Chattering removal, LCD controller, memories, and instruction fetch

1 Today's goal

- Learn circuits for chattering removal and LCD control and how to use them.
- Learn the distributed RAM and the block RAM in the FPGA.
- Learn how to design the instruction fetch circuit.

2 Today's contents

Step 1 *Check 1* Write chattering removal circuit (List 1), LCD controller (List 2, you can download it from the web page), and the state machine counter (List 3). Implement them in the FPGA and check if it works properly.

Step 2 Write the distributed ram (List 5) and its test bench (List 7) to confirm that it works properly. In the same way write the block RAM (List 6) and its test bench, and perform the simulation.

Step 3 *Check 2* Modify the test bench (List 6) to instantiate both the distributed ram (List 5) and the block RAM (List 6) and show their output in the same time.

Step 4 *Check 3* Write the block ram (List 8) with initialization, the top module (List 9) and its UCF (List 10), and implement it in the FPGA to confirm that it works correctly.

Step 5 *Check 4* Simulate the instruction fetch circuit. More specifically, write instruction fetch circuit (List 11), its test bench (List 12), and block RAM (List 13) with initialization, and perform the simulation.

Step 5 *Check 5* Implement the instruction fetch circuit. More specifically, write top module (List 14) and its UCF (List 15), and implement the bit file in the FPGA.

3 Chattering removal

The chattering removal circuit (List 1) supports N (default value is 1) inputs that may have chattering. The idea of chattering removal is to sample the input every $2^{16} = 65536$ clock cycles. Since the clock frequency of the FPGA board is 50MHz(20ns), the sampling period is $20ns \times 65536 = 1.3ms$. If the period of chattering is no more than 1.3ms, then this circuit removes chattering correctly.

List 1: Chattering removal chattering.v

```
1  module chattering(clk,reset,in,out);
2    parameter N=1;
3
4    input clk,reset;
5    input [N-1:0]in;
6    output [N-1:0]out;
7    reg [N-1:0]out;
8    reg [15:0] count;
9
10   always @(posedge clk or negedge reset)
11     if(!reset) count <= 0;
12     else count <= count + 1;
13
14   always @(posedge clk)
15     if(count==0) out <= in;
16
17 endmodule
```

4 LCD controller

List 2 is an LCD controller for the LCD on the FPGA board. Six 16-bit numbers data0, data1, data2, data3, data4, data5 given to the circuit are displayed in the LCD. Each 16-bit number is represented by 4-digit hexadecimal.

List 2: LCD controller lcdctrl.v

```
1  `define INIT 2'b00
2  `define SETPOS0 2'b01
3  `define SETPOS1 2'b10
4  `define WRITE 2'b11
5
```

```
module lcdctrl(clk,reset,lcd_e,lcd_rs,lcd_rw,sf_d,data0,data1,
    data2,data3,data4,data5);

  input clk,reset;
  input [15:0] data0,data1,data2,data3,data4,data5;
  output lcd_e,lcd_rs,lcd_rw;
  output [11:8] sf_d;
  reg lcd_e;

  reg [1:0] state;
  reg [2:0] index;
  reg [3:0] addr;
  reg [31:0] counter;
  reg [26:0] ctrl;
  wire set_enb;
  wire [19:0] wait_cnt;
  wire ret;
  reg [7:0] ascii;
  reg [3:0] hex;

  assign lcd_rw = 0;
  assign ret = ctrl[26];
  assign lcd_rs = ctrl[25];
  assign set_enb = ctrl[24];
  assign sf_d = ctrl[23:20];
  assign wait_cnt = ctrl[19:0];

  always @(posedge clk or negedge reset)
    if(!reset) begin
      addr <= 0;
      counter <= 0;
      state <= `INIT;
      index <= 0;
    end else if(counter<wait_cnt)
      counter <= counter + 1;
    else begin
      counter <= 0;
      if(!ret)
        addr <= addr + 1;
      else begin
        addr <= 0;
        if(state==`INIT)
          state <= `SETPOS0;
        else if(state==`SETPOS0 || state==`SETPOS1)
          state <= `WRITE;
        else if(state == `WRITE) begin
          if(index== 5)
            index <= 0;
          else
            index <= index + 1;
          if(index == 2)
            state <= `SETPOS1;
          else if(index == 5)
            state <= `SETPOS0;
        end
      end
    end

  always @(posedge clk or negedge reset)
    if(!reset) lcd_e <= 0;
    else if(set_enb && counter >= 1 && counter <= 12)
      lcd_e <= 1;
    else lcd_e <= 0;

  always @(state or addr or ascii)
    case(state)
    `INIT:
      case(addr)
        4'h0: ctrl = {3'b000, 4'h0, 20'hB71B0};
        4'h1: ctrl = {3'b001, 4'h3, 20'h320D7};
        4'h2: ctrl = {3'b001, 4'h3, 20'h01397};
        4'h3: ctrl = {3'b001, 4'h3, 20'h00800};
        4'h4: ctrl = {3'b001, 4'h2, 20'h00800};
        4'h5: ctrl = {3'b001, 4'h2, 20'h00041};
        4'h6: ctrl = {3'b001, 4'h8, 20'h00800};
        4'h7: ctrl = {3'b001, 4'h0, 20'h00041};
        4'h8: ctrl = {3'b001, 4'h6, 20'h00800};
        4'h9: ctrl = {3'b001, 4'h0, 20'h00041};
        4'hA: ctrl = {3'b001, 4'hC, 20'h00800};
        4'hB: ctrl = {3'b001, 4'h0, 20'h00041};
        4'hC: ctrl = {3'b101, 4'h1, 20'h1482F};
        default: ctrl = {3'bxxx, 4'hx, 20'hxxxxx};
        endcase
    `SETPOS0:
      case(addr)
        4'h0: ctrl = {3'b001, 4'h8, 20'h00041};
        4'h1: ctrl = {3'b101, 4'h0, 20'h00800};
        default: ctrl = {3'bxxx, 4'hx, 20'hxxxxx};
        endcase
    `SETPOS1:
      case(addr)
        4'h0 : ctrl = {3'b001, 4'hC, 20'h00041};
        4'h1 : ctrl = {3'b101, 4'h0, 20'h00800};
        default: ctrl = {3'bxxx, 4'hx, 20'hxxxxx};
        endcase
    `WRITE:
      case(addr)
        4'h0 : ctrl = {3'b011, ascii[7:4], 20'h00041};
        4'h1 : ctrl = {3'b011, ascii[3:0], 20'h00800};
        4'h2 : ctrl = {3'b011, ascii[7:4], 20'h00041};
        4'h3 : ctrl = {3'b011, ascii[3:0], 20'h00800};
        4'h4 : ctrl = {3'b011, ascii[7:4], 20'h00041};
        4'h5 : ctrl = {3'b011, ascii[3:0], 20'h00800};
        4'h6 : ctrl = {3'b011, ascii[7:4], 20'h00041};
        4'h7 : ctrl = {3'b011, ascii[3:0], 20'h00800};
        4'h8 : ctrl = {3'b011, 4'hA, 20'h00041};
        4'h9 : ctrl = {3'b111, 4'h0, 20'h00800};
        default: ctrl = {3'bxxx, 4'hx, 20'hxxxxx};
        endcase
    default: ctrl = {3'bxxx, 4'hx, 20'hxxxxx};
    endcase

  reg [15:0] data;
  always @(posedge clk or negedge reset)
    if(!reset) data <= 0;
    else if(addr == 0)
      case(index)
        3'd0: data <= data0;
        3'd1: data <= data1;
        3'd2: data <= data2;
        3'd3: data <= data3;
        3'd4: data <= data4;
        3'd5: data <= data5;
        endcase
```

```
128
129     always @(addr or data)
130       case(addr)
131       4'h0,4'h1: hex = data[15:12];
132       4'h2,4'h3: hex = data[11:8];
133       4'h4,4'h5: hex = data[7:4];
134       4'h6,4'h7: hex = data[3:0];
135       default: hex = 4'hx;
136       endcase
137
138     always @(hex)
139       case(hex)
140       4'h0 : ascii = 8'h30;
141       4'h1 : ascii = 8'h31;
142       4'h2 : ascii = 8'h32;
143       4'h3 : ascii = 8'h33;
144       4'h4 : ascii = 8'h34;
145       4'h5 : ascii = 8'h35;
146       4'h6 : ascii = 8'h36;
147       4'h7 : ascii = 8'h37;
148       4'h8 : ascii = 8'h38;
149       4'h9 : ascii = 8'h39;
150       4'hA : ascii = 8'h41;
151       4'hB : ascii = 8'h42;
152       4'hC : ascii = 8'h43;
153       4'hD : ascii = 8'h44;
154       4'hE : ascii = 8'h45;
155       4'hF : ascii = 8'h46;
156       default : ascii = 8'hxx;
157       endcase
158
159 endmodule
```

5 State machine counter

List 3 is a top module that contains state machine, chattering removal circuit, LCD controller, and six counters. Six counters are used to enumerate the number of states as follows:

counter0 incremented if IDLE.

counter1 incremented if FETCHA.

counter2 incremented if FETCHB.

counter3 incremented if EXECA.

counter4 incremented if EXECB.

counter5 always incremented.

The output of counter0, ..., counter5 are connected to the LCD controller through wires (nets) data0, ..., data5.

List 3: Top module of the state machine counter state_top.v

```
1  `define IDLE 3'b000
2  `define FETCHA 3'b001
3  `define FETCHB 3'b010
4  `define EXECA 3'b011
5  `define EXECB 3'b100
6
7  module state_top(CLK50MHZ,ROT_A,BTN_SOUTH,SW,LED,
       LCD_E,LCD_RS,LCD_RW,SF_D);
8
9    input CLK50MHZ,ROT_A,BTN_SOUTH;
10   input [2:0] SW;
11   output [4:0] LED;
12   output LCD_RS,LCD_E,LCD_RW;
13   output [11:8] SF_D;
14   wire clk, reset;
15   wire [2:0] cs;
16   wire [15:0] data0,data1,data2,data3,data4,data5;
17
18   state satate0(.clk(clk),.reset(reset),.run(SW[2]),.cont(SW[1])
         ,.halt(SW[0]),.cs(cs));
19   chattering chattering0(.clk(CLK50MHZ),.reset(reset),.in(
         ROT_A),.out(clk));
20   lcdctrl lcdctrl0(.clk(CLK50MHZ),.reset(reset),.lcd_e(LCD_E)
         ,.lcd_rs(LCD_RS),.lcd_rw(LCD_RW),.sf_d(SF_D),.data0(
         data0),.data1(data1),.data2(data2),.data3(data3),.
         data4(data4),.data5(data5));
21   counter counter0 (.clk(clk),.reset(reset),.load(0),.inc(cs==
         `IDLE),.d(0),.q(data0));
22   counter counter1 (.clk(clk),.reset(reset),.load(0),.inc(cs==
         `FETCHA),.d(0),.q(data1));
23   counter counter2 (.clk(clk),.reset(reset),.load(0),.inc(cs==
         `FETCHB),.d(0),.q(data2));
24   counter counter3 (.clk(clk),.reset(reset),.load(0),.inc(cs==
         `EXECA),.d(0),.q(data3));
25   counter counter4 (.clk(clk),.reset(reset),.load(0),.inc(cs==
         `EXECB),.d(0),.q(data4));
26   counter counter5 (.clk(clk),.reset(reset),.load(0),.inc(1),.d(0)
         ,.q(data5));
27
28   assign reset = ~BTN_SOUTH;
29   assign LED[4] = (cs==`IDLE);
30   assign LED[3] = (cs==`FETCHA);
31   assign LED[2] = (cs==`FETCHB);
32   assign LED[1] = (cs==`EXECA);
33   assign LED[0] = (cs==`EXECB);
34
35 endmodule
```

List 4: UCF for the state machine counter state_top.ucf

```
1  # ROT PUSH BUTTON
2  NET "ROT_A" LOC = "T13" | IOSTANDARD = LVTTL |
       PULLUP ;
3
4  # PUSH SWITCH
5  NET "BTN_SOUTH" LOC = "T15" | IOSTANDARD =
       LVTTL | PULLDOWN ;
6
7  # LED
8  NET "LED<4>" LOC = "V19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
9  NET "LED<3>" LOC = "U19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
```

```
10 NET "LED<2>" LOC = "U20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
11 NET "LED<1>" LOC = "T19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
12 NET "LED<0>" LOC = "R20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
13
14 # SLIDE SWITCH
15 NET "SW<2>" LOC = "U8" | IOSTANDARD = LVTTL |
       PULLUP ;
16 NET "SW<1>" LOC = "U10" | IOSTANDARD = LVTTL |
       PULLUP ;
17 NET "SW<0>" LOC = "V8" | IOSTANDARD = LVTTL |
       PULLUP ;
18
19 # CLOCK
20 NET "CLK50MHZ" LOC = "E12" | IOSTANDARD =
       LVCMOS33 ;
21 NET "CLK50MHZ" PERIOD = 20.0ns HIGH 40%;
22
23 # LCD
24 NET "LCD_E" LOC = "AB4" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
25 NET "LCD_RS" LOC = "Y14" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
26 NET "LCD_RW" LOC = "W13" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
27 NET "SF_D<8>" LOC = "AA12" | IOSTANDARD = LVTTL
       | DRIVE = 4 | SLEW = QUIETIO ;
28 NET "SF_D<9>" LOC = "Y16" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
29 NET "SF_D<10>" LOC = "AB16" | IOSTANDARD =
       LVTTL | DRIVE = 4 | SLEW = QUIETIO ;
30 NET "SF_D<11>" LOC = "Y15" | IOSTANDARD = LVTTL
       | DRIVE = 4 | SLEW = QUIETIO ;
```

6 Memories: Distributed RAM and Block RAM

The FPGA has two types of memories: *distributed RAM*(List 5) and *block RAM*. Distributed RAM is implemented using flip flops and combinational circuits in the FPGA. Thus, Distributed RAM is costly, and its capacity is small. It supports *asynchronous read* and *synchronous write*.

On the other hand, the FPGA has *block RAM* as a building block. It has larger capacity than the distributed RAM and supports *synchronous read* and *synchronous write*.

Both types of memories have inputs, clk, load, addr, d and output q. Input load is an control bit and write operation is performed if it is 1. Input port addr (AWIDTH bits) specifies the address to be read/written, and d (DWIDTH bits) is the data to be written. Port q (DWIDTH bits) outputs the data specified by addr.

Both memories uses the array mem that has WORDS elements of DWIDTH bits each. Note that WORDS must be equal to 2^{AWIDTH}. In the distributed RAM, mem[addr] is always output to q. Thus, it supports asynchronous read, because the output q is determined regardless of clk. In the block RAM, mem[addr] is latched by q in the rising edge of clk. Hence it supports synchronous read. Both memories supports synchronous write, that is, input d is latched by mem[addr] in the rising edge of clk if load is 1.

List 5: Distributed ram dram.v

```
1  module dram(clk, load, addr, d, q);
2    parameter DWIDTH=8, AWIDTH=8, WORDS=256;
3
4    input clk, load;
5    input [AWIDTH-1:0] addr;
6    input [DWIDTH-1:0] d;
7    output [DWIDTH-1:0] q;
8    reg [DWIDTH-1:0] mem [WORDS-1:0];
9
10   always @(posedge clk)
11     if(load) mem[addr] <= d;
12
13   assign q = mem[addr];
14
15 endmodule
```

List 6: Block ram ram.v

```
1  module ram(clk, load, addr, d, q);
2    parameter DWIDTH=16,AWIDTH=10, WORDS=1024;
3
4    input clk,load;
5    input [AWIDTH-1:0] addr;
6    input [DWIDTH-1:0] d;
7    output [DWIDTH-1:0] q;
8    reg [DWIDTH-1:0] q;
9    reg [DWIDTH-1:0] mem [WORDS-1:0];
10
11   always @(posedge clk)
12     begin
13       if(load) mem[addr] <= d;
14       q <= mem[addr];
15     end
16
17 endmodule
```

Block RAM can be initialized using initial statements in List 8.

List 7: Test Bench ram_tb.v

```
1  `timescale 1ns / 1ps
2  module ram_tb;
3
4    reg clk, load;
5    reg [7:0] addr;
6    reg [15:0] d;
7    wire [15:0] q;
8
9    dram #(16,8,256) ram0(.clk(clk),.load(load),.addr(addr),.d(d
         ),.q(q));
10
11   initial begin
12     clk = 0;
13     forever
14       #50 clk = ~clk;
15   end
16
17   initial begin
18     addr = 0; load = 0; d = 0;
19     #100 addr = 1; load = 1; d = 3;
20     #100 addr = 2; load = 1; d = 6;
21     #100 addr = 1; load = 0; d = 0;
22     #100 addr = 2;
23     #100 addr = 3;
24     #100 addr = 0;
25   end
26
27 endmodule
```

List 8: Block ram with initialization ram.v

```
1  module ram(clk, load, addr, d, q);
2    parameter DWIDTH=16,AWIDTH=10,WORDS=1024;
3
4    input clk,load;
5    input [AWIDTH-1:0] addr;
6    input [DWIDTH-1:0] d;
7    output [DWIDTH-1:0] q;
8    reg [DWIDTH-1:0] q;
9    reg [DWIDTH-1:0] mem [WORDS-1:0];
10
11   always @(posedge clk)
12     begin
13       if(load) mem[addr] <= d;
14       q <= mem[addr];
15     end
16
17   integer i;
18   initial begin
19     for(i=0;i<WORDS;i=i+1)
20       mem[i]=0;
21     mem[0]=3;
22     mem[1]=6;
23     mem[2]=9;
24     mem[3]=12;
25     mem[4]=15;
26     mem[5]=18;
27     mem[6]=21;
28     mem[7]=24;
29   end
30
31 endmodule
```

7 Instruction fetch circuit

List 11 shows the instruction fetch circuit. It instantiates five modules, pc0 (program counter, 8-bit counter), ir0 (instruction register, 16-bit counter), obuf0 (output buffer, 16-bit counter), state0 (state machine), and ram0 (memory, 8-bit 256-word block RAM). Module pc0 is used to specify the address to be read to get (fetch) instruction code from ram0. The instruction code is stored in ir0. This instruction fetch operation is performed in two states, FETCHA and FETCHB.

In states EXECA andEXECB, the operation in Table 1 is executed. If the instruction code stored in ir0 is 0, then the next state is IDLE. If the lower 8 bits of the instruction is 0 and the upper 8 bits is non-0, then read the memory specified the upper 8 bits, and write it to obuf0. If the lower 8 bits of the instruction is non-0, then write the lower 8 bits to the memory with address upper 8 bits.

8 Homework

Illustrate the block diagram of the instruction fetch circuit. The block diagram should includes:

Table 1: Operation of instruction fetch circuit

Instruction Register ir0		Behavior
upper 8 bits	lower 8 bits	
0	0	move to IDLE
A ($\neq 0$)	0	read mem[A] and write it to obuf0
A	B ($\neq 0$)	write B in mem[A]

List 9: Top module for ramram_top.v

```
1  module ram_top(CLK50MHZ,ROT_A,BTN_SOUTH,LCD_E,
       LCD_RS,LCD_RW,SF_D);
2
3    input CLK50MHZ,ROT_A,BTN_SOUTH;
4    output LCD_RS,LCD_E,LCD_RW;
5    output [11:8] SF_D;
6    wire clk, reset;
7    wire [15:0] data0,data3;
8    reg [15:0] data1,data2,data4,data5;
9
10   chattering chattering0(.clk(CLK50MHZ),.reset(reset),.in(~
         ROT_A),.out(clk));
11   lcdctrl lcdctrl0(.clk(CLK50MHZ),.reset(reset),.lcd_e(LCD_E)
         ,.lcd_rs(LCD_RS),.lcd_rw(LCD_RW),.sf_d(SF_D),.data0(
         data0),.data1(data1),.data2(data2),.data3(data3),.
         data4(data4),.data5(data5));
12   counter counter0(.clk(clk),.reset(reset),.load(0),.inc(1),.d(0),.
         q(data0));
13   ram #(16,8,256) ram0(.clk(clk),.load(0),.addr(data0[7:0]),.d
         (0),.q(data3));
14
15   assign reset = ~BTN_SOUTH;
16
17   always @(posedge clk or negedge reset)
18   if(!reset)
19     begin
20       data1<=0; data2<=0; data4<=0; data5<=0;
21     end
22   else
23     begin
24       data1<=data0; data2<=data1; data4<=data3; data5
             <=data4;
25     end
26
27 endmodule
```

List 10: UCF ram_top.ucf for ram_top.v

```
1  # ROT PUSH BUTTON
2  NET "ROT_A" LOC = "T13" | IOSTANDARD = LVTTL |
       PULLUP ;
3
4  # PUSH SWITCH
5  NET "BTN_SOUTH" LOC = "T15" | IOSTANDARD =
       LVTTL | PULLDOWN ;
6
7  # CLOCK
8  NET "CLK50MHZ" LOC = "E12" | IOSTANDARD =
       LVCMOS33 ;
9  NET "CLK50MHZ" PERIOD = 20.0ns HIGH 40%;
10
11 # LCD
12 NET "LCD_E" LOC = "AB4" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
13 NET "LCD_RS" LOC = "Y14" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
14 NET "LCD_RW" LOC = "W13" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
15 NET "SF_D<8>" LOC = "AA12" | IOSTANDARD = LVTTL
       | DRIVE = 4 | SLEW = QUIETIO ;
16 NET "SF_D<9>" LOC = "Y16" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
17 NET "SF_D<10>" LOC = "AB16" | IOSTANDARD =
       LVTTL | DRIVE = 4 | SLEW = QUIETIO ;
18 NET "SF_D<11>" LOC = "Y15" | IOSTANDARD = LVTTL
       | DRIVE = 4 | SLEW = QUIETIO ;
```

List 11: Instruction fetch circuit fetch.v

```
1  `define IDLE 3'b000
2  `define FETCHA 3'b001
3  `define FETCHB 3'b010
4  `define EXECA 3'b011
5  `define EXECB 3'b100
6
7  module fetch(clk, reset, run, cs, pcout, irout, ramout, abus,
       dbus, out);
8
9      input clk, reset, run;
10     output [2:0] cs;
11     output [7:0] abus, pcout;
12     output [15:0] irout, ramout, dbus, out;
13     reg [7:0]     abus;
14     wire halt,read,write;
15
16     counter #(8) pc0(.clk(clk),.reset(reset),.load(0),.inc(cs ==
           `FETCHA),.d(0),.q(pcout));
17     counter #(16) ir0(.clk(clk),.reset(reset),.load(cs ==
           `FETCHB),.inc(0),.d(dbus),.q(irout));
18     state state0(.clk(clk),.reset(reset),.run(run),.halt(halt),.cont
           (read),.cs(cs));
19     ram #(16,8,256) ram0(.clk(clk),.load(cs == `EXECA &&
           write),.addr(abus),.d(dbus),.q(ramout));
20     counter #(16) obuf0(.clk(clk),.reset(reset),.load(cs ==
           `EXECB),.inc(0),.d(dbus),.q(out));
21
22     assign halt = (irout==0);
23     assign read = (irout[15:8]!=0 && irout[7:0]==0);
24     assign write = (!halt && !read);
25
26     always @(cs or irout or pcout)
27       if(cs == `FETCHA) abus = pcout;
28       else if(cs == `EXECA) abus = irout[15:8];
29       else abus = 12'hxxx;
30
31     assign dbus = (cs == `FETCHB || cs == `EXECB ?
           ramout : 16'hzzzz);
32     assign dbus = (cs == `EXECA ? {8'h00,irout[7:0]} : 16'
           hzzzz);
33
34 endmodule
```

List 12: Test bench fetch_tb.v for instruction fetch circuit

```
1  `timescale 1ns / 1ps
2  module fetch_tb;
3
4    reg clk, reset, run;
5    wire [2:0] cs;
6    wire [7:0] pcout, abus;
7    wire [15:0] irout, ramout, dbus, out;
8
9    fetch fetch0(.clk(clk),.reset(reset),.run(run),.cs(cs),.pcout(
         pcout),.irout(irout),.ramout(ramout),.abus(abus),.dbus(
         dbus),.out(out));
10
11   initial begin
12     clk = 0;
13     forever
14       #50 clk = ~clk;
15   end
16
17   initial begin
18     reset = 0; run = 0;
19     #100 reset = 1; run=1;
20     #100 run = 0;
21   end
22
23 endmodule
```

- All instantiated modules with their ports.
- Connection of ports.

You can illustrate abus and dbus as buses. After that, write the data and control flows in the block diagram for for all possible configuration of the instruction fetch circuit. More specifically, write the data and control flows when each of the following events occur:

1. the state transition from IDLE to FETCHA,
2. the state transition from FETCHA to FETCHB,
3. the state transition from FETCHB to EXECA,
4. the state transition from EXECA to next state, and each of the following cases:

   case 1 irout[15:8]==0 and irout[7:0]==0

   case 2 irout[15:8]!=0 and irout[7:0]==0

   case 3 irout[15:8]!=0 and irout[7:0]!=0

5. the state transition from EXECB to IDLE

List 13: Block ram ram.v for instruction fetch circuit

```
1  module ram(clk, load, addr, d, q);
2    parameter DWIDTH=16,AWIDTH=10,WORDS=1024;
3
4    input clk,load;
5    input [AWIDTH-1:0] addr;
6    input [DWIDTH-1:0] d;
7    output [DWIDTH-1:0] q;
8    reg [DWIDTH-1:0] q;
9    reg [DWIDTH-1:0] mem [WORDS-1:0];
10
11   always @(posedge clk)
12     begin
13       if(load) mem[addr] <= d;
14       q <= mem[addr];
15     end
16
17   integer i;
18   initial begin
19     for(i=0;i<WORDS;i=i+1)
20       mem[i]=0;
21     mem[0]=16'h1012;
22     mem[1]=16'h1134;
23     mem[2]=16'h1256;
24     mem[3]=16'h1378;
25     mem[4]=16'h1000;
26     mem[5]=16'h1100;
27     mem[6]=16'h1200;
28     mem[7]=16'h1300;
29     mem[8]=16'h0000;
30   end
31
32 endmodule
```

List 14: top module fetch_top.v for instruction fetch circuit

```
1  module fetch_top(CLK50MHZ,ROT_A,ROT_CENTER,
       BTN_SOUTH,LED,LCD_E,LCD_RS,LCD_RW,SF_D);
2
3    input CLK50MHZ,ROT_A,ROT_CENTER,BTN_SOUTH;
4    output [4:0] LED;
5    output LCD_RS,LCD_E,LCD_RW;
6    output [11:8] SF_D;
7    wire clk, reset;
8    wire [2:0] cs;
9    wire [15:0] data0,data3,data1,data2,data4,data5;
10
11   chattering chattering0(.clk(CLK50MHZ), .reset(reset), .in(~
         ROT_A), .out(clk));
12   lcdctrl lcdctrl0(.clk(CLK50MHZ), .reset(reset), .lcd_e(LCD_E
         ), .lcd_rs(LCD_RS), .lcd_rw(LCD_RW), .sf_d(SF_D), .
         data0(data0), .data1(data1), .data2(data2), .data3(
         data3), .data4(data4), .data5(data5));
13   fetch fetch0(.clk(clk), .reset(reset), .run(ROT_CENTER), .cs
         (cs), .pcout(data0), .irout(data1), .ramout(data2), .
         abus(data3), .dbus(data4), .out(data5));
14
15   assign reset = ~BTN_SOUTH;
16   assign LED[4] = (cs=='IDLE);
17   assign LED[3] = (cs=='FETCHA);
18   assign LED[2] = (cs=='FETCHB);
19   assign LED[1] = (cs=='EXECA);
20   assign LED[0] = (cs=='EXECB);
21
22 endmodule
```

List 15: UCF fetch_top.ucf for top module of instruction fetch circuit

```
1  # ROT PUSH BUTTON
2  NET "ROT_A" LOC = "T13" | IOSTANDARD = LVTTL |
       PULLUP ;
3  NET "ROT_CENTER" LOC = "R13" | IOSTANDARD =
       LVTTL | PULLDOWN ;
4
5  # PUSH BUTTON
6  NET "BTN_SOUTH" LOC = "T15" | IOSTANDARD =
       LVTTL | PULLDOWN ;
7
8  # LED
9  NET "LED<4>" LOC = "V19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
10 NET "LED<3>" LOC = "U19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
11 NET "LED<2>" LOC = "U20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
12 NET "LED<1>" LOC = "T19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
13 NET "LED<0>" LOC = "R20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
14
15 # CLOCK
16 NET "CLK50MHZ" LOC = "E12" | IOSTANDARD =
       LVCMOS33 ;
17 NET "CLK50MHZ" PERIOD = 20.0ns HIGH 40%;
18
19 # LCD
20 NET "LCD_E" LOC = "AB4" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
21 NET "LCD_RS" LOC = "Y14" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
22 NET "LCD_RW" LOC = "W13" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
23 NET "SF_D<8>" LOC = "AA12" | IOSTANDARD = LVTTL
       | DRIVE = 4 | SLEW = QUIETIO ;
24 NET "SF_D<9>" LOC = "Y16" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
25 NET "SF_D<10>" LOC = "AB16" | IOSTANDARD =
       LVTTL | DRIVE = 4 | SLEW = QUIETIO ;
26 NET "SF_D<11>" LOC = "Y15" | IOSTANDARD = LVTTL
       | DRIVE = 4 | SLEW = QUIETIO ;
```