# Q1

|  |
| --- |
| C:\Users\YC\Downloads\HW5\Q1.png |
| module debouncing(clk, rst, in, out);  input clk, rst, in;  output reg out;  reg [2:0] state, next\_state;  parameter WAIT = 3'b000,  h1 = 3'b001,  h2 = 3'b010,  h3 = 3'b011,  l1 = 3'b100,  l2 = 3'b101;  always@(posedge clk) begin  if(rst) state <= WAIT;  else state <= next\_state;  end  always@(state or in) begin *//excitation logic*  case(state)  WAIT: next\_state = (in) ? h1 : WAIT;  h1 : next\_state = (in) ? h2 : WAIT;  h2 : next\_state = (in) ? h3 : WAIT;  h3 : next\_state = (in) ? h3 : l1 ;  l1 : next\_state = (in) ? h3 : l2 ;  l2 : next\_state = (in) ? h3 : WAIT;  default: next\_state = WAIT;  endcase  end  always@(state) begin *//output logic*  case(state)  WAIT: out = 1'b0;  h1 : out = 1'b0;  h2 : out = 1'b0;  h3 : out = 1'b1;  l1 : out = 1'b1;  l2 : out = 1'b1;  default: out = 1'b0;  endcase  end  endmodule |

# Q2

|  |
| --- |
| C:\Users\YC\Downloads\HW5\Q2.png |
| module ram1024X16(clk, addr, data, rw, cs);  input clk, rw, cs;  input [9:0] addr;  inout [15:0] data;    reg [15:0] ram0 [0:1023];  reg [15:0] ram1 [0:1023];  reg [15:0] ram2 [0:1023];  reg [15:0] ram3 [0:1023];  reg [15:0] dout;  *//dataflow*  assign data = (!rw)&&(!cs) ? dout : 16'bzzzz\_zzzz\_zzzz\_zzzz;  *// read*  always@(posedge clk) begin  *// read*  if(!rw) begin  case(addr[9:8])  2'b00 : dout <= ram0[addr[7:0]];  2'b01 : dout <= ram1[addr[7:0]];  2'b10 : dout <= ram2[addr[7:0]];  2'b11 : dout <= ram3[addr[7:0]];  default : $display("error");  endcase  $display("%t : reading ram%d[%d] = %d, cs = %d", $time, addr[9:8], addr[7:0], dout, cs);  end  *//write*  else begin  case(addr[9:8])  2'b00 : ram0[addr[7:0]] <= data;  2'b01 : ram1[addr[7:0]] <= data;  2'b10 : ram2[addr[7:0]] <= data;  2'b11 : ram3[addr[7:0]] <= data;  default : $display("error");  endcase  $display("%t : writing ram%d[%d] <= %d, cs = %d", $time, addr[9:8], addr[7:0], data, cs);  end  end  endmodule |

# Q3

|  |
| --- |
| C:\Users\YC\Downloads\HW5\Q3.png |
| */\*3bit counter head pointer\*/*  module head\_ptr(  input clk,  input en,  output reg [2:0] Q,  input rst  );  always@(posedge clk) begin  if(rst) Q <= 3'b001;  else begin  if(en) Q <= Q + 1;  else Q <= Q; *//也可不assign*  end  end  endmodule  */\*3bit counter tail pointer\*/*  module tail\_ptr(  input clk,  input en,  output reg [2:0] Q,  input rst  );  always@(negedge clk) begin  if(rst) Q <= 3'b001;  else begin  if(en) Q <= Q + 1;  else Q <= Q; *//也可不assign*  end  end  endmodule  */\*reg for full & empty flag\*/*  module Flags(  input [2:0] head,  input [2:0] tail,  output reg full,  output reg empty,  input clear  );  always@(clear, head, tail) begin  if(clear) {full, empty} = 2'b01;  else begin  if(head == tail) {full, empty} = 2'b01;  else if((tail + 1) == head) {full, empty} = 2'b10;  else {full, empty} = 2'b00;  end  end  endmodule  */\*Circular buffer for Queue storage\*/*  module Circular\_buffer(  input [7:0] data\_in,  output reg [7:0] data\_out,  input [2:0] addr, *//for head or tail*  input rw *//rw = 0 (read); rw = 1 (write)*  );  reg [7:0] mem [0:7];  always@(rw, addr) begin  if(rw) begin *//rw = 1 write*  mem[addr] = data\_in;  data\_out = 8'bzzzz\_zzzz;  end  else begin *//rw = 0 read*  data\_out = mem[addr];  end  end  endmodule  */\*Insert\_FSM\*/*  module Insert\_FSM(  input clk,  input ins,  input full,  input clear,  output reg tail\_en,  output reg rw  );  reg state, next\_state;  parameter A = 1'b0,  B = 1'b1;  always@(posedge clk) begin  if(clear) state <= 1'b0;  else state <= next\_state;  end  always@(ins, full, state) begin  case(state)  A : next\_state = ({ins, full} == 2'b10) ? B : A;  B : next\_state = ({ins, full} == 2'b10) ? B : A;  endcase  end  always@(ins, full, state) begin  case(state)  A : {rw, tail\_en} = ({ins, full} == 2'b10) ? 2'b11 : 2'b00;  B : {rw, tail\_en} = ({ins, full} == 2'b10) ? 2'b11 : 2'b00;  endcase  end  endmodule  */\*Retrive\_FSM\*/*  module Retrive\_FSM(  input clk,  input ret,  input empty,  input clear,  output reg head\_en,  output reg rw  );  reg state, next\_state;  parameter C = 1'b0,  D = 1'b1;  always@(negedge clk) begin  if(clear) state <= 1'b0;  else state <= next\_state;  end  always@(ret, empty, state) begin  case(state)  C : next\_state = ({ret, empty} == 2'b10) ? D : C;  D : next\_state = ({ret, empty} == 2'b10) ? D : C;  endcase  end  always@(ret, empty, state) begin  case(state)  C : {rw, head\_en} = ({ret, empty} == 2'b10) ? 2'b01 : 2'b00;  D : {rw, head\_en} = ({ret, empty} == 2'b10) ? 2'b01 : 2'b00;  endcase  end  endmodule  module Queue(  input clk,  input clear,  input [7:0] data\_in,  input insert,  input retrieve,  output [7:0] data\_out,  output full,  output empty  );  wire h\_en, t\_en;  wire [2:0] h\_addr, t\_addr, in\_addr;  wire in\_full, in\_empty;  wire Ins\_rw, Ret\_rw, in\_rw;  head\_ptr head\_ptr(clk, h\_en, h\_addr, clear);  tail\_ptr tail\_ptr(clk, t\_en, t\_addr, clear);  Flags Flags(h\_addr, t\_addr, in\_full, in\_empty, clear);  Circular\_buffer Circular\_buffer(data\_in, data\_out, in\_addr, in\_rw);  Insert\_FSM Insert\_FSM(clk, insert, in\_full, clear, t\_en, Ins\_rw);  Retrive\_FSM Retrive\_FSM(clk, retrieve, in\_empty, clear, h\_en, Ret\_rw);  assign in\_addr = (in\_rw) ? t\_addr : h\_addr;  assign in\_rw = Ins\_rw | Ret\_rw;  assign full = in\_full;  assign empty = in\_empty;  endmodule |