11/03/2019

EE446 LABORATORY

EXPERIMENT 1

PRELIMINARY REPORT

Muttalip Caner TOL

2031466

Tuesday Afternoon

1.2. Module Design with Verilog HDL

1.2.1. Constant Value Generator

```verilog
module Constant_Value_Generator #(parameter W = 4, V = 4'b0000) (bus_in);

    output wire [(W-1):0] bus_in;

    assign bus_in = V;

endmodule
```

1.2.2. Decoder

1. Implementation

```verilog
module decoder #(parameter IN_WIDTH = 3) (
    in, out
);
    input [(IN_WIDTH-1):0] in;
    output reg [((1<<IN_WIDTH)-1):0] out;

    always @(in)
    begin
        out = {(1<<IN_WIDTH){1'b0}};
        out[in] = 1'b1;
    end

endmodule
```

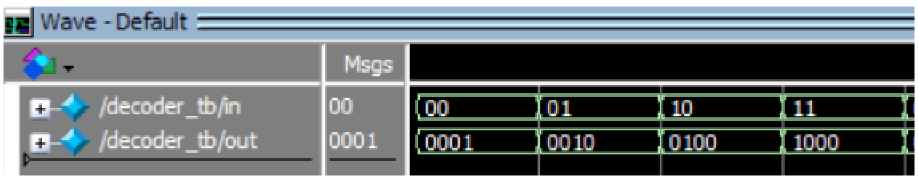
2. Test bench module

```verilog
module decoder_tb();
    reg [1:0] in;
    wire [3:0] out;
    reg[1:0] testvectors[3:0]; // array of testvectors
    reg [1:0] vectornum;

    decoder #2 DUT(in, out);

    // load vectors
    initial
    begin
    $readmemb("C:/Users/Caner/Documents/GitHub/Course-Projects/EE446/LAB1/decoder_tb.tv",testvectors);
    vectornum = 0;
    end

    //apply test vectors
    always
    begin
     #1;
     in = testvectors[vectornum];
     vectornum = vectornum +1;
    end

endmodule
```

3. Vector table

```
00
01
10
11
```

4. Verification



```
/decoder_tb/in   00     00   | 01   | 10   | 11
/decoder_tb/out  0001   0001 | 0010 | 0100 | 1000
```

1.2.3. Multiplexers

1. Implementations

```verilog
module mux2 #(parameter W = 4)( select, mux_in1, mux_in2, mux_out );

input      select;
input[W-1:0] mux_in1, mux_in2;
output wire [W-1:0] mux_out;

assign mux_out = select ? mux_in2 : mux_in1;

endmodule
```

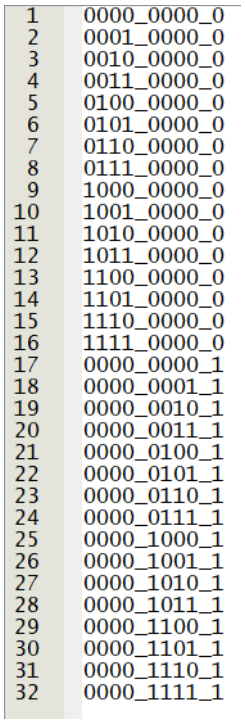
```verilog
module mux4 #(parameter W = 3)( select, mux_in1,mux_in2,mux_in3,mux_in4, mux_out );

    input[1:0] select;
    input[W:0] mux_in1,mux_in2,mux_in3,mux_in4;
    output reg [W:0] mux_out;
    always @(select, mux_in1,mux_in2,mux_in3,mux_in4)
    begin
        case(select)
            2'b00: mux_out = mux_in1;
            2'b01: mux_out = mux_in2;
            2'b10: mux_out = mux_in3;
            2'b11: mux_out = mux_in4;
        endcase
    end

endmodule
```

2. Test bench modules

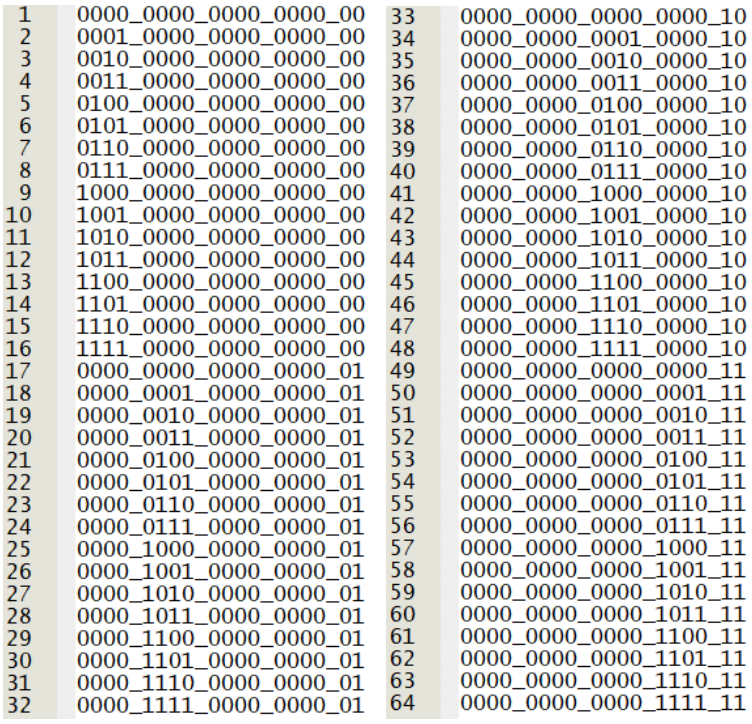
```verilog
module mux2_tb();
    reg select;
    reg[3:0] mux_in1, mux_in2;
    wire [3:0] mux_out;
    reg[8:0] testvectors[31:0]; // array of testvectors
    reg [4:0] vectornum;

    mux2 #4 DUT(select, mux_in1, mux_in2, mux_out);

    // load vectors
    initial
    begin
    $readmemb("C:/Users/Caner/Documents/GitHub/Course-Projects/EE446/LAB1/mux2_tb.tv",testvectors);
    vectornum = 0;
    end

    //apply test vectors
    always
    begin
     #5;
     {mux_in1, mux_in2,select} = testvectors[vectornum];
     vectornum = vectornum +1;
    end

endmodule
```

3. Vector tables

   Test vectors of mux2 and mux4:



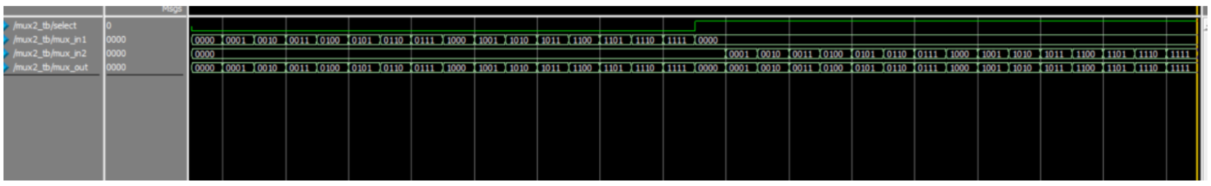
```
0000_0000_0
0001_0000_0
0010_0000_0
0011_0000_0
0100_0000_0
0101_0000_0
0110_0000_0
0111_0000_0
1000_0000_0
1001_0000_0
1010_0000_0
1011_0000_0
1100_0000_0
1101_0000_0
1110_0000_0
1111_0000_0
0000_0000_1
0000_0001_1
0000_0010_1
0000_0011_1
0000_0100_1
0000_0101_1
0000_0110_1
0000_0111_1
0000_1000_1
0000_1001_1
0000_1010_1
0000_1011_1
0000_1100_1
0000_1101_1
0000_1110_1
0000_1111_1
```



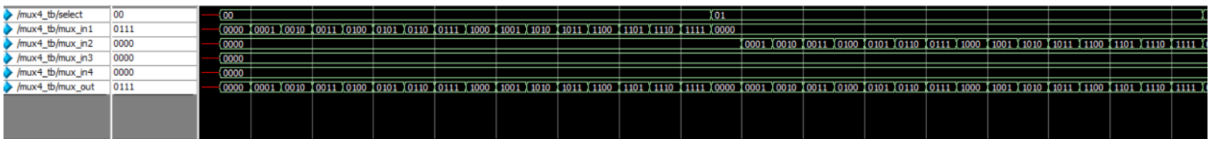
```
0000_0000_0000_0000_00
0001_0000_0000_0000_00
0010_0000_0000_0000_00
0011_0000_0000_0000_00
0100_0000_0000_0000_00
0101_0000_0000_0000_00
0110_0000_0000_0000_00
0111_0000_0000_0000_00
1000_0000_0000_0000_00
1001_0000_0000_0000_00
1010_0000_0000_0000_00
1011_0000_0000_0000_00
1100_0000_0000_0000_00
1101_0000_0000_0000_00
1110_0000_0000_0000_00
1111_0000_0000_0000_00
0000_0000_0000_0000_01
0000_0001_0000_0000_01
0000_0010_0000_0000_01
0000_0011_0000_0000_01
0000_0100_0000_0000_01
0000_0101_0000_0000_01
0000_0110_0000_0000_01
0000_0111_0000_0000_01
0000_1000_0000_0000_01
0000_1001_0000_0000_01
0000_1010_0000_0000_01
0000_1011_0000_0000_01
0000_1100_0000_0000_01
0000_1101_0000_0000_01
0000_1110_0000_0000_01
0000_1111_0000_0000_01
0000_0000_0000_0000_10
0000_0000_0001_0000_10
0000_0000_0010_0000_10
0000_0000_0011_0000_10
0000_0000_0100_0000_10
0000_0000_0101_0000_10
0000_0000_0110_0000_10
0000_0000_0111_0000_10
0000_0000_1000_0000_10
0000_0000_1001_0000_10
0000_0000_1010_0000_10
0000_0000_1011_0000_10
0000_0000_1100_0000_10
0000_0000_1101_0000_10
0000_0000_1110_0000_10
0000_0000_1111_0000_10
0000_0000_0000_0000_11
0000_0000_0000_0001_11
0000_0000_0000_0010_11
0000_0000_0000_0011_11
0000_0000_0000_0100_11
0000_0000_0000_0101_11
0000_0000_0000_0110_11
0000_0000_0000_0111_11
0000_0000_0000_1000_11
0000_0000_0000_1001_11
0000_0000_0000_1010_11
0000_0000_0000_1011_11
0000_0000_0000_1100_11
0000_0000_0000_1101_11
0000_0000_0000_1110_11
0000_0000_0000_1111_11
```

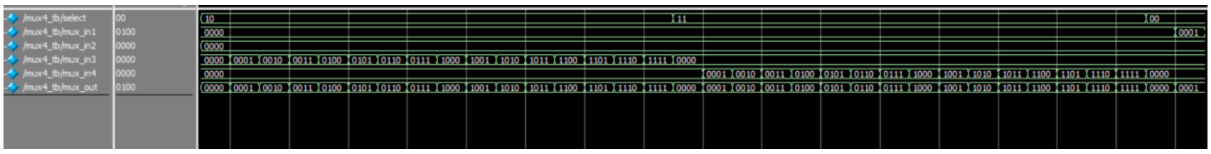
4. Verification

   Simulation results for mux2:



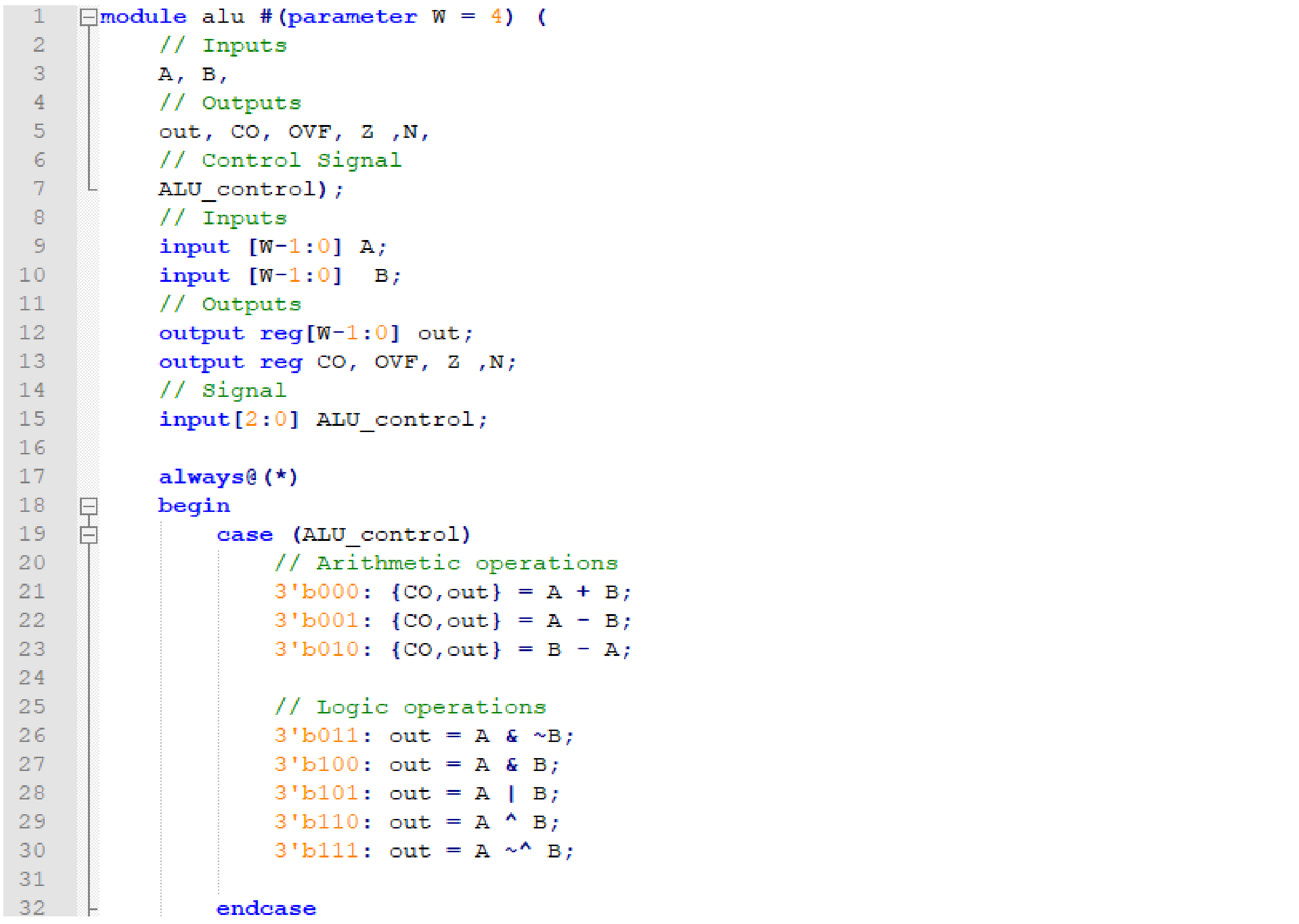
   Simulation results for mux4:



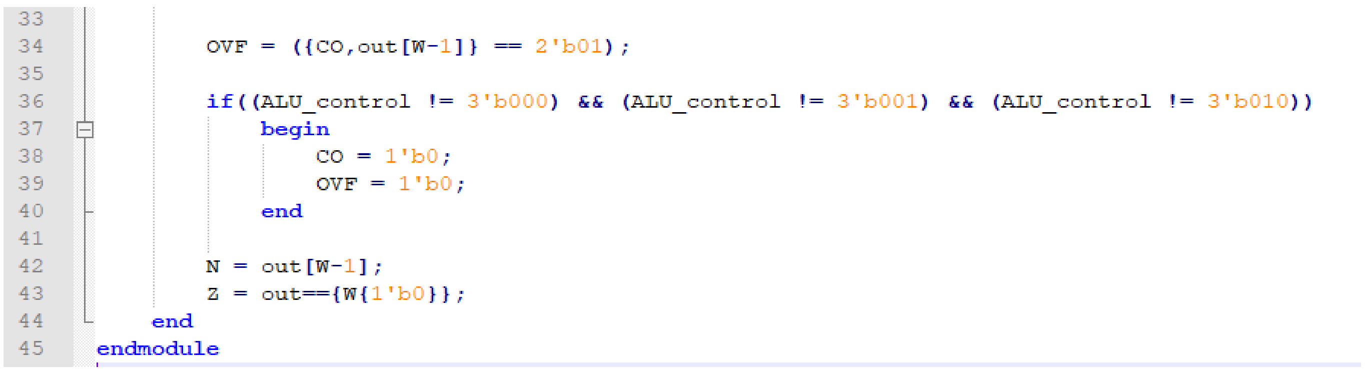


1.2.4. Arithmetic Logic Unit (ALU)

1. Implementation



```verilog
module alu #(parameter W = 4) (
    // Inputs
    A, B,
    // Outputs
    out, CO, OVF, Z ,N,
    // Control Signal
    ALU_control);
    // Inputs
    input [W-1:0] A;
    input [W-1:0]  B;
    // Outputs
    output reg[W-1:0] out;
    output reg CO, OVF, Z ,N;
    // Signal
    input[2:0] ALU_control;

    always@(*)
    begin
        case (ALU_control)
            // Arithmetic operations
            3'b000: {CO,out} = A + B;
            3'b001: {CO,out} = A - B;
            3'b010: {CO,out} = B - A;

            // Logic operations
            3'b011: out = A & ~B;
            3'b100: out = A & B;
            3'b101: out = A | B;
            3'b110: out = A ^ B;
            3'b111: out = A ~^ B;

        endcase
```

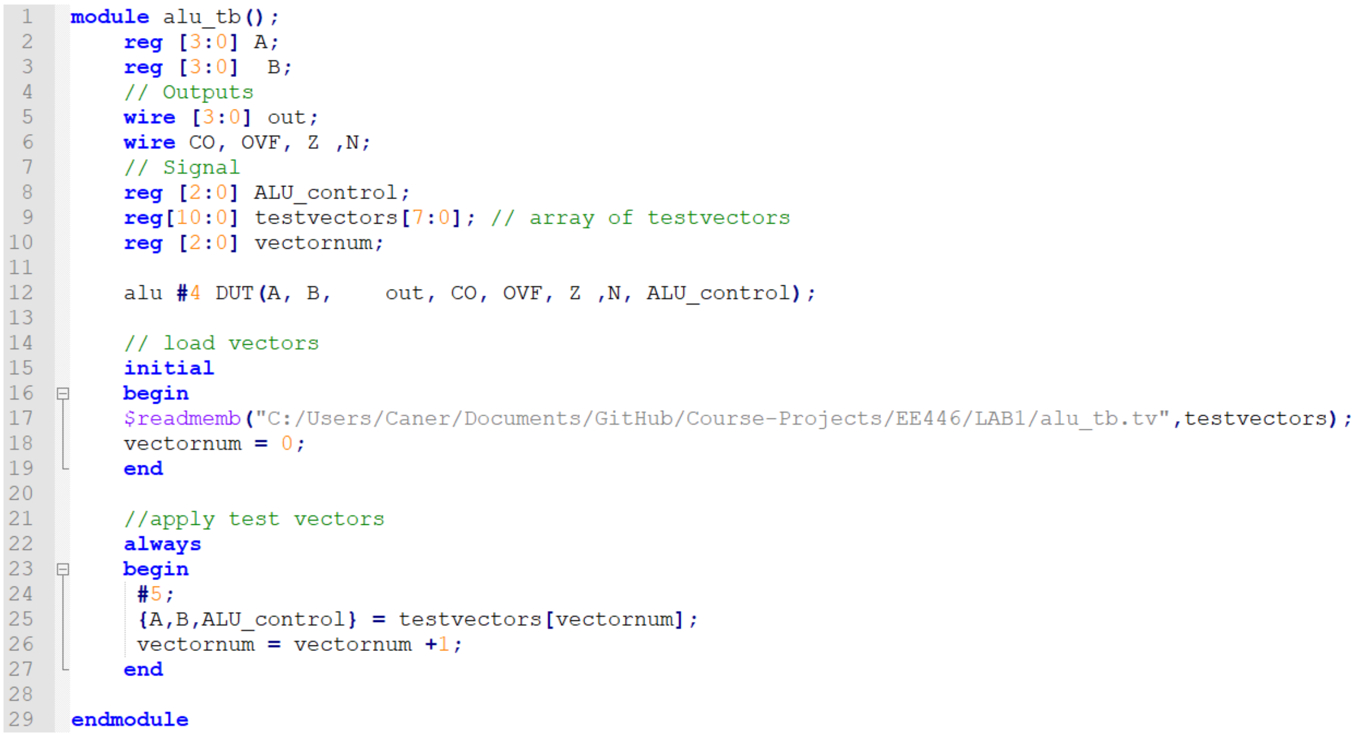


```verilog

        OVF = ({CO,out[W-1]} == 2'b01);

        if((ALU_control != 3'b000) && (ALU_control != 3'b001) && (ALU_control != 3'b010))
            begin
                CO = 1'b0;
                OVF = 1'b0;
            end

        N = out[W-1];
        Z = out=={W{1'b0}};
    end
endmodule
```

2. Overflow detection

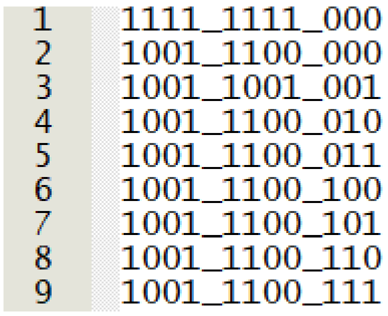
   I have put checkpoints on the arithmetic operations. Overflow flag is set when:
   - adding two positives yields a negative
   - or, adding two negatives gives a positive
   - or, subtract a negative from a positive gives a negative
   - or, subtract a positive from a negative gives a positive

3. Test bench module



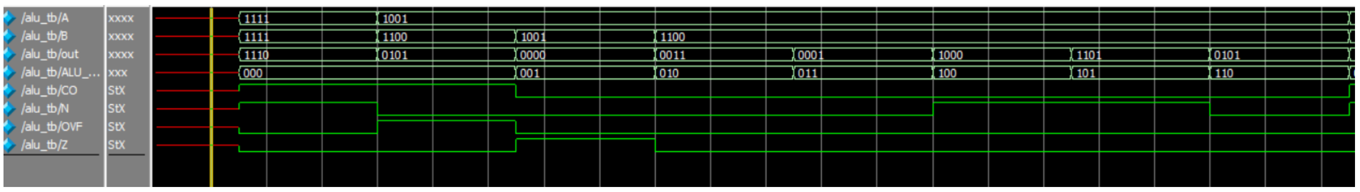
```verilog
module alu_tb();
    reg [3:0] A;
    reg [3:0]  B;
    // Outputs
    wire [3:0] out;
    wire CO, OVF, Z ,N;
    // Signal
    reg [2:0] ALU_control;
    reg[10:0] testvectors[7:0]; // array of testvectors
    reg [2:0] vectornum;

    alu #4 DUT(A, B,    out, CO, OVF, Z ,N, ALU_control);

    // load vectors
    initial
    begin
    $readmemb("C:/Users/Caner/Documents/GitHub/Course-Projects/EE446/LAB1/alu_tb.tv",testvectors);
    vectornum = 0;
    end

    //apply test vectors
    always
    begin
     #5;
     {A,B,ALU_control} = testvectors[vectornum];
     vectornum = vectornum +1;
    end

endmodule
```

4. Vector table



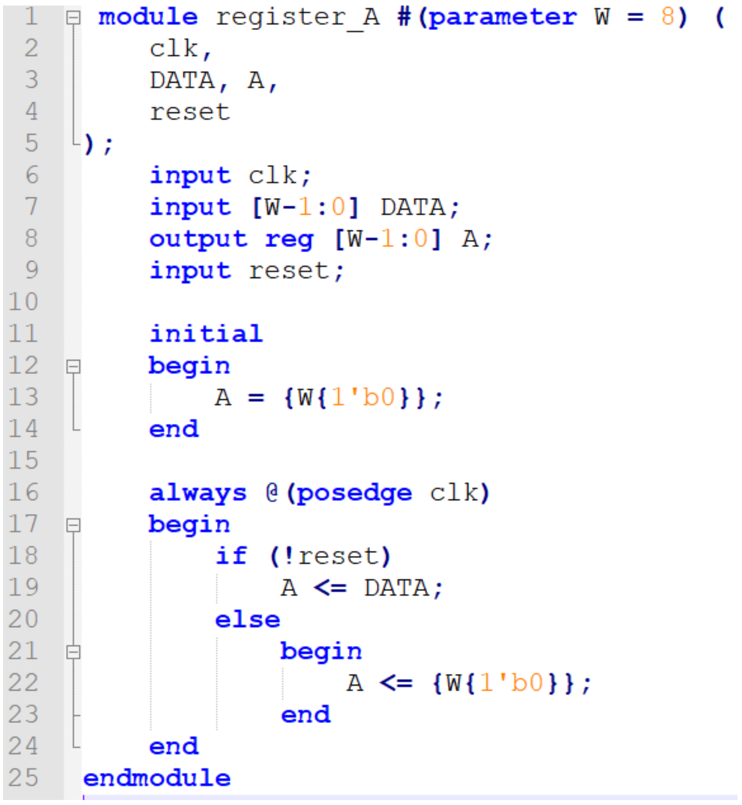
```
1111_1111_000
1001_1100_000
1001_1001_001
1001_1100_010
1001_1100_011
1001_1100_100
1001_1100_101
1001_1100_110
1001_1100_111
```

5. Verification



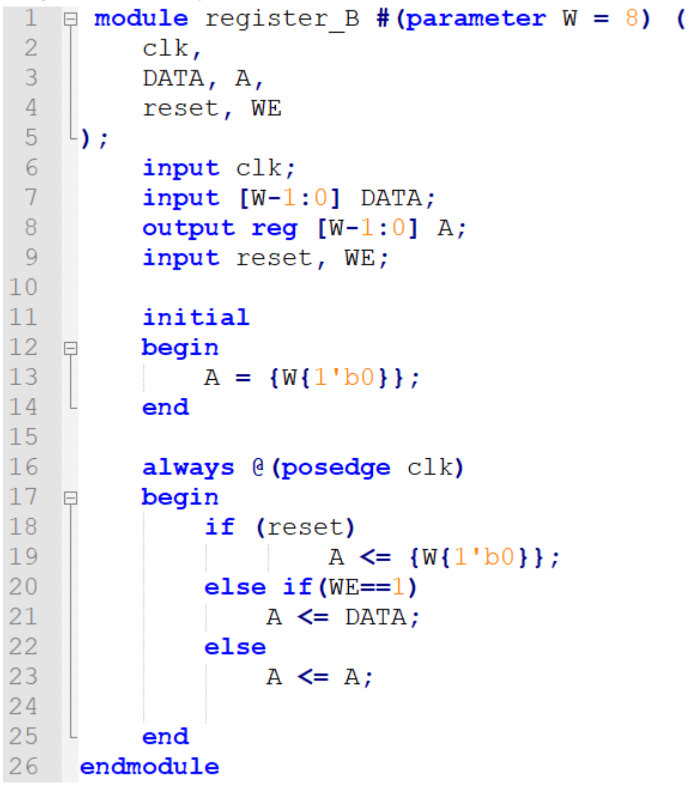
1.2.5. Registers

1. Simple register with synchronous reset



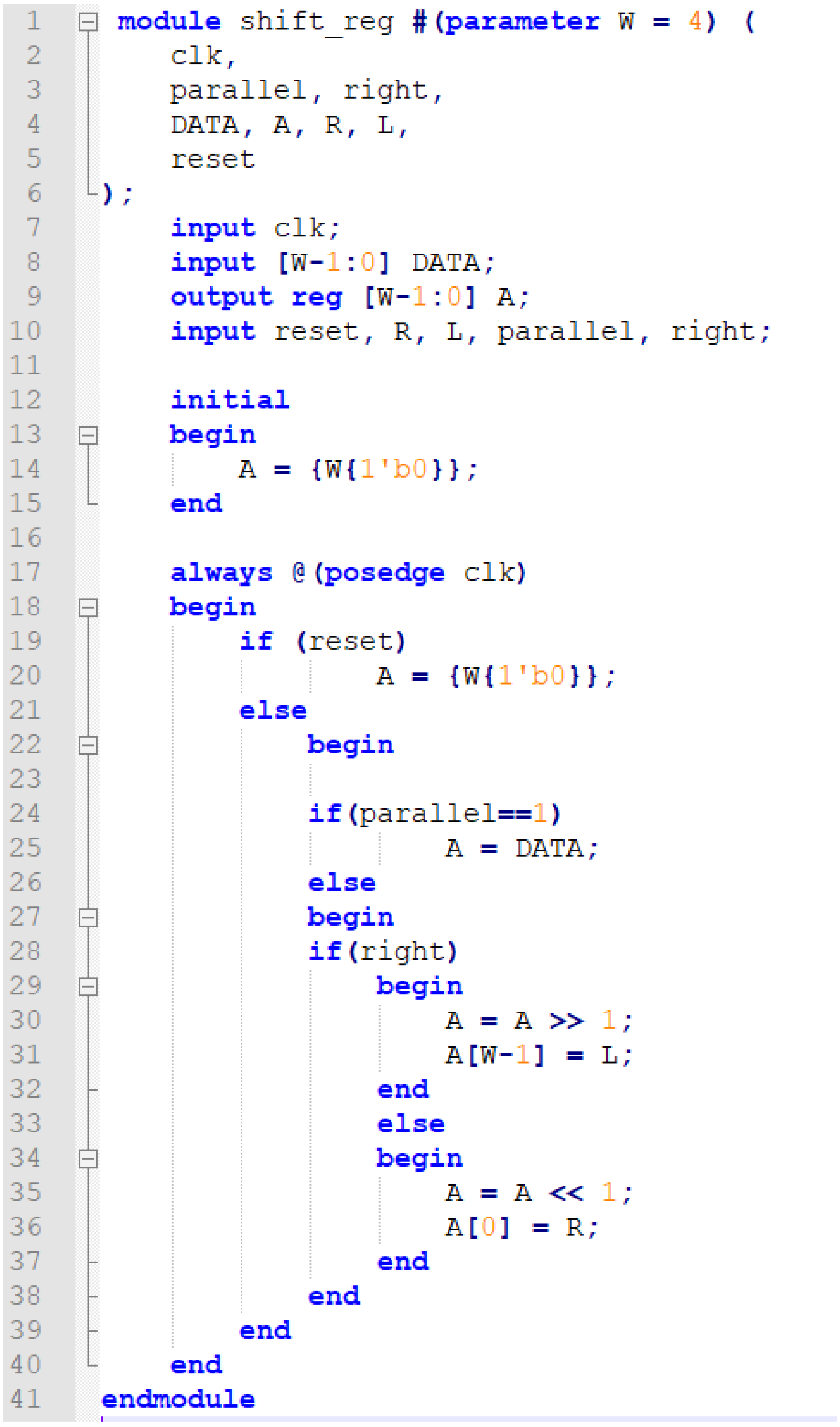
```verilog
module register_A #(parameter W = 8)  (
    clk,
    DATA, A,
    reset
);
    input clk;
    input [W-1:0] DATA;
    output reg [W-1:0] A;
    input reset;

    initial
    begin
        A = {W{1'b0}};
    end

    always @(posedge clk)
    begin
        if (!reset)
            A <= DATA;
        else
            begin
                A <= {W{1'b0}};
            end
    end
endmodule
```

2. Register with synchronous reset and write enable



```verilog
module register_B #(parameter W = 8)  (
    clk,
    DATA, A,
    reset, WE
);
    input clk;
    input [W-1:0] DATA;
    output reg [W-1:0] A;
    input reset, WE;

    initial
    begin
        A = {W{1'b0}};
    end

    always @(posedge clk)
    begin
        if (reset)
                A <= {W{1'b0}};
        else if(WE==1)
            A <= DATA;
        else
            A <= A;

    end
endmodule
```

3. Shift register with parallel and serial load



```
module shift_reg #(parameter W = 4) (
    clk,
    parallel, right,
    DATA, A, R, L,
    reset
);
    input clk;
    input [W-1:0] DATA;
    output reg [W-1:0] A;
    input reset, R, L, parallel, right;

    initial
    begin
        A = {W{1'b0}};
    end

    always @(posedge clk)
    begin
        if (reset)
                A = {W{1'b0}};
        else
            begin

            if(parallel==1)
                    A = DATA;
            else
            begin
            if(right)
                begin
                    A = A >> 1;
                    A[W-1] = L;
                end
                else
                begin
                    A = A << 1;
                    A[0] = R;
                end
            end
        end
    end
endmodule
```

4. Test bench modules

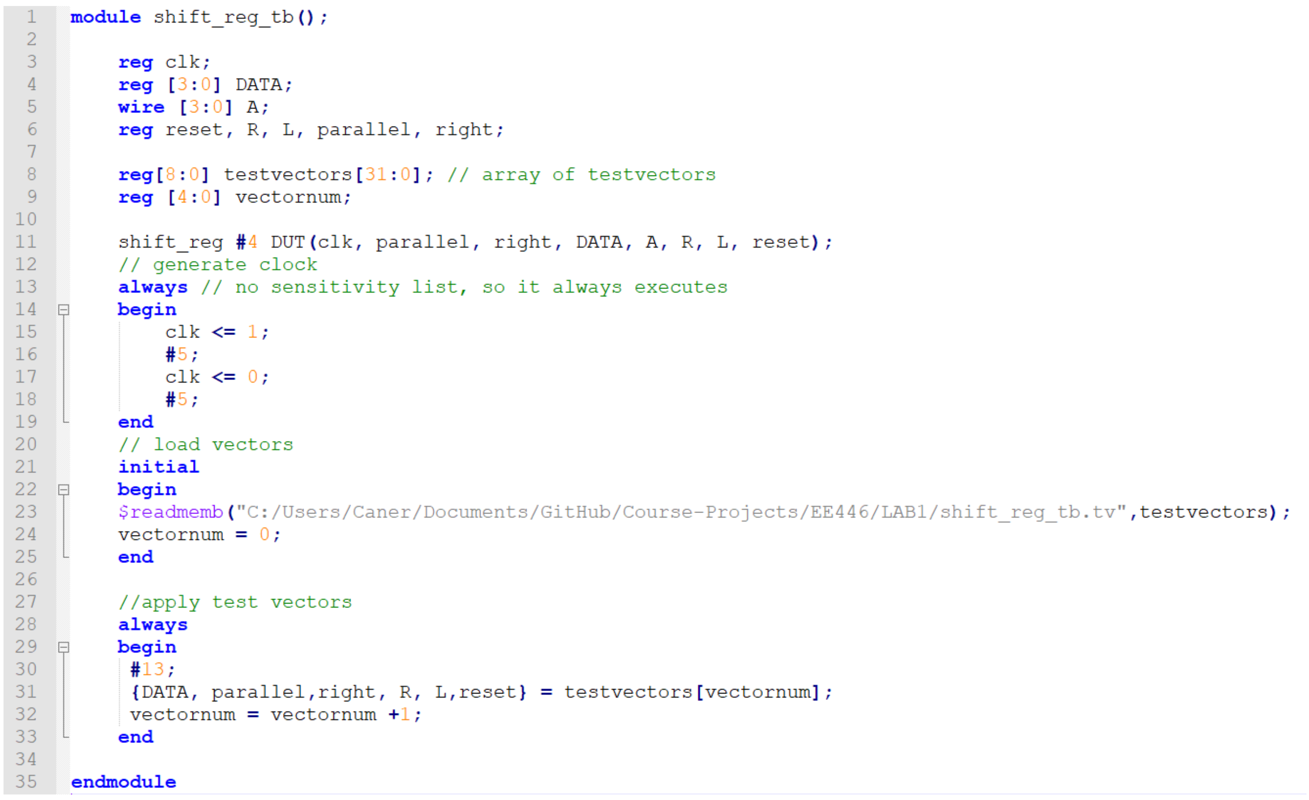
For Simple Register with synchronous reset:

```verilog
module register_A_tb();

    reg clk;
    reg [3:0] DATA;
    wire [3:0] A;
    reg reset;

    reg[4:0] testvectors[31:0]; // array of testvectors
    reg [4:0] vectornum;

    register_A #4 DUT(clk, DATA, A, reset);
    // generate clock
    always // no sensitivity list, so it always executes
    begin
        clk <= 1;
        #5;
        clk <= 0;
        #5;
    end
    // load vectors
    initial
    begin
    $readmemb("C:/Users/Caner/Documents/GitHub/Course-Projects/EE446/LAB1/register_A_tb.tv",testvectors);
    vectornum = 0;
    end

    //apply test vectors
    always
    begin
     #13;
     {DATA, reset} = testvectors[vectornum];
     vectornum = vectornum +1;
    end

endmodule
```

For Register with synchronous reset and write enable:

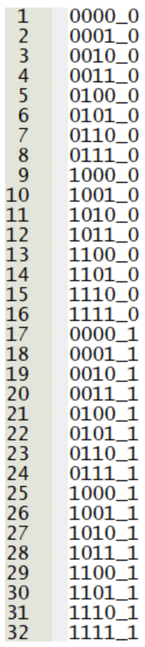
```verilog
module register_B_tb();

    reg clk;
    reg [3:0] DATA;
    wire [3:0] A;
    reg reset, WE;

    reg[5:0] testvectors[31:0]; // array of testvectors
    reg [4:0] vectornum;

    register_B #4 DUT(clk, DATA, A, reset, WE);
    // generate clock
    always // no sensitivity list, so it always executes
    begin
        clk <= 1;
        #5;
        clk <= 0;
        #5;
    end
    // load vectors
    initial
    begin
    $readmemb("C:/Users/Caner/Documents/GitHub/Course-Projects/EE446/LAB1/register_B_tb.tv",testvectors);
    vectornum = 0;
    end

    //apply test vectors
    always
    begin
     #13;
     {DATA, reset, WE} = testvectors[vectornum];
     vectornum = vectornum +1;
    end

endmodule
```

For Shift register with parallel and serial load:

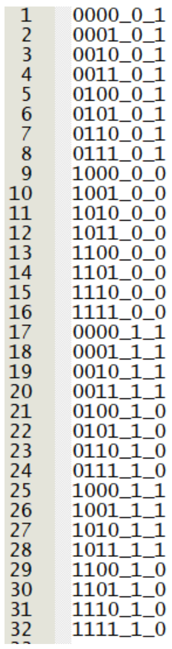


```verilog
module shift_reg_tb();

    reg clk;
    reg [3:0] DATA;
    wire [3:0] A;
    reg reset, R, L, parallel, right;

    reg[8:0] testvectors[31:0]; // array of testvectors
    reg [4:0] vectornum;

    shift_reg #4 DUT(clk, parallel, right, DATA, A, R, L, reset);
    // generate clock
    always // no sensitivity list, so it always executes
    begin
        clk <= 1;
        #5;
        clk <= 0;
        #5;
    end
    // load vectors
    initial
    begin
    $readmemb("C:/Users/Caner/Documents/GitHub/Course-Projects/EE446/LAB1/shift_reg_tb.tv",testvectors);
    vectornum = 0;
    end

    //apply test vectors
    always
    begin
     #13;
     {DATA, parallel,right, R, L,reset} = testvectors[vectornum];
     vectornum = vectornum +1;
    end

endmodule
```

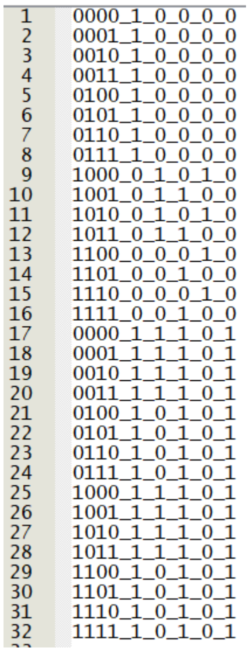
5. Vector tables



```
0000_0
0001_0
0010_0
0011_0
0100_0
0101_0
0110_0
0111_0
1000_0
1001_0
1010_0
1011_0
1100_0
1101_0
1110_0
1111_0
0000_1
0001_1
0010_1
0011_1
0100_1
0101_1
0110_1
0111_1
1000_1
1001_1
1010_1
1011_1
1100_1
1101_1
1110_1
1111_1
```



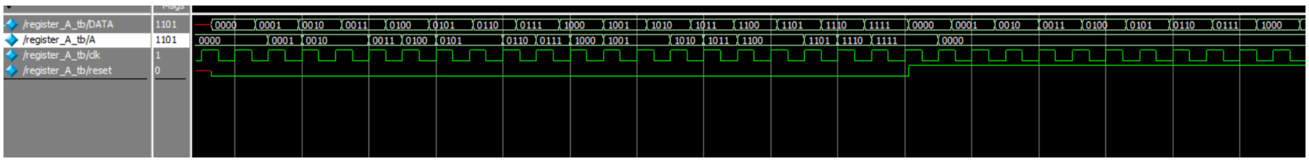
```
0000_0_1
0001_0_1
0010_0_1
0011_0_1
0100_0_1
0101_0_1
0110_0_1
0111_0_1
1000_0_0
1001_0_0
1010_0_0
1011_0_0
1100_0_0
1101_0_0
1110_0_0
1111_0_0
0000_1_1
0001_1_1
0010_1_1
0011_1_1
0100_1_0
0101_1_0
0110_1_0
0111_1_0
1000_1_1
1001_1_1
1010_1_1
1011_1_1
1100_1_0
1101_1_0
1110_1_0
1111_1_0
```



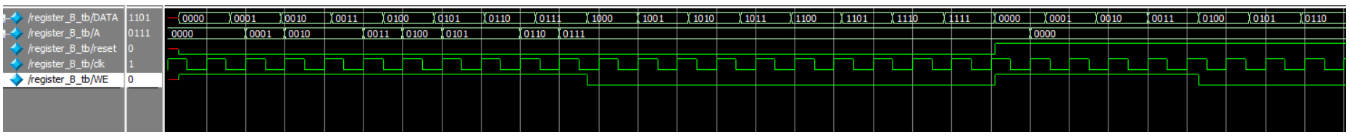
```
0000_1_0_0_0_0
0001_1_0_0_0_0
0010_1_0_0_0_0
0011_1_0_0_0_0
0100_1_0_0_0_0
0101_1_0_0_0_0
0110_1_0_0_0_0
0111_1_0_0_0_0
1000_0_1_0_1_0
1001_0_1_1_0_0
1010_0_1_0_1_0
1011_0_1_1_0_0
1100_0_0_0_1_0
1101_0_0_1_0_0
1110_0_0_0_1_0
1111_0_0_1_0_0
0000_1_1_1_0_1
0001_1_1_1_0_1
0010_1_1_1_0_1
0011_1_1_1_0_1
0100_1_0_1_0_1
0101_1_0_1_0_1
0110_1_0_1_0_1
0111_1_0_1_0_1
1000_1_1_1_0_1
1001_1_1_1_0_1
1010_1_1_1_0_1
1011_1_1_1_0_1
1100_1_0_1_0_1
1101_1_0_1_0_1
1110_1_0_1_0_1
1111_1_0_1_0_1
```

6. Verification

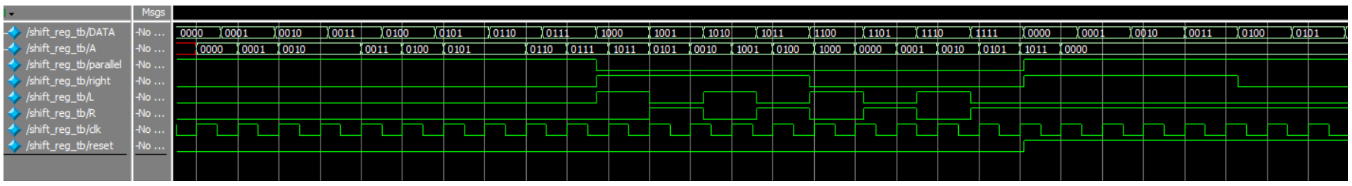
Simulation of Simple register with synchronous reset



Simulation of Register with synchronous reset and write enable

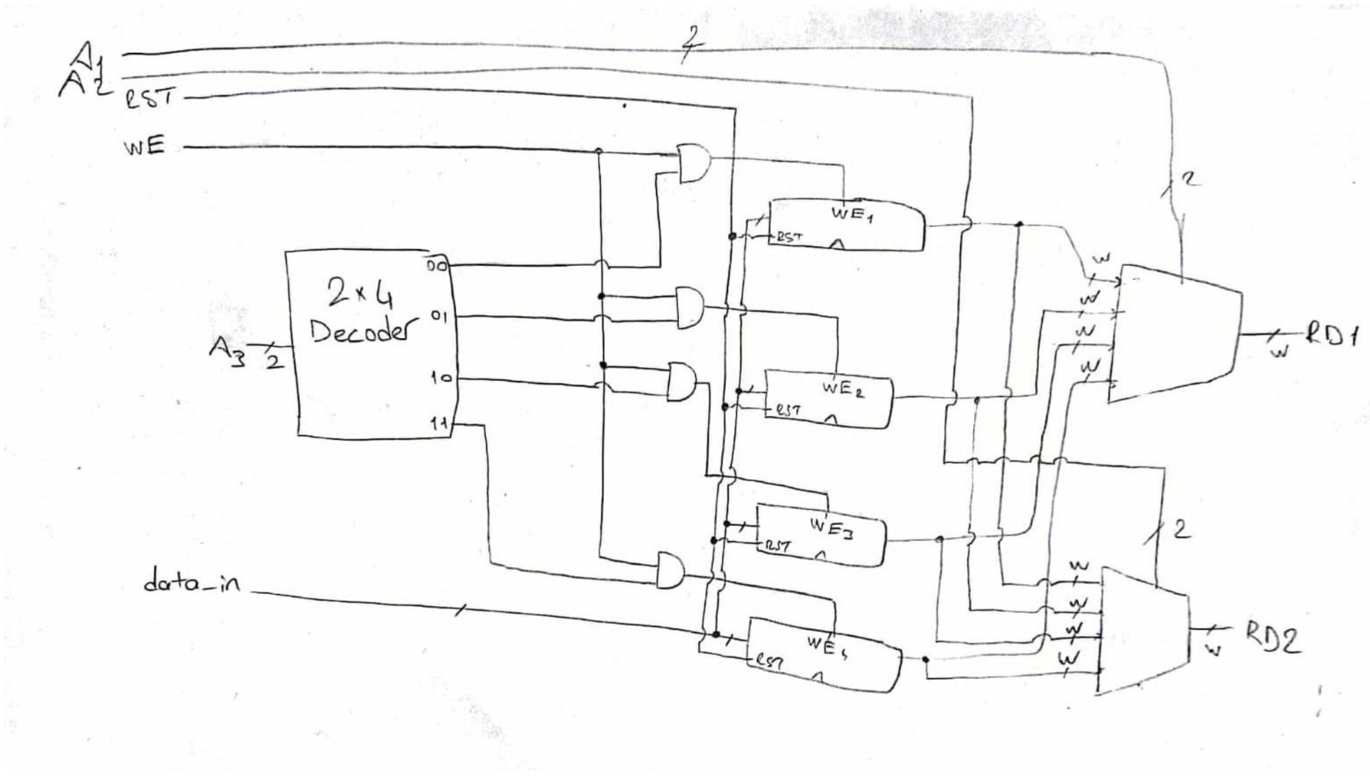


Simulation of Shift register with parallel and serial load

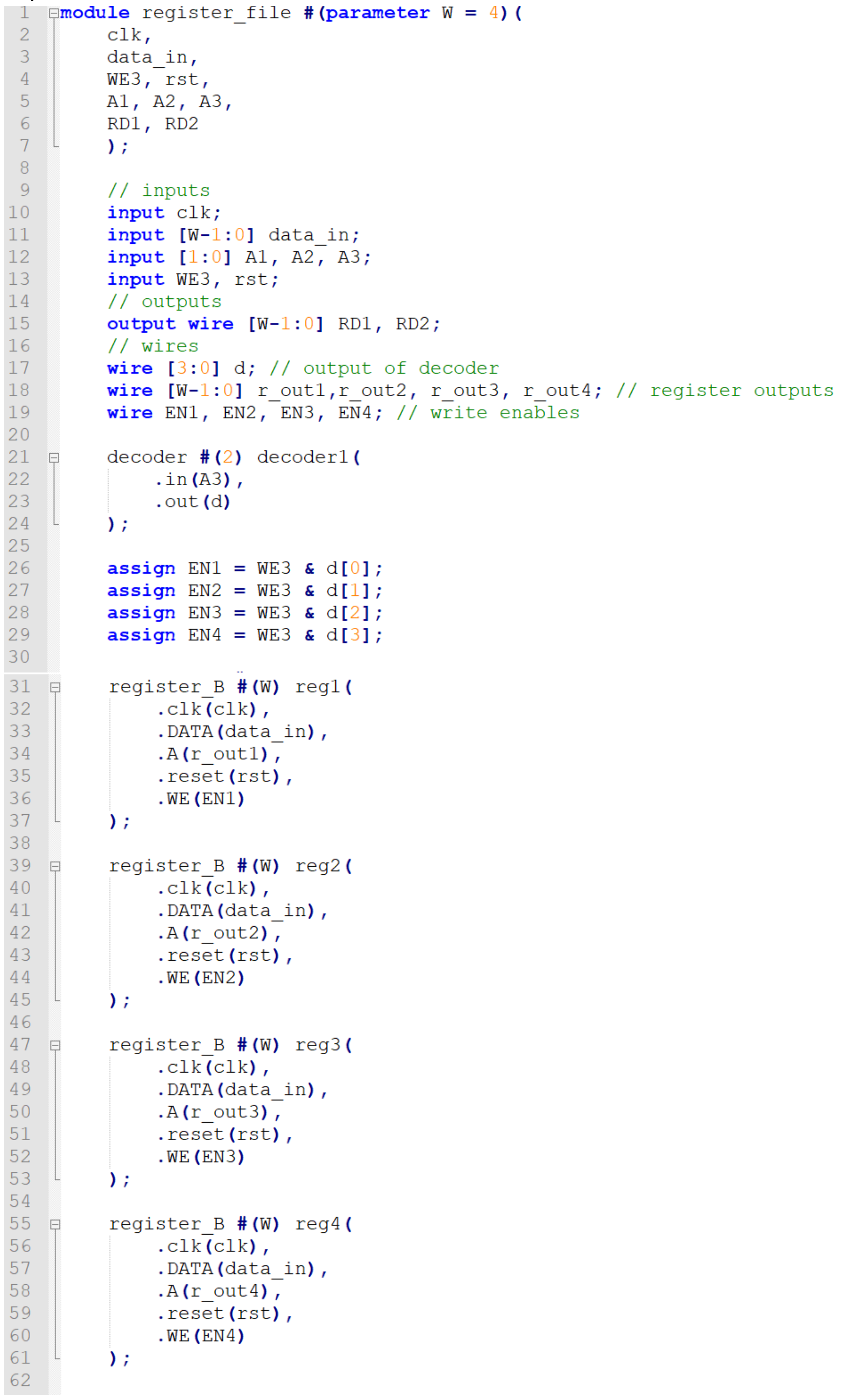


1.3. Register File

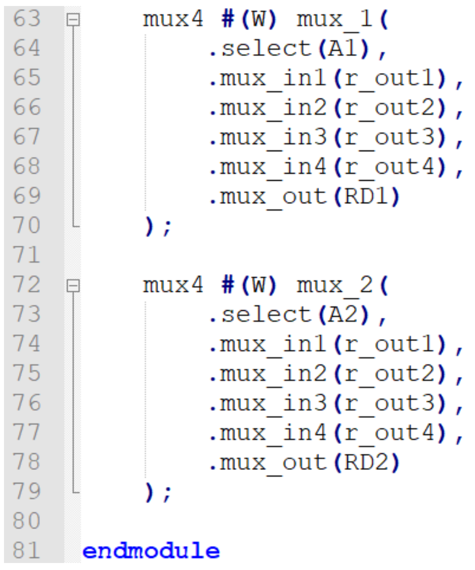
1. Design



2. Implementation

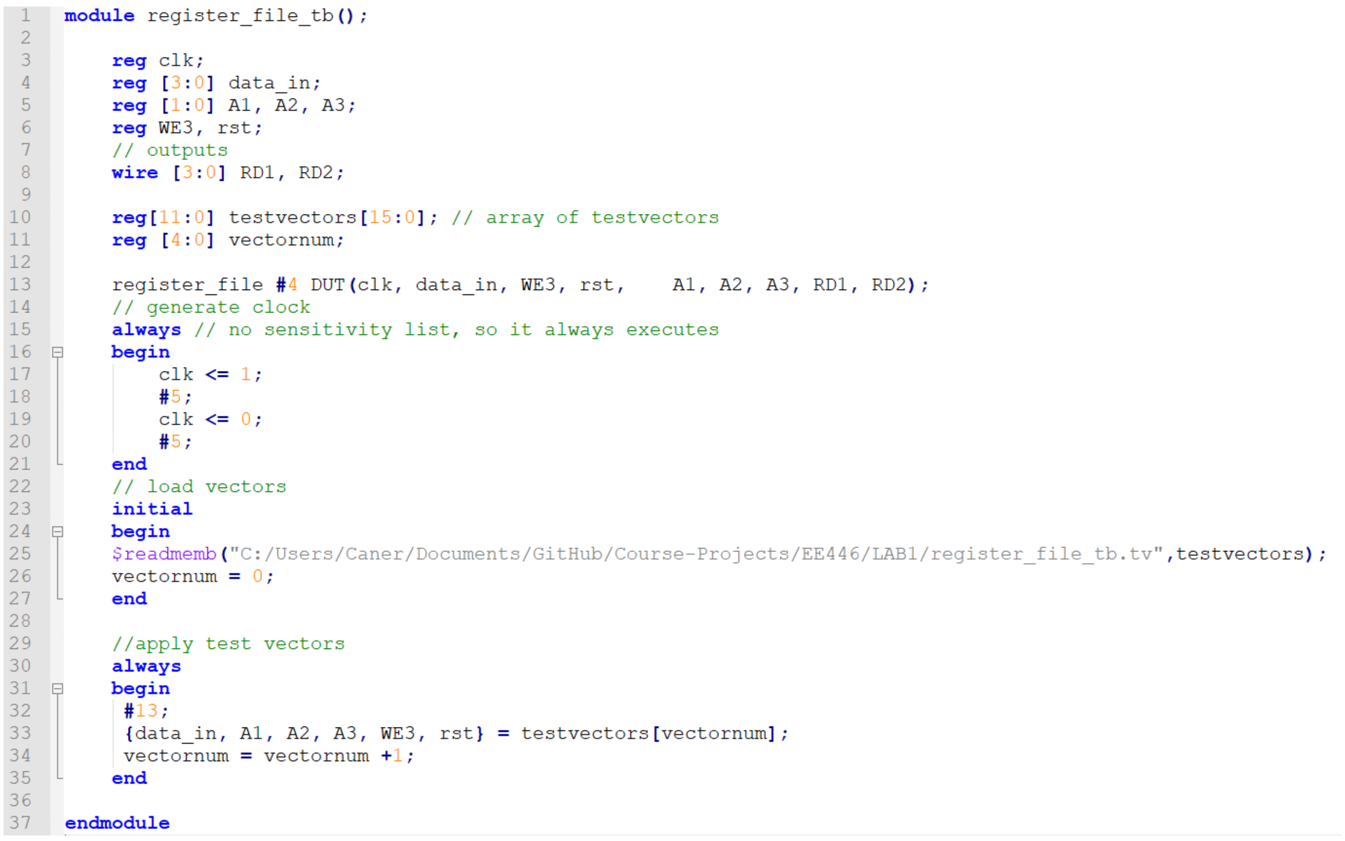


```
module register_file #(parameter W = 4)(
  clk,
  data_in,
  WE3, rst,
  A1, A2, A3,
  RD1, RD2
  );

  // inputs
  input clk;
  input [W-1:0] data_in;
  input [1:0] A1, A2, A3;
  input WE3, rst;
  // outputs
  output wire [W-1:0] RD1, RD2;
  // wires
  wire [3:0] d; // output of decoder
  wire [W-1:0] r_out1,r_out2, r_out3, r_out4; // register outputs
  wire EN1, EN2, EN3, EN4; // write enables

  decoder #(2) decoder1(
    .in(A3),
    .out(d)
  );

  assign EN1 = WE3 & d[0];
  assign EN2 = WE3 & d[1];
  assign EN3 = WE3 & d[2];
  assign EN4 = WE3 & d[3];

  register_B #(W) reg1(
    .clk(clk),
    .DATA(data_in),
    .A(r_out1),
    .reset(rst),
    .WE(EN1)
  );

  register_B #(W) reg2(
    .clk(clk),
    .DATA(data_in),
    .A(r_out2),
    .reset(rst),
    .WE(EN2)
  );

  register_B #(W) reg3(
    .clk(clk),
    .DATA(data_in),
    .A(r_out3),
    .reset(rst),
    .WE(EN3)
  );

  register_B #(W) reg4(
    .clk(clk),
    .DATA(data_in),
    .A(r_out4),
    .reset(rst),
    .WE(EN4)
  );

```



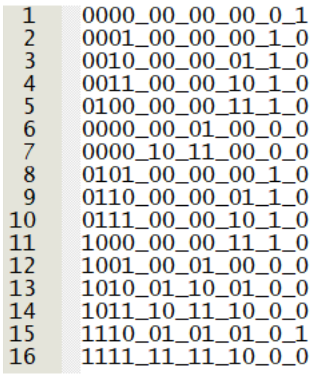
```verilog
	mux4 #(W) mux_1(
		.select(A1),
		.mux_in1(r_out1),
		.mux_in2(r_out2),
		.mux_in3(r_out3),
		.mux_in4(r_out4),
		.mux_out(RD1)
	);

	mux4 #(W) mux_2(
		.select(A2),
		.mux_in1(r_out1),
		.mux_in2(r_out2),
		.mux_in3(r_out3),
		.mux_in4(r_out4),
		.mux_out(RD2)
	);

endmodule
```

3. Test bench module



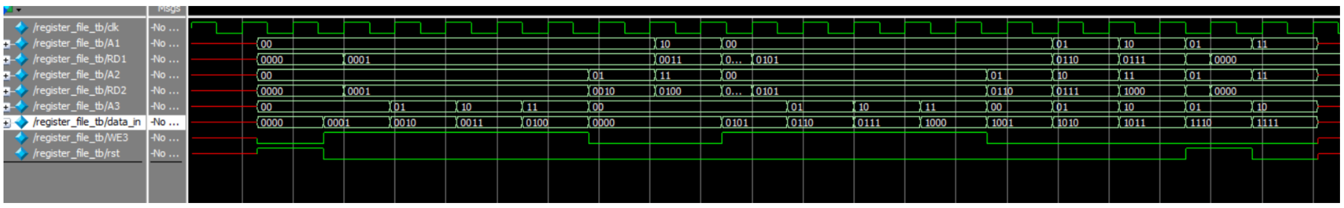
```verilog
module register_file_tb();

    reg clk;
    reg [3:0] data_in;
    reg [1:0] A1, A2, A3;
    reg WE3, rst;
    // outputs
    wire [3:0] RD1, RD2;

    reg[11:0] testvectors[15:0]; // array of testvectors
    reg [4:0] vectornum;

    register_file #4 DUT(clk, data_in, WE3, rst,    A1, A2, A3, RD1, RD2);
    // generate clock
    always // no sensitivity list, so it always executes
    begin
        clk <= 1;
        #5;
        clk <= 0;
        #5;
    end
    // load vectors
    initial
    begin
    $readmemb("C:/Users/Caner/Documents/GitHub/Course-Projects/EE446/LAB1/register_file_tb.tv",testvectors);
    vectornum = 0;
    end

    //apply test vectors
    always
    begin
     #13;
     {data_in, A1, A2, A3, WE3, rst} = testvectors[vectornum];
     vectornum = vectornum +1;
    end

endmodule
```

4. Vector table



```
0000_00_00_00_0_1
0001_00_00_00_1_0
0010_00_00_01_1_0
0011_00_00_10_1_0
0100_00_00_11_1_0
0000_00_01_00_0_0
0000_10_11_00_0_0
0101_00_00_00_1_0
0110_00_00_01_1_0
0111_00_00_10_1_0
1000_00_00_11_1_0
1001_00_01_00_0_0
1010_01_10_01_0_0
1011_10_11_10_0_0
1110_01_01_01_0_1
1111_11_11_10_0_0
```

5. Verification



1.4. Datapath Design for an Architecture

- How many control pins for the control signals does your architecture have?
  R1Src, ASrc, ALUControl[2:0], Acc_parallel, Acc_Lsrc[1..0], Q_right and the reset signals of the four registers.
  There are 9 control pins.

- How many different control signals does your architecture use to perform the desired tasks?
  R1Src, ASrc, ALUControl[2:0], Acc_parallel, Acc_Lsrc[1..0], Q_right
  6 control signals.

- Can you reduce the number of the control pins? Why not or how?
  We can ignore the reset pins of the registers.

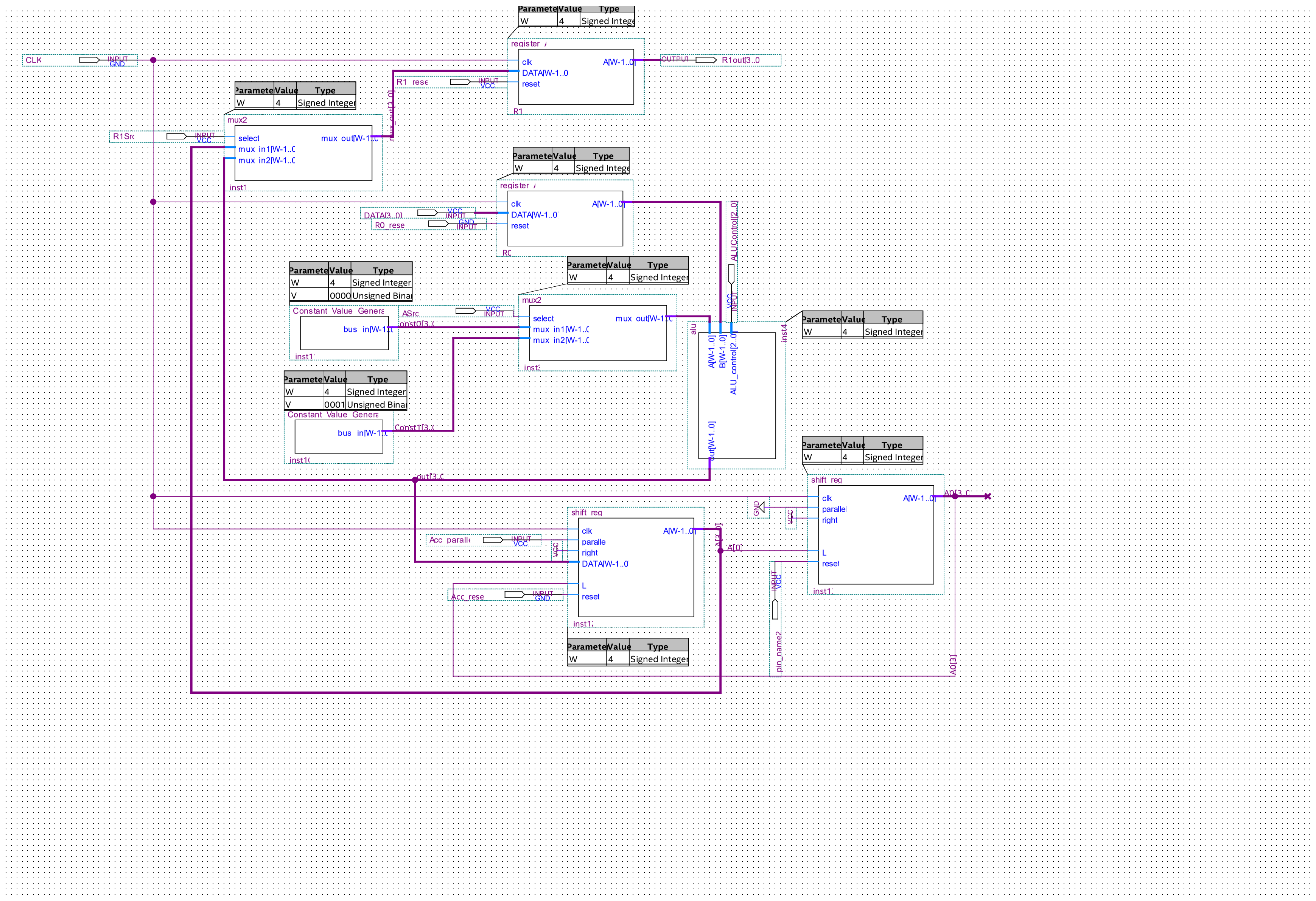
- Write down the sequence of the control signals for REVERSED LOAD operation. How many cycle does this operation take?

  1st Clock: ASrc = 0, ALUControl = 00
  R0 <- DATA
  2nd Clock: Acc_parallel = 1
   Acc <- DATA + 0x0
  3rd Clock: Acc_parallel = 0, Acc_Lsrc[1..0] = 00, Q_right=1
  Acc[3] <- Q[0]
  Q[3] <- Acc[0]
  4th Clock:
  Acc[3] <- Q[0]
  Q[3] <- Acc[0]

5th Clock:
Acc[3] <- Q[0]
Q[3] <- Acc[0]
6th Clock:
Acc[3] <- Q[0]
Q[3] <- Acc[0]
7th Clock: Q_right = 0, Acc_Lsrc[1..0] = 01
Acc[3] <- Q[3]
Q[0] <- 0
8th Clock:
Acc[3] <- Q[3]
Q[0] <- 0
9th Clock:
Acc[3] <- Q[3]
Q[0] <- 0

10th Clock:
Acc[3] <- Q[3]
Q[0] <- 0
11th Clock: Acc_parallel = 1, R1Src = 1
R1 <- Acc

11 Clock cycles in total.


CLK
INPUT GND
R1Src
INPUT VCC
mux2
select
mux in1[W-1..0]
mux in2[W-1..0]
mux out[W-1..0]
inst1
mux_out[3..0]
register
clk
DATA[W-1..0]
reset
A[W-1..0]
R1
R1 reset
OUTPUT
R1out[3..0]
DATA[3..0]
R0_reset
R0
Parameter Value Type
W 4 Signed Integer
V 0000 Unsigned Binary
V 0001 Unsigned Binary
Constant Value Generator
bus in[W-1..0]
inst1
inst10
ASrc
const0[3..0]
const1[3..0]
alu
A[W-1..0]
B[W-1..0]
ALU_control[2..0]
out[W-1..0]
ALUControl[2..0]
inst4
out[3..0]
shift reg
clk
parallel
right
L
reset
Acc parallel
Acc reset
A[3..0]
A[0]
A0[3]
GND
VCC
pin_name2
inst11
inst12