Digital System Design Applications
PROJECT - 1

Barrel Shifter with Logical/Arithmetic Right Shift and Rotate

EHB436E CRN: 10345

Group:1
Hatice Nur Andı

Introduction

Our project centered around the development of a shifter/rotator designed to operate on 32-bit data. In formulating our approach, I initially gleaned insights from the provided source, setting a course for a systematic and modular progression. Adopting a modular strategy has proven advantageous, aiding not only in the identification of potential errors but also in facilitating the project's scalability.

The modules I crafted are designed to handle operations on varying bit sizes, ranging from 1 bit to 16 bits. The data undergoes a sequential process, starting with the 16-bit shifter/rotator, followed successively by the 8-bit, 4-bit, 2-bit, and ultimately the 1-bit modules. The decision of whether these modules will execute a shift operation is contingent upon the 'op' input, intricately linked to the specified bit of 'b'. Notably, for rotate and sra operations, the 6th and 5th bits of 'b' play pivotal roles, respectively.

Moving forward, I will introduce each of these modular components individually, outlining their unique functionalities. Subsequently, I will showcase the top module where these individual modules are integrated seamlessly. In conclusion, the results of the integrated system will be presented, followed by a comprehensive analysis of its performance and efficiency.

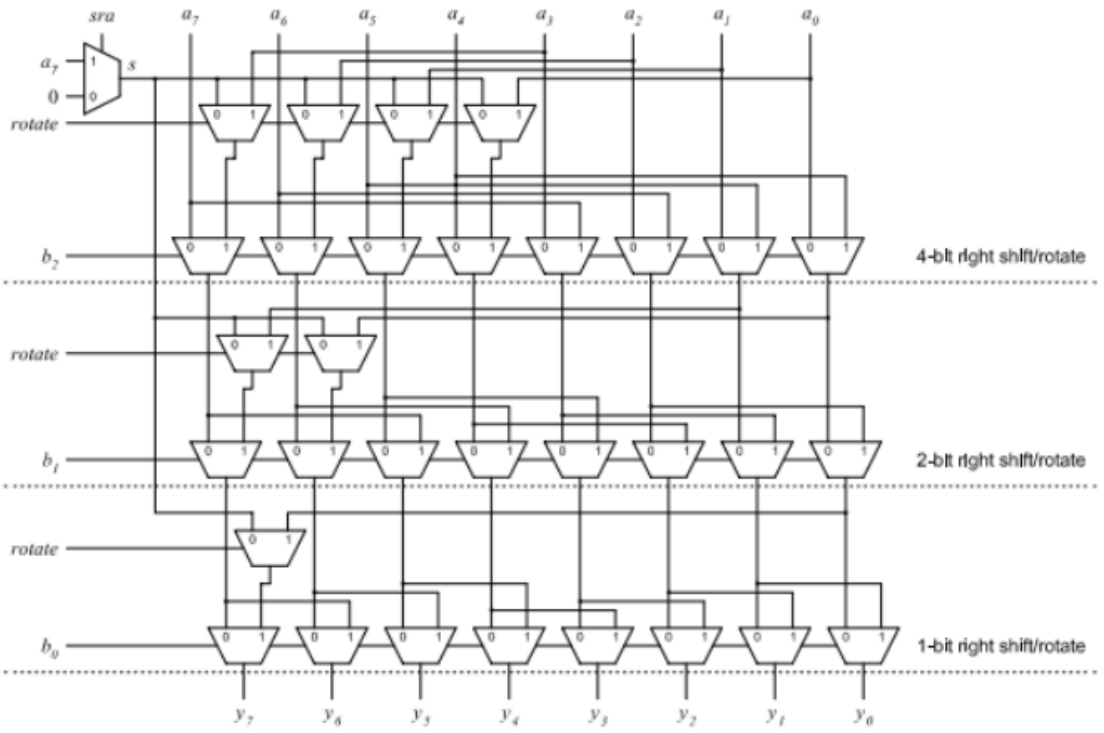


Figure 3. 8-bit mux-based right shifter/rotator.

op1bit

verilog code:

```
`timescale 1ns / 1ps
module op1bit(
    input sra,
    input rotate,
    input [31:0]a,
    input op,
    output [31:0]out1

);
wire rotate0;

genvar m;
generate
    mux2to1 m1(.D0(sra), .D1(a[0]), .S(rotate), .O(rotate0));
    mux2to1 m2(.D0(a[31]), .D1(rotate0), .S(op), .O(out1[31]));

    for (m=0;m<31;m=m+1)begin
    mux2to1 m2(.D0(a[m]), .D1(a[m+1]), .S(op), .O(out1[m]));
   end
endgenerate
endmodule
```

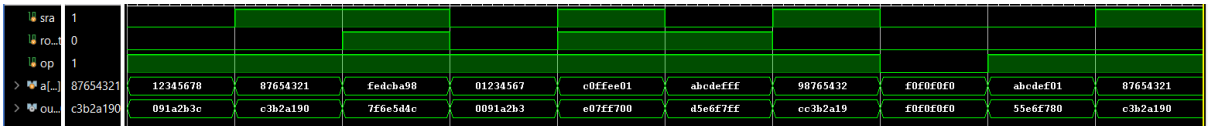
testbench code:

```
`timescale 1ns / 1ps

module tb_op1bit;

reg sra, rotate, op;
reg [31:0] a;
wire [31:0] out1;

op1bit uut(
    .sra(sra),
    .rotate(rotate),
    .a(a),
    .op(op),
    .out1(out1)
);

initial begin
```

```
    sra = 0; rotate = 0; op = 1; a = 32'h12345678;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 0; op = 1; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 1; op = 1; a = 32'hFEDCBA98;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 0; rotate = 0; op = 1; a = 32'h01234567;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 1; op = 1; a = 32'hC0FFEE01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 0; rotate = 1; op = 1; a = 32'hABCDEFFF;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 0; op = 1; a = 32'h98765432;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 0; rotate = 0; op = 0; a = 32'hF0F0F0F0;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 0; rotate = 0; op = 1; a = 32'hABCDEF01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 0; op = 1; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    $stop;
end
```

```
endmodule
```

simulation result:



op2bit

verilog code:

```
`timescale 1ns / 1ps
module op2bit(
    input sra,
    input rotate,
    input [31:0] a,
    input op,
    output [31:0] out1

    );

    wire [1:0] rotate1;
    genvar i,j,k;
    generate
        for(i = 0; i<2; i=i+1)begin
            mux2to1 m1(sra , a[i] , rotate, rotate1[i]);
        end

         for(j = 30; j<32; j=j+1)begin
            mux2to1 m2(a[j] , rotate1[j-30] , op, out1[j]);
        end

        for(k = 0; k<30; k=k+1)begin
            mux2to1 m3(a[k] , a[k+2] , op, out1[k]);
        end

     endgenerate
endmodule
```

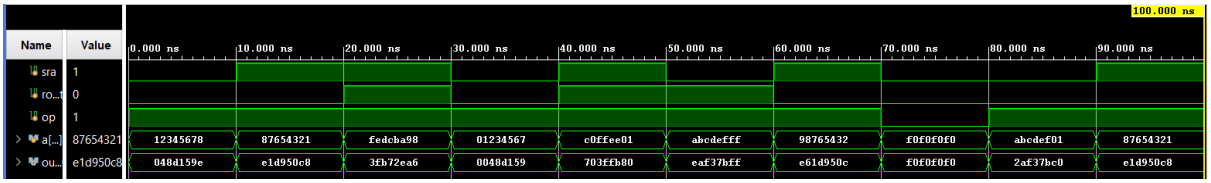
testbench code:

```
`timescale 1ns / 1ps

module tb_op2bit;
```

```
reg sra, rotate, op;
reg [31:0] a;
wire [31:0] out1;

op2bit uut(
    .sra(sra),
    .rotate(rotate),
    .a(a),
    .op(op),
    .out1(out1)
);

initial begin
    sra = 0; rotate = 0; op = 1; a = 32'h12345678;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 0; op = 1; a = 32'h87654321;
    #10; $display("sra=%b, rotaate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 1; op = 1; a = 32'hFEDCBA98;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 0; rotate = 0; op = 1; a = 32'h01234567;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 1; op = 1; a = 32'hC0FFEE01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 0; rotate = 1; op = 1; a = 32'hABCDEFFF;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 0; op = 1; a = 32'h98765432;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 0; rotate = 0; op = 0; a = 32'hF0F0F0F0;
```

```
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 0; rotate = 0; op = 1; a = 32'hABCDEF01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    sra = 1; rotate = 0; op = 1; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out1=%h", sra,
rotate, op, a, out1);

    $stop;
end

endmodule
```

simulation result:



op4bit

verilog code:

```
`timescale 1ns / 1ps

module op4bit(
    input sra,
    input rotate,
    input [31:0] a,
    input op,
    output [31:0] out2

    );

    wire [3:0] rotate2;
//4. katman
    genvar i,j,k;
    generate
```

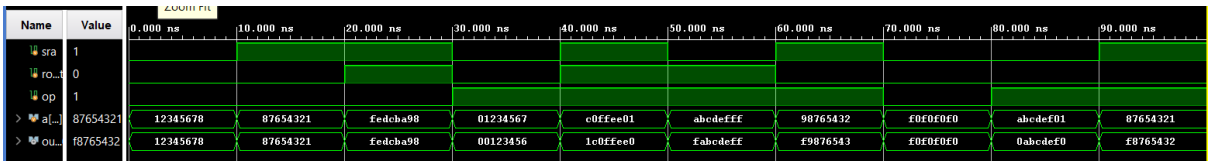
```
        for(i = 0; i<4; i=i+1)begin
            mux2to1 m1(sra , a[i] , rotate, rotate2[i]);
        end

        for(j = 28; j<32; j=j+1)begin
            mux2to1 m1(a[j] , rotate2[j-28] , op, out2[j]);
        end

        for(k = 0; k<28; k=k+1)begin
            mux2to1 m1(a[k] , a[k+4] , op, out2[k]);
        end

    endgenerate



endmodule
```

testbench code:

```
`timescale 1ns / 1ps

module tb_op4bit;

reg sra, rotate, op;
reg [31:0] a;
wire [31:0] out2;

op4bit uut(
    .sra(sra),
    .rotate(rotate),
    .a(a),
    .op(op),
    .out2(out2)
);

initial begin
    sra = 0; rotate = 0; op = 0; a = 32'h12345678;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    sra = 1; rotate = 0; op = 0; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);
```

```
    sra = 1; rotate = 1; op = 0; a = 32'hFEDCBA98;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    sra = 0; rotate = 0; op = 1; a = 32'h01234567;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    sra = 1; rotate = 1; op = 1; a = 32'hC0FFEE01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    sra = 0; rotate = 1; op = 1; a = 32'hABCDEFFF;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    sra = 1; rotate = 0; op = 1; a = 32'h98765432;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    sra = 0; rotate = 0; op = 0; a = 32'hF0F0F0F0;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    sra = 0; rotate = 0; op = 1; a = 32'hABCDEF01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    sra = 1; rotate = 0; op = 1; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out2=%h", sra,
rotate, op, a, out2);

    $stop;
end

endmodule
```

simulation result:



op8bit

verilog code:

```
`timescale 1ns / 1ps


module op8bit(
    input sra,
    input rotate,
    input [31:0]a,
    input op,
    output [31:0]out8

);
wire [7:0] rotate3;
wire [31:0] out4;
genvar i,m;
generate
    for(i=0;i<8;i=i+1)begin
    mux2to1 m1(.D0(sra), .D1(a[i]), .S(rotate), .O(rotate3[i]));
    mux2to1 m2(.D0(a[i+24]), .D1(rotate3[i]), .S(op), .O(out8[i+24]));
    end

    for (m=0;m<24;m=m+1)begin
    mux2to1 m2(.D0(a[m]), .D1(a[m+8]), .S(op), .O(out8[m]));
   end
endgenerate
endmodule
```

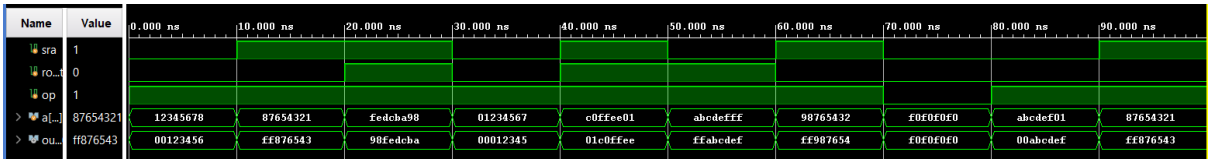
testbench code:

```
`timescale 1ns / 1ps

module tb_op8bit;

reg sra, rotate, op;
reg [31:0] a;
wire [31:0] out8;

op8bit uut(
    .sra(sra),
    .rotate(rotate),
    .a(a),
    .op(op),
```

```
    .out8(out8)
);

initial begin
    sra = 0; rotate = 0; op = 1; a = 32'h12345678;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 1; rotate = 0; op = 1; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 1; rotate = 1; op = 1; a = 32'hFEDCBA98;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 0; rotate = 0; op = 1; a = 32'h01234567;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 1; rotate = 1; op = 1; a = 32'hC0FFEE01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 0; rotate = 1; op = 1; a = 32'hABCDEFFF;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 1; rotate = 0; op = 1; a = 32'h98765432;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 0; rotate = 0; op = 0; a = 32'hF0F0F0F0;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 0; rotate = 0; op = 1; a = 32'hABCDEF01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);

    sra = 1; rotate = 0; op = 1; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out8=%h", sra,
rotate, op, a, out8);
```

```
    $stop;
end

endmodule
```

simulation result:



op16bit:

verilog code:

```
module op16bit(
    input sra,
    input rotate,
    input [31:0] a,
    input op,
    output [31:0] out4
    );

    wire [15:0] rotate4;
//4. katman
    genvar i;
    generate
        for(i = 0; i<16; i=i+1)begin
            mux2to1 m1(sra , a[i] , rotate, rotate4[i]);
            mux2to1 m2 (a[i+16], rotate4[i], op, out4[i+16]);
            mux2to1 m3 (a[i], a[i+16], op, out4[i]);
        end
    endgenerate

    endmodule
```

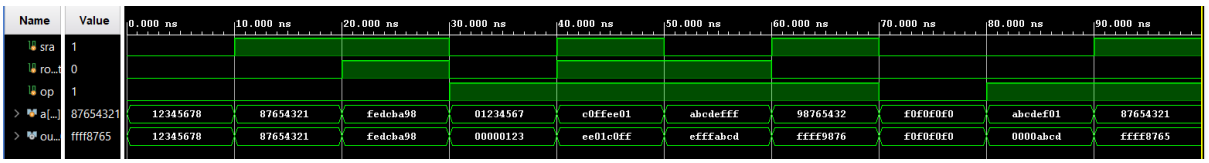
testbenvh code:

```
`timescale 1ns / 1ps

module tb_op16bit;

reg sra, rotate, op;
```

```
reg [31:0] a;
wire [31:0] out4;

op16bit uut(
    .sra(sra),
    .rotate(rotate),
    .a(a),
    .op(op),
    .out4(out4)
);

initial begin
    sra = 0; rotate = 0; op = 0; a = 32'h12345678;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    sra = 1; rotate = 0; op = 0; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    sra = 1; rotate = 1; op = 0; a = 32'hFEDCBA98;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    sra = 0; rotate = 0; op = 1; a = 32'h01234567;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    sra = 1; rotate = 1; op = 1; a = 32'hC0FFEE01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);


    sra = 0; rotate = 1; op = 1; a = 32'hABCDEFFF;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    sra = 1; rotate = 0; op = 1; a = 32'h98765432;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    sra = 0; rotate = 0; op = 0; a = 32'hF0F0F0F0;
```

```
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    sra = 0; rotate = 0; op = 1; a = 32'hABCDEF01;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    sra = 1; rotate = 0; op = 1; a = 32'h87654321;
    #10; $display("sra=%b, rotate=%b, op=%b, a=%h, out4=%h", sra,
rotate, op, a, out4);

    $stop;
end

endmodule
```

simulation result:



shifter/rotator

After combining these modules, we created the 'top' module, interconnecting the necessary components. For the result bit, all bits underwent a logical OR operation and were complemented, specifically for the carry. The value for the SRA operation was selected using a MUX, as shown in Figure 3.

verilog code:

```
(* DONT_TOUCH = "TRUE" *)
module shifter_rotator(
      input [6:0] B,
      input [31:0] A,
      output carry,
      output [31:0] C
);
//B[5] = a or 1
```

```
//B[6] = rotate
      wire s;
      wire [31:0] in8, in4,in2,in1;
      mux_2 muxxxx(.D1(A[31]), .D0(1'b0), .S(B[5]), .O(s));
      op16bit op1 (
            .a(A),
            .out16(in8),
            .sra(s),
            .rotate(B[6]),
            .op(B[4])
      );
      op8bit op2 (
            .a(in8),
            .out8(in4),
            .sra(s),
            .rotate(B[6]),
            .op(B[3])
      );
      op4bit op3 (
            .a(in4),
            .out4(in2),
            .sra(s),
            .rotate(B[6]),
            .op(B[2])
      );
      op2bit op4 (
            .a(in2),
            .out2(in1),
            .sra(s),
            .rotate(B[6]),
            .op(B[1])
      );
      op1bit op5 (
            .a(in1),
            .out1(C),
            .sra(s),
            .rotate(B[6]),
            .op(B[0])
      );
      wire [31:0] cw;
      genvar i;
      generate
            for(i=0;i<32;i=i+1)begin
```

```
            assign cw[i+1] = cw[i] | C[i];
            end
      endgenerate
      assign carry= ~cw[31];

endmodule
```

Simulation part
We generated 100 values each for 'a' and 'b' using Python code.

random values for a:

```
11111010000011010000000010111010 11000110101111001011110000000010 10111001101100101111010011001010
10000110000001101000100001110110 10110111100001110010110010001010 11010101000101010110001100101110
01100000101111100101101000101101 01000011101000010000100001100010 10101101101010100100111110010010
10011000000011011011100101101001 10100011101111111101110001100101 11001010000010011011010010111001
00010110100100111001100011101000 11100101000101110011100001000111 01001010000100110010010111110101
11110011000111110000100000011110 10110011010111011011101101110100 10000110111101100111101000011100
01010010010001011010011110011110 00111011111100010011111010011101 11111011110101110001010101010110
00101000001000111000010100011111 11110010100111110111000100000011 00111110000110001101000000001000
01100101111110111000101111101110 00100010110110110111001001001111 11000100100010101110110001000011
00110001000111001100111000101100 01100100110010101001101101111011 01010000001100000111001001000001
01101100011101010011110010001001 10101110001111110010110000010010 01110000011000011101100000001110
01111010001111000101000101111101 11100000011011010111110100010011 10011111001101111101101011001110
01110000101100010101110111001111 00000100001101111001010011001101 11011100100011011110111010011011
01001000101110100100010110110011 00100010100000101001110110100011 01110011011011101001000101101111
10001101011011100111001101011110 11110010001110001011111010010100 11110000001011101101001111101011
00010010011000001100001100000111 01101100001111111111011001111000 00000000000111111101011110010111
11100110100111101110110100000000 11001011100010101100110100101010 01110101000101010001000000010001
10101101101010011001001000101100 00101001100110100000111100010000 10101100100100011010001101011011
00010101100001000101111100101001 00101000101100000101001110010001 00000000010011111010001110111101
10010110111110011000001001011001 01110100011100100100010101100001 01101001011000011011110000000110
01111111101011101111000011001101 01000000100010011100000011110101 11001110001110010011010110011111
01000111010100111000011110100111 00000101111111111101010100010010 11100101100101010100010101000111
00000110110100111001011100011101 00101001101101101100000100111110 00110111000110011101100110101000
11110001001010100100111111001111 00100001101111011010111010100111 10000100001010000011000001001000
11100110100111110010100111010001 11100011001001101100010100101100 10101111100011010010110101110100
11111001100100101010100100111101 01101100101010001101100010000101 00110111000010001011101111111011
10101011111111110000000110011010 10011011100101111011111111011100 01000101110010001110100011011110
00011110101011000110110000000110 01011101110101011100011100100110 10101101010001110011011110111111
10101000000101001010111001100111 00000101010000001101101001010011 00100001010100001011111001011101
01010011000111000110111111001011 10101011100110001101101110100100 11100011001000101011110001001011
                                  00001010001101000000000101100111 00011010100011000100000100000101
```

```
11011110100100111101101010100011
10110101001011010000010111010001
10110000100000100110110100111001
01000010001001101000011010110111
11000000001011110000010000010100
11110110000010000110000101111100
00001111101000000000110001110010
```

random values for b:

```
0111101
1101001
1111000
1100101
1101101
0001110
1101101
0100001
1111110
1100101
0010000
1110000
0010011
1001000
0011010
1111110
0001001
0111011
1011011
0111011
0000011
0011011
0101011
0010010
0001101
1011010
0110111
1001011
1010000
0001101
```

```
0000001
1101110
1100010
0111101
0100011
0010000
0011011
1100111
1110110
0100001
1000001
0010011
1110010
1000010
0001101
0001010
0001011
0111111
1100000
1010001
0001001
0001001
1011010
0011101
1000101
0100101
1110001
1001010
0100000
0001001
0011110
```

```
0010100
1011111
0111100
1101101
1001011
0001101
1011110
0101000
0010101
1010001
0100110
1111001
1110010
0111010
0001000
1000101
1101111
0010111
0001100
0011110
0111111
0110100
0001001
1001000
1000101
0111110
0111010
1010010
1011010
1011101
1101010
```

```
0111110
1101010
1001010
0011111
1000000
```

testbench code:

```
module tb_shifer_rotator;
reg [31:0] a1;
reg [6:0] b1;
wire [31:0] y1 ;
reg [31:0] reals;
reg [31:0] i,j; // Reg tipinde sayacı tanımla
reg [31:0] a [0:99];
reg [31:0] r [0:99];
reg [6:0] b [0:99];
reg [31:0] y [0:99];
// Instantiate the shifter_rotator module 100 times for different test
cases
genvar k;
generate
for (k = 0; k < 100; k = k + 1) begin : gen_block
Top_Module uut (
.A(a1),
.B(b1),
.C(y1),
.carry(carry)
```

```
);
end
endgenerate
// Initial block to read input from files
initial begin
$readmemb("C:/sstu_project1/random_values.txt", a);
$readmemb("C:/sstu_project1/random_values_7.txt", b);
$readmemb("C:/sstu_project1/real_results.txt", r);
// Run simulations for 100 test cases
#10;
for (i = 0; i < 100; i = i + 1) begin
a1 = a[i];
b1 = b[i];
reals = r[i];
#10; // Add any necessary delay between test cases
y[i] = y1;
$writememb("C:/sstu_project1_1/results.txt", y);
// Display inputs in binary and decimal
$write("A=\"bin=%b,dec=%0d\"; B=\"bin=%b,dec=%0d\"; ", a1, a1, b1, b1);
// Display expected result in binary and decimal
$write("C=\"bin=%b,dec=%0d\"; ", reals, reals);
// Display circuit result in binary and decimal
$write("C_circ=\"bin=%b,dec=%0d\"; ", y1, y1);
// Compare expected result with circuit result
$write("status=%s\n", (y1 == reals) ? "TRUE" : "FALSE");
end
// End simulation
$finish;
end
endmodule
```

daha sonra vivado üzerinde sonuçları karşılaştırmak için bir kod daha yazdık

```
module realvalue (
    input [31:0] data,
    input [6:0] b,
    output reg [31:0] shifted
);
    wire [4:0] shift_amount;
    wire [1:0] select;
    assign select = b[6:5];
    assign shift_amount = b[4:0];
    always@* begin
```

```
        casex (select)
        // Logical right shift
        2'b00 : assign shifted = data >> shift_amount;
        // Arithmetic right shift
        2'b01 : assign shifted = (data[31] == 0) ? (data >>
shift_amount) :
        {32{data[31]}};
        // Rotate right
        2'b1x :assign shifted = (data >> shift_amount) | (data << (32 -
        shift_amount));
    endcase
    end
endmodule
```

simulation results:

TCL Console:
A="bin=11110100000110100000000101110101,dec=4095345013";
B="bin=0111101,dec=61"; C="bin=11111111111111111111111111111111,dec=4294967295";
C_circ="bin=11111111111111111111111111111111,dec=4294967295"; status= TRUE
A="bin=10000110000001101000100001110110,dec=2248575094";
B="bin=1101001,dec=105";
C="bin=00111011010000110000001101000100,dec=994247492";
C_circ="bin=00111011010000110000001101000100,dec=994247492"; status= TRUE
A="bin=01100000101111100101101000101101,dec=1623087661";
B="bin=1111000,dec=120";
C="bin=10111110010110100010110101100000,dec=3193580896";
C_circ="bin=10111110010110100010110101100000,dec=3193580896"; status= TRUE
A="bin=10011000000011011011100101101001,dec=2551036265";
B="bin=1100101,dec=101";
C="bin=01001100110000000110110111001011,dec=1287679435";
C_circ="bin=01001100110000000110110111001011,dec=1287679435"; status= TRUE
A="bin=00010110100100111001100011101000,dec=378771688";
B="bin=1101101,dec=109";
C="bin=11000111010000001011010010011100,dec=3342906524";
C_circ="bin=11000111010000001011010010011100,dec=3342906524"; status= TRUE
A="bin=11110011000111110000100000011110,dec=4078897182"; B="bin=0001110,dec=14";
C="bin=00000000000000111100110001111100,dec=248956";
C_circ="bin=00000000000000111100110001111100,dec=248956"; status= TRUE
A="bin=01010010010001011010011110011110,dec=1380296606";

B="bin=1101101,dec=109";
C="bin=00111100111100101001001000101101,dec=1022530093";
C_circ="bin=00111100111100101001001000101101,dec=1022530093"; status= TRUE
A="bin=00101000001000111000010100011111,dec=673416479"; B="bin=0100001,dec=33";
C="bin=000101000001000111000001010001111,dec=336708239";
C_circ="bin=000101000001000111000001010001111,dec=336708239"; status= TRUE
A="bin=01100101111110111000101111101110,dec=1710984174"; B="bin=1111110,dec=126";
C="bin=10010111111011100010111110111001,dec=2548969401";
C_circ="bin=10010111111011100010111110111001,dec=2548969401"; status= TRUE
A="bin=00110001000111001100111000101100,dec=823971372";
B="bin=1100101,dec=101";
C="bin=01100001100010001110011001110001,dec=1636361841";
C_circ="bin=01100001100010001110011001110001,dec=1636361841"; status= TRUE
A="bin=01101100011101010011110010001001,dec=1819622537";
B="bin=0010000,dec=16"; C="bin=00000000000000000110110001110101,dec=27765";
C_circ="bin=00000000000000000110110001110101,dec=27765"; status= TRUE
A="bin=01111010001111000101000101111101,dec=2050773373";
B="bin=1110000,dec=112";
C="bin=01010001011111010111101000111100,dec=1367177788";
C_circ="bin=01010001011111010111101000111100,dec=1367177788"; status= TRUE
A="bin=01110000101100010101101111001111,dec=1890671567";
B="bin=0010011,dec=19"; C="bin=00000000000000000000111000010110,dec=3606";
C_circ="bin=00000000000000000000111000010110,dec=3606"; status= TRUE
A="bin=01001000101110100100010110110011,dec=1220167091";
B="bin=1001000,dec=72";
C="bin=10110011010010001011101001000101,dec=3007887941";
C_circ="bin=10110011010010001011101001000101,dec=3007887941"; status= TRUE
A="bin=10001101011011100111001101011110,dec=2372825950";
B="bin=0011010,dec=26"; C="bin=00000000000000000000000000100011,dec=35";
C_circ="bin=00000000000000000000000000100011,dec=35"; status= TRUE
A="bin=00010010011000001100001100000111,dec=308331271";
B="bin=1111110,dec=126";
C="bin=01001001100000110000110000011100,dec=1233325084";
C_circ="bin=01001001100000110000110000011100,dec=1233325084"; status= TRUE
A="bin=11100110100111110111011010000000,dec=3869210240"; B="bin=0001001,dec=9";
C="bin=00000000111001101001111101110111,dec=7557051";
C_circ="bin=00000000111001101001111101110111,dec=7557051"; status= TRUE
A="bin=00010101100001000101111100101001,dec=360996649"; B="bin=1011011,dec=91";
C="bin=10110000100010111110010100100010,dec=2961958178";
C_circ="bin=10110000100010111110010100100010,dec=2961958178"; status= TRUE
A="bin=01111111101011101111000011001101,dec=2142171341"; B="bin=0000011,dec=3";
C="bin=00001111111101011101111000011001,dec=267771417";
C_circ="bin=00001111111101011101111000011001,dec=267771417"; status= TRUE
A="bin=01000111010100111000011110100111,dec=1196656551"; B="bin=0011011,dec=27";
C="bin=00000000000000000000000000001000,dec=8";
C_circ="bin=00000000000000000000000000001000,dec=8"; status= TRUE
A="bin=00000110110100110010111000110101,dec=114503221"; B="bin=0101011,dec=43";
C="bin=00000000000000001101101001100101,dec=55909";

C_circ="bin=00000000000000001101101001100101,dec=55909"; status= TRUE
A="bin=11110001001010100100111111001111,dec=4046081999";
B="bin=0010010,dec=18"; C="bin=00000000000000000011110001001010,dec=15434";
C_circ="bin=00000000000000000011110001001010,dec=15434"; status= TRUE
A="bin=11100110100111110010100111010001,dec=3869190609";
B="bin=0001101,dec=13"; C="bin=00000000000001110011010011111001,dec=472313";
C_circ="bin=00000000000001110011010011111001,dec=472313"; status= TRUE
A="bin=11110011001001010101001001111101,dec=4079309437";
B="bin=1011010,dec=90";
C="bin=11001001010101001001111101111100,dec=3377766268";
C_circ="bin=11001001010101001001111101111100,dec=3377766268"; status= TRUE
A="bin=00011110101011000110110000000110,dec=514616326"; B="bin=1001011,dec=75";
C="bin=10000000110000111101010110001101,dec=2160317837";
C_circ="bin=10000000110000111101010110001101,dec=2160317837"; status= TRUE
A="bin=10101000000101001010111001100111,dec=2819927655";
B="bin=1010000,dec=80";
C="bin=10101110011001111010100000010100,dec=2926028820";
C_circ="bin=10101110011001111010100000010100,dec=2926028820"; status= TRUE
A="bin=01010011000111000110111111001011,dec=1394372555";
B="bin=0001101,dec=13"; C="bin=00000000000000101001100011100011,dec=170211";
C_circ="bin=00000000000000101001100011100011,dec=170211"; status= TRUE
A="bin=00001110110000011001010100001101,dec=247567629"; B="bin=1000000,dec=64";
C="bin=00001110110000011001010100001101,dec=247567629";
C_circ="bin=00001110110000011001010100001101,dec=247567629"; status= TRUE
A="bin=11000110101111001011110000000010,dec=3334257666";
B="bin=1011100,dec=92";
C="bin=01101011110010111100000000101100,dec=1808515116";
C_circ="bin=01101011110010111100000000101100,dec=1808515116"; status= TRUE
A="bin=10110111100001110010110010001010,dec=3079089290";
B="bin=0001011,dec=11"; C="bin=00000000001011011111000011100101,dec=1503461";
C_circ="bin=00000000001011011111000011100101,dec=1503461"; status= TRUE
A="bin=01000011101000010000100001100010,dec=1134626914"; B="bin=0000001,dec=1";
C="bin=00100001110100001000010000110001,dec=567313457";
C_circ="bin=00100001110100001000010000110001,dec=567313457"; status= TRUE
A="bin=10100011101111111101110001100101,dec=2747260005";
B="bin=1101110,dec=110";
C="bin=01110001100101101000111011111111,dec=1905692415";
C_circ="bin=01110001100101101000111011111111,dec=1905692415"; status= TRUE
A="bin=11100101000101110011100001000111,dec=3843504199";
B="bin=1100010,dec=98";
C="bin=11111001010001011100111000010001,dec=4182101521";
C_circ="bin=11111001010001011100111000010001,dec=4182101521"; status= TRUE
A="bin=00111011111100010011111010011101,dec=1005665949"; B="bin=0100011,dec=35";
C="bin=00000111011111100010011111010011,dec=125708243";
C_circ="bin=00000111011111100010011111010011,dec=125708243"; status= TRUE
A="bin=11110010100111110111000100000011,dec=4070535427";
B="bin=0010000,dec=16"; C="bin=00000000000000001111001010011111,dec=62111";
C_circ="bin=00000000000000001111001010011111,dec=62111"; status= TRUE

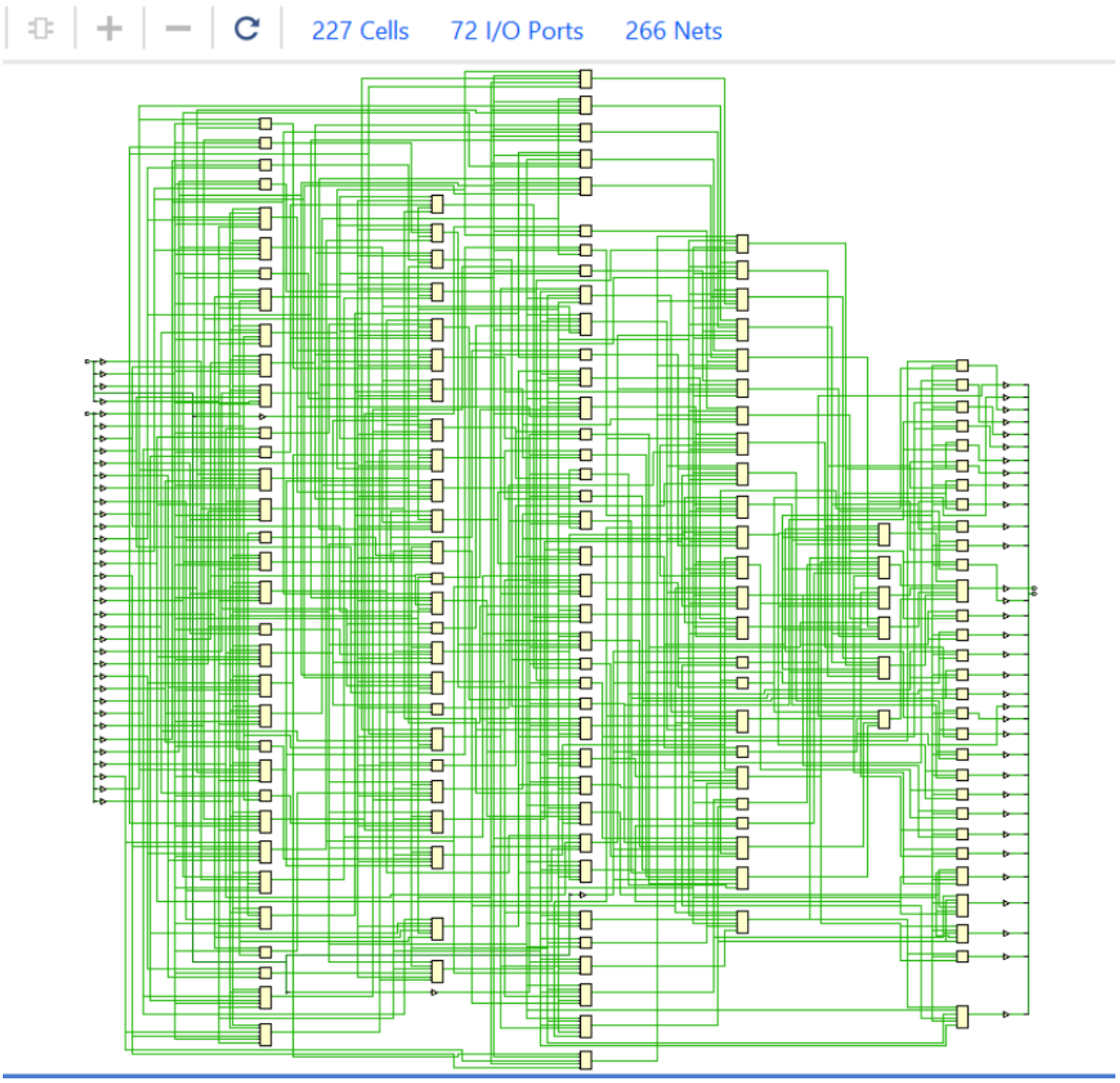
A="bin=00100010110110110111001001001111,dec=584806991"; B="bin=0011011,dec=27";
C="bin=00000000000000000000000000000100,dec=4";
C_circ="bin=00000000000000000000000000000100,dec=4"; status= TRUE
A="bin=01100100110010101001101101111011,dec=1690999675";
B="bin=1100111,dec=103";
C="bin=11110110110010011001010100110110,dec=4140406070";
C_circ="bin=11110110110010011001010100110110,dec=4140406070"; status= TRUE
A="bin=10101111000111111001011000010010,dec=2938082834";
B="bin=1110110,dec=118";
C="bin=01111110010110000100101010111100,dec=2119715516";
C_circ="bin=01111110010110000100101010111100,dec=2119715516"; status= TRUE
A="bin=00000100001101111001010011001101,dec=70751437"; B="bin=1000001,dec=65";
C="bin=10000100001101111001010011001110,dec=2182859366";
C_circ="bin=10000100001101111001010011001110,dec=2182859366"; status= TRUE
A="bin=00100010100001010011101110100011,dec=579156899"; B="bin=0010011,dec=19";
C="bin=00000000000000000000010001010000,dec=1104";
C_circ="bin=00000000000000000000010001010000,dec=1104"; status= TRUE
A="bin=11110010001110001011111010010100,dec=4063805076";
B="bin=1110010,dec=114";
C="bin=00101111101001010011110010001110,dec=799358094";
C_circ="bin=00101111101001010011110010001110,dec=799358094"; status= TRUE
A="bin=01101100001111111110110011111000,dec=1816128760"; B="bin=1000010,dec=66";
C="bin=00011011000011111111101100111110,dec=454032190";
C_circ="bin=00011011000011111111101100111110,dec=454032190"; status= TRUE
A="bin=11001011100010101100110100101010,dec=3414871338";
B="bin=0001101,dec=13"; C="bin=00000000000011001011110001010110,dec=416854";
C_circ="bin=00000000000011001011110001010110,dec=416854"; status= TRUE
A="bin=00101001100110100000111100010000,dec=697962256"; B="bin=0001010,dec=10";
C="bin=00000000000010100110011010000011,dec=681603";
C_circ="bin=00000000000010100110011010000011,dec=681603"; status= TRUE
A="bin=00101000101100000101001110010001,dec=682644369"; B="bin=0001011,dec=11";
C="bin=00000000000010100010110000001010,dec=333322";
C_circ="bin=00000000000010100010110000001010,dec=333322"; status= TRUE
A="bin=01110100011100100100010101100001,dec=1953645921";
B="bin=0111111,dec=63"; C="bin=00000000000000000000000000000000,dec=0";
C_circ="bin=00000000000000000000000000000000,dec=0"; status= TRUE
A="bin=01000000100010011100000011110101,dec=1082769653";
B="bin=1100000,dec=96";
C="bin=01000000100010011100000011110101,dec=1082769653";
C_circ="bin=01000000100010011100000011110101,dec=1082769653"; status= TRUE
A="bin=00000101111111111101010100010010,dec=100652306"; B="bin=1010001,dec=81";
C="bin=11101010100010010000001011111111,dec=3934847743";
C_circ="bin=11101010100010010000001011111111,dec=3934847743"; status= TRUE
A="bin=00101001101101101100000100111110,dec=699842878"; B="bin=0001001,dec=9";
C="bin=00000000001010011011011011100000,dec=1366880";
C_circ="bin=00000000001010011011011011100000,dec=1366880"; status= TRUE
A="bin=00100001101110110101110110100111,dec=565927335"; B="bin=0001001,dec=9";
C="bin=00000000001000011011101101011110,dec=1105326";

C_circ="bin=000000000001000011011101101011110,dec=1105326"; status= TRUE
A="bin=11100011001001101100010100101100,dec=3810968876";
B="bin=1011010,dec=90";
C="bin=11001001101100010100101100111000,dec=3383839544";
C_circ="bin=11001001101100010100101100111000,dec=3383839544"; status= TRUE
A="bin=01101100101010001101100010000101,dec=1823004805";
B="bin=0011101,dec=29"; C="bin=00000000000000000000000000000011,dec=3";
C_circ="bin=00000000000000000000000000000011,dec=3"; status= TRUE
A="bin=10011011100101111011111111011100,dec=2610413532"; B="bin=1000101,dec=69";
C="bin=11100100110111001011110111111110,dec=3839671806";
C_circ="bin=11100100110111001011110111111110,dec=3839671806"; status= TRUE
A="bin=01011101110101011100011100100110,dec=1574291238";
B="bin=0100101,dec=37"; C="bin=00000010111011101010111000111001,dec=49196601";
C_circ="bin=00000010111011101010111000111001,dec=49196601"; status= TRUE
A="bin=00001101010000001101101001010011,dec=222354003";
B="bin=1110001,dec=113";
C="bin=01101101001010011000011010100000,dec=1831437984";
C_circ="bin=01101101001010011000011010100000,dec=1831437984"; status= TRUE
A="bin=10101011100110001101101110100100,dec=2878921636";
B="bin=1001010,dec=74";
C="bin=11101001001010101110011000110110,dec=3911902774";
C_circ="bin=11101001001010101110011000110110,dec=3911902774"; status= TRUE
A="bin=00001010001101000000000101100111,dec=171180391"; B="bin=0100000,dec=32";
C="bin=00001010001101000000000101100111,dec=171180391";
C_circ="bin=00001010001101000000000101100111,dec=171180391"; status= TRUE
A="bin=10111001101100101111010011001010,dec=3115513034"; B="bin=0001001,dec=9";
C="bin=00000000101110011010010101111010,dec=6084986";
C_circ="bin=00000000101110011010010101111010,dec=6084986"; status= TRUE
A="bin=11010101000101010110001100101110,dec=3574948654";
B="bin=0011110,dec=30"; C="bin=00000000000000000000000000000011,dec=3";
C_circ="bin=00000000000000000000000000000011,dec=3"; status= TRUE
A="bin=10101101101010100100111110010010,dec=2913619858";
B="bin=0010100,dec=20"; C="bin=00000000000000000000101011011010,dec=2778";
C_circ="bin=00000000000000000000101011011010,dec=2778"; status= TRUE
A="bin=11001010000010011011010010111001,dec=3389633721";
B="bin=1011111,dec=95";
C="bin=10010100000100110110100101110011,dec=2484300147";
C_circ="bin=10010100000100110110100101110011,dec=2484300147"; status= TRUE
A="bin=01001010000100110010010111110101,dec=1242768885";
B="bin=0111100,dec=60"; C="bin=00000000000000000000000000000100,dec=4";
C_circ="bin=00000000000000000000000000000100,dec=4"; status= TRUE
A="bin=10000110111101100111101000011100,dec=2264300060";
B="bin=1101101,dec=109";
C="bin=11010000111001000011011110110011,dec=3504617395";
C_circ="bin=11010000111001000011011110110011,dec=3504617395"; status= TRUE
A="bin=11111011110101110001010101010110,dec=4225176918";
B="bin=1001011,dec=75";
C="bin=10101010110111110111101011100010,dec=2866772706";

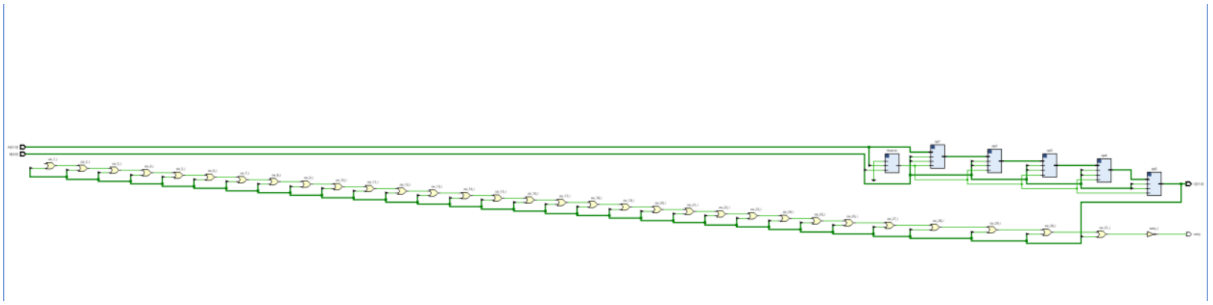
C_circ="bin=10101010110111110111101011100010,dec=2866772706"; status= TRUE
A="bin=00111110000110001101000000001000,dec=1041813512";
B="bin=0001101,dec=13"; C="bin=00000000000000011111000011000110,dec=127174";
C_circ="bin=00000000000000011111000011000110,dec=127174"; status= TRUE
A="bin=11000100100010101110110001000011,dec=3297438787";
B="bin=1011110,dec=94"; C="bin=00010010001010111011000100001111,dec=304853263";
C_circ="bin=0001001000101011101100010000 1111,dec=304853263"; status= TRUE
A="bin=01010000001100000111001001000001,dec=1345352257";
B="bin=0101000,dec=40"; C="bin=00000000010100000011000001110010,dec=5255282";
C_circ="bin=00000000010100000011000001110010,dec=5255282"; status= TRUE
A="bin=01110000011000011101100000001110,dec=1885460494";
B="bin=0010101,dec=21"; C="bin=00000000000000000000001110000011,dec=899";
C_circ="bin=00000000000000000000001110000011,dec=899"; status= TRUE
A="bin=10011111001101111101101011001110,dec=2671237838"; B="bin=1010001,dec=81";
C="bin=11101101011001110100111110011011,dec=3982970779";
C_circ="bin=11101101011001110100111110011011,dec=3982970779"; status= TRUE
A="bin=01110011011011101001000101101111,dec=1936626031";
B="bin=1111001,dec=121";
C="bin=10110111010010001011011110111001,dec=3074996153";
C_circ="bin=10110111010010001011011110111001,dec=3074996153"; status= TRUE
A="bin=11110000010111011010011111101011,dec=4029600747";
B="bin=1110010,dec=114";
C="bin=10110100111110101111110000001011,dec=3036347403";
C_circ="bin=10110100111110101111110000001011,dec=3036347403"; status= TRUE
A="bin=00000000001111111010111110010111,dec=2086807"; B="bin=0111010,dec=58";
C="bin=00000000000000000000000000000000,dec=0";
C_circ="bin=00000000000000000000000000000000,dec=0"; status= TRUE
A="bin=01110101000101010001000000010001,dec=1964314641";
B="bin=0001000,dec=8"; C="bin=00000000111010100010101000010000,dec=7673104";
C_circ="bin=0000000011101010001010100010000,dec=7673104"; status= TRUE
A="bin=10101100100100011010001101011011,dec=2895225691";
B="bin=1000101,dec=69";
C="bin=11011101011001001000110100011010,dec=3714354458";
C_circ="bin=11011101011001001000110100011010,dec=3714354458"; status= TRUE
A="bin=00000000100111111010001110111101,dec=5219261"; B="bin=1101111,dec=111";
C="bin=01000111011110100000000010011111,dec=1199177887";
C_circ="bin=01000111011110100000000010011111,dec=1199177887"; status= TRUE
A="bin=01101001011000011011110000000110,dec=1768012806";
B="bin=0010111,dec=23"; C="bin=00000000000000000000000011010010,dec=210";
C_circ="bin=00000000000000000000000011010010,dec=210"; status= TRUE
A="bin=11001110001110010011010110011111,dec=3459855775"; B="bin=0001100,dec=12";
C="bin=00000000000011001110001110010011,dec=844691";
C_circ="bin=00000000000011001110001110010011,dec=844691"; status= TRUE
A="bin=11100101100101010100010101000111,dec=3851765063";
B="bin=0011110,dec=30"; C="bin=00000000000000000000000000000011,dec=3";
C_circ="bin=00000000000000000000000000000011,dec=3"; status= TRUE
A="bin=00110111000110011101100110101000,dec=924441000"; B="bin=0111111,dec=63";
C="bin=00000000000000000000000000000000,dec=0";

C_circ="bin=00000000000000000000000000000000,dec=0"; status= TRUE
A="bin=10101111100011010010110101110100,dec=2945265012"; B="bin=0001001,dec=9";
C="bin=00000000101010111100011010010110,dec=5752470";
C_circ="bin=00000000101010111100011010010110,dec=5752470"; status= TRUE
A="bin=00110111000010001011110111111011,dec=923319803"; B="bin=1001000,dec=72";
C="bin=11111011001101110000100010111101,dec=4214687933";
C_circ="bin=11111011001101110000100010111101,dec=4214687933"; status= TRUE
A="bin=01000101110010001110100011011110,dec=1170794718";
B="bin=1000101,dec=69";
C="bin=11110010001011100100011101000110,dec=4063119174";
C_circ="bin=11110010001011100100011101000110,dec=4063119174"; status= TRUE
A="bin=00100001010100001011111001011101,dec=558939741"; B="bin=0111010,dec=58";
C="bin=00000000000000000000000000001000,dec=8";
C_circ="bin=00000000000000000000000000001000,dec=8"; status= TRUE
A="bin=11100011001000101011110001001011,dec=3810704459";
B="bin=1010010,dec=82";
C="bin=10101111000100101111100011001000,dec=2937256136";
C_circ="bin=10101111000100101111100011001000,dec=2937256136"; status= TRUE
A="bin=00011010100011000100000100000101,dec=445399301";
B="bin=1011010,dec=90";
C="bin=10100011000100000100000101000110,dec=2735751494";
C_circ="bin=10100011000100000100000101000110,dec=2735751494"; status= TRUE
A="bin=11011110100100111101101010100011,dec=3734231715";
B="bin=1011101,dec=93";
C="bin=11110100100111101101010100011110,dec=4104049950";
C_circ="bin=11110100100111101101010100011110,dec=4104049950"; status= TRUE
A="bin=10110101001011010000010111010001,dec=3039626705";
B="bin=1101010,dec=106";
C="bin=01110100011011010100101101000001,dec=1953319745";
C_circ="bin=01110100011011010100101101000001,dec=1953319745"; status= TRUE
A="bin=01000010001001101000011010110111,dec=1109821111";
B="bin=1101010,dec=106";
C="bin=10101101110100001000100110100001,dec=2916125089";
C_circ="bin=10101101110100001000100110100001,dec=2916125089"; status= TRUE
A="bin=11000000001011110000010000010100,dec=3224306708";
B="bin=1001010,dec=74"; C="bin=00000101001100000000101111000001,dec=87034817";
C_circ="bin=00000101001100000000101111000001,dec=87034817"; status= TRUE
A="bin=11110110000010000110000101111100,dec=4127744380"; B="bin=0011111,dec=31";
C="bin=00000000000000000000000000000001,dec=1";
C_circ="bin=00000000000000000000000000000001,dec=1"; status= TRUE xsim: Time (s):
cpu = 00:00:09 ; elapsed = 00:00:05 . Memory (MB): peak = 1620.414 ; gain = 132.227
INFO: [USF-XSim-96] XSim completed. Design snapshot 'TB_behav' loaded. INFO:
[USF-XSim-97] XSim simulation ran for 1000ns launch_simulation: Time (s): cpu = 00:00:11 ;
elapsed = 00:00:14 . Memory (MB): peak = 1620.414 ; gain = 132.637

Technology schematic:



RTL Schematic:

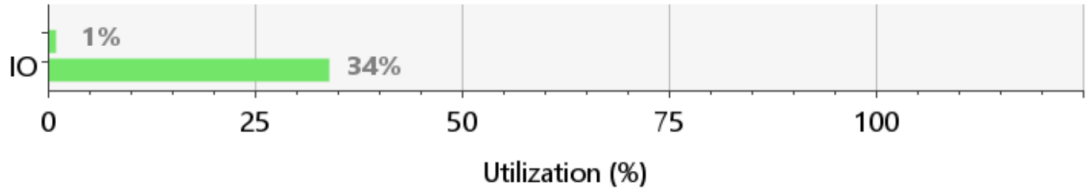


Utilization, power and Timing Reports:

112 LUTs were utilized. The maximum delay is associated with the B[5] input (16-bit shift rotate level mux enable input) to the carry output, registering at 18.354 ns. The critical path is defined by the B input to the carry output due to its prolonged delay. The maximum attainable clock frequency is 55.5 MHz, as the critical delay must allow for a complete clock cycle.

Summary

Resource	Utilization	Available	Utilization %
LUT	122	63400	0.19
IO	72	210	34.29



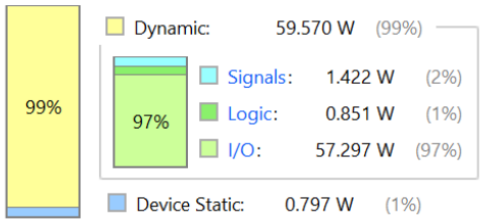
Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**60.367 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-215.4°C (-46.6 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power



From Port	To Port	Max Delay 1	Max Process Corner	Min Delay	Min Process Corner
B[5]	carry	18.354	SLOW	4.750	FAST
B[5]	C[17]	17.607	SLOW	5.203	FAST
B[5]	C[16]	17.376	SLOW	5.482	FAST
B[4]	carry	17.361	SLOW	4.697	FAST
B[5]	C[30]	17.289	SLOW	6.002	FAST
B[5]	C[5]	17.232	SLOW	5.680	FAST
B[6]	carry	17.201	SLOW	4.576	FAST
B[5]	C[6]	17.192	SLOW	5.522	FAST
B[5]	C[14]	17.134	SLOW	5.303	FAST
B[5]	C[26]	17.026	SLOW	5.209	FAST
B[4]	C[30]	17.023	SLOW	5.777	FAST
B[5]	C[19]	16.992	SLOW	5.211	FAST
B[5]	C[15]	16.950	SLOW	5.318	FAST
B[5]	C[18]	16.949	SLOW	4.793	FAST
B[5]	C[10]	16.840	SLOW	5.257	FAST
B[5]	C[20]	16.827	SLOW	5.053	FAST
B[5]	C[23]	16.809	SLOW	4.939	FAST
B[5]	C[21]	16.795	SLOW	5.110	FAST
B[4]	C[17]	16.768	SLOW	5.230	FAST
B[6]	C[30]	16.761	SLOW	5.644	FAST
B[4]	C[26]	16.749	SLOW	5.261	FAST
B[5]	C[12]	16.694	SLOW	5.213	FAST
A[9]	C[30]	16.647	SLOW	6.373	FAST
B[5]	C[9]	16.603	SLOW	5.028	FAST