Active-HDL PDF Export
project workspace

Contents

2 8-bit

2.1 bc.v

```
module bc(in, rw, sel, out);

  input in, rw, sel;
  output out;

  wire w1, w2, w3, w4;
  // not n1(w6, in);
  // not n2(w7, rw);
  and a1(w1, sel, in, ~rw);
  and a2(w2, sel, ~rw, ~in);
  nor nr1(w3, w1, w4);
  nor nr2(w4, w2, w3);
  and a3(out, sel, w4, rw);

endmodule
```

2.2 decoder.v

```
// module sram(id[0], id[1], id[2], o0, o1, o2, o3, o4, o5, o6, o7);

//   input id[0], id[1], id[2];
//   output o0, o1, o2, o3, o4, o5, o6, o7;

//   o0(~id[0]&~id[1]&~id[2]),
//   o1(~id[0]&~id[1]&id[2]),
//   o2(~id[0]&id[1]&~id[2]),
//   o3(~id[0]&id[1]&id[2]),
//   o4(id[0]&~id[1]&~id[2]),
//   o5(id[0]&~id[1]&id[2]),
//   o6(id[0]&id[1]&~id[2]),
//   o7(id[0]&id[1]&id[2]);

// endmodule

module decoder(id, od);

  input [2:0]id;
  output [7:0]od;
  and a0(od[0], ~id[0], ~id[1], ~id[2]);
  and a1(od[1], ~id[0], ~id[1], id[2]);
  and a2(od[2], ~id[0], id[1], ~id[2]);
  and a3(od[3], ~id[0], id[1], id[2]);
  and a4(od[4], id[0], ~id[1], ~id[2]);
  and a5(od[5], id[0], ~id[1], id[2]);
  and a6(od[6], id[0], id[1], ~id[2]);
  and a7(od[7], id[0], id[1], id[2]);

endmodule
```

2.3 sram.v

```
module sram(ins, rws, addr, outs);

  input [7:0]ins;
  input [2:0]addr;
```

```
  input rws;
  output [7:0]outs;
  reg [7:0]rowout[0:7];
  wire [7:0]word;
  wire [15:0]halfout;

  //   bt(inb, rwb, selb, outb);
  decoder sel(.od(word), .id(addr));

  // module bt(inb[8], rwb, selb, outb[8]);
  bt row0(.outb(rowout[0]), .inb(ins), .rwb(rws), .selb(word[0]));
  bt row1(.outb(rowout[1]), .inb(ins), .rwb(rws), .selb(word[1]));
  bt row2(.outb(rowout[2]), .inb(ins), .rwb(rws), .selb(word[2]));
  bt row3(.outb(rowout[3]), .inb(ins), .rwb(rws), .selb(word[3]));
  bt row4(.outb(rowout[4]), .inb(ins), .rwb(rws), .selb(word[4]));
  bt row5(.outb(rowout[5]), .inb(ins), .rwb(rws), .selb(word[5]));
  bt row6(.outb(rowout[6]), .inb(ins), .rwb(rws), .selb(word[6]));
  bt row7(.outb(rowout[7]), .inb(ins), .rwb(rws), .selb(word[7]));

  //   or outbit0(outs[0], rowout[0][0], rowout[1][0], rowout[2][0], rowout[3
][0],
  //              rowout[4][0], rowout[5][0], rowout[6][0], rowout[7][0]);
  // output bit 0
  or outh00(halfout[0],  rowout[0][0], rowout[1][0], rowout[2][0], rowout[3][
0]);
  or outh01(halfout[1], rowout[4][0], rowout[5][0], rowout[6][0], rowout[7][0
]);
  or outbit0(outs[0], halfout[0], halfout[1]);


  //   or outbit1(outs[1], rowout[0][1], rowout[1][1], rowout[2][1], rowout[3
][1],
  //              rowout[4][1], rowout[5][1], rowout[6][1], rowout[7][1]);
  //   outout bit 1
  or outbit10(halfout[2], rowout[0][1], rowout[1][1], rowout[2][1], rowout[3]
[1]);
  or outbit11(halfout[3], rowout[4][1], rowout[5][1], rowout[6][1], rowout[7]
[1]);
  or outbit1(outs[1], halfout[2], halfout[3]);

  //   or outbit2(outs[2], rowout[0][2], rowout[1][2], rowout[2][2], rowout[3
][2],
  //              rowout[4][2], rowout[5][2], rowout[6][2], rowout[7][2]);
  or outbit20(halfout[4], rowout[0][2], rowout[1][2], rowout[2][2], rowout[3]
[2]);
  or outbit21(halfout[5], rowout[4][2], rowout[5][2], rowout[6][2], rowout[7]
[2]);
  or boutbit2(outs[2], halfout[4], halfout[5]);


  //   or outbit3(outs[3], rowout[0][3], rowout[1][3], rowout[2][3], rowout[3
][3],
  //              rowout[4][3], rowout[5][3], rowout[6][3], rowout[7][3]);
  or outbit30(halfout[6], rowout[0][3], rowout[1][3], rowout[2][3], rowout[3]
[3]);
  or outbit31(halfout[7], rowout[4][3], rowout[5][3], rowout[6][3], rowout[7]
[3]);
  or outbit3(outs[3], halfout[6], halfout[7]);

  //   or outbit4(outs[4], rowout[0][4], rowout[1][4], rowout[2][4], rowout[3
][4],
  //              rowout[4][4], rowout[5][4], rowout[6][4], rowout[7][4]);
  or outbit40(halfout[8], rowout[0][4], rowout[1][4], rowout[2][4], rowout[3]
```

```
[4]);
  or outbit41(halfout[9], rowout[4][4], rowout[5][4], rowout[6][4], rowout[7]
[4]);
  or outbit4(outs[4], halfout[8], halfout[9]);

  //   or outbit5(outs[5], rowout[0][5], rowout[1][5], rowout[2][5], rowout[3
][5],
  //              rowout[4][5], rowout[5][5], rowout[6][5], rowout[7][5]);
  or outbit50(halfout[10], rowout[0][5], rowout[1][5], rowout[2][5], rowout[3
][5]);
  or outbit51(halfout[11], rowout[4][5], rowout[5][5], rowout[6][5], rowout[7
][5]);
  or outbit5(outs[5], halfout[10], halfout[11]);


  //   or outbit6(outs[6], rowout[0][6], rowout[1][6], rowout[2][6], rowout[3
][6],
  //              rowout[4][6], rowout[5][6], rowout[6][6], rowout[7][6]);
  or outbit60(halfout[12], rowout[0][6], rowout[1][6], rowout[2][6], rowout[3
][6]);
  or outbit61(halfout[13], rowout[4][6], rowout[5][6], rowout[6][6], rowout[7
][6]);
  or outbit6(outs[6], halfout[12], halfout[13]);


  //   or outbit7(outs[7], rowout[0][7], rowout[1][7], rowout[2][7], rowout[3
][7],
  //              rowout[4][7], rowout[5][7], rowout[6][7], rowout[7][7]);
  or outbit70(halfout[14], rowout[0][7], rowout[1][7], rowout[2][7], rowout[3
][7]);
  or outbit71(halfout[15], rowout[4][7], rowout[5][7], rowout[6][7], rowout[7
][7]);
  or outbit7(outs[7], halfout[14], halfout[15]);

endmodule
```

2.4 sram.awc

2.5 bt.v

```
module bt(inb, rwb, selb, outb);

  input [7:0]inb;
  input selb;
  input rwb;
  output [7:0]outb;

  // module bc(in, rw, sel, out);
  bc clm0(.out(outb[0]), .in(inb[0]), .rw(rwb), .sel(selb));
  bc clm1(.out(outb[1]), .in(inb[1]), .rw(rwb), .sel(selb));
  bc clm2(.out(outb[2]), .in(inb[2]), .rw(rwb), .sel(selb));
  bc clm3(.out(outb[3]), .in(inb[3]), .rw(rwb), .sel(selb));
  bc clm4(.out(outb[4]), .in(inb[4]), .rw(rwb), .sel(selb));
  bc clm5(.out(outb[5]), .in(inb[5]), .rw(rwb), .sel(selb));
  bc clm6(.out(outb[6]), .in(inb[6]), .rw(rwb), .sel(selb));
  bc clm7(.out(outb[7]), .in(inb[7]), .rw(rwb), .sel(selb));

endmodule
```

2.6 state.v

```
module state(rwin, sclk, enable, datain, saddr, dataout);
  input sclk, enable, rwin;
  input [0:7]datain;
  input [2:0]saddr;
  output [7:0]dataout;
  wire ddata, dout, sout;

  dflipflop states(.qata(dout), .clk(sclk), .data(rwin));
  or dd(sout, dout, ~enable);
  and fsm(ddata, sout, enable);
  sram ss(.outs(dataout), .ins(datain), .rws(dout), .addr(saddr));

endmodule
```

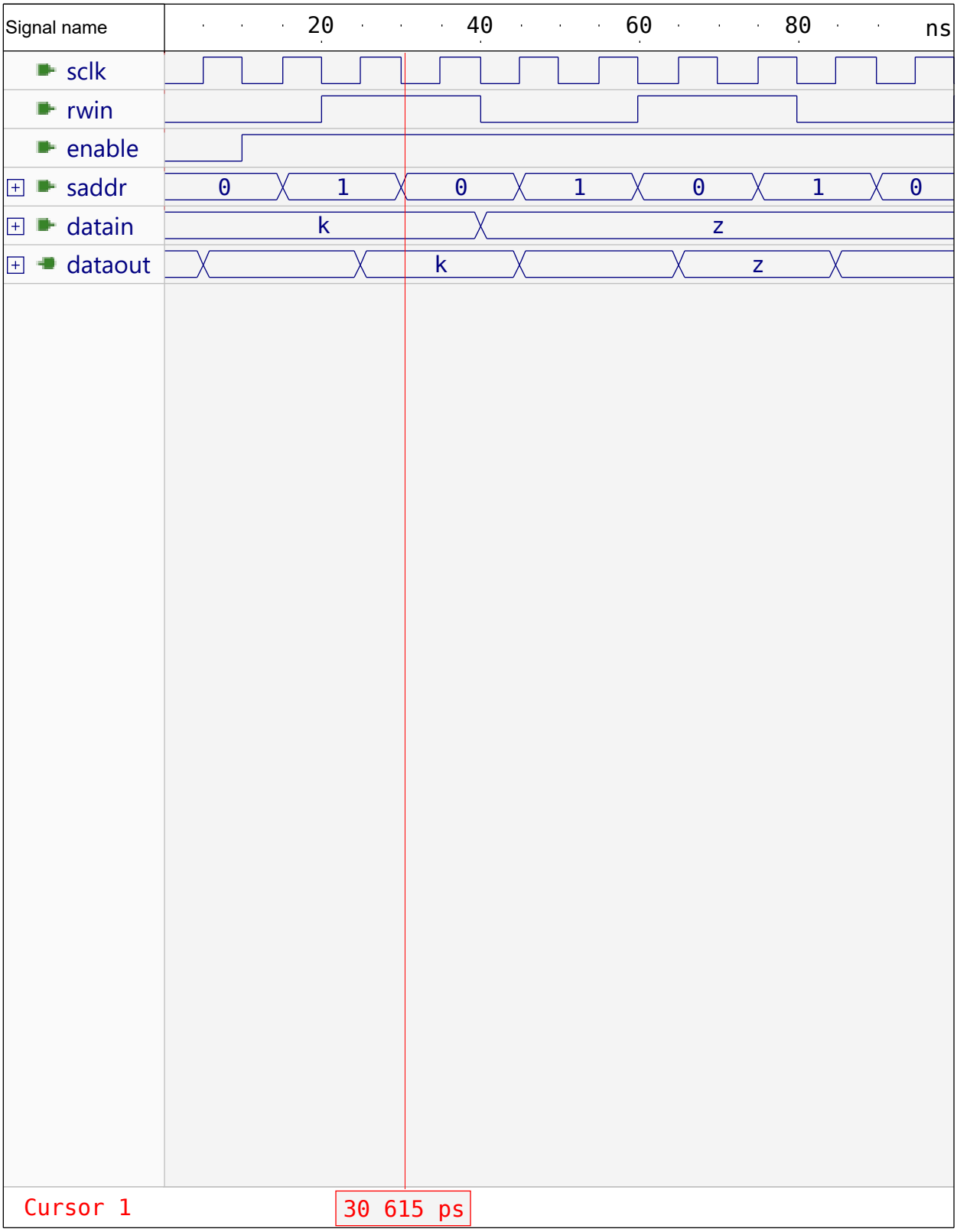
2.7 dflipflop.v

```
module dflipflop(data, clk, qata);

  input data, clk;
  output qata;

  wire w1, w2, w3, w4, w5, w6;
  nand a1(w1, w2, w4);
  nand a2(w2, w1, clk);
  nand a3(w3, w2, clk, w4);
  nand a4(w4, w3, data);
  nand a5(qata, w2, w6);
  nand a6(w6, w3, qata);

endmodule
```

2.8 state_tb.v

```
module state_tb;

  // Parameters

  // Ports
  reg sclk = 0;
  reg  enable = 0;
  reg  rwin = 0;
  reg [7:0] datain;
  reg [2:0] saddr;
  wire [7:0] dataout;

  state
    state_dut (
      .sclk (sclk ),
      . enable ( enable ),
      . rwin ( rwin ),
      .datain (datain ),
      .saddr (saddr ),
      .dataout  ( dataout)
    );

  initial
  begin
    repeat (10)
    begin
      #5 sclk = !sclk;
      #3;
      rwin = ~rwin;
      enable = 1'b1;
      datain = 8'b01101011;
      saddr = 3'b000;
    end
  end

  // always
  //   #5  sclk = ! sclk ;

endmodule
```

2.9 state.awc


Signal name
20
40
60
80
ns
sclk
rwin
enable
saddr
0
1
0
1
0
1
0
datain
k
z
dataout
k
z
Cursor 1
30 615 ps