高等計算機演算法

Advanced Computer Algorithms

Term Projects

2024.05.29

學號:M11215075

姓名:胡劭

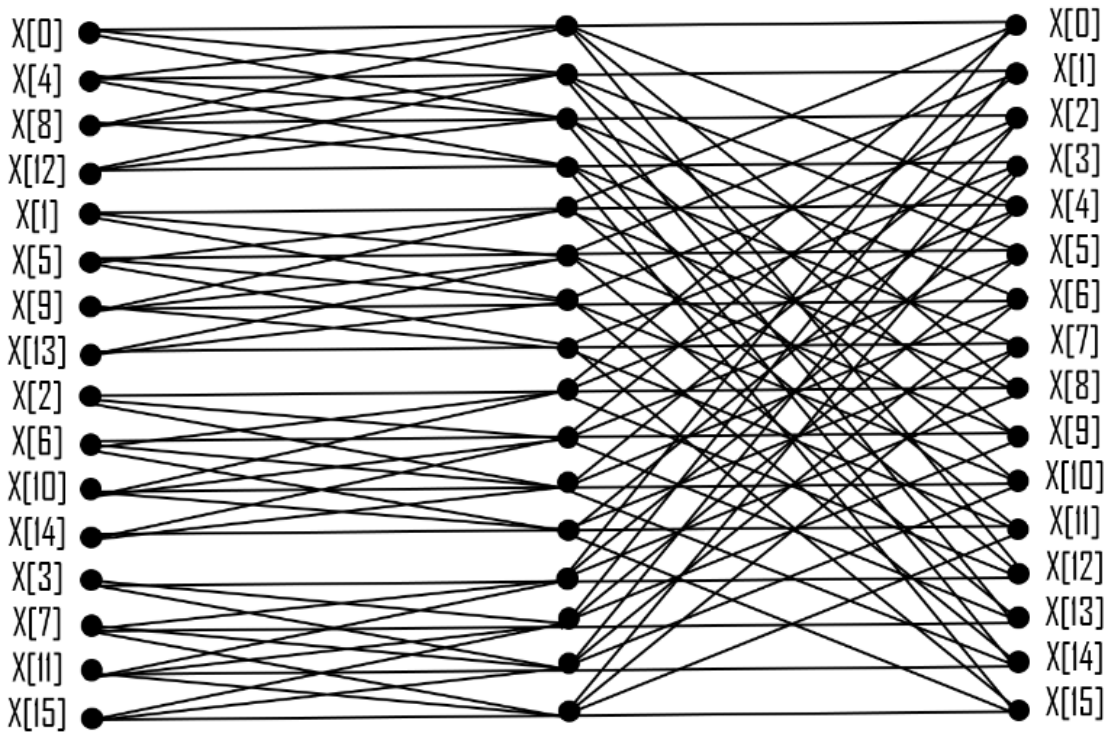
1. Design:

我選擇的是 FFT network，以下是我的設計以及說明:

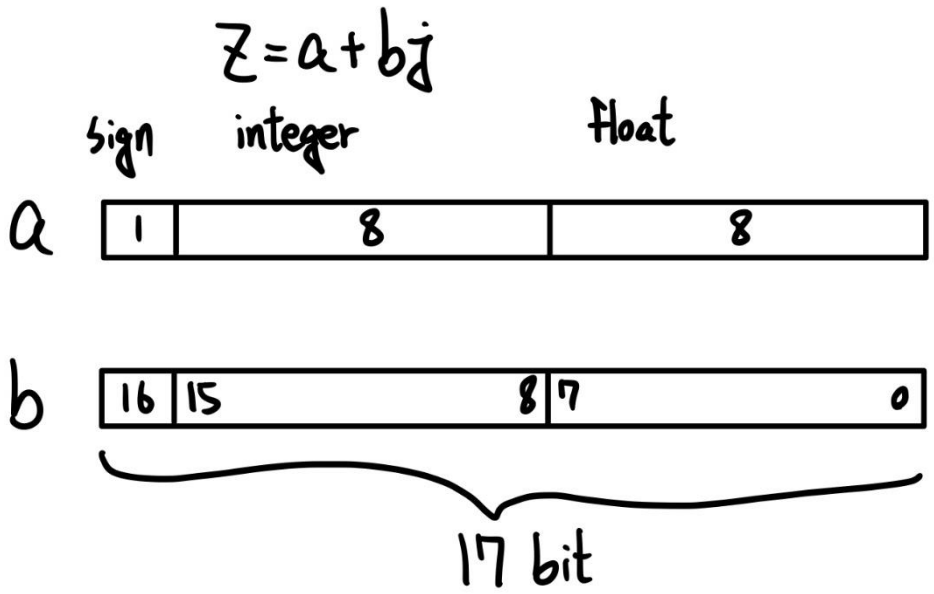
- **Algorithm**

我採用的是 radix-4

1.16-point radix-4 FFT schematic diagram



2. Data represent

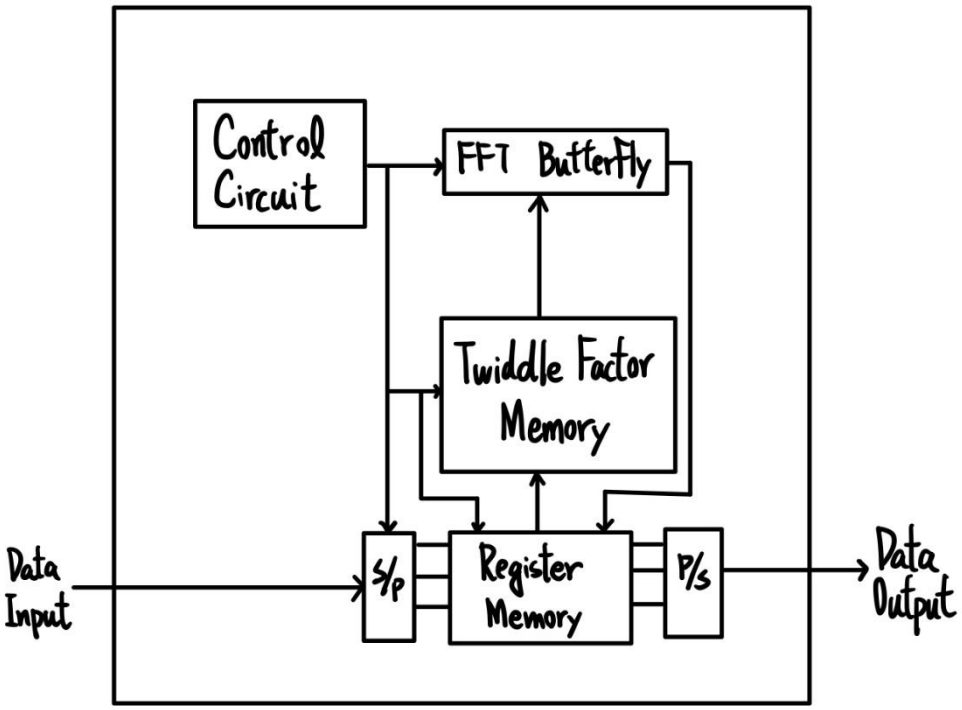


- **Architecture**

  1.Block diagram

  作法:主要可以切為主要三個區塊

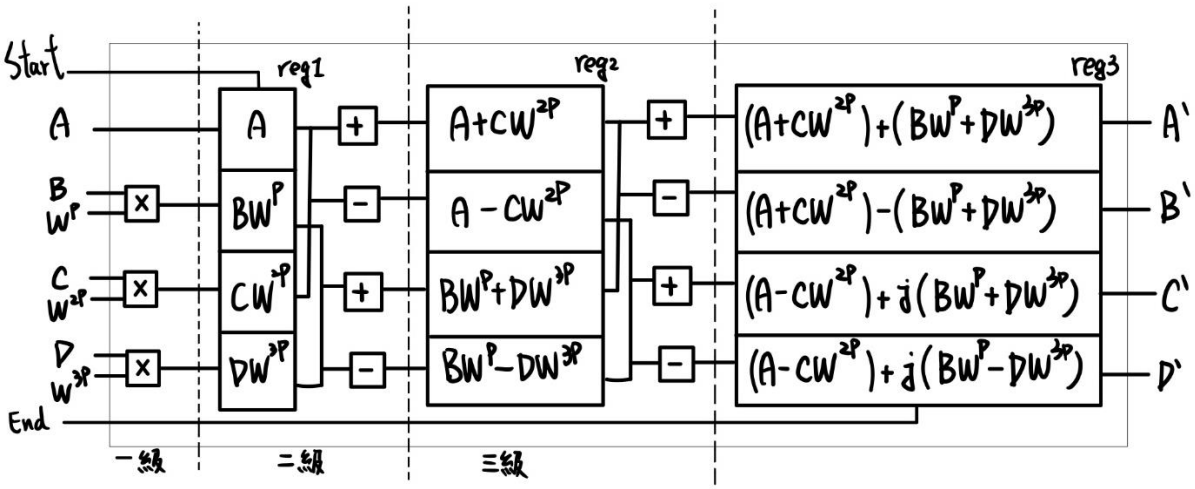
2.Processor architecture



- **Butterfly**

1.完整設計圖

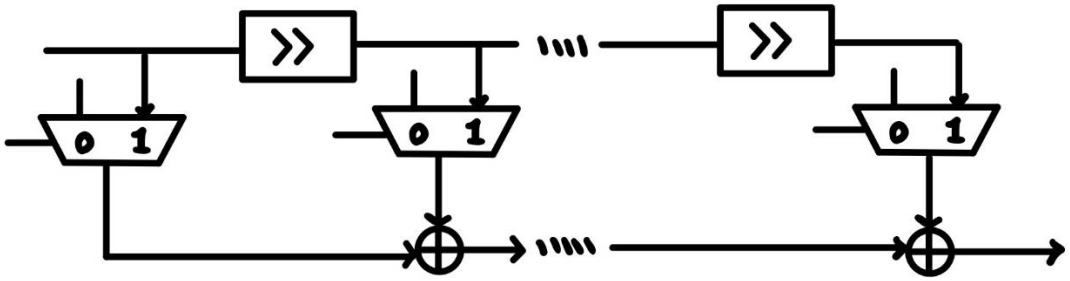
作法:用三級 pipeline 做，透過 pipeline 跟運算因子在每個 stage 內做運算



2.加減法器直接用運算，所以只要做乘法的部分

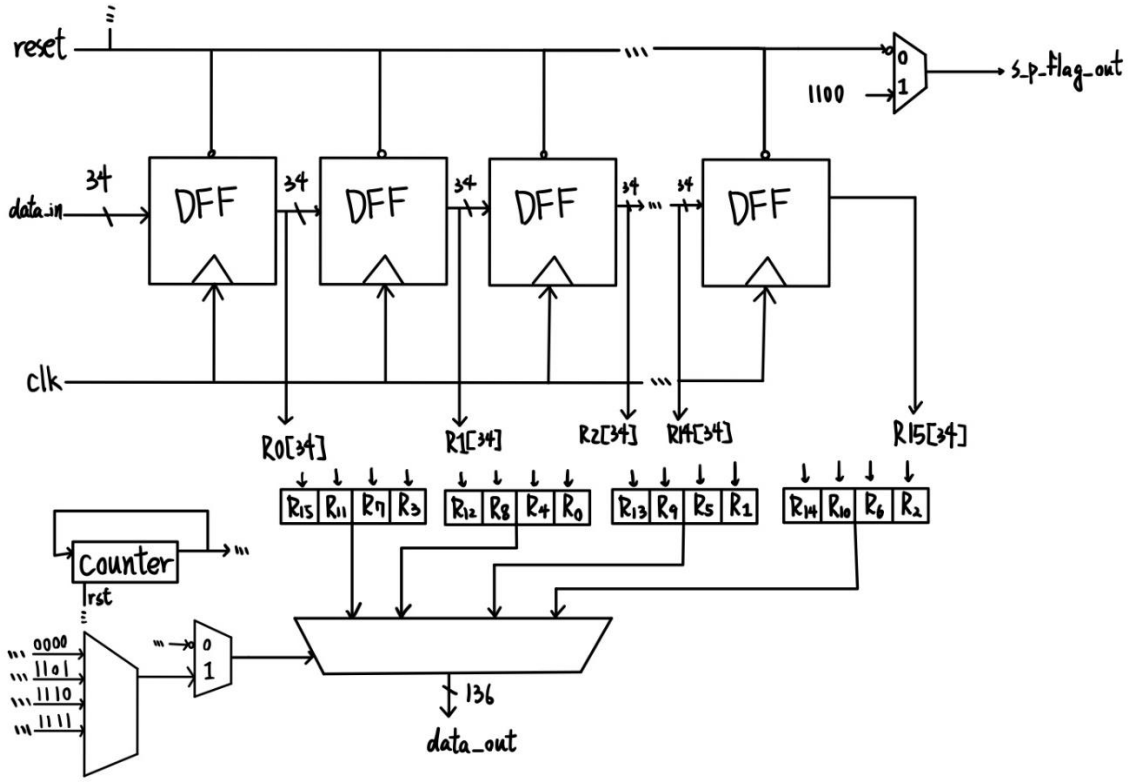
3.乘法器

利用位移方式計算乘法



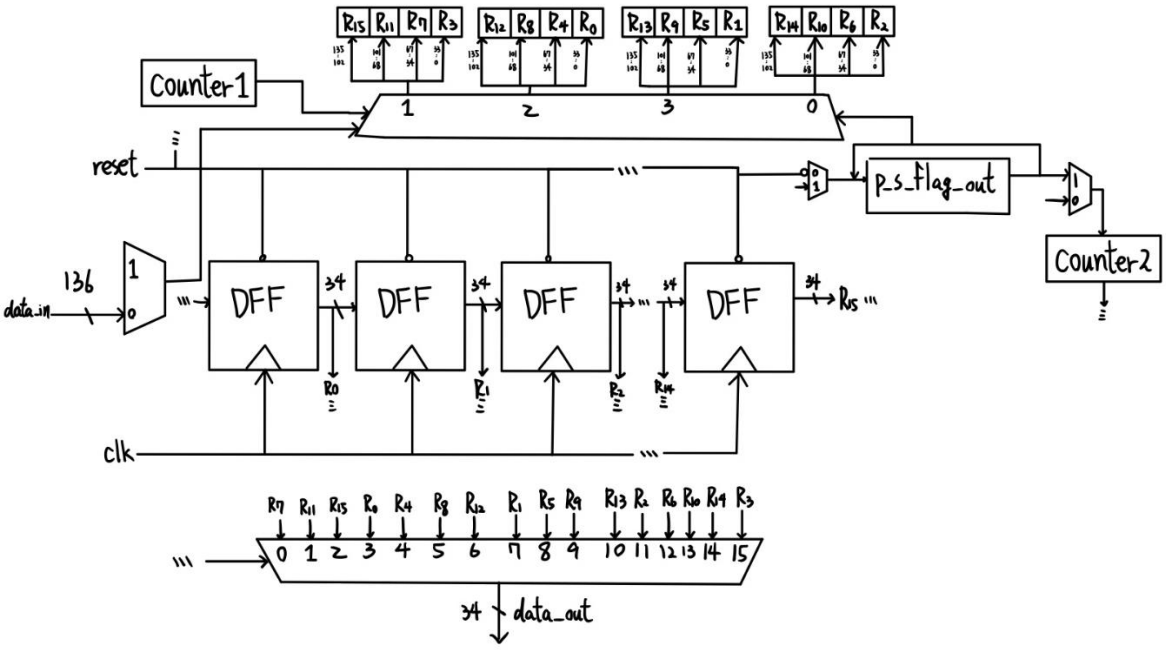
- SIPO

  作法:用 16 顆 DFF，每一顆 34bit，利用一個計數器來計數，達到指定數字去找指定的 DFF 來平行化輸出 data



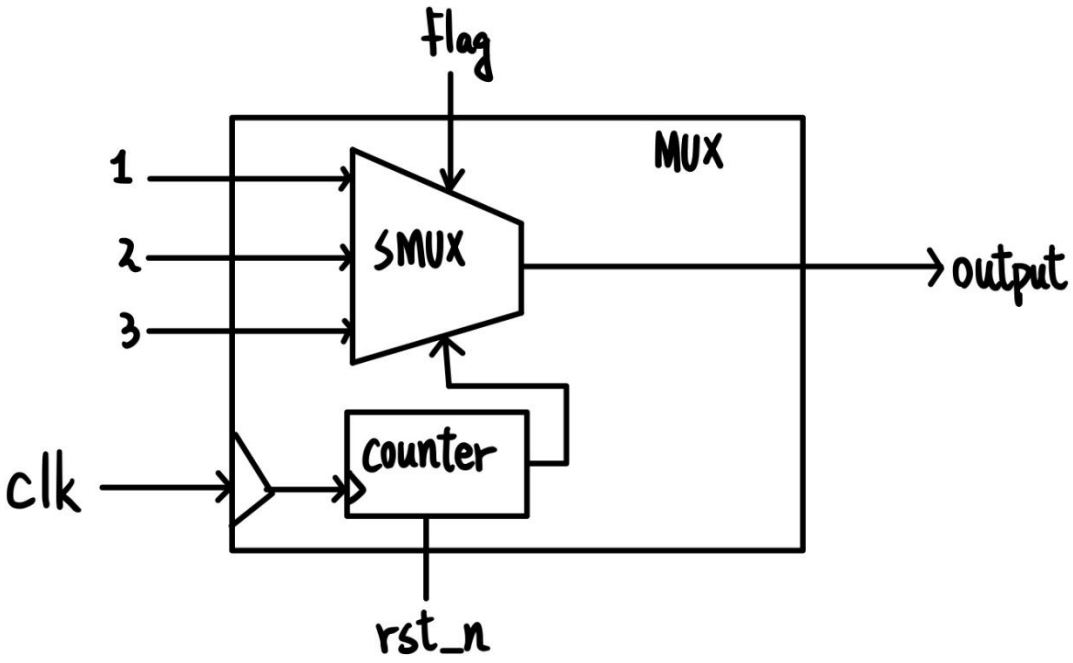
- PISO

  作法:跟 SIPO 類似，只是要讓輸出轉回串型輸出

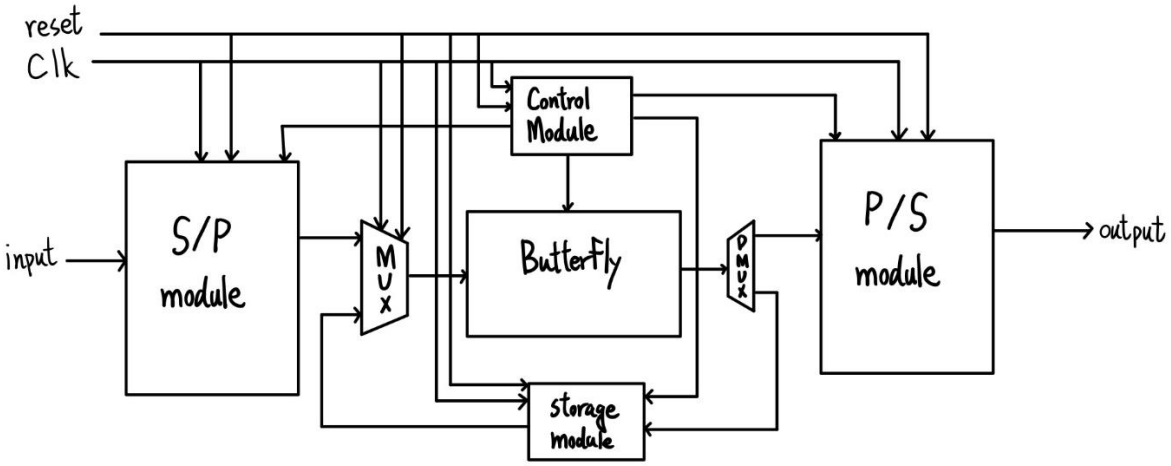


- **Mux**

  作法: 使用 flag 控制 data 選擇，counter 計數器的值來做其 Mux 功能，並透過 clk 和 rst_n 同步和 reset 內部的計數器跟 register



- **完整設計圖**



3. HDL Entry:

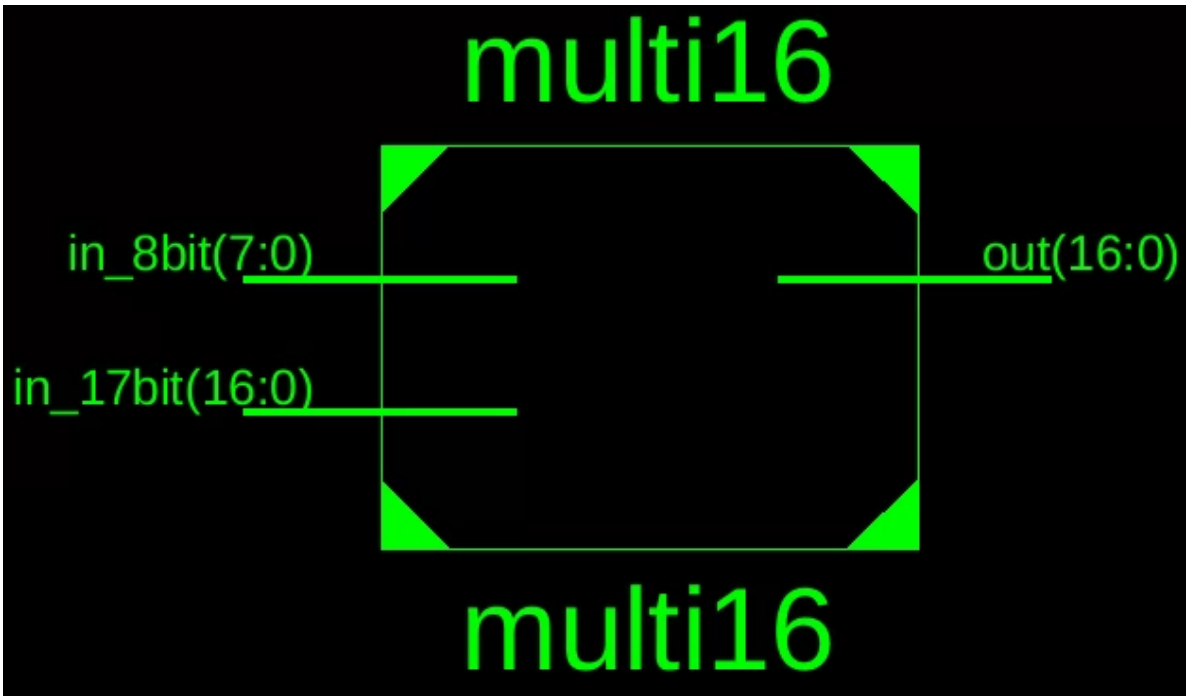
- Butterfly

1.multi16

Verilog code:

```
module multi16(

  input  wire [16:0] in_17bit,  // 17-bit input  data
  input  wire [7:0]  in_8bit,   // 8-bit  input  data
  output wire [16:0] out        // 17-bit output data

  );

  wire        flag;             // determine the sign of the product
  wire [16:0] in_17bit_b;       // store 17-bit true form data
  wire [7:0]  in_8bit_b;        // store 8-bit  true form data
  wire [24:0] mul;
  wire [16:0] mul_b;
  reg  [24:0] neg_mul;

  //*************************** The following are assign statements ***************************

  assign  in_17bit_b = (in_17bit[16] == 1) ? ~in_17bit[16:0] + 1'b1 : in_17bit;
  // If in_17bit is a negative number, transform to 2's complement, otherwise remain the same.

  always @ ( in_17bit or in_8bit ) begin
    case ( in_8bit )
      8'b00000000: neg_mul = 25'b0;
      8'b00110001: neg_mul = ( in_17bit_b << 0 ) + ( in_17bit_b << 4 ) + ( in_17bit_b << 5 );
      8'b11001111: neg_mul = ( in_17bit_b << 0 ) + ( in_17bit_b << 4 ) + ( in_17bit_b << 5 );
      8'b01011010: neg_mul = ( in_17bit_b << 1 ) + ( in_17bit_b << 3 ) + ( in_17bit_b << 4 ) + ( in_17bit_b << 6 );
      8'b10100110: neg_mul = ( in_17bit_b << 1 ) + ( in_17bit_b << 3 ) + ( in_17bit_b << 4 ) + ( in_17bit_b << 6 );
      8'b01110110: neg_mul = ( in_17bit_b << 1 ) + ( in_17bit_b << 2 ) + ( in_17bit_b << 4 ) + ( in_17bit_b << 5 ) + ( in_17bit_b << 6 );
      8'b10001010: neg_mul = ( in_17bit_b << 1 ) + ( in_17bit_b << 2 ) + ( in_17bit_b << 4 ) + ( in_17bit_b << 5 ) + ( in_17bit_b << 6 );
      8'b01111111: neg_mul = ( in_17bit_b << 7 );
      8'b10000001: neg_mul = ( in_17bit_b << 7 );
    endcase
  end

//  assign  in_8bit_b  = ( in_8bit[7]  == 1) ?  ~in_8bit[7:0]  + 1'b1 : in_8bit;
  // If in_8bit is a negative number, transform to 2's complement, otherwise remain the same.
  assign  flag  = in_17bit[16] + in_8bit[7];                // Determine the sign of the product.
//  assign  mul   = in_17bit_b[16:0] * in_8bit_b[7:0];          // Calculate the absolute value of the product.
  assign  mul = neg_mul;
  assign  mul_b = {mul[23:15], mul[14:7]};                  // Shift, combining integer and fractional parts.
  assign  out   = (flag == 1) ? ~mul_b[16:0] + 1'b1 : mul_b;  // Output only keep 17 bits.

endmodule
```

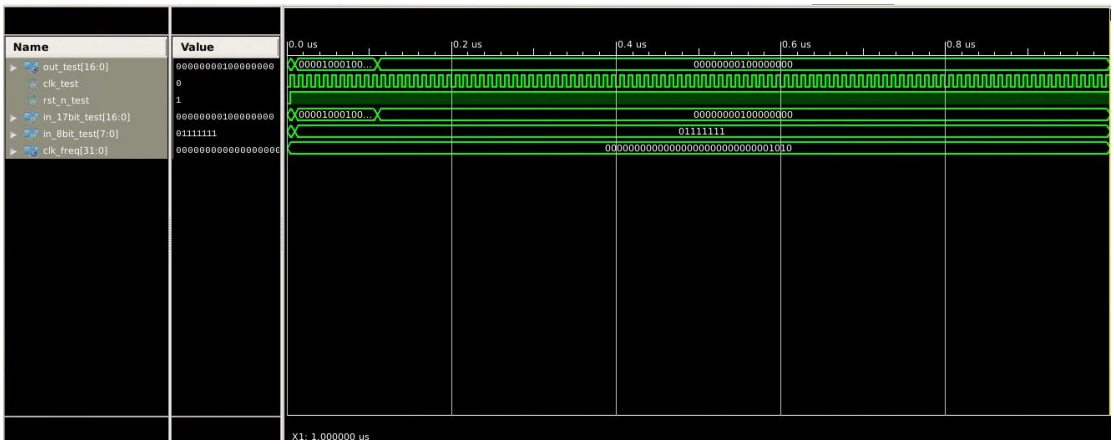
RTL:



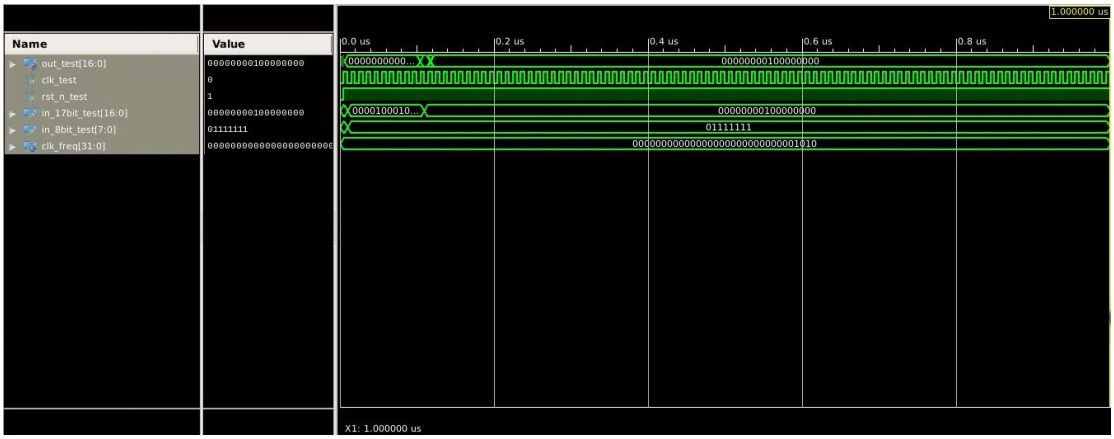
Testbench:

```
1  module multi16_tb( );
2
3    reg          clk_test;
4    reg          rst_n_test;
5    reg  [16:0]  in_17bit_test;
6    reg  [7:0]   in_8bit_test;
7    wire [16:0]  out_test;
8
9    parameter clk_freq = 10;
10
11   multi16 multi160(.in_17bit(in_17bit_test),
12                    .in_8bit(in_8bit_test),
13                    .out(out_test)
14   );
15
16   initial begin
17     clk_test = 0;
18     rst_n_test = 0;
19     in_17bit_test = 17'b0;
20     in_8bit_test = 8'b0;
21   end
22
23   always begin
24     #(clk_freq / 2) clk_test = ~ clk_test;  // create a 100MHz clock
25     rst_n_test = 1;
26   end
28   initial begin
29     #10 begin
30       in_17bit_test = 17'b00001000100001000;
31       in_8bit_test  = 8'b01111111;
32     end
33
34     #100 begin
35       in_17bit_test = 17'b00000000100000000;
36       in_8bit_test  = 8'b01111111;
37     end
38
39   end
40
41 endmodule
```

Behavioral:



Post-Route:



2.Butterfly

Verilog code:

```
module butterfly(
  input  wire [135:0] calc_in,  // The 4 numbers which need to be calculated. Format: in4(Re,Im), in3(Re,Im), in2(Re,Im), in1(Re,Im)
  input  wire [2:0]   rotation, // Number of each butterfly computation (8 in total)
  output wire [135:0] calc_out  // The 4 output numbers. Format: out4(Re,Im), out3(Re,Im), out2(Re,Im), out1(Re,Im)
  );

  parameter para0000 = 8'b00000000;  //  0.0000. sin(0pi/8) = sin(8pi/8) = cos(4pi/8) = - cos(4pi/8)
  parameter para3827 = 8'b00110001;  //  0.3827. sin(1pi/8) = sin(7pi/8) = cos(3pi/8) = - cos(5pi/8) 0.011000011111100010100000100100000010110111100000000
  parameter parn3827 = 8'b11001111;  // -0.3827. sin(1pi/8) = sin(7pi/8) = cos(3pi/8) = - cos(5pi/8)
  parameter para7071 = 8'b01011010;  //  0.7071. sin(2pi/8) = sin(6pi/8) = cos(2pi/8) = - cos(6pi/8) 0.101101010000010010000001011011110000000011010001110
  parameter parn7071 = 8'b10100110;  // -0.7071. sin(2pi/8) = sin(6pi/8) = cos(2pi/8) = - cos(6pi/8)
  parameter para9239 = 8'b01110110;  //  0.9239. sin(3pi/8) = sin(5pi/8) = cos(1pi/8) = - cos(7pi/8) 0.111011001000010010110101110111001100011000111111000 1
  parameter parn9239 = 8'b10001010;  // -0.9239. sin(3pi/8) = sin(5pi/8) = cos(1pi/8) = - cos(7pi/8)
  parameter para1111 = 8'b01111111;  //  1.0000. sin(4pi/8) = sin(4pi/8) = cos(0pi/8) = - cos(8pi/8)
  parameter parn1111 = 8'b10000001;  // -1.0000.

  reg  signed [7:0]  rotation_factor1_real;  // For input B
  reg  signed [7:0]  rotation_factor1_imag;  // For input B
  reg  signed [7:0]  rotation_factor2_real;  // For input C
  reg  signed [7:0]  rotation_factor2_imag;  // For input C
  reg  signed [7:0]  rotation_factor3_real;  // For input D
  reg  signed [7:0]  rotation_factor3_imag;  // For input D

  wire signed [7:0]  in_8bit_1_real, in_8bit_1_imag;
  wire signed [7:0]  in_8bit_2_real, in_8bit_2_imag;
  wire signed [7:0]  in_8bit_3_real, in_8bit_3_imag;

  wire signed [33:0] temp_0, temp_1, temp_2, temp_3;

  wire signed [16:0] temp_0_real, temp_0_imag;
  wire signed [16:0] temp_1_real, temp_1_imag;
  wire signed [16:0] temp_2_real, temp_2_imag;
  wire signed [16:0] temp_3_real, temp_3_imag;

  wire signed [16:0] in_17bit_0_real, in_17bit_0_imag;
  wire signed [16:0] in_17bit_1_real, in_17bit_1_imag;
  wire signed [16:0] in_17bit_2_real, in_17bit_2_imag;
    wire signed [16:0] in_17bit_3_real, in_17bit_3_imag;

    wire signed [16:0] out_0_real, out_0_imag;
    wire signed [16:0] out_1_real, out_1_imag;
    wire signed [16:0] out_2_real, out_2_imag;
    wire signed [16:0] out_3_real, out_3_imag;

    wire signed [16:0] temp_2_1_real, temp_2_2_real, temp_2_1_imag, temp_2_2_imag;
    wire signed [16:0] temp_3_1_real, temp_3_2_real, temp_3_1_imag, temp_3_2_imag;
    wire signed [16:0] temp_4_1_real, temp_4_2_real, temp_4_1_imag, temp_4_2_imag;

  //***************************** The following are state machines ******************

// This always part controls signal rotation_factor1_real.
  always @ ( rotation ) begin
    case ( rotation )
      3'b000: rotation_factor1_real = para1111;
      3'b001: rotation_factor1_real = para1111;
      3'b010: rotation_factor1_real = para1111;
      3'b011: rotation_factor1_real = para1111;
      3'b100: rotation_factor1_real = para1111;  //   cos(0)     =   1        W_16^0
      3'b101: rotation_factor1_real = para9239;  //   cos(pi/8)  =   0.9239   W_16^1
      3'b110: rotation_factor1_real = para7071;  //   cos(pi/4)  =   0.7071   W_16^2
      3'b111: rotation_factor1_real = para3827;  //   cos(3pi/8) =   0.3827   W_16^3
    endcase
  end

// This always part controls signal rotation_factor1_imag.
  always @ ( rotation ) begin
    case ( rotation )
      3'b000: rotation_factor1_imag = para0000;
      3'b001: rotation_factor1_imag = para0000;
      3'b010: rotation_factor1_imag = para0000;
      3'b011: rotation_factor1_imag = para0000;
```

```
      3'b100: rotation_factor1_imag = para0000;  // - sin(0)     =   0        W_16^0
      3'b101: rotation_factor1_imag = parn3827;  // - sin(pi/8)  = - 0.3827   W_16^1
      3'b110: rotation_factor1_imag = parn7071;  // - sin(pi/4)  = - 0.7071   W_16^2
      3'b111: rotation_factor1_imag = parn9239;  // - sin(3pi/8) = - 0.9239   W_16^3
    endcase
  end

// This always part controls signal rotation_factor2_real.
  always @ ( rotation ) begin
    case ( rotation )
      3'b000: rotation_factor2_real = para1111;
      3'b001: rotation_factor2_real = para1111;
      3'b010: rotation_factor2_real = para1111;
      3'b011: rotation_factor2_real = para1111;
      3'b100: rotation_factor2_real = para1111;  //   cos(0)     =   1        W_16^0
      3'b101: rotation_factor2_real = para7071;  //   cos(pi/4)  =   0.7071   W_16^2
      3'b110: rotation_factor2_real = para0000;  //   cos(pi/2)  =   0        W_16^4
      3'b111: rotation_factor2_real = parn7071;  //   cos(3pi/8) = - 0.7071   W_16^6
    endcase
  end

// This always part controls signal rotation_factor2_imag.
  always @ ( rotation ) begin
    case ( rotation )
      3'b000: rotation_factor2_imag = para0000;
      3'b001: rotation_factor2_imag = para0000;
      3'b010: rotation_factor2_imag = para0000;
      3'b011: rotation_factor2_imag = para0000;
      3'b100: rotation_factor2_imag = para0000;  // - sin(0)     =   0        W_16^0
      3'b101: rotation_factor2_imag = parn7071;  // - sin(pi/8)  = - 0.7071   W_16^2
      3'b110: rotation_factor2_imag = parn1111;  // - sin(pi/4)  = - 1        W_16^4
      3'b111: rotation_factor2_imag = parn7071;  // - sin(3pi/8) = - 0.7071   W_16^6
    endcase
  end
```

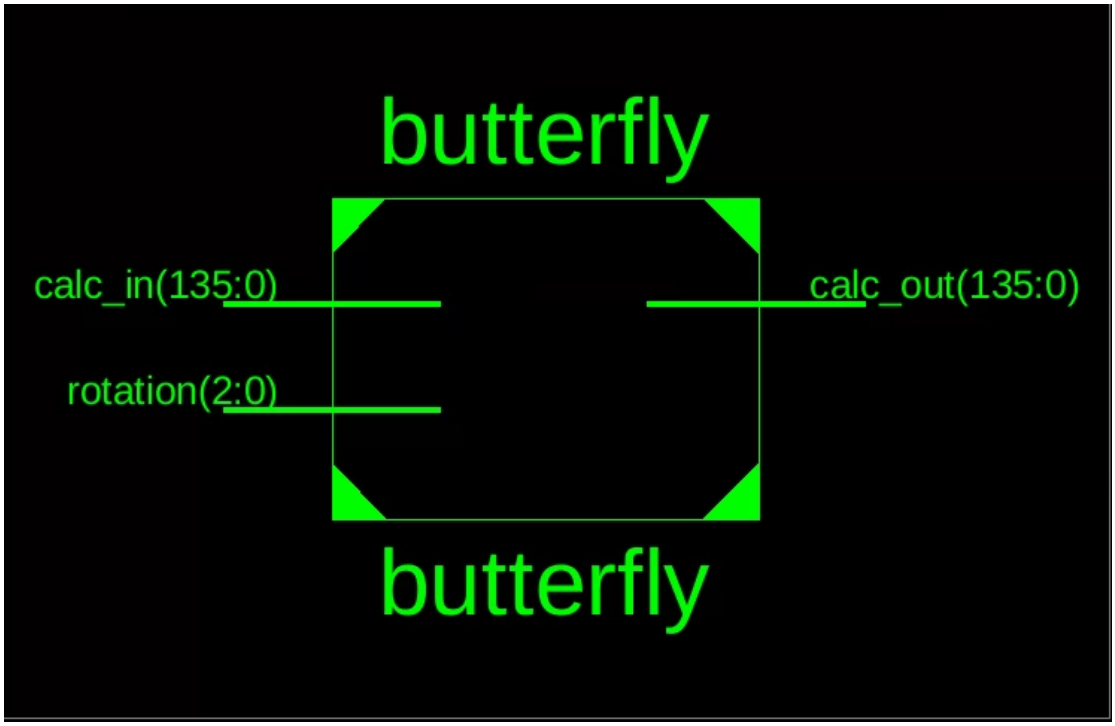
```
107   // This always part controls signal rotation_factor3_real.
108     always @ ( rotation ) begin
109       case ( rotation )
110         3'b000: rotation_factor3_real = para1111;
111         3'b001: rotation_factor3_real = para1111;
112         3'b010: rotation_factor3_real = para1111;
113         3'b011: rotation_factor3_real = para1111;
114         3'b100: rotation_factor3_real = para1111;  //   cos(0)     =   1        W_16^0
115         3'b101: rotation_factor3_real = para3827;  //   cos(3pi/8) =   0.7071   W_16^3
116         3'b110: rotation_factor3_real = parn7071;  //   cos(6pi/8) =   0        W_16^6
117         3'b111: rotation_factor3_real = parn9239;  //   cos(9pi/8) = - 0.7071   W_16^9
118       endcase
119     end
120
121   // This always part controls signal rotation_factor3_imag.
122     always @ ( rotation ) begin
123       case ( rotation )
124         3'b000: rotation_factor3_imag = para0000;
125         3'b001: rotation_factor3_imag = para0000;
126         3'b010: rotation_factor3_imag = para0000;
127         3'b011: rotation_factor3_imag = para0000;
128         3'b100: rotation_factor3_imag = para0000;  // - sin(0)     = - 0        W_16^0
129         3'b101: rotation_factor3_imag = parn9239;  // - sin(3pi/8) = - 0.7071   W_16^3
130         3'b110: rotation_factor3_imag = parn7071;  // - sin(6pi/8) = - 0.1      W_16^6
131         3'b111: rotation_factor3_imag = para3827;  // - sin(9pi/8) =   0.7071   W_16^9
132       endcase
133     end
134
135   //***************************** The following are instantiations ******************
136
137     multi16 multiBRR( .in_17bit(in_17bit_1_real),
138                       .in_8bit(in_8bit_1_real),
139                       .out(temp_2_1_real)
140                     );
```

```
141
142     multi16 multiBII( .in_17bit(in_17bit_1_imag),
143                       .in_8bit(in_8bit_1_imag),
144                       .out(temp_2_2_real)
145                     );
146
147     multi16 multiBRI( .in_17bit(in_17bit_1_real),
148                       .in_8bit(in_8bit_1_imag),
149                       .out(temp_2_1_imag)
150                     );
151
152     multi16 multiBIR( .in_17bit(in_17bit_1_imag),
153                       .in_8bit(in_8bit_1_real),
154                       .out(temp_2_2_imag)
155                     );
156
157   //************************************************
158
159     multi16 multiCRR( .in_17bit(in_17bit_2_real),
160                       .in_8bit(in_8bit_2_real),
161                       .out(temp_3_1_real)
162                     );
163
164     multi16 multiCII( .in_17bit(in_17bit_2_imag),
165                       .in_8bit(in_8bit_2_imag),
166                       .out(temp_3_2_real)
167                     );
168
169     multi16 multiCRI( .in_17bit(in_17bit_2_real),
170                       .in_8bit(in_8bit_2_imag),
171                       .out(temp_3_1_imag)
172                     );
173
174     multi16 multiCIR( .in_17bit(in_17bit_2_imag),
175                       .in_8bit(in_8bit_2_real),
176                       .out(temp_3_2_imag)
177                     );
178
179   //*****************************************************************************
180
181     multi16 multiDRR( .in_17bit(in_17bit_3_real),
182                       .in_8bit(in_8bit_3_real),
183                       .out(temp_4_1_real)
184                     );
185
186     multi16 multiDII( .in_17bit(in_17bit_3_imag),
187                       .in_8bit(in_8bit_3_imag),
188                       .out(temp_4_2_real)
189                     );
190
191     multi16 multiDRI( .in_17bit(in_17bit_3_real),
192                       .in_8bit(in_8bit_3_imag),
193                       .out(temp_4_1_imag)
194                     );
195
196     multi16 multiDIR( .in_17bit(in_17bit_3_imag),
197                       .in_8bit(in_8bit_3_real),
198                       .out(temp_4_2_imag)
199                     );
200
201   //************************** The following are assign statements ***************
202
203     assign in_17bit_0_real = calc_in[33:17];         // A (real)
204     assign in_17bit_0_imag = calc_in[16:0];          // A (imag)
205
206     assign in_8bit_1_real  = rotation_factor1_real;  // rotation factor for B (real)
```

```
  assign in_8bit_1_imag  = rotation_factor1_imag;  // rotation factor for B (imag)
  assign in_17bit_1_real = calc_in[67:51];         // B (real)
  assign in_17bit_1_imag = calc_in[50:34];         // B (imag)

  assign in_8bit_2_real  = rotation_factor2_real;  // rotation factor for C (real)
  assign in_8bit_2_imag  = rotation_factor2_imag;  // rotation factor for C (imag)
  assign in_17bit_2_real = calc_in[101:85];        // C (real)
  assign in_17bit_2_imag = calc_in[84:68];         // C (imag)

  assign in_8bit_3_real  = rotation_factor3_real;  // rotation factor for D (real)
  assign in_8bit_3_imag  = rotation_factor3_imag;  // rotation factor for D (imag)
  assign in_17bit_3_real = calc_in[135:119];       // D (real)
  assign in_17bit_3_imag = calc_in[118:102];       // D (imag)

//*****************************************************************************

  assign temp_0_real = {in_17bit_0_real[16], in_17bit_0_real};  // A real
  assign temp_0_imag = {in_17bit_0_imag[16], in_17bit_0_imag};  // A imag

  assign temp_1_real = temp_2_1_real - temp_2_2_real;  //B * W^P     real
  assign temp_1_imag = temp_2_1_imag + temp_2_2_imag;  //B * W^P     imag

  assign temp_2_real = temp_3_1_real - temp_3_2_real;  //C * W^{2p}  real
  assign temp_2_imag = temp_3_1_imag + temp_3_2_imag;  //C * W^{2p}  imag

  assign temp_3_real = temp_4_1_real - temp_4_2_real;  //D * W^{3P}  real
  assign temp_3_imag = temp_4_1_imag + temp_4_2_imag;  //D * W^{3P}  imag


  assign temp_0 = {temp_0_real,temp_0_imag};   // A
  assign temp_1 = {temp_1_real, temp_1_imag};  // B * Wp
  assign temp_2 = {temp_2_real, temp_2_imag};  // C * W2p
  assign temp_3 = {temp_3_real, temp_3_imag};  // D * W3p
```

```
   assign out_0_real = temp_0_real + temp_1_real + temp_2_real + temp_3_real;
   assign out_0_imag = temp_0_imag + temp_1_imag + temp_2_imag + temp_3_imag;

   assign out_1_real = temp_0_real - temp_1_real + temp_2_real - temp_3_real;
   assign out_1_imag = temp_0_imag - temp_1_imag + temp_2_imag - temp_3_imag;

   assign out_2_real = temp_0_real - temp_1_imag - temp_2_real + temp_3_imag;
   assign out_2_imag = temp_0_imag + temp_1_real - temp_2_imag - temp_3_real;
   assign out_3_real = temp_0_real + temp_1_imag - temp_2_real - temp_3_imag;
   assign out_3_imag = temp_0_imag - temp_1_real - temp_2_imag + temp_3_real;

   assign calc_out = { out_2_real, out_2_imag,
                       out_1_real, out_1_imag,
                       out_3_real, out_3_imag,
                       out_0_real, out_0_imag };

endmodule
```

HDL:



Testbench:

```
1   module butterfly_tb();
2
3     reg          clk_test;
4     reg          rst_n_test;
5     reg  [135:0] calc_in_test;
6     reg  [2:0]   rotation_test;
7     wire [135:0] calc_out_test;
8
9     reg  [135:0] temp1, temp2, temp3, temp4;
10
11    parameter clk_freq = 10;  // or 8 for 125 MHz
12
13    butterfly butterfly0(.calc_in(calc_in_test),
14                         .rotation(rotation_test),
15                         .calc_out(calc_out_test)
16    );
17
18    initial begin
19      clk_test = 0;
20      rst_n_test = 0;
21      calc_in_test = 135'b0;
22      rotation_test = 3'b0;
23    end
24
25    always begin
26        #(clk_freq / 2) clk_test = ~ clk_test;  // create a 100 MHz clock
27        rst_n_test = 1;  // enable
28    end
```

```
30      initial begin
31    //******************************** The following is the 16 inputs ************************
32        $display("\n\nLoad Data\n");
33        #10 begin
34          calc_in_test = 136'b0000000010000000000000000000000000000000100000000000000000000000
35          //                  |   x[12]Real   ||   x[12]Imag   ||    x[8]Real   ||    x[8]Imag
36          rotation_test = 3'b000;
37        end
38
39        #10 temp1 = calc_out_test;
40
41        #10 begin
42          calc_in_test = 136'b0000000010000000000000000000000000000000100000000000000100000000
43          //                  |   x[13]Real   ||   x[13]Imag   ||    x[9]Real   ||    x[9]Imag
44          rotation_test = 3'b001;
45        end
46
47        #10 temp2 = calc_out_test;
48
49        #10 begin
50          calc_in_test = 136'b0000000010000000000000000000000000000000100000000000000000000000
51          //                  |   x[14]Real   ||   x[14]Imag   ||    x[10]Real  ||    x[10]Imag
52          rotation_test = 3'b010;
53        end
```

```
0000000011000000000000000000000000000000100000000000000000000000;
||    x[4]Real   ||    x[4]Imag   ||    x[0]Real   ||   x[0]Imag    |
```

```
0000000010000000000000010000000000000000000000000000000000000000;
||    x[5]Real   ||    x[5]Imag   ||    x[1]Real   ||   x[1]Imag    |
```

```
0000000001000000000000010000000000000000100000000000000000000000;
||    x[6]Real   ||    x[6]Imag   ||    x[2]Real   ||   x[2]Imag    |
```

```
    #10 temp3 = calc_out_test;

    #10 begin
      calc_in_test = 136'b0000000010000000000000000000000000000000000000000000000000000000
    //                    |   x[15]Real   ||    x[15]Imag  ||    x[11]Real  ||    x[11
      rotation_test = 3'b011;
    end

    #10 temp4 = calc_out_test;

//********************************** The following is stage 2 **********************
    $display("\n\nOUTPUT\n");

    #10 begin
      calc_in_test = {temp4[33:0], temp3[33:0], temp2[33:0], temp1[33:0]};
      rotation_test = 3'b100;
    end

    #10 begin
      $display("[0]  %b_%b_%b %b_%b_%b\n", calc_out_test[33],  calc_out_test[32:25],
      $display("[4]  %b_%b_%b %b_%b_%b\n", calc_out_test[67],  calc_out_test[66:59],
      $display("[8]  %b_%b_%b %b_%b_%b\n", calc_out_test[101], calc_out_test[100:93],
      $display("[12] %b_%b_%b %b_%b_%b\n", calc_out_test[135], calc_out_test[134:127],
    end
```

```
)0000000000000010000000000000010000000000000000000000000000000000000000;
|Imag  ||    x[7]Real   ||    x[7]Imag   ||    x[3]Real   ||   x[3]Imag    |
```

```
**********
```

```
calc_out_test[24:17],   calc_out_test[16],  calc_out_test[15:8],    calc_out_test[7:0]);
calc_out_test[58:51],   calc_out_test[50],  calc_out_test[49:42],   calc_out_test[41:34]);
calc_out_test[92:85],   calc_out_test[84],  calc_out_test[83:76],   calc_out_test[75:68]);
calc_out_test[126:119], calc_out_test[118], calc_out_test[117:110], calc_out_test[109:102]);
```

```
    #10 begin
      calc_in_test = {temp4[67:34], temp3[67:34], temp2[67:34], temp1[67:34]};
      rotation_test = 3'b101;
    end

    #10 begin
      $display("[1]  %b_%b_%b %b_%b_%b\n", calc_out_test[33],  calc_out_test[32:25],
      $display("[5]  %b_%b_%b %b_%b_%b\n", calc_out_test[67],  calc_out_test[66:59],
      $display("[9]  %b_%b_%b %b_%b_%b\n", calc_out_test[101], calc_out_test[100:93],
      $display("[13] %b_%b_%b %b_%b_%b\n", calc_out_test[135], calc_out_test[134:127],
    end

    #10 begin
      calc_in_test = {temp4[101:68], temp3[101:68], temp2[101:68], temp1[101:68]};
      rotation_test = 3'b110;
    end

    #10 begin
      $display("[2]  %b_%b_%b %b_%b_%b\n", calc_out_test[33],  calc_out_test[32:25],
      $display("[6]  %b_%b_%b %b_%b_%b\n", calc_out_test[67],  calc_out_test[66:59],
      $display("[10] %b_%b_%b %b_%b_%b\n", calc_out_test[101], calc_out_test[100:93],
      $display("[14] %b_%b_%b %b_%b_%b\n", calc_out_test[135], calc_out_test[134:127],
    end
```

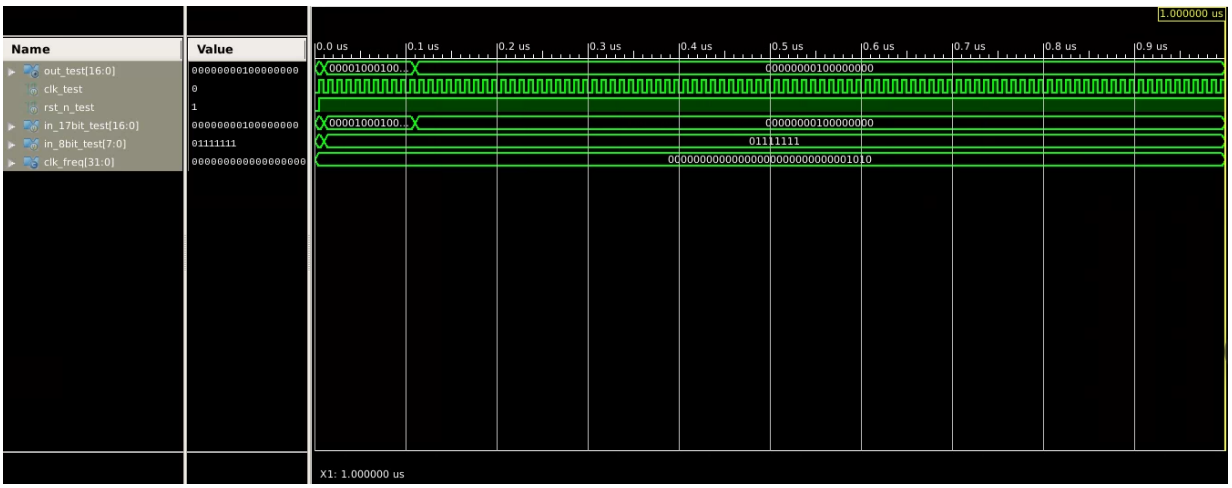
```
calc_out_test[24:17],   calc_out_test[16],  calc_out_test[15:8],    calc_out_test[7:0]);
calc_out_test[58:51],   calc_out_test[50],  calc_out_test[49:42],   calc_out_test[41:34]);
calc_out_test[92:85],   calc_out_test[84],  calc_out_test[83:76],   calc_out_test[75:68]);
calc_out_test[126:119], calc_out_test[118], calc_out_test[117:110], calc_out_test[109:102]);
```


```
calc_out_test[24:17],   calc_out_test[16],  calc_out_test[15:8],    calc_out_test[7:0]);
calc_out_test[58:51],   calc_out_test[50],  calc_out_test[49:42],   calc_out_test[41:34]);
calc_out_test[92:85],   calc_out_test[84],  calc_out_test[83:76],   calc_out_test[75:68]);
calc_out_test[126:119], calc_out_test[118], calc_out_test[117:110], calc_out_test[109:102]);
```

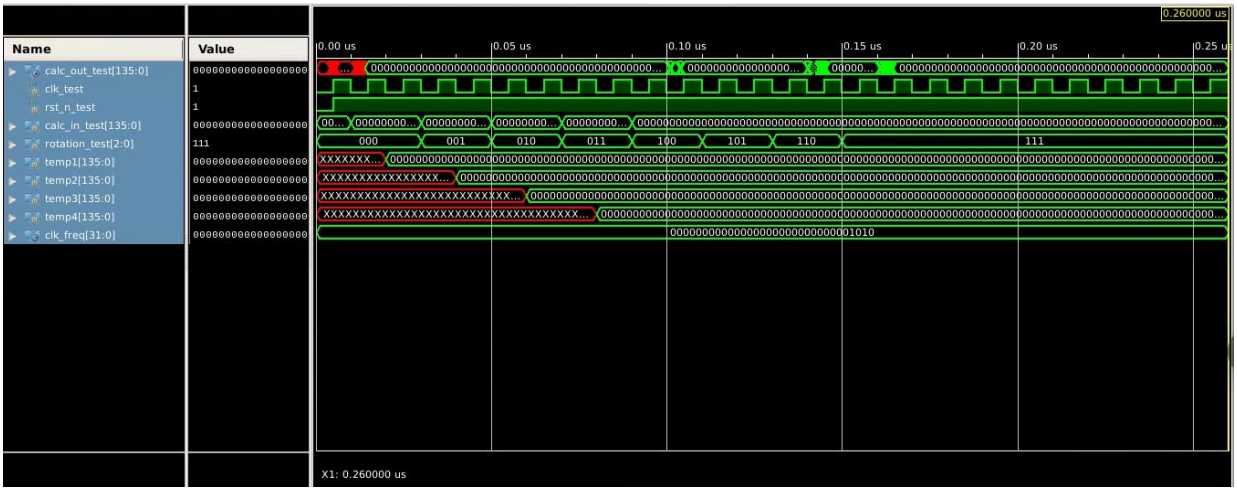
```
104         #10 begin
105           calc_in_test = {temp4[135:102], temp3[135:102], temp2[135:102], temp1[135:102]};
106           rotation_test = 3'b111;
107         end
108
109         #10 begin
110           $display("[3]  %b_%b_%b %b_%b_%b\n", calc_out_test[33],  calc_out_test[32:25],
111           $display("[7]  %b_%b_%b %b_%b_%b\n", calc_out_test[67],  calc_out_test[66:59],
112           $display("[11] %b_%b_%b %b_%b_%b\n", calc_out_test[101], calc_out_test[100:93],
113           $display("[15] %b_%b_%b %b_%b_%b\n", calc_out_test[135], calc_out_test[134:127],
114         end
115
116         #100 $stop;
117
118       end
119
120     endmodule
```

```
calc_out_test[24:17],   calc_out_test[16],  calc_out_test[15:8],    calc_out_test[7:0]);
calc_out_test[58:51],   calc_out_test[50],  calc_out_test[49:42],   calc_out_test[41:34]);
calc_out_test[92:85],   calc_out_test[84],  calc_out_test[83:76],   calc_out_test[75:68]);
calc_out_test[126:119], calc_out_test[118], calc_out_test[117:110], calc_out_test[109:102]);
```

Behavioral:



Post-Route:



- SIPO

Verilog code:

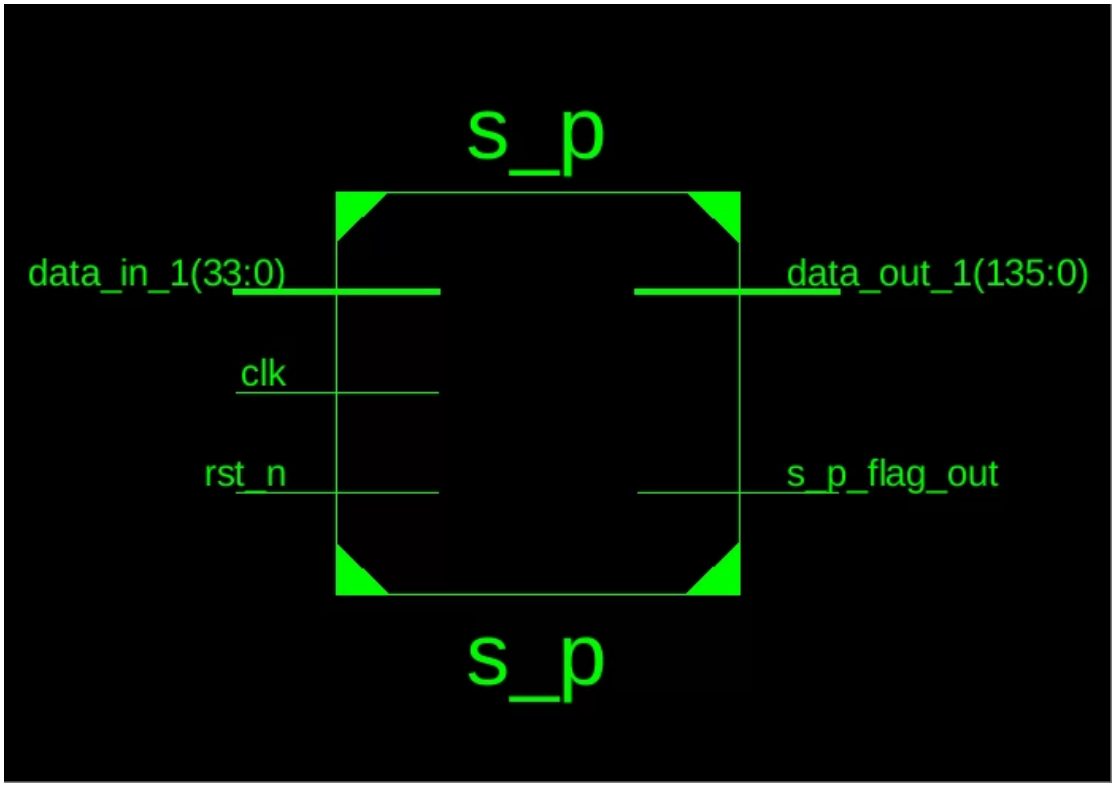
```
1   module s_p(
2     input wire clk,
3     input wire rst_n,
4     input wire [33:0] data_in_1,
5     output reg [135:0] data_out_1,
6     output reg s_p_flag_out
7     );
8     reg [33:0] R0;
9     reg [33:0] R1;
10    reg [33:0] R2;
11    reg [33:0] R3;
12    reg [33:0] R4;
13    reg [33:0] R5;
14    reg [33:0] R6;
15    reg [33:0] R7;
16    reg [33:0] R8;
17    reg [33:0] R9;
18    reg [33:0] R10;
19    reg [33:0] R11;
20    reg [33:0] R12;
21    reg [33:0] R13;
22    reg [33:0] R14;
23    reg [33:0] R15;
```

```
25    reg [3:0] counter;
26
27    reg s_p_flag_mux;
28   // This always part controls signal counter.
29    always @ ( posedge clk or negedge rst_n ) begin
30      if ( !rst_n )
31        counter <= 4'b0;
32      else if (counter == 4'b1111)
33        counter <= 4'b0;
34      else
35        counter <= counter + 4'b0001;
36    end
37
38   // This always part controls signal s_p_flag_out.
39    always @ ( posedge clk or negedge rst_n ) begin
40      if ( !rst_n )
41        s_p_flag_out <= 0;
42      else if (counter == 4'b1100)
43        s_p_flag_out <= 1'b1;
44      else
45        s_p_flag_out <= 1'b0;
46   end
47
48  // This always part controls signal s_p_flag_mux.
49
50   always @ ( posedge clk or negedge rst_n ) begin
51     if ( !rst_n )
52       s_p_flag_mux <= 0;
53     else begin
54       case ( counter )
55         4'b0000: s_p_flag_mux <= 1'b1;
56         4'b1101: s_p_flag_mux <= 1'b1;
57         4'b1110: s_p_flag_mux <= 1'b1;
58         4'b1111: s_p_flag_mux <= 1'b1;
59         default: s_p_flag_mux <= 1'b0;
```

```
      endcase
    end
  end

// This always part controls signal data_in_1. or negedge rst_n

  always @ ( posedge clk ) begin
    case ( counter )
      4'b0000: data_out_1  = {R15,  R11, R7,  R3};
      4'b1101: data_out_1  = {R12,  R8,  R4,  R0};
      4'b1110: data_out_1  = {R13,  R9,  R5,  R1};
      4'b1111: data_out_1  = {R14,  R10, R6,  R2};
    endcase
end
```

```
  always @(posedge clk) begin
    case(counter)
      4'b0000: R0  <= data_in_1;
      4'b0001: R1  <= data_in_1;
      4'b0010: R2  <= data_in_1;
      4'b0011: R3  <= data_in_1;
      4'b0100: R4  <= data_in_1;
      4'b0101: R5  <= data_in_1;
      4'b0110: R6  <= data_in_1;
      4'b0111: R7  <= data_in_1;
      4'b1000: R8  <= data_in_1;
      4'b1001: R9  <= data_in_1;
      4'b1010: R10 <= data_in_1;
      4'b1011: R11 <= data_in_1;
      4'b1100: R12 <= data_in_1;
      4'b1101: R13 <= data_in_1;
      4'b1110: R14 <= data_in_1;
      4'b1111: R15 <= data_in_1;
    endcase
  end
endmodule
```

HDL:



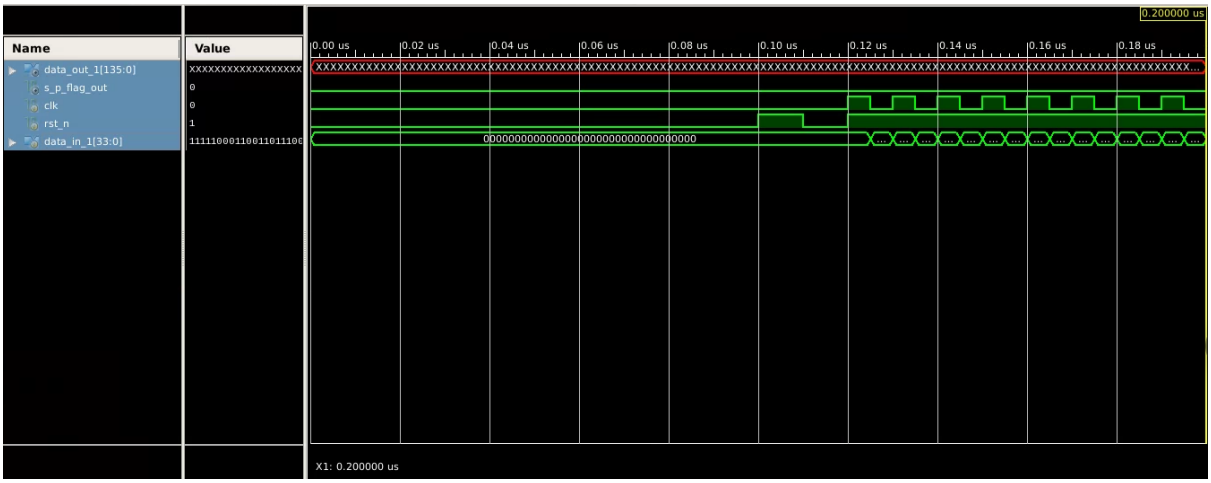
Testbench:

```
24  module s_p_tb;
25
26  // Inputs
27  reg clk;
28  reg rst_n;
29  reg [33:0] data_in_1;
30
31  // Outputs
32  wire [135:0] data_out_1;
33  wire s_p_flag_out;
34
35  // Instantiate the Unit Under Test (UUT)
36  s_p uut (
37      .clk(clk),
38      .rst_n(rst_n),
39      .data_in_1(data_in_1),
40      .data_out_1(data_out_1),
41      .s_p_flag_out(s_p_flag_out)
42  );
```

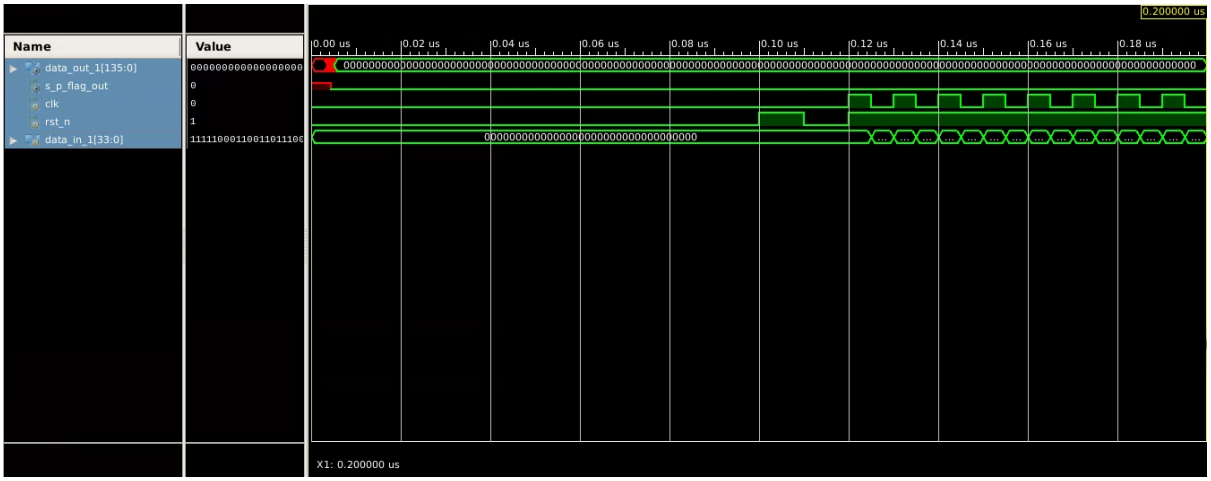
```
44    initial begin
45        // Initialize Inputs
46        clk = 0;
47        rst_n = 0;
48        data_in_1 = 0;
49
50        // Wait 100 ns for global reset to finish
51        #100;
52
53        // Reset sequence
54        rst_n = 1;
55        #10;
56        rst_n = 0;
57        #10;
58        rst_n = 1;
60        // Clocking and data input
61        repeat (16) begin
62            // Toggle clock
63            clk = ~clk;
64            #5;
65
66            // Assign data_in_1 with some test data
67            data_in_1 = $random;
68
69            // Display input and output for debugging
70            $display("Time=%t, clk=%b, rst_n=%b, data_in_1=%h,
71        end
72
73        // End simulation
74        $finish;
75    end
76
77    endmodule
```

```
data_out_1=%h, s_p_flag_out=%b", $time, clk, rst_n, data_in_1, data_out_1, s_p_flag_out);
```

Behavioral:



Post-Route:



- PISO

Verilog code:

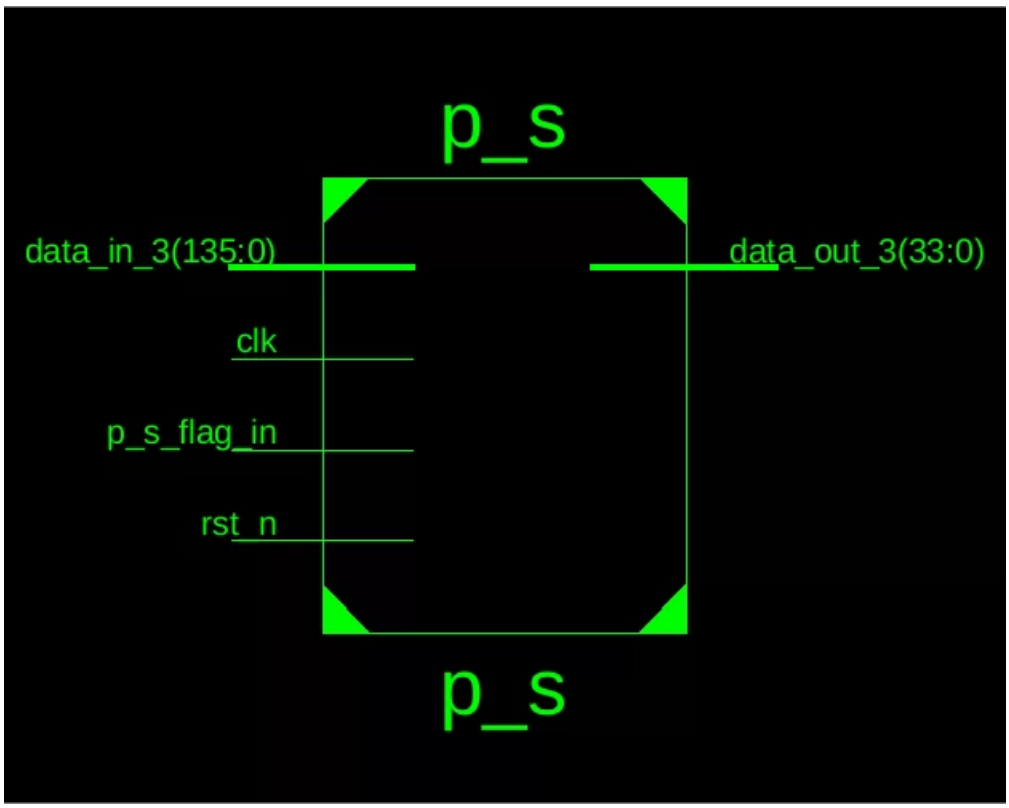
```
1   module p_s(
2     input wire clk,
3     input wire rst_n,
4     input wire [135:0] data_in_3,
5     input wire p_s_flag_in,
6     output reg [33:0] data_out_3
7     );
8     reg [33:0] R0;
9     reg [33:0] R1;
10    reg [33:0] R2;
11    reg [33:0] R3;
12    reg [33:0] R4;
13    reg [33:0] R5;
14    reg [33:0] R6;
15    reg [33:0] R7;
16    reg [33:0] R8;
17    reg [33:0] R9;
18    reg [33:0] R10;
19    reg [33:0] R11;
20    reg [33:0] R12;
21    reg [33:0] R13;
22    reg [33:0] R14;
23    reg [33:0] R15;

25    reg [1:0] counter_1;
26
27    reg [3:0] counter_2;
28
29    reg p_s_flag_out;
30
31  // This always part controls signal counter_1.
32
33    always @ ( posedge clk or negedge rst_n ) begin
34
35      if ( !rst_n )
36        counter_1 <= 2'b0;
37      else if ( counter_1 == 2'b11 )
38        counter_1 <= 2'b0;
39      else
40        counter_1 <= counter_1 + 2'b01;
41    end
42
```

```
// This always part controls registers.

 always @ ( posedge clk ) begin
   if ( !p_s_flag_in ) begin
     case ( counter_1 )
       2'b10: begin
         R0 <= data_in_3[33:0];
         R4 <= data_in_3[67:34];
         R8 <= data_in_3[101:68];
         R12 <= data_in_3[135:102];
       end
       2'b11: begin
         R1 <= data_in_3[33:0];
         R5 <= data_in_3[67:34];
         R9 <= data_in_3[101:68];
         R13 <= data_in_3[135:102];
       end
       2'b00: begin
         R2 <= data_in_3[33:0];
         R6 <= data_in_3[67:34];
         R10 <= data_in_3[101:68];
         R14 <= data_in_3[135:102];
       end
       2'b01: begin
         R3 <= data_in_3[33:0];
         R7 <= data_in_3[67:34];
         R11 <= data_in_3[101:68];
         R15 <= data_in_3[135:102];
       end
       endcase
   end
end
```

```
// This always part controls signal counter_2.

 always @ ( posedge clk or negedge rst_n ) begin

   if ( !rst_n )
     counter_2 <= 4'b0;
   else if ( counter_2 == 4'b1111 )
     counter_2 <= 4'b0;
   else
     counter_2 <= counter_2 + 4'b0001;
 end

// This always part controls p_s_flag_out.

 always @ ( posedge clk or negedge rst_n ) begin
    if ( !rst_n )
     p_s_flag_out <= 1'b0;
    else if ( !p_s_flag_in )
     p_s_flag_out <= 1'b1;
    else
     p_s_flag_out <= p_s_flag_out;
 end
```

```
  always @ ( posedge clk ) begin
    if ( p_s_flag_out ) begin
      case( counter_2 )
        4'b0000: data_out_3 <= R7;
        4'b0001: data_out_3 <= R11;
        4'b0010: data_out_3 <= R15;
        4'b0011: data_out_3 <= R0;
        4'b0100: data_out_3 <= R4;
        4'b0101: data_out_3 <= R8;
        4'b0110: data_out_3 <= R12;
        4'b0111: data_out_3 <= R1;
        4'b1000: data_out_3 <= R5;
        4'b1001: data_out_3 <= R9;
        4'b1010: data_out_3 <= R13;
        4'b1011: data_out_3 <= R2;
        4'b1100: data_out_3 <= R6;
        4'b1101: data_out_3 <= R10;
        4'b1110: data_out_3 <= R14;
        4'b1111: data_out_3 <= R3;
      endcase
    end
  end
```

HDL:



Testbench:

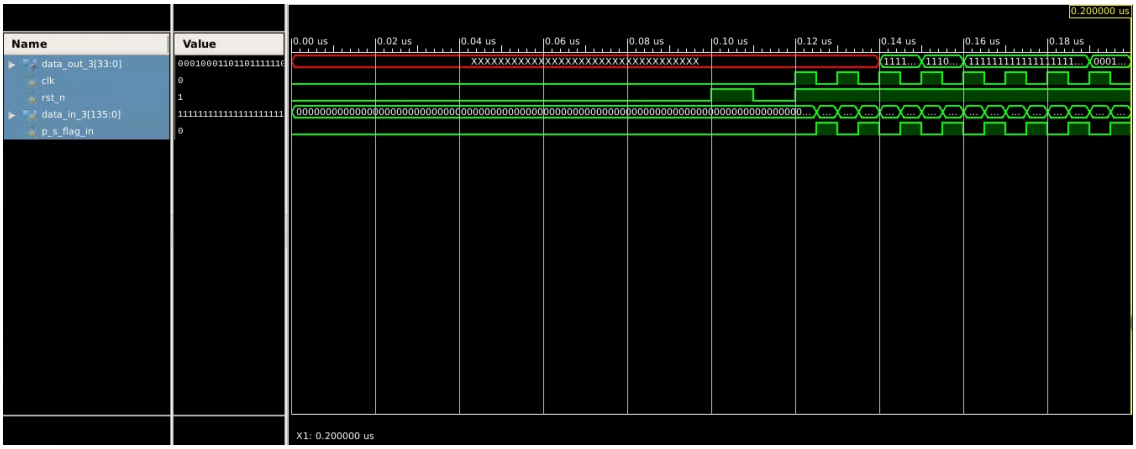
```
25  module p_s_tb;
26
27      // Inputs
28      reg clk;
29      reg rst_n;
30      reg [135:0] data_in_3;
31      reg p_s_flag_in;
32
33      // Outputs
34      wire [33:0] data_out_3;
35
36      // Instantiate the Unit Under Test (UUT)
37      p_s uut (
38          .clk(clk),
39          .rst_n(rst_n),
40          .data_in_3(data_in_3),
41          .p_s_flag_in(p_s_flag_in),
42          .data_out_3(data_out_3)
43      );
```

```
45      initial begin
46          // Initialize Inputs
47          clk = 0;
48          rst_n = 0;
49          data_in_3 = 0;
50          p_s_flag_in = 0;
51
52          // Wait 100 ns for global reset to finish
53          #100;
54
55          // Add stimulus here
56          // Reset sequence
57          rst_n = 1;
58          #10;
59          rst_n = 0;
60          #10;
61          rst_n = 1;
```

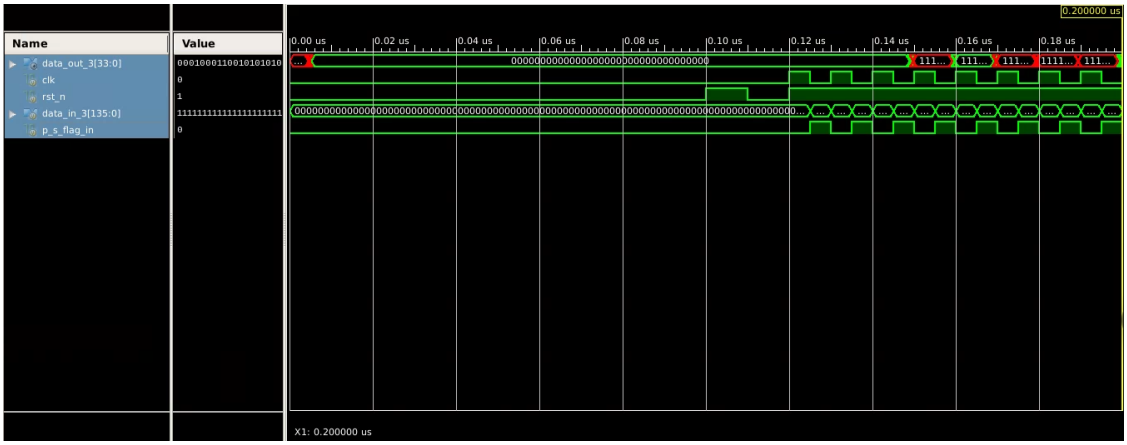
```
64          repeat (16) begin
65              // Toggle clock
66              clk = ~clk;
67              #5;
68
69              // Assign data_in_3 with some test data
70              data_in_3 = $random;
71
72              // Toggle p_s_flag_in every other clock cycle
73              p_s_flag_in = clk;
74
75              // Display input and output for debugging
76              $display("Time=%t, clk=%b, rst_n=%b, data_in_3=%h, p_s_flag_in=%b,
77          end
78
79          // End simulation
80          $finish;
81      end
82
83  endmodule
```

```
data_out_3=%h", $time, clk, rst_n, data_in_3, p_s_flag_in, data_out_3);
```

Behavioral:



Post-Route:



- ctrl

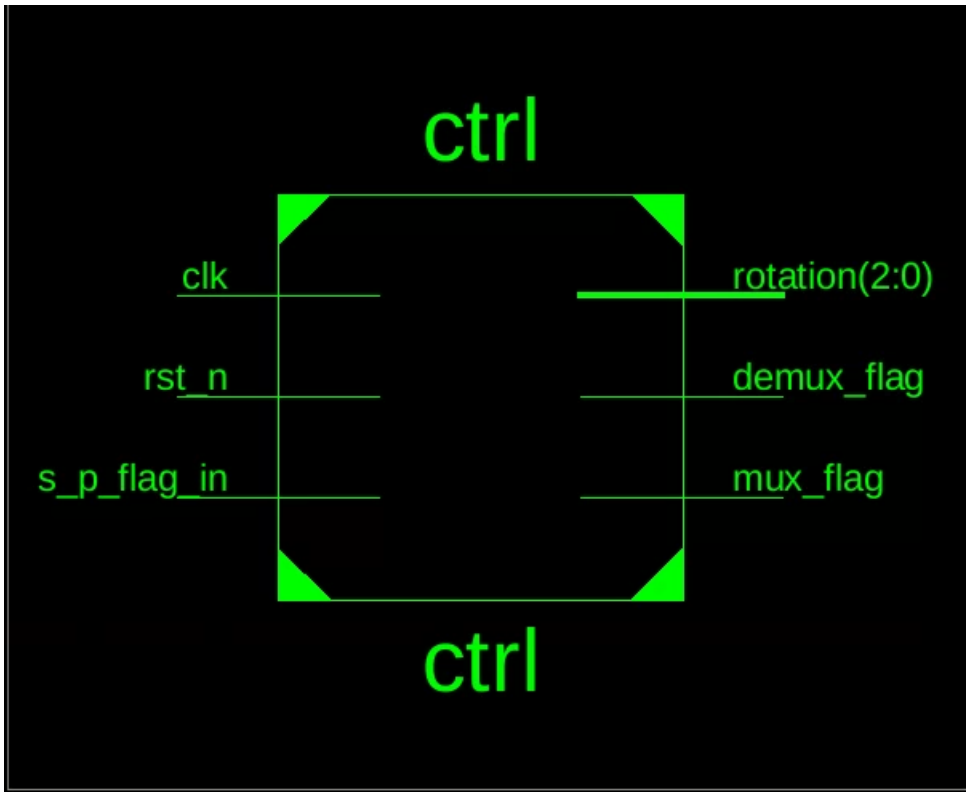
Verilog code:

```
1  module ctrl(
2      input  wire       clk,
3      input  wire       rst_n,
4      input  wire       s_p_flag_in,
5      output reg        mux_flag,
6      output reg  [2:0] rotation,
7      output reg        demux_flag
8      );
9
10     parameter STOP = 3'b0;
11     parameter MUX_IDLE = 1'b0;
12     parameter ROT_IDLE = 3'b0;
13     parameter DEMUX_IDLE = 1'b0;
14     parameter S_P_SEL_0 = 1'b0;
15     parameter S_P_SEL_1 = 1'b0;
16     parameter S_P_SEL_2 = 1'b0;
17     parameter S_P_SEL_3 = 1'b0;
18     parameter REG_SEL_0 = 1'b1;
19     parameter REG_SEL_1 = 1'b1;
20     parameter REG_SEL_2 = 1'b1;
21     parameter REG_SEL_3 = 1'b1;
22     parameter P_S_SEL_0 = 1'b0;
23     parameter P_S_SEL_1 = 1'b0;
```

```
23    parameter P_S_SEL_1 = 1'b0;
24    parameter P_S_SEL_2 = 1'b0;
25    parameter P_S_SEL_3 = 1'b0;
26
27    reg[2:0] core_tick;  // count how many sequential logic
28
29  // This always part controls signal core_tick.
30    always @ ( posedge clk or negedge rst_n) begin
31      if ( !rst_n )
32        core_tick <= STOP; // 3 bits
33      else if ( core_tick == STOP ) begin
34        case ( s_p_flag_in )
35          1'b0: core_tick <= STOP;
36          1'b1: core_tick <= core_tick + 1;
37        endcase
38      end else
39        core_tick <= core_tick + 1;
40    end
41
42  // This always part controls signal mux_flag.
43    always @ ( posedge clk or negedge rst_n ) begin
44      if ( !rst_n )
45        mux_flag <= MUX_IDLE;
46      else begin
47        case ( core_tick )
48          3'b000: mux_flag <= S_P_SEL_0;  // Com
49          3'b001: mux_flag <= S_P_SEL_1;  // 15
50          3'b010: mux_flag <= S_P_SEL_2;  // 16
51          3'b011: mux_flag <= S_P_SEL_3;  // 17
52          3'b100: mux_flag <= REG_SEL_0;  // 18
53          3'b101: mux_flag <= REG_SEL_1;  // 19
54          3'b110: mux_flag <= REG_SEL_2;  // 20
55          3'b111: mux_flag <= REG_SEL_3;  // 21
56          // If MUX/DEMUX has 3 OPTIONS (SP/PS,
57        endcase
58      end
59    end
60
61   // This always part controls signal demux_flag
62    always @ ( posedge clk or negedge rst_n ) be
63      if ( !rst_n )
64        demux_flag <= DEMUX_IDLE;
65      else begin
66        case ( core_tick )
```

```
67              3'b000: demux_flag <= REG_SEL_0;  // Cor
68              3'b001: demux_flag <= REG_SEL_1;  // 15
69              3'b010: demux_flag <= REG_SEL_2;  // 16
70              3'b011: demux_flag <= REG_SEL_3;  // 17
71              3'b100: demux_flag <= P_S_SEL_0;  // 18
72              3'b101: demux_flag <= P_S_SEL_1;  // 19
73              3'b110: demux_flag <= P_S_SEL_2;  // 20
74              3'b111: demux_flag <= P_S_SEL_3;  // 21
75              // If MUX/DEMUX has 3 OPTIONS (SP/PS, RI
76            endcase
77          end
78        end
79
80      // This always part controls signal rotation.
81      always @ ( posedge clk or negedge rst_n ) begin
82        if ( !rst_n )
83          rotation <= ROT_IDLE;
84        else
85          rotation <=core_tick;
86      end
87
88      endmodule
```

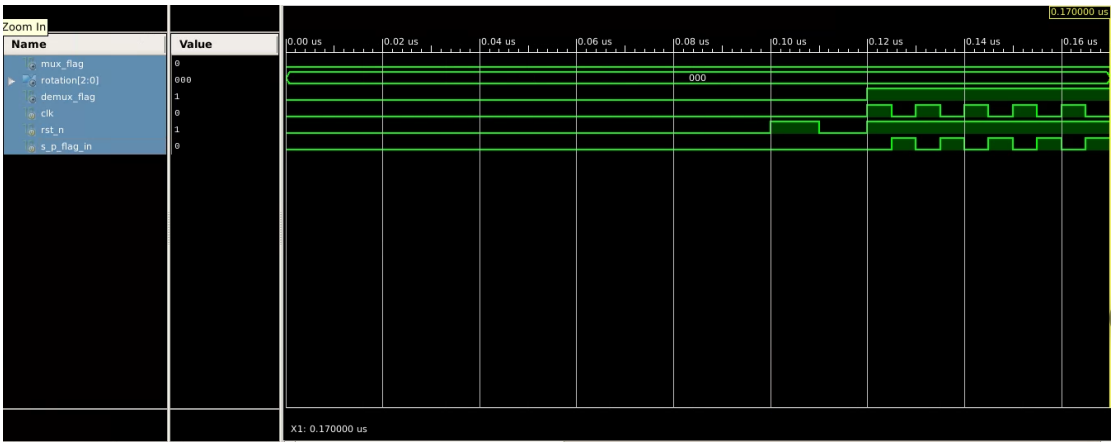
HDL:



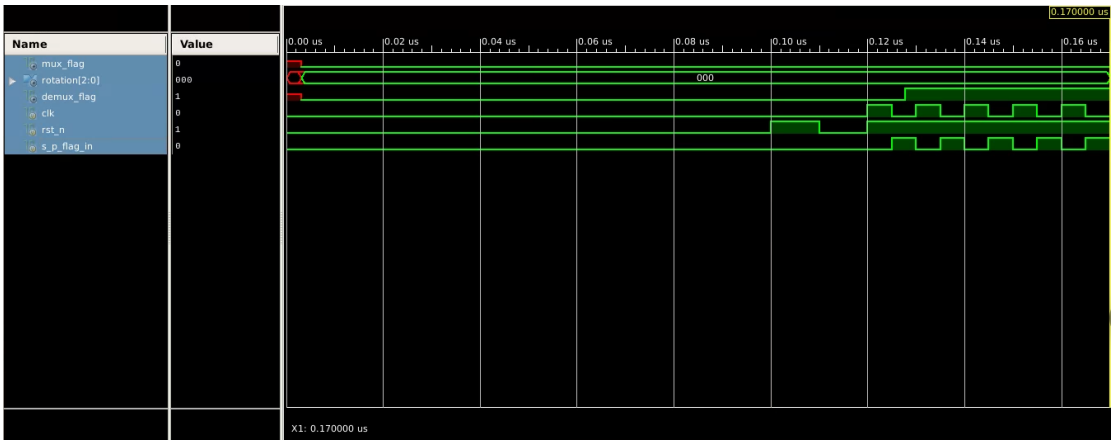
Testbench:

```
25   module counter_tb;
26
27   // Inputs
28   reg clk;
29   reg rst_n;
30
31   // Outputs
32   wire [3:0] counter;
33
34   // Instantiate the Unit Under Test (UUT)
35   counter uut (
36       .clk(clk),
37       .rst_n(rst_n),
38       .counter(counter)
39   );
40
41   initial begin
42       // Initialize Inputs
43       clk = 0;
44       rst_n = 0;
47       #100;
48
49       // Reset sequence
50       rst_n = 1;
51       #10;
52       rst_n = 0;
53       #10;
54       rst_n = 1;
55
56       // Test counter increment
57       repeat (20) begin
58           #10;
59           $display("Time=%t, counter=%d", $time, counter);
60       end
61
62       // End simulation
63       $finish;
64   end
65
66   // Clock generation
67   always #5 clk = ~clk;
```

Behavioral:



Post-Route:



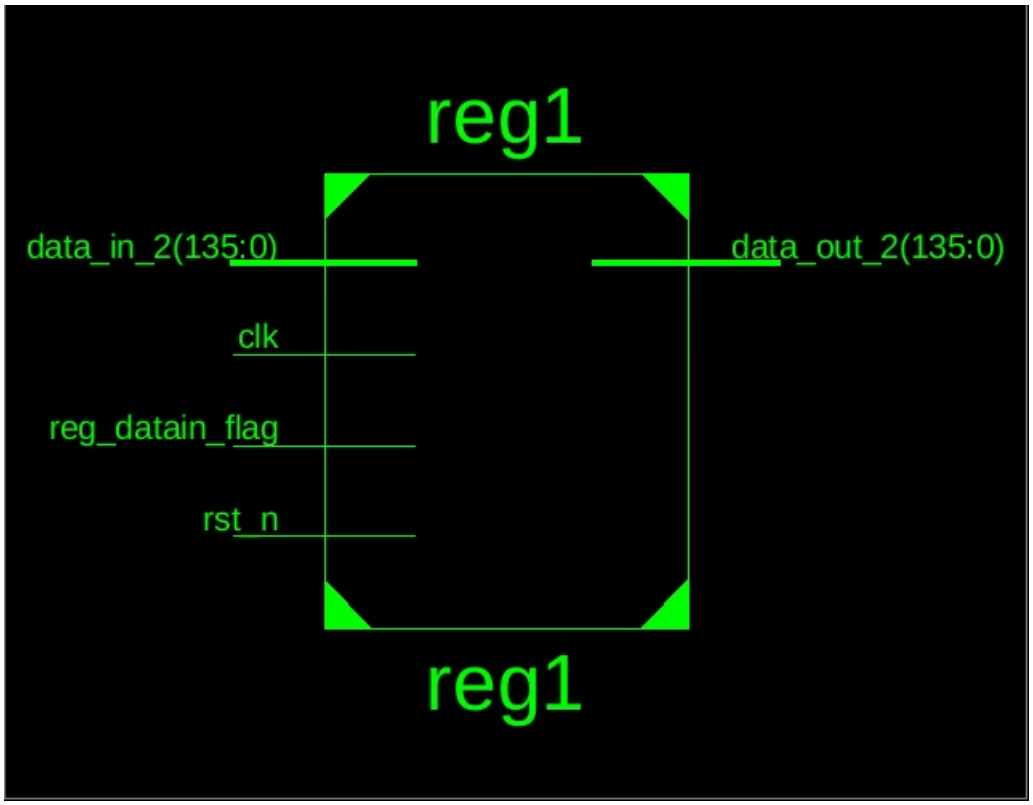
- Buffer storage

Verilog code:

```
module reg1(
  input wire clk,
  input wire rst_n,
  input wire [135:0] data_in_2,
  input wire reg_datain_flag,
  output reg [135:0] data_out_2
  );
  reg reg_flag_mux;  // control

  reg [33:0] R0;
  reg [33:0] R1;
  reg [33:0] R2;
  reg [33:0] R3;
  reg [33:0] R4;
  reg [33:0] R5;
  reg [33:0] R6;
  reg [33:0] R7;
  reg [33:0] R8;
  reg [33:0] R9;
  reg [33:0] R10;
  reg [33:0] R11;
  reg [33:0] R12;
  reg [33:0] R13;
  reg [33:0] R14;
  reg [33:0] R15;

  reg [1:0] counter1;
  reg [1:0] counter2;

// This always part controls signal
  always @ ( posedge clk ) begin
    if ( !rst_n )
      counter1 <= 2'b00;
    else if ( reg_datain_flag )
      counter1 <= counter1 + 1'b1;
  end

// This always part controls regist
  always @ ( posedge clk ) begin
    if ( reg_datain_flag )begin
      case ( counter1 )
        2'b01: begin
          R0  <= data_in_2[33:0];
          R1  <= data_in_2[67:34];
          R2  <= data_in_2[101:68];
          R3  <= data_in_2[135:102];
        end
        2'b10: begin
          R4  <= data_in_2[33:0];
          R5  <= data_in_2[67:34];
          R6  <= data_in_2[101:68];
          R7  <= data_in_2[135:102];
        end
        2'b11: begin
          R8  <= data_in_2[33:0];
          R9  <= data_in_2[67:34];
          R10 <= data_in_2[101:68];
          R11 <= data_in_2[135:102];
        end
        2'b00: begin
          R12 <= data_in_2[33:0];
          R13 <= data_in_2[67:34];
          R14 <= data_in_2[101:68];
          R15 <= data_in_2[135:102];
        end
      endcase
    end

  always @ ( posedge clk ) begin
    if ( !rst_n )
      reg_flag_mux <= 1'b0;
    else if ( counter1 == 2'b11 )
      reg_flag_mux <= 1'b1;
    else if ( counter2 == 2'b11 )
      reg_flag_mux <= 1'b0;
  end

// This always part controls signal
  always @ ( posedge clk ) begin
    if ( !rst_n )
      counter2 <= 2'b00;
    else if ( reg_flag_mux )
      counter2 <= counter2 + 2'b01;
  end

```

```
89      always @ ( posedge clk ) begin
90        if ( reg_flag_mux ) begin
91          case ( counter2 )
92            2'b00: data_out_2 <= {R12,  R8,  R4,  R0};
93            2'b01: data_out_2 <= {R13,  R9,  R5,  R1};
94            2'b10: data_out_2 <= {R14,  R10, R6,  R2};
95            2'b11: data_out_2 <= {R15,  R11, R7,  R3};
96        endcase
97        end
98      end
99
100   endmodule
```

HDL:

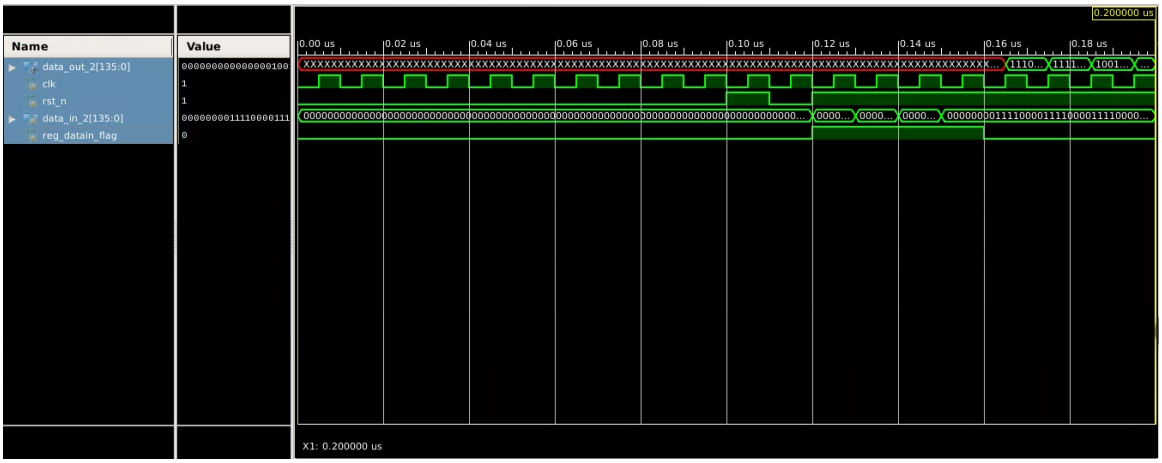


Testbench:

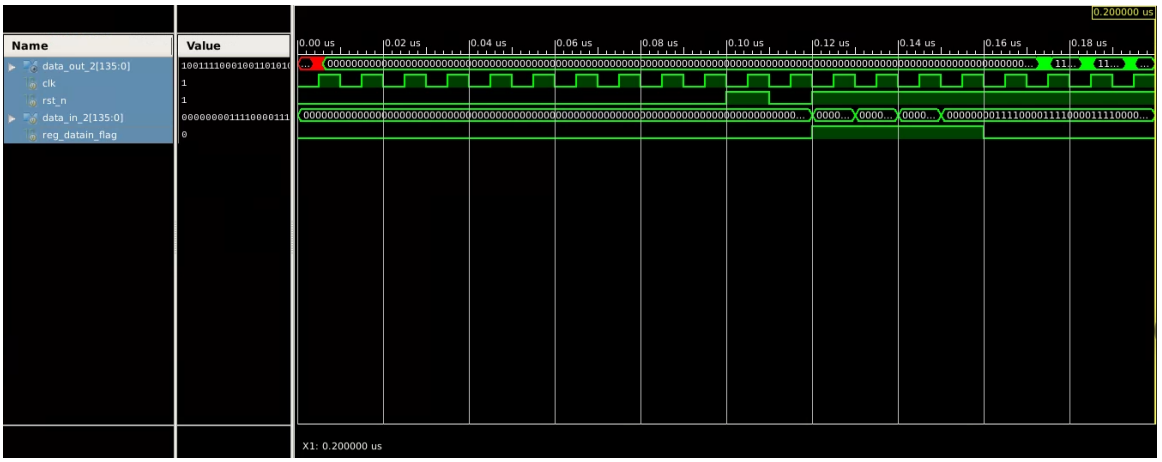
```
25  module reg1_tb;
26
27  // Inputs
28  reg clk;
29  reg rst_n;
30  reg [135:0] data_in_2;
31  reg reg_datain_flag;
32
33  // Outputs
34  wire [135:0] data_out_2;
35
36  // Instantiate the Unit Under Test (UUT)
37  reg1 uut (
38      .clk(clk),
39      .rst_n(rst_n),
40      .data_in_2(data_in_2),
41      .reg_datain_flag(reg_datain_flag),
42      .data_out_2(data_out_2)
43  );
```

```
45  initial begin
46      // Initialize Inputs
47      clk = 0;
48      rst_n = 0;
49      data_in_2 = 0;
50      reg_datain_flag = 0;
51
52      // Wait 100 ns for g
53      #100;
54
55      // Reset sequence
56      rst_n = 1;
57      #10;
58      rst_n = 0;
59      #10;
60      rst_n = 1;
61
62      // Test case 1: Check
63      reg_datain_flag = 1;
64      data_in_2 = 136'h012
65      #10;
66      data_in_2 = 136'hfedcba9876543210fedcba9876543210;
67      #10;
68      data_in_2 = 136'h0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f;
69      #10;
70      data_in_2 = 136'hf0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0;
71      #10;
72      reg_datain_flag = 0;
73
74      // Check output data after loading
75      #40;
76      $display("Output data: %h", data_out_2);
77
78      // End simulation
79      $finish;
80  end
81
82  always #5 clk = ~clk;
83
84  endmodule
```

Behavioral:



Post-Route:



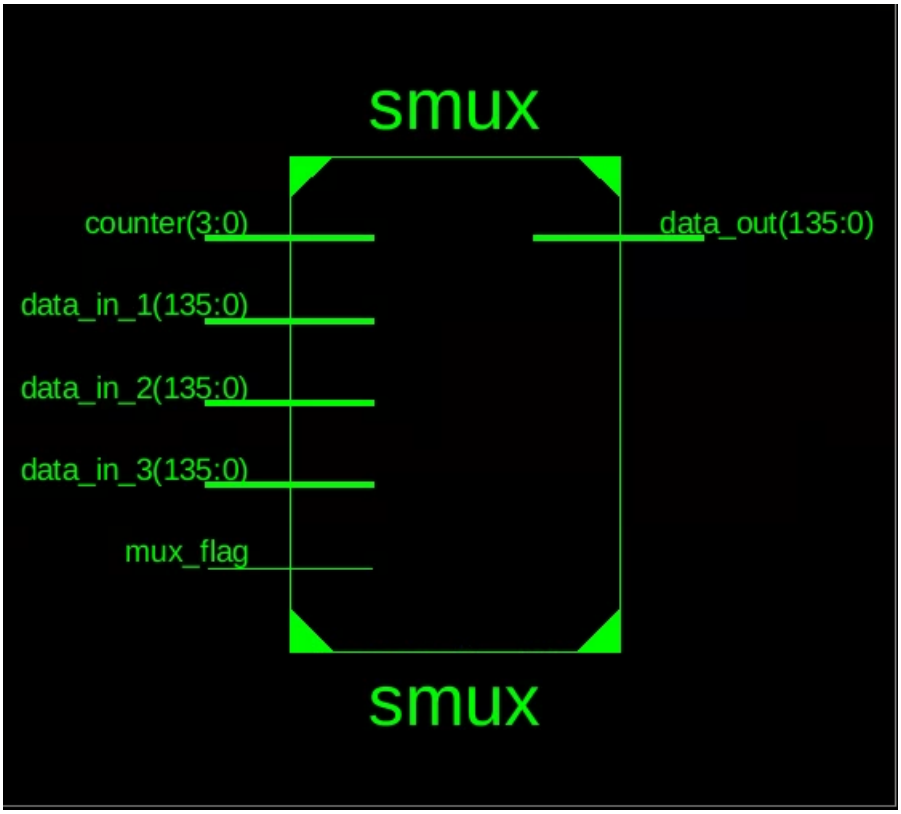
- Mux

1. Smux

Verilog code:

```
module smux(
    input mux_flag,
    input [135:0] data_in_1,
    input [135:0] data_in_2,
    input [135:0] data_in_3,
    input [3:0] scounter,
    output reg [135:0] data_out
);
reg [33:0] R1, R2, R3, R4;
    always @(*) begin
        case(scounter)
            2: data_out <= {R4, R3, R2, R1};
            14: R1 <= data_in_3[33:0];
            15: R2 <= data_in_3[33:0];
            0: R3 <= data_in_3[33:0];
            1: R4 <= data_in_3[33:0];
            default: data_out <= data_out;
        endcase
        data_out <= mux_flag ? data_in_1 : data_in_2;
    end

endmodule
```

HDL:

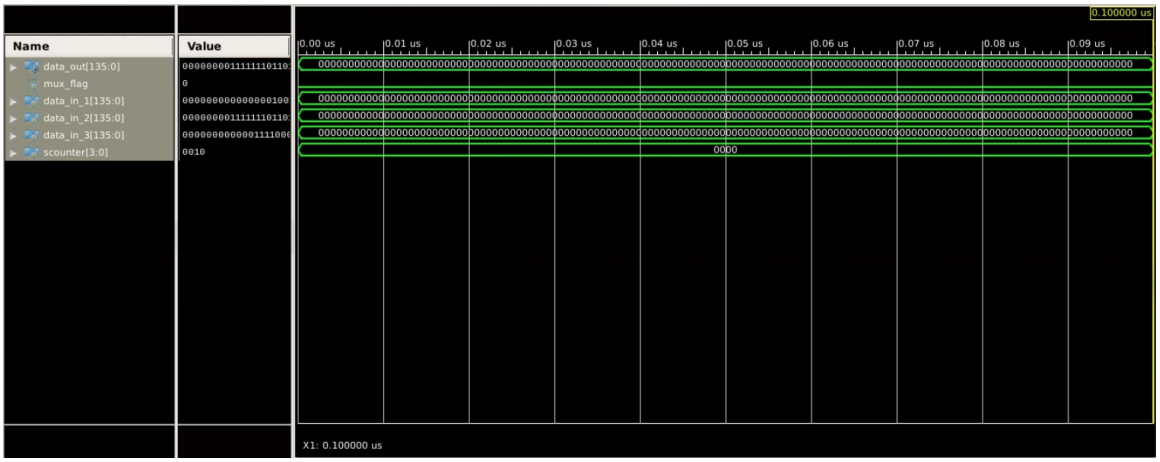


Testbench:

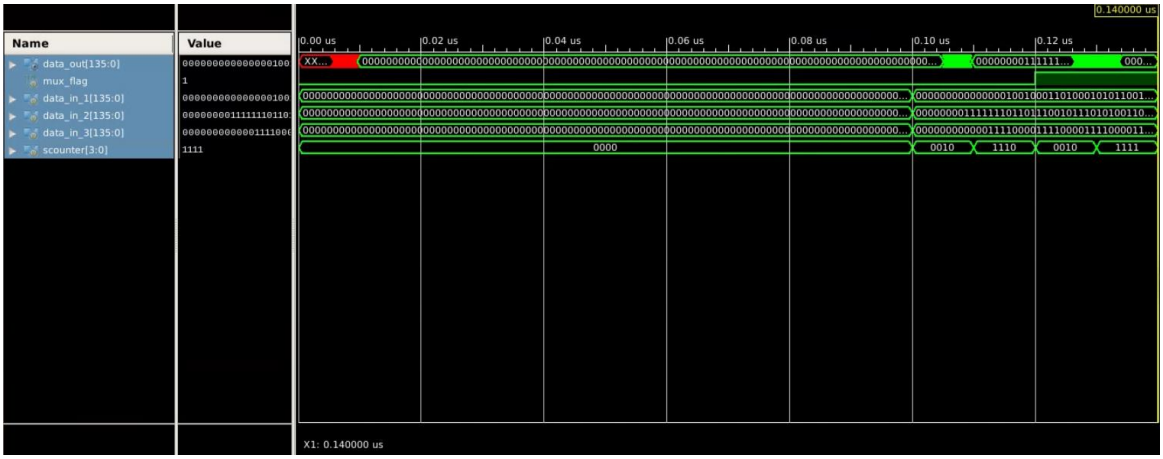
```
25  module smux_tb;
26
27  // Inputs
28  reg mux_flag;
29  reg [135:0] data_in_1;
30  reg [135:0] data_in_2;
31  reg [135:0] data_in_3;
32  reg [3:0] counter;
33
34  // Outputs
35  wire [135:0] data_out;
36
37  // Instantiate the Unit Under Test (UUT)
38  smux uut (
39      .mux_flag(mux_flag),
40      .data_in_1(data_in_1),
41      .data_in_2(data_in_2),
42      .data_in_3(data_in_3),
43      .counter(counter),
44      .data_out(data_out)
45  );
```

```
47  initial begin
48      // Initialize Inputs
49      mux_flag = 0;
50      data_in_1 = 0;
51      data_in_2 = 0;
52      data_in_3 = 0;
53      counter = 0;
54
55      // Wait 100 ns for global reset to finish
56      #100;
57
58      // Test case 1: mux_flag = 0, counter = 2
59      mux_flag = 0;
60      data_in_1 = 136'h0123456789abcdef0123456789abcdef;
61      data_in_2 = 136'hfedcba9876543210fedcba9876543210;
62      data_in_3 = 136'h0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f;
63      counter = 2;
64      #10;
65      $display("Test case 1: mux_flag = 0, counter = 2, data_out = %h", data_out);
66
67      // Test case 2: mux_flag = 0, counter = 14
68      counter = 14;
69      #10;
70      $display("Test case 2: mux_flag = 0, counter = 14, data_out = %h", data_out);
71
72      // Test case 3: mux_flag = 1, counter = 2
73      mux_flag = 1;
74      counter = 2;

75      #10;
76      $display("Test case 3: mux_flag = 1, counter = 2, data_out = %h", data_out);
77
78      // Test case 4: mux_flag = 1, counter = 15
79      counter = 15;
80      #10;
81      $display("Test case 4: mux_flag = 1, counter = 15, data_out = %h", data_out);
82
83      // End simulation
84      $finish;
85  end
86
87  endmodule
```

Behavioral:



Post-Route:

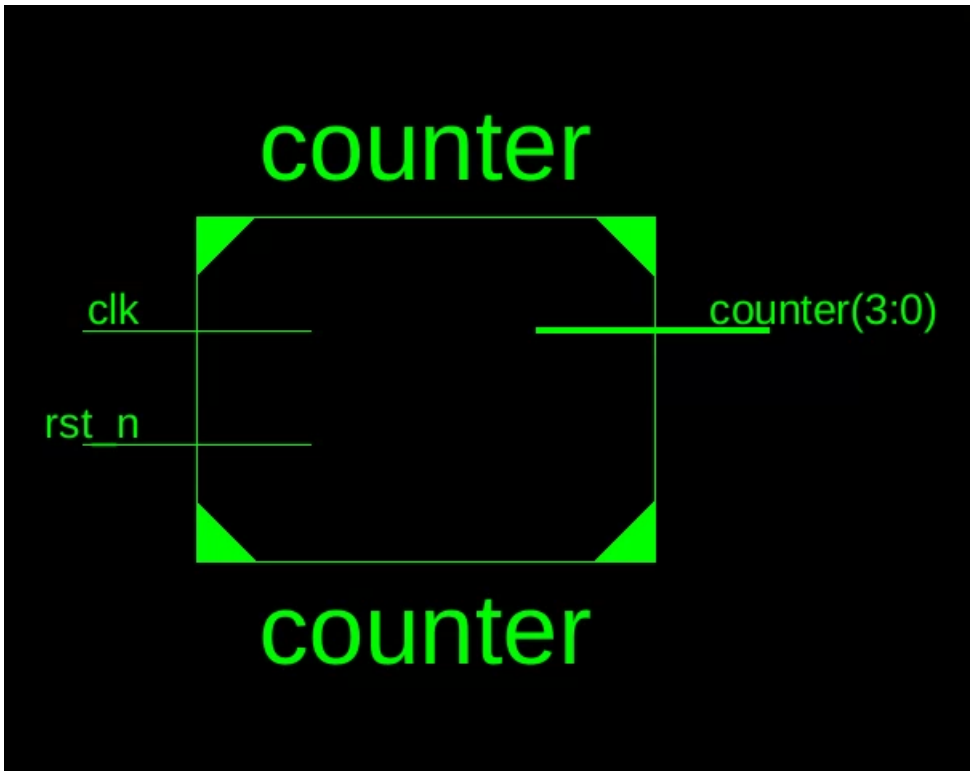


2. Counter

Verilog code:

```
1   module counter(
2       input clk,
3       input rst_n,
4       output reg [3:0] ccounter
5   );
6
7   always @(*) begin
8       if(!rst_n)
9           ccounter <= 0;
10      else
11          ccounter <= ccounter + 1;
12  end
13
14  endmodule
```

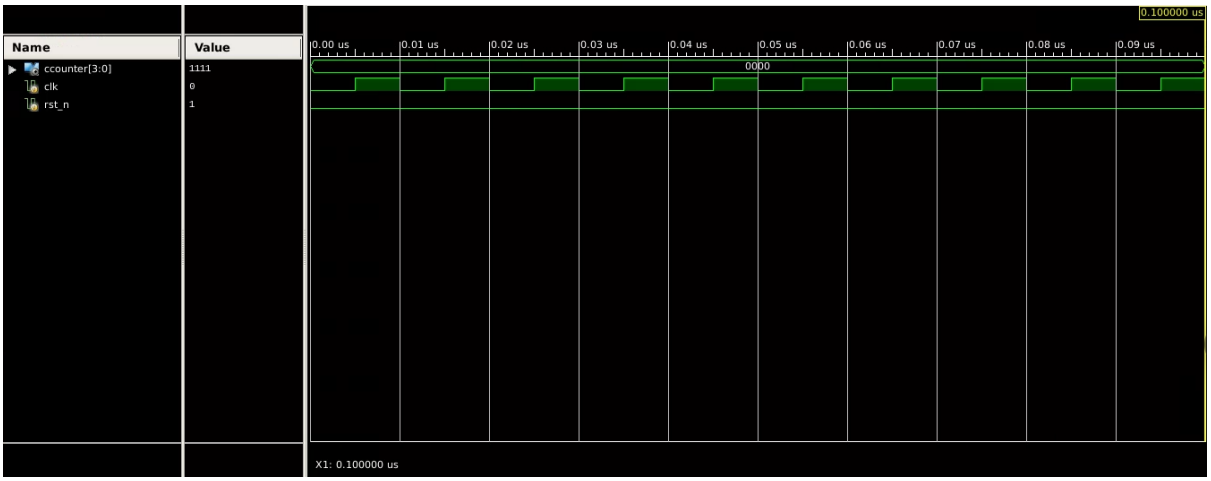
HDL:



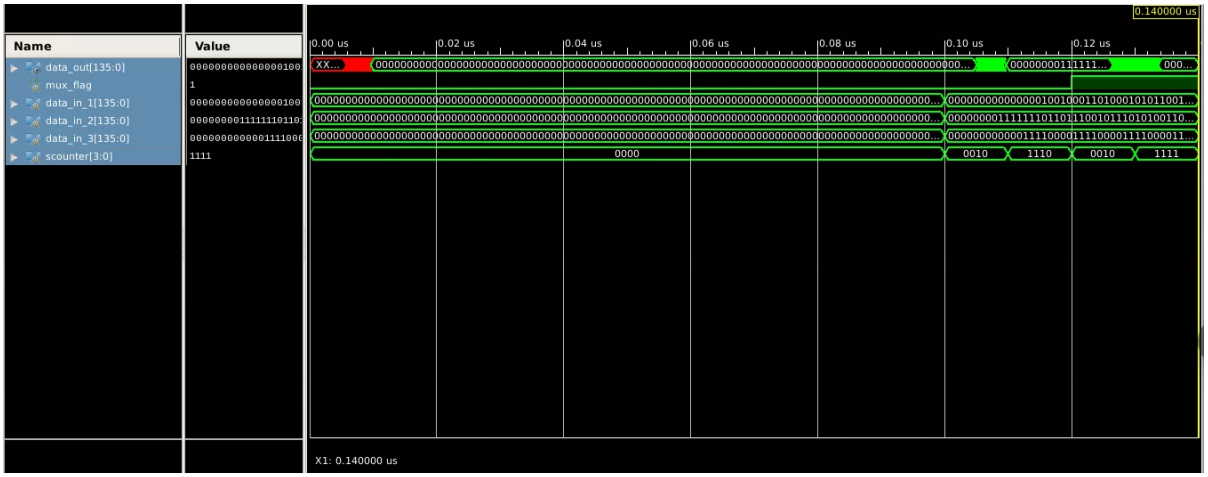
Testbench:

```
module counter_tb;

// Inputs
reg clk;
reg rst_n;

// Outputs
wire [3:0] counter;

// Instantiate the Un
counter uut (
    .clk(clk),
    .rst_n(rst_n),
    .counter(counter)
);

initial begin
    // Initialize Inp
    clk = 0;
    rst_n = 0;

    // Wait 100 ns fo
    #100;

    // Reset sequence
    rst_n = 1;
    #10;
    rst_n = 0;

    #10;
    rst_n = 1;

    // Test counter increment
    repeat (20) begin
        #10;
        $display("Time=%t, counter=%d", $time, counter);
    end

    // End simulation
    $finish;
end

// Clock generation
always #5 clk = ~clk;

endmodule
```

Behavioral:



Post-Route:

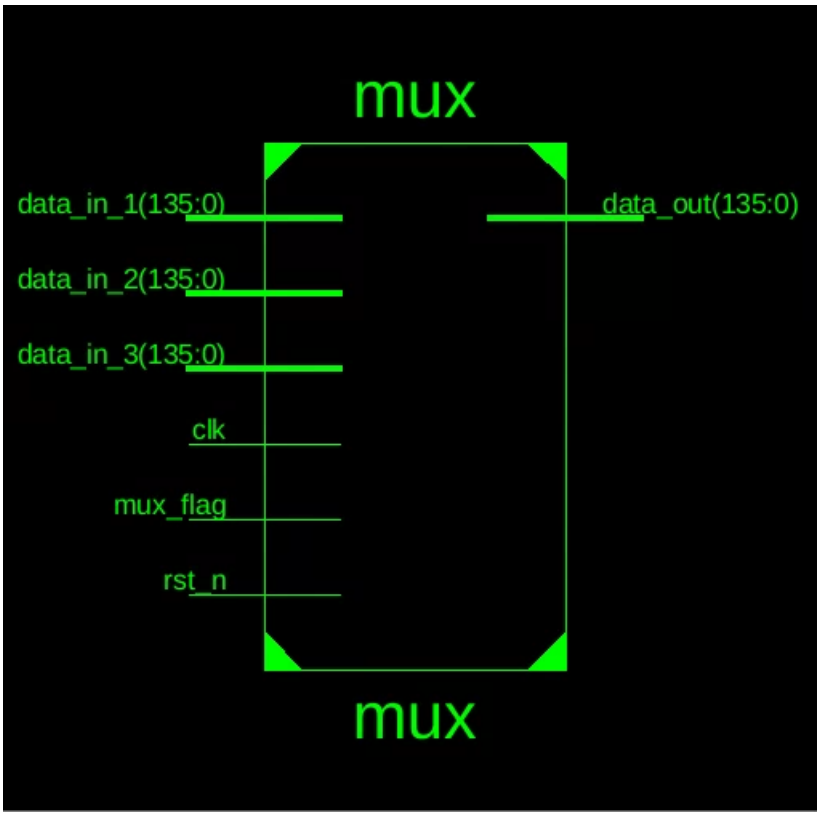


3. Mux

Verilog code:

```
21  module mux(
22      input mux_flag,
23      input clk,
24      input rst_n,
25      input [135:0] data_in_1,
26      input [135:0] data_in_2,
27      input [135:0] data_in_3,
28      output [135:0] data_out
29  );
30
31  wire [3:0] cscounter;
32
33  counter counter_inst (
34      .clk(clk),
35      .rst_n(rst_n),
36      .ccounter(cscounter)
37  );
38
39
40  smux mux_inst (
41      .mux_flag(mux_flag),
42      .data_in_1(data_in_1),
43      .data_in_2(data_in_2),
44      .data_in_3(data_in_3),
45      .scounter(cscounter),
46      .data_out(data_out)
47  );
```

HDL:

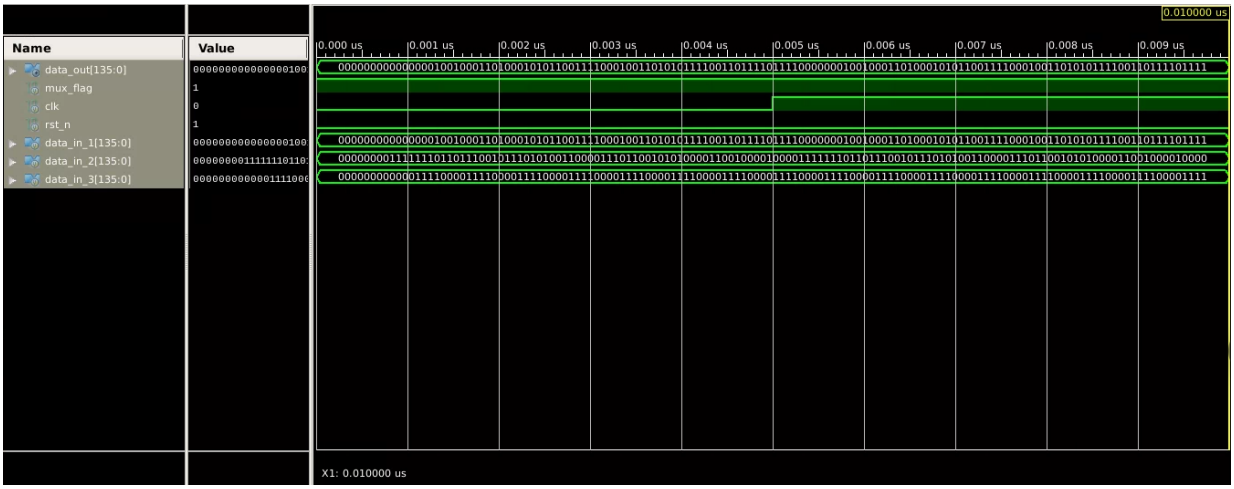


Testbench:

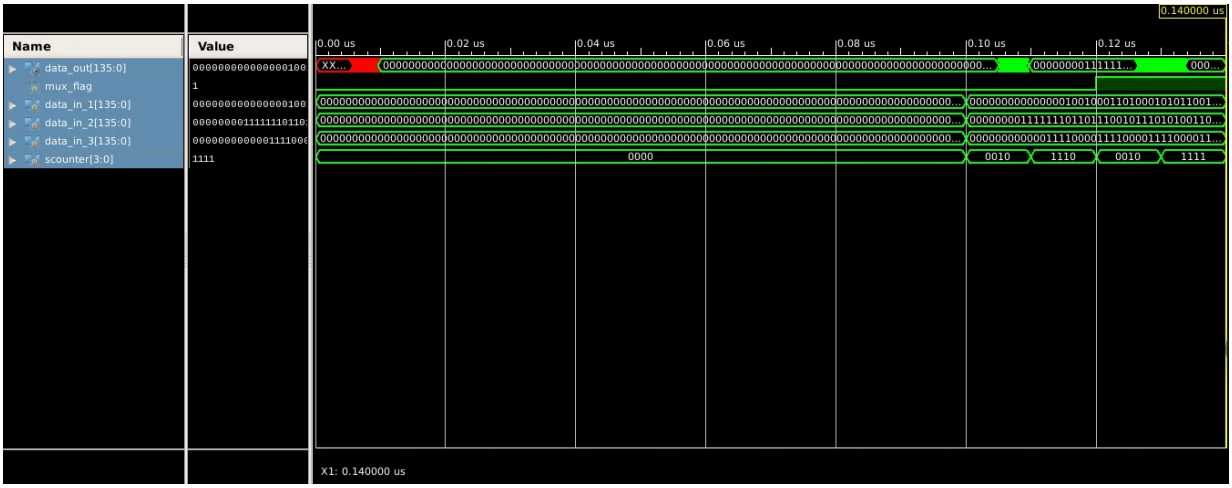
```
1   `timescale 1ns / 1ps
2
3   module mux_tb;
4
5       // Inputs
6       reg mux_flag_tb;
7       reg clk_tb;
8       reg rst_n_tb;
9       reg [135:0] data_in_1_tb;
10      reg [135:0] data_in_2_tb;
11      reg [135:0] data_in_3_tb;
12
13      // Outputs
14      wire [135:0] data_out_tb;
15
16      // Instantiate the Unit Under
17      mux uut (
18          .mux_flag(mux_flag_tb),
19          .clk(clk_tb),
20          .rst_n(rst_n_tb),
21          .data_in_1(data_in_1_tb),
22          .data_in_2(data_in_2_tb),
23          .data_in_3(data_in_3_tb),
24          .data_out(data_out_tb)
25      );
28      always begin
29          clk_tb = 0;
30          #5;
31          clk_tb = 1;
32          #5;
33      end
34
35      // Reset generation
36      initial begin
37          rst_n_tb = 0;
38          #10;
39          rst_n_tb = 1;
40          #10;
41          rst_n_tb = 0;
42          #10;
43          rst_n_tb = 1;
44          #10;
45          $finish;
46      end
```

```
49      initial begin
50          mux_flag_tb = 0;
51          data_in_1_tb = 136'h0123456789abcdef0123456789abcdef;
52          data_in_2_tb = 136'hfedcba9876543210fedcba9876543210;
53          data_in_3_tb = 136'h0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f;
54          #200;
55          $display("Test case 1: mux_flag = 0, data_out = %h", data_out_tb);
56      end
57
58      // Test case 2: mux_flag = 1
59      initial begin
60          mux_flag_tb = 1;
61          #200;
62          $display("Test case 2: mux_flag = 1, data_out = %h", data_out_tb);
63      end
64
65  endmodule
```

Behavioral:



Post-Route:



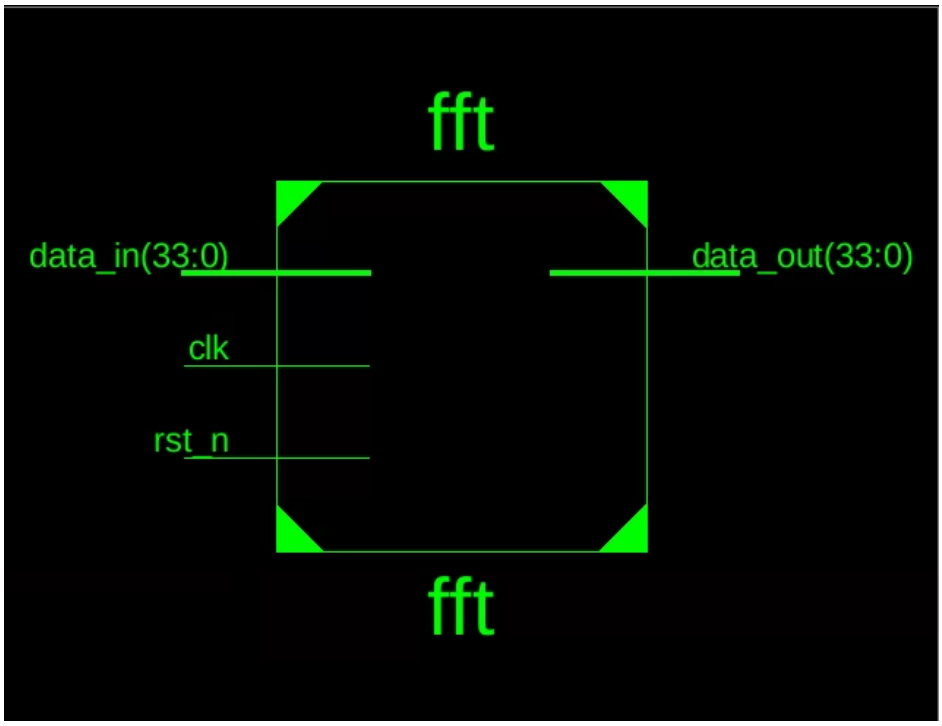
- FFT

  Verilog code:

```
module fft(

    input  wire         clk,        // clock
    input  wire         rst_n,      // reset
    input  wire [33:0] data_in,    // input from
    output [33:0] data_out    // output to pin

    );

    wire         mux_flag, demux_flag, s_p_flag;
    wire[2:0]    rotation;
    wire[135:0] data_1, data_2, data_3, data_4;

    ctrl ctrl0(
      .clk(clk),                  // input from t
      .rst_n(rst_n),              // input from t
      .s_p_flag_in(s_p_flag),     // input from s
      .mux_flag(mux_flag),        // output to mu
      .rotation(rotation),        // output to bu
      .demux_flag(demux_flag)     // output to p_
    );

    s_p s_p0(
      .clk(clk),                  // input from t
      .rst_n(rst_n),              // input from t
      .data_in_1(data_in),        // input from t
      .data_out_1(data_1),        // output to mu
      .s_p_flag_out(s_p_flag)     // output to ct
    );
```

```
31    mux mux0(
32      .mux_flag(mux_flag),       //
33      .clk(clk),
34      .rst_n(rst_n),
35      .data_in_2(data_1),        //
36      .data_in_1(data_2),        //
37      .data_in_3(data_4),
38      .data_out(data_3)          //
39    );
40
41    butterfly butterfly0(
42      .calc_in(data_3),          //
43      .rotation(rotation),       //
44      .calc_out(data_4)          //
45    );
46
47    reg1 reg10(
48      .clk(clk),                 //
49      .rst_n(rst_n),             //
50      .data_in_2(data_4),
51      .data_out_2(data_2),
52      .reg_datain_flag(demux_flag)
53    );
55    p_s p_s0(
56      .clk(clk),                 ,
57      .rst_n(rst_n),             ,
58      .p_s_flag_in(demux_flag),  ,
59      .data_in_3(data_4),        ,
60      .data_out_3(data_out)      ,
61    );
62
63  endmodule
```

HDL:

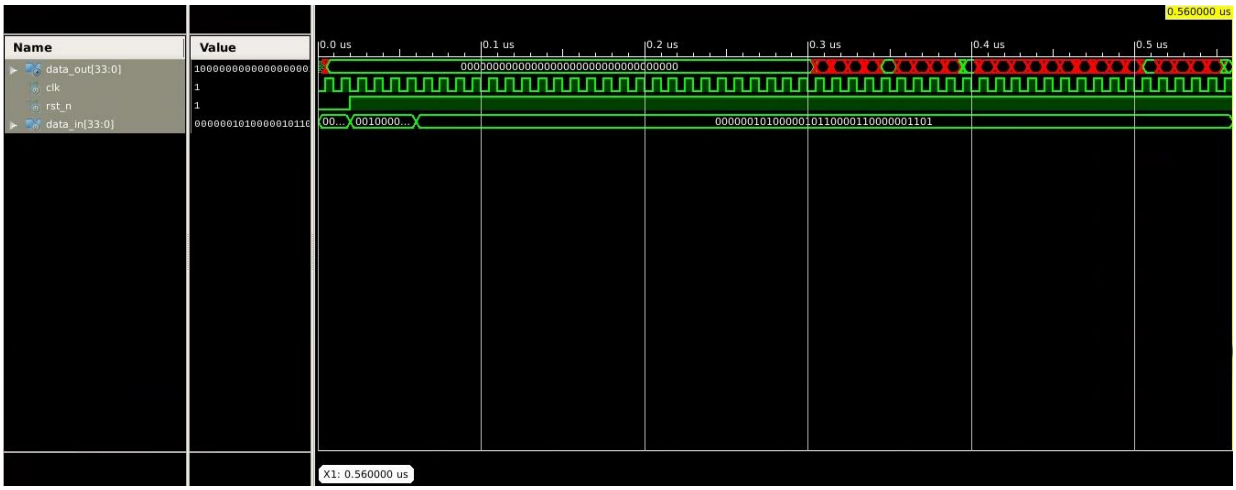


Testbench:

```
1   `timescale 1ns / 1ps
2
3   module fft_tb;
4
5       // Inputs
6       reg clk;
7       reg rst_n;
8       reg [33:0] data_in;
9
10      // Outputs
11      wire [33:0] data_out;
12
13      // Instantiate the Unit
14      fft uut (
15          .clk(clk),
16          .rst_n(rst_n),
17          .data_in(data_in),
18          .data_out(data_out)
19      );
20
21      // Clock generation
22      always begin
23          clk = 0;
24          #5;
25          clk = 1;
26          #5;
27      end
29      // Reset generation
30      initial begin
31          rst_n = 0;
32          #10;
33          rst_n = 1;
34          #10;
35          // Add more delay or te
36          $finish;
37      end
38
39      // Stimulus generation
40      initial begin
41          // Initialize Inputs
42          data_in = 32'h12345678;
43
44          // Apply stimulus
45          #20; // Wait for some t
46          data_in = 32'h87654321;
47
48          // Add more stimulus as
49
50          // End simulation
51          #100;
52          $finish;
53      end
54
55  endmodule
```

Behavioral:

Post-Route:



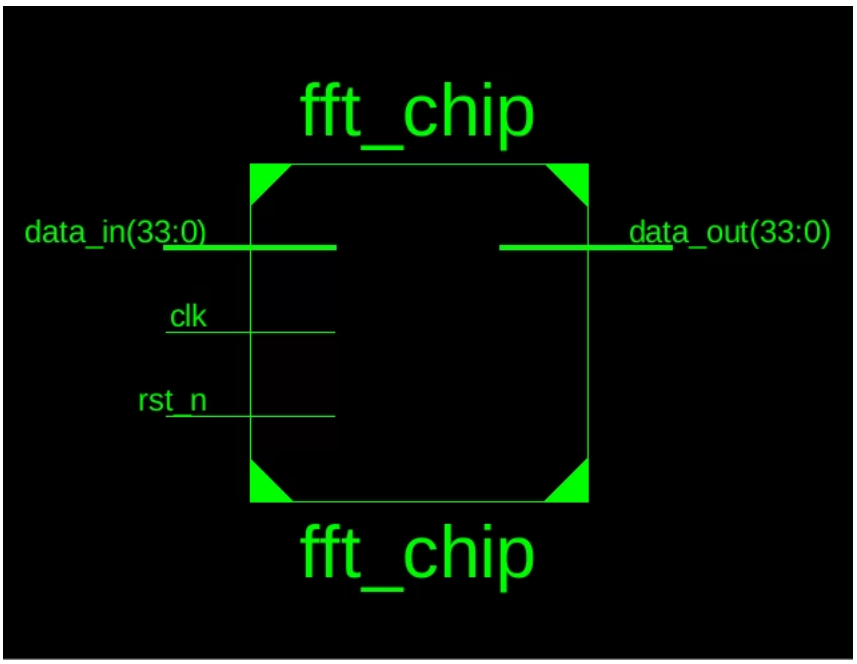
- FFT Chip

Verilog code:

```
module fft_chip(
    input clk,
    input rst_n,
    input [33:0] data_in,
    output [33:0] data_out
);

wire clk_buf, rst_n_buf;
wire [33:0] data_in_buf, data_out_buf;

// Input buffers
IBUF clk_ibuf (.O(clk_buf), .I(clk));
IBUF rst_n_ibuf (.O(rst_n_buf), .I(rst_n));
genvar i;
generate
    for (i = 0; i < 34; i = i + 1) begin
        IBUF data_in_ibuf (.O(data_in_buf[i]), .I(data_in[i]));
    end
endgenerate

// Output buffers
genvar j;
generate
    for (j = 0; j < 34; j = j + 1) begin
        OBUF data_out_obuf (.O(data_out[j]), .I(data_out_buf[j]));
    end
endgenerate
```

```
  fft inst_fft (
      .clk(clk_buf),
      .rst_n(rst_n_buf),
      .data_in(data_in_buf),
      .data_out(data_out_buf)
  );

  endmodule
```

HDL:

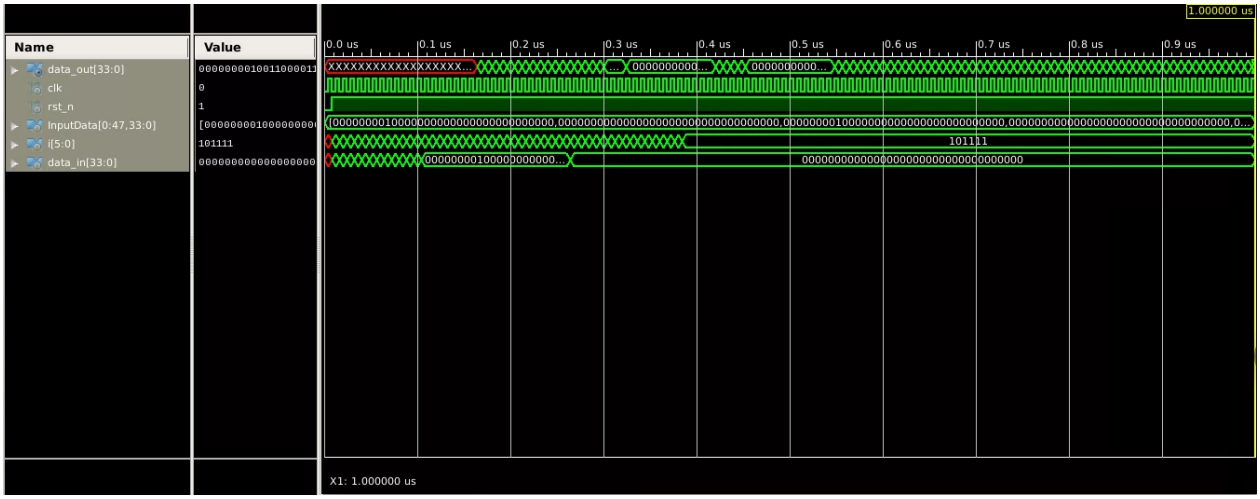


Testbench:

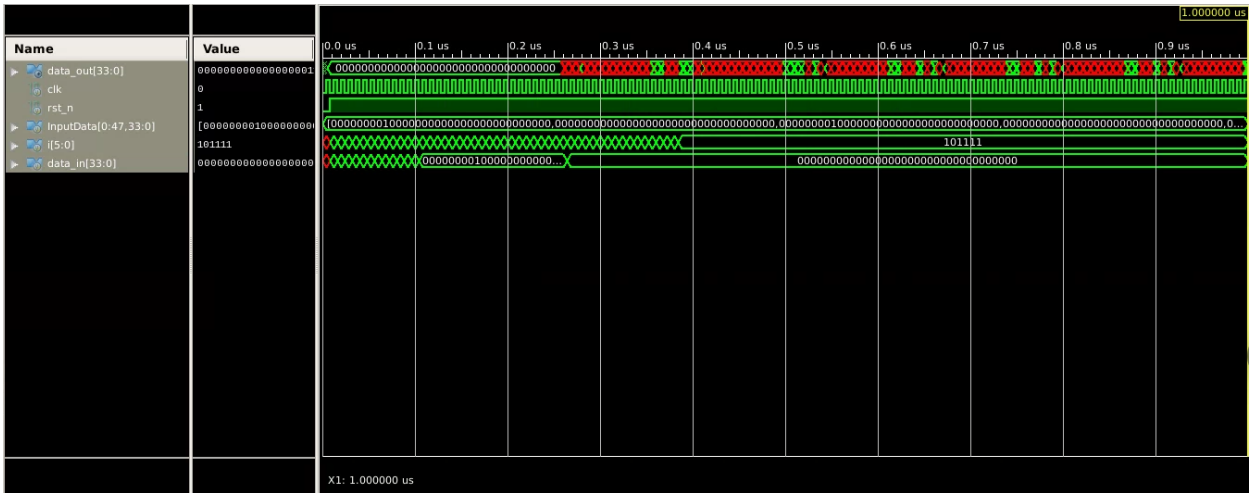
```
25  module fft_chip_tb;
26
27      // Inputs
28      reg clk;
29      reg rst_n;
30      reg [33:0] InputData [0:
31      reg [5:0] i;
32      reg [33:0] data_in;
33
34      // Outputs
35      wire [33:0] data_out;
36
37      // Instantiate the Unit
38      fft_chip uut (
39          .clk(clk),
40          .rst_n(rst_n),
41          .data_in(data_in),
42          .data_out(data_out)
43      );
44
45      initial begin
46          clk = 1'b0;
47          forever #4 clk = ~clk;
48      end
```

```
50      initial begin
51          rst_n = 0;
52          $display("\nLoad Data\n");
53          $readmemb("/home/ise/NTUST_Advanced_Computer_Algorithms/fftchip/Data_Input.txt", InputData);
54
55         #8 begin
56          rst_n = 1'b1;
57          for (i = 0; i < 47; i = i + 1) begin
58            data_in = InputData[i];
59            #8
60            $display("[%d] %b_%b_%b %b_%b_%b\n", (i+13)%16, data_out[33], data_out[32:25], data_out[24:17],
61             data_out[16], data_out[15:8], data_out[7:0]);
62          end
63        end
64      end
65
66  endmodule
```

Behavioral:



Post-Route:



4. Discussion:

1.本次作業的主要目的是探討如何調用IP核，我選擇的是FFT。雖然四個主題都非常有趣，但我認為FFT對未來的幫助最大，因此選擇了它作為我的題目。我覺得這是一個非常有意義的專案，使我能夠將上學期在FPGA課程中學到的知識應用到這個專案中。這次的專案不僅鞏固了我對FPGA的基礎知識，還讓我有機會實際操作和應用先進的演算法。

2.最主要的問題是理解算法。由於這個算法是radix-4 FFT，是一種快速DFT的方法之一，而課堂上沒有學過這種改良的方法，所以我需要花一些時間來學習和理解它。這個過程涉及到對數學理論的深入研究，以及對演算法步驟的詳細理解。在理解這個改良過的算法時，我查閱了許多資料，並且觀看了一些相關的教學視頻，這讓我對FFT有了更全面的認識。

3.其次是在模組調用方面，需要理解這個晶片的設計原理以及如何進行模組間的連接，才能精確地使用這個算法。在這個過程中，我學習了如何分析和理解不同模組的功能和作用，並且掌握了如何通過正確的接口將它們連接起來。

這不僅增強了我對硬體設計的理解，也提高了我在實際應用中解決問題的能力。整個過程充滿挑戰，但也非常有成就感，因為我能夠看到自己從理論學習到實踐操作的整個過程。

5. Conclusion

- Performance

1.Estimation

FFT operation

$$\frac{1s}{16\ x\ 7.4\ ns} = 8.445946\ x\ 10^6$$

Average Bandwidth

$$135\ MHz\ x\ \frac{34\ bits}{9} = 510.51\ MB/S$$

Pipeline throughput

$$\frac{1s}{16\ x\ 7.4\ ns}\ x\ 16\ x\ (17 + 17) = 4.5946\ x\ 10^9\ bps$$

2.Timing Constraints

	^	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1	Yes	Autotimespec constraint for clock net clk_buf_BUFG	SETUP HOLD	0.043ns	12.155ns	0	0 0
2	Yes	Autotimespec constraint for clock net inst_fft/mux0/GND_7_o_GND_7_o_equal_12_o_BUFG	SETUP HOLD	2.066ns	11.644ns	0	0 0
3	Yes	Autotimespec constraint for clock net inst_fft/mux0/GND_7_o_GND_7_o_equal_10_o_BUFG	SETUP HOLD	2.003ns	10.393ns	0	0 0
4	Yes	Autotimespec constraint for clock net inst_fft/mux0/GND_7_o_GND_7_o_equal_6_o_BUFG	SETUP HOLD	0.556ns	3.807ns	0	0 0
5	Yes	Autotimespec constraint for clock net inst_fft/mux0/GND_7_o_GND_7_o_equal_8_o_BUFG	SETUP HOLD	1.736ns	11.032ns	0	0 0

3..Timing results

Item	Time (ns)
clock clk (rise edge)	7.40
clock network delay (ideal)	4.30
clock uncertainty	-0.50
Clk slack(MET)	0.00

換算工作頻率為135 MHZ

- Power dissipation

  Power dissipation

Item	Voltage/ Power dissipation
Global Operating Voltage	1.62 V
Cell Internal Power	756.0848 mW
Net Switching Power	375.6310 mW
Total Dynamic Power	1.1317 W
Cell Leakage Power	28.5281 uW

- Hardware cost

1.Hardware cost

fft_chip Project Status (05/22/2024 - 09:43:47)			
Project File:	fftchip.xise	**Parser Errors:**	No Errors
Module Name:	fft_chip	**Implementation State:**	Placed and Routed
Target Device:	xc7vx330t-2ffg1157	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	464 Warnings (390 new)
Design Goal:	Balanced	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	All Constraints Met
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)

Device Utilization Summary [-]				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	2,551	408,000	1%	
Number used as Flip Flops	1,949			
Number used as Latches	136			
Number used as Latch-thrus	0			
Number used as AND/OR logics	466			
Number of Slice LUTs	3,789	204,000	1%	
Number used as logic	3,756	204,000	1%	
Number using O6 output only	2,870			
Number using O5 output only	22			
Number using O5 and O6	864			
Number used as ROM	0			
Number used as Memory	0	70,200	0%	
Number used exclusively as route-thrus	33			
Number with same-slice register load	24			
Number with same-slice carry load	9			
Number with other load	0			
Number of occupied Slices	1,796	51,000	3%	
Number of LUT Flip Flop pairs used	5,312			
Number with an unused Flip Flop	2,790	5,312	52%	
Number with an unused LUT	1,523	5,312	28%	
Number of fully used LUT-FF pairs	999	5,312	18%	
Number of unique control sets	34			
Number of slice register sites lost to control set restrictions	155	408,000	1%	
Number of bonded IOBs	70	600	11%	
Number of RAMB36E1/FIFO36E1s	0	750	0%	
Number of RAMB18E1/FIFO18E1s	0	1,500	0%	
Number of BUFG/BUFGCTRLs	5	32	15%	
Number used as BUFGs	5			
Number used as BUFGCTRLs	0			
Number of IDELAYE2/IDELAYE2_FINEDELAYs	0	700	0%	
Number of ILOGICE2/ILOGICE3/ISERDESE2s	0	700	0%	
Number of ODELAYE2/ODELAYE2_FINEDELAYs	0	650	0%	
Number of OLOGICE2/OLOGICE3/OSERDESE2s	0	700	0%	
Number of PHASER_IN/PHASER_IN_PHYs	0	56	0%	
Number of PHASER_OUT/PHASER_OUT_PHYs	0	56	0%	
Number of BSCANs	0	4	0%	
Number of BUFHCEs	0	168	0%	
Number of BUFRs	0	56	0%	
Number of CAPTUREs	0	1	0%	
Number of DNA_PORTs	0	1	0%	
Number of DSP48E1s	0	1,120	0%	
Number of EFUSE_USRs	0	1	0%	
Number of FRAME_ECCs	0	1	0%	
Number of GTHE2_CHANNELs	0	28	0%	
Number of GTHE2_COMMONs	0	7	0%	
Number of IBUFDS_GTE2s	0	14	0%	
Number of ICAPs	0	2	0%	
Number of IDELAYCTRLs	0	14	0%	
Number of IN_FIFOs	0	56	0%	
Number of MMCME2_ADVs	0	14	0%	
Number of OUT_FIFOs	0	56	0%	
Number of PCIE_3_0s	0	2	0%	
Number of PHASER_REFs	0	14	0%	
Number of PHY_CONTROLs	0	14	0%	
Number of PLLE2_ADVs	0	14	0%	
Number of STARTUPs	0	1	0%	
Number of XADCs	0	1	0%	
Average Fanout of Non-Clock Nets	3.75			

Performance Summary [-]			
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports [-]					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Wed May 22 09:21:42 2024	0	392 Warnings (390 new)	22 Infos (22 new)
Translation Report	Current	Wed May 22 09:22:09 2024	0	0	0
Map Report	Current	Wed May 22 09:23:08 2024	0	72 Warnings (0 new)	5 Infos (0 new)
Place and Route Report	Current	Wed May 22 09:24:09 2024	0	0	3 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Wed May 22 09:24:28 2024	0	0	4 Infos (0 new)
Bitgen Report					

Secondary Reports [-]		
Report Name	**Status**	**Generated**
ISIM Simulator Log	Current	Wed May 22 09:49:03 2024
Post-Synthesis Simulation Model Report	Current	Wed May 22 09:21:46 2024
Post-Place and Route Simulation Model Report	Current	Wed May 22 09:44:25 2024

2. Area result

Item	Number	Item	Area(um^2)
Ports	70	Combinational	1483546.414358
Nets	140	Non-combinational	153599.850292
Cells	71	Net Interconnect	2089790.568665
References	3	Total Area	3726936.83315

3. Evaluation index

Parameter	Value
Process	SMIC 0.18 um
Number of Pins	90
Operating frequency	135.135 MHz
Operating voltage	1.62 V
Total area	4.695300 mm^2
Power consumption per calculate operation	5.25118208x10^6 mJ
FFT operands per unit area and unit power consumption	41987.688 times/($\mathrm{mm}^2 mWs$)
Bandwidth	510.51 MB/s
Parallel throughput	4.5946x10^9 bps

5.Reference

[1]Design and Implementation of an FFT IP Generator

[2] Garrido M, Huang S J, Chen S G. Feedforward FFT Hardware Architectures Based on Rotator Allocation[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2018, 65(2):581-592.

[3] Zulkifli, Siva. Design of 16-point Radix-4 Fast Fourier Transform in 0.18μm CMOS Technology. American Journal of Applied Sciences. 2007.

[4] 丁晓磊, 朱恩, 赵梅. 16点基4-FFT芯片设计技术研究[J]. 信息技术, 2007(01): 64-67+71.

[5] DJJ 工程數學特論 講義

[6]DJJ ADSP 講義