
    @( posedge CLK);
    // !!!!!!!!!!!!!!!!!
    // .... please type some additional data here ... //
    // !!!!!!!!!!!!!!!!!
    push_dat <= 55;
    @( posedge CLK);
    push_stb <= 0;
    #1000;
    @(posedge CLK);
    pop_stb <= 1;
  end

lifo
#(
       .WIDTH(8)
)
f1
(
       .CLK(CLK),
       .RST(RST),
       .PUSH_STB(push_stb),
       .PUSH_DAT(push_dat),
       .POP_STB(pop_stb),
       .POP_DAT(pop_dat),
       .FULL(lifo_full)
);
endmodule
```

lifo.v file:

```
module lifo
#(
  parameter WIDTH = 8
)
(
  input wire CLK,
  input wire RST,
  input wire PUSH_STB,
  input wire [WIDTH-1:0] PUSH_DAT,
  input wire POP_STB,
  output wire [WIDTH-1:0] POP_DAT,
  output wire FULL
);
//----------------------------------------
reg [WIDTH-1:0] fl_dat [0:15];
reg [3:0] fl_head_ptr;
wire fl_in_full;
reg fl_out_stb;
//----------------------------------------
integer fl_state;
//---------------------------------------
assign fl_in_full = (fl_state==3);
//----------------------------------------
assign FULL = fl_in_full;
//---------------------------------------
always@(posedge CLK or posedge RST)
if(RST)
  begin
    fl_state <= 0;
    fl_head_ptr <= 4'h0;
  end
else casex(fl_state)
//.............................
// clear state
//.............................
0:
  begin
    fl_head_ptr <= 4'hF;
    fl_state <= 1;
  end
//.............................
// empty
//.............................
1:
  if(PUSH_STB)
    begin
      fl_head_ptr <= fl_head_ptr + 4'h1;
      fl_state <= 2;
    end
//.............................
// not empty
//.............................
2:
  begin
    if(PUSH_STB & !POP_STB & (fl_head_ptr == 4'hE)) // after a new data is pushed LIFO
will be full
      fl_state <= 3;
    else if(!PUSH_STB & POP_STB & (fl_head_ptr == 4'd0)) // after next data is poped
LIFO will be empty
      fl_state <= 1;
    else
      fl_state <= 2;
    if(PUSH_STB)
      fl_head_ptr <= fl_head_ptr + 4'h1;
    else if(POP_STB)
      fl_head_ptr <= fl_head_ptr - 4'h1;
  end
//.............................
// full
//.............................
3:
  if(POP_STB)
    begin
      fl_head_ptr <= fl_head_ptr - 4'h1;
      fl_state <= 2;
    end
  else
    begin
      fl_head_ptr <= fl_head_ptr;
      fl_state <= 3;
    end
//.............................
default: fl_state <= 0;
//.............................
endcase
//--------------------------------------------------------------------------------------
always@(posedge CLK)
  if(PUSH_STB) begin
    fl_dat[fl_head_ptr + 4'd1] <= PUSH_DAT;
  end
//---------------------------------------------------------------------------------------
assign POP_DAT = fl_dat [fl_head_ptr];
//--------------------------------------------------------------------------------------
endmodule
```

6. Compile and run the simulation. Analyze waveforms of the stack module.
7. Does the module work correctly? Check with different numbers and different diagrams of data.
8. Add the EMPTY signal to lifo ports indicating that there is no data stored on a stack.