گزارش کار آزمایشگاه DSD

آزمایش شماره 3

10 فروردین 1400

عرشا اخوان
محمدحسین عبدی
علیرضا ایلامی

شماره آزمایش: **3**	موضوع: توصیف جریان داده	تاریخ آزمایش: 10 فروردین 1400
عرشیا اخوان **97110422**	محمدحسین عبدی **97110285**	علیرضا ایلامی **97101286**

مقدمه:

در این آزمایش قصد داریم دو مقایسه‌گر، یکی cascadable 4 bit comparator و دیگری یک serial comparator بسازیم.

- این آزمایش صرفا کد وریلاگ است.

شرح آزمایش:

در هر دو حالت سریال و حالت 4 بیت ثابت، یک بیت LGN داریم. (Less / Greater Not) که در مقایسه دو عدد، اگر اولی از دومی کوچکتر بود خروجی یک میشود. در غیر اینصورت خروجی صفر است.

البته حالت تساوی اصلا اینجا چک نمیشود.

ابتدا شرط مساوی بودن دو عدد را با یکدیگر چک میکنیم اگر مساوی بود، اصلا LGN بررسی نمیشود و صفر خروجی میدهد. در غیر اینصورت محاسبه LGN آغاز میشود.

برای حالت سریال، باید با کلاک کار کنیم. و برای این آزمایش نیاز بود که DFF را خودمان طراحی کنیم.

ابتدا ماژول فلیپ‌فلاپ را در فایل وریلاگ dff.v طراحی کردیم و پس از تست کارکرد آن، آن را داخل بقیه کد وریلاگ comp_seq.v (خطوط 17 تا 20) آوردیم. (در صورت آزمایش گفته شده از هیچ توصیف اضافه‌ای تعریف نکنید.)

برای طراحی DFF، دقیقا مانند معماری خود فلیپ فلاپ از 4 گیت NAND استفاده کرده‌ایم.

گزارش تست بنچ:
تست بنچ در فایلی به نام testbench.v نوشته شده است.
در ابتدا دو ماژول ترتیبی و ترکیبی Comparator (مقایسه‌کننده) را تعریف کرده و کلاک را نیز ست کرده‌ایم.

در فایل seed.dat یک عدد رندوم قرار داده ایم.
این عدد رندوم، همان random$ ماست.

برای اینکه با هر سیمولیت این رندوم بودن ما تفاوت کند، ما به کمک یک بش در فایل seed.dat عدد رندوم را نوشته ایم. (به علت محدودیت لینوکس اینکار را کرده ایم. در نسخه ویندوز random flag$ وجود دارد.)

سپس، ابتدا ماژول ترکیبی را سیمولیت کرده (خطوط 55-62) و سپس ماژول ترتیبی را شبیه سازی کرده ایم. (63-79)

یک فایل تست بنچ نیز برای خود فلیپ فلاپ نوع D که در مقایسه‌کننده ترتیبی از آن استفاده کرده‌ایم، نوشته شده است. (برای زمانی که جدا نوشته شده بود و تستش کردیم.)

کد وریلاگ مقایسه‌گر ترتیبی در شکل زیر موجود است:

```
// digits are comming from MSB (right to left)
module Comp_seq (a,
                 b,
                 reset,
                 lgn_out,
                 e_out,
                 clk);

    input a, b, reset, clk;
    output lgn_out, e_out;

    wire y_lgn_out, x_lgn_out, qp_lgn_out;
    wire y_e_out, x_e_out, qp_e_out;


    // lgn_out dff implementation
    assign x_lgn_out  = ~(clk & (~reset & ((~e_out & lgn_out) | (e_out & ~a & b))));
    assign y_lgn_out  = ~(clk & x_lgn_out);
    assign lgn_out    = ~(qp_lgn_out & x_lgn_out);
    assign qp_lgn_out = ~(lgn_out & y_lgn_out);


    // e_out dff implementaion
    assign x_e_out  = ~(clk & reset | (e_out & ~(a ^ b)));
    assign y_e_out  = ~(clk & x_e_out);
    assign e_out    = ~(qp_e_out & x_e_out);
    assign qp_e_out = ~(e_out & y_e_out);

endmodule //Comp_seq
```

کد وریلاگ مقایسه‌گر ترکیبی در شکل زیر موجود است:

مقایسه گر یک بیتی:

```
module Comp_1bit (a,
                  b,
                  lgn_in,
                  e_in,
                  lgn_out,
                  e_out,
                  );
    input a, b, gin, e_in, lgn_in;
    output lgn_out, e_out;

    assign lgn_out = (~e_in & lgn_in) | (e_in & ~a & b);
    assign e_out   = (e_in & ~(a ^ b));

endmodule //Comp_1bit
```

مقایسه گر 4 بیتی:

```
module Comp_comb (a, b, lgn_in, e_in, lgn_out, e_out,);
    parameter BIT_LEN = 4;
    input wire [BIT_LEN-1:0] a, b;
    input wire lgn_in, e_in;
    output lgn_out, e_out;

    wire [BIT_LEN:0] lgn, e;

    assign lgn[BIT_LEN] = lgn_in;
    assign e[BIT_LEN]   = e_in;
    assign lgn_out      = lgn[0];
    assign e_out        = e[0];

    genvar i;
    generate
    for (i = BIT_LEN - 1 ; i >= 0; i = i - 1) begin
        Comp_1bit c1b(
        .a(a[i]),
        .b(b[i]),
        .lgn_in(lgn[i+1]),
        .lgn_out(lgn[i]),
        .e_in(e[i+1]),
        .e_out(e[i])
        );
    end
    endgenerate

endmodule //Comp_4bit
```

تمامی نتایج تست در پوشه report در فایل result.txt قرار داده شده است.

برای مثال، تست مقایسه‌گر ترکیبی به شرح زیر است:

```
combinational comparator module test:
          0   A: 6  B:12  LGN:1  EQ:0
          1   A: 9  B:13  LGN:1  EQ:0
          2   A: 4  B:12  LGN:1  EQ:0
          3   A: 2  B: 9  LGN:1  EQ:0
          4   A: 8  B: 7  LGN:0  EQ:0
          5   A: 0  B: 9  LGN:1  EQ:0
          6   A: 5  B:12  LGN:1  EQ:0
          7   A: 5  B:11  LGN:1  EQ:0
          8   A: 3  B:11  LGN:1  EQ:0
          9   A:12  B:13  LGN:1  EQ:0
         10   A: 4  B: 0  LGN:0  EQ:0
         11   A: 1  B:15  LGN:1  EQ:0
         12   A: 6  B:12  LGN:1  EQ:0
         13   A: 7  B: 2  LGN:0  EQ:0
         14   A: 4  B: 1  LGN:0  EQ:0
         15   A:12  B: 3  LGN:0  EQ:0
```

این هم مثالی از تست مقایسه‌گر ترتیبی:

```
                    188   A:1  B:0  LGN:1  EQ:0
                    190   A:0  B:0  LGN:1  EQ:0
                    192   A:0  B:0  LGN:1  EQ:0
                    194   A:1  B:1  LGN:1  EQ:0
                    196   A:0  B:0  LGN:1  EQ:0
                    198   A:1  B:1  LGN:1  EQ:0
reset module  -->  LGN:0  EQ:1
                    202   A:1  B:1  LGN:0  EQ:1
                    204   A:0  B:0  LGN:0  EQ:1
                    206   A:0  B:0  LGN:0  EQ:1
                    208   A:0  B:0  LGN:0  EQ:1
                    210   A:1  B:0  LGN:0  EQ:0
                    212   A:1  B:0  LGN:0  EQ:0
                    214   A:0  B:1  LGN:0  EQ:0
                    216   A:1  B:0  LGN:0  EQ:0
                    218   A:0  B:0  LGN:0  EQ:0
reset module  -->  LGN:0  EQ:1
                    222   A:0  B:1  LGN:1  EQ:0
                    224   A:0  B:0  LGN:1  EQ:0
                    226   A:1  B:1  LGN:1  EQ:0
                    228   A:1  B:0  LGN:1  EQ:0
                    230   A:1  B:0  LGN:1  EQ:0
                    232   A:0  B:0  LGN:1  EQ:0
                    234   A:1  B:1  LGN:1  EQ:0
                    236   A:0  B:1  LGN:1  EQ:0
                    238   A:0  B:1  LGN:1  EQ:0
reset module  -->  LGN:0  EQ:1
                    242   A:1  B:0  LGN:0  EQ:0
```

فایل تست بنچ به شرح زیر است:

```
`timescale 1ns/1ps
`define NULL 0
module testbench;()
   reg [3:0] a, b;
   reg A, B;
   reg reset, clk;
   wire lgn, e, LGN, E;

   Comp_comb #(.BIT_LEN(4)) comp0)
.   a(a),
.   b(b),
.   lgn_in(1'b0),
.   e_in(1'b1),
.   lgn_out(lgn),
.   e_out(e)
;(

   Comp_seq comp1)
.   a(A),
.   b(B),
.   reset(reset),
.   lgn_out(LGN),
.   e_out(E),
.   clk(clk)
;(

   always
   begin
   clk          = 0;
   forever #1 clk = ~clk;
   end

   integer data_file;
   integer scan_file;
   integer seed;
   initial begin
      data_file = $fopen("seed.dat", "r");
      if (data_file == `NULL) begin
$         display("data_file handle was NULL");
$         finish;
      end
      scan_file = $fscanf(data_file, "%d", seed);
      if (scan_file == `NULL) begin
$         display("integer read error");
$         finish;
```

```
      end
   end



   integer i, j;
   initial begin

 // 2#       waiting for reading random seed from file "seed.dat"
$       display("combinational comparator module test:");
      for (i = 0; i < 16; i = i + 1) begin
         a = {$random(seed)}%16;
         b = {$random(seed)}%16;
1#
$          display(i, "   A:%d  B:%d  LGN:%b  EQ:%b", a, b, lgn, e);
      end

$       display("sequantial comparator module test:");
      for (i = 0; i < 16; i = i + 1) begin
         A     = 0;
         B     = 0;
         reset = 1;
;2#
$          display("reset module  -->  LGN:%b  EQ:%b", LGN, E);
         j = {$random(seed)}%20+1;
         while (j > 0) begin
            A     = {$random(seed)}%2;
            B     = {$random(seed)}%2;
            reset = 0;
2#
$             display($time, "   A:%d  B:%d  LGN:%b  EQ:%b", A, B, LGN, E);
            j = j - 1;
         end
      end

$       finish;
   end

endmodule
```

پایان