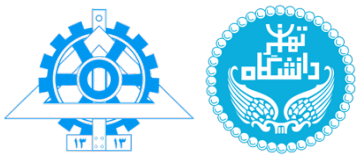
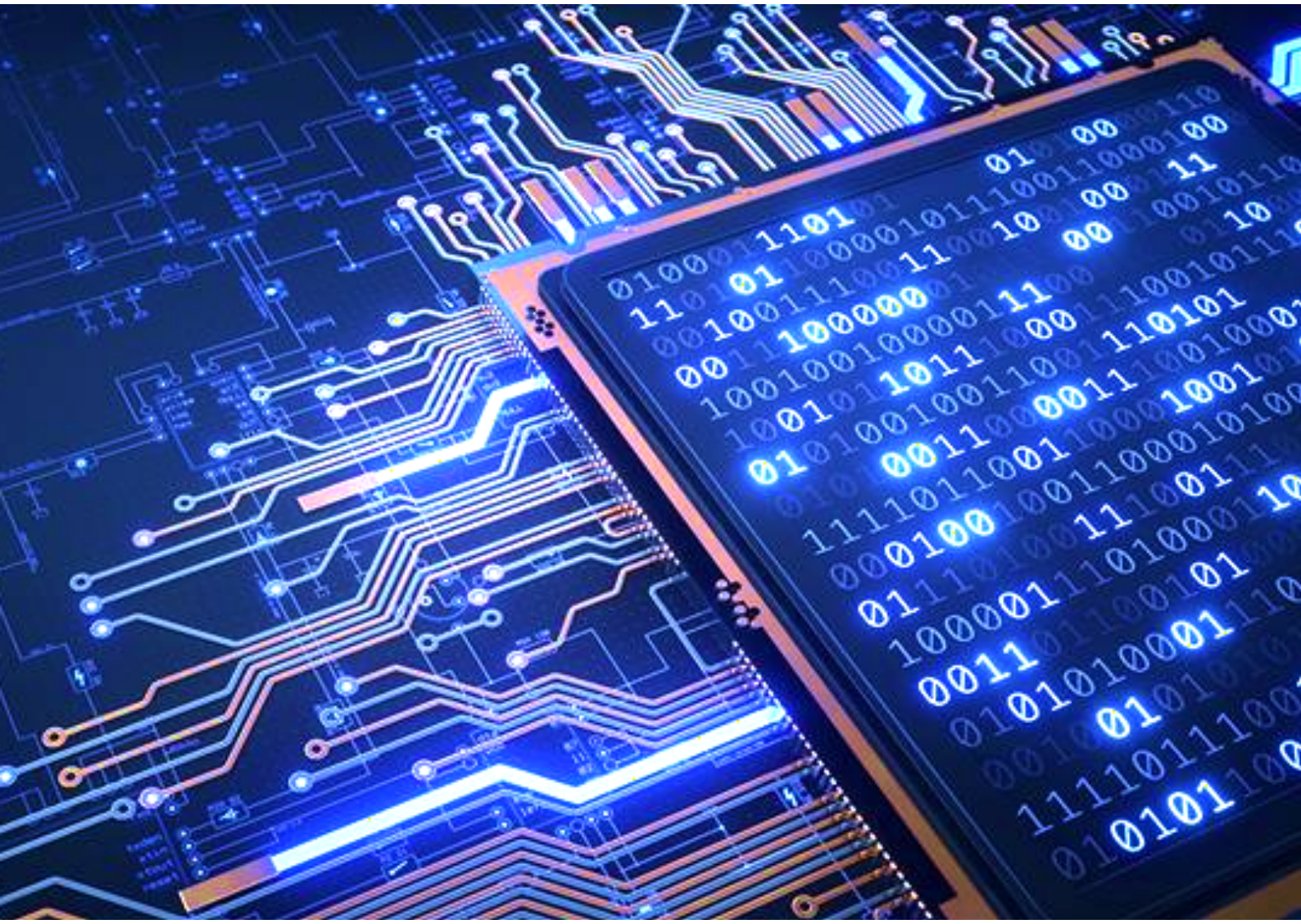


بهار ۱۴۰۳

تمرین کامپیوتری سوم

"مدار های منطقی"

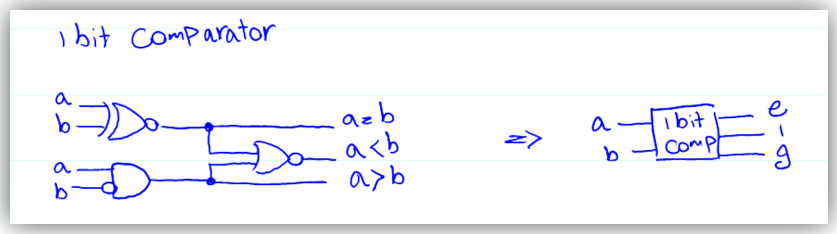


مجید صادقی نژاد ۸۱۰۱۰۱۴۵۹

بخش اول:

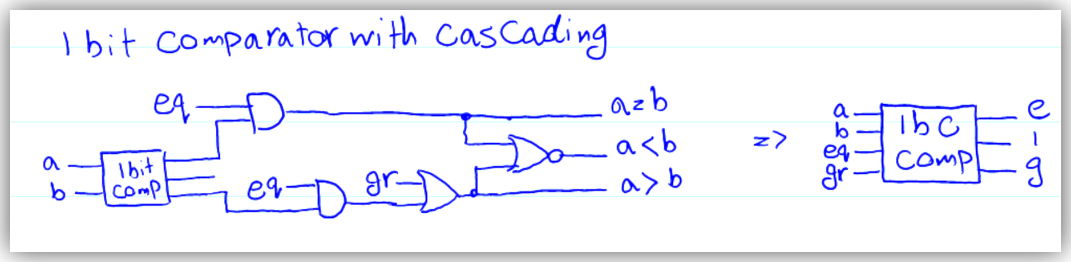
ابتدا بلوک بیشینه را طراحی می کنیم برای ساخت این بلوک نیاز به مقایسه گر داریم پس در ابتدا یک مقایسه گر تک بیتی در سطح گیت طراحی می کنیم:

```
module comparator_1b(input a,b, output e,g,l);
 wire w1,w2,w3;

 xor x1(w1,a,b);
 not n1(e,w1);

 not n2(w2,b);
 and a1(g,a,w2);

 or o1(w3,e,g);
 not n3(l,w3);

endmodule
```



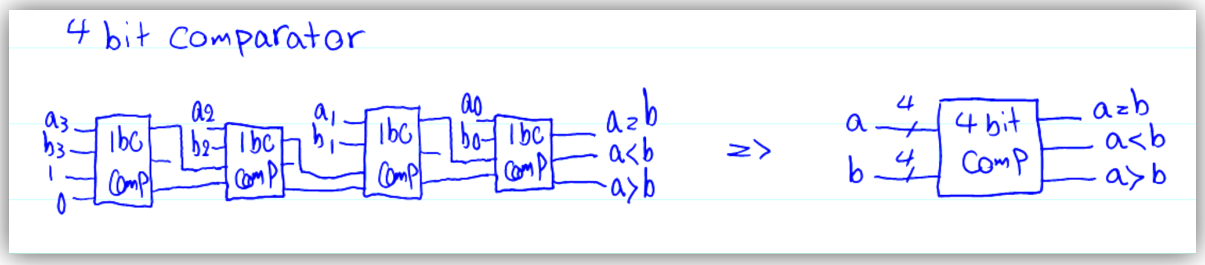
سپس با استفاده از مقایسه گر تک بیتی یک مقایسه گر تک بیتی با قابلیت cascading طراحی می کنیم :

```
module comparator_1b_c(input eq,gr, input a,b, output e,g,l);
 wire w1,w2,w3;
 wire w4,w5;

 comparator_1b comp1(a,b,w1,w2,w3);

 and a1(e,eq,w1);

 and a2(w4,eq,w2);
 or o1(g,w4,gr);

 or o2(w5,e,g);
 not n1(l,w5);

endmodule
```



حال با به هم بستن چهار عدد از این مقایسه گر ها یک مقایسه کننده چهاربیتی طراحی می کنیم:

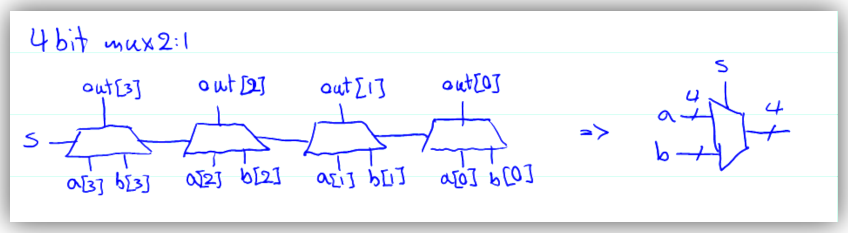
```
module comparator_4b(input[3:0] a,b, output e,g,l);
 wire [0:11] w;

 comparator_1b_c comp1(1'b1,1'b0,a[3],b[3],w[0],w[1],w[2]);
 comparator_1b_c comp2(w[0],w[1],a[2],b[2],w[3],w[4],w[5]);
 comparator_1b_c comp3(w[3],w[4],a[1],b[1],w[6],w[7],w[8]);
 comparator_1b_c comp4(w[6],w[7],a[0],b[0],e,g,l);

endmodule
```



بلوک دیگری که برای محاسبه ی بیشینه نیاز داریم یک مالتی پلکسر ۲ به ۱ است که ورودی ها و خروجی چهاربیتی دارد که در ابتدا یک مالتی پلکسر ۲ به ۱ تک بیتی در سطح گیت طراحی کرده و با به هم بستن چهار عدد از این مالتی پلکسر ها مالتی پلکسر چهار بیتی می سازیم:

```
module mux2(input a,b,select, output out);
 wire w0,w1,w2;

 not n1(w0,select);
 and a1(w1,select,b);
 and a2(w2,w0,a);
 or o1(out,w1,w2);

endmodule
```

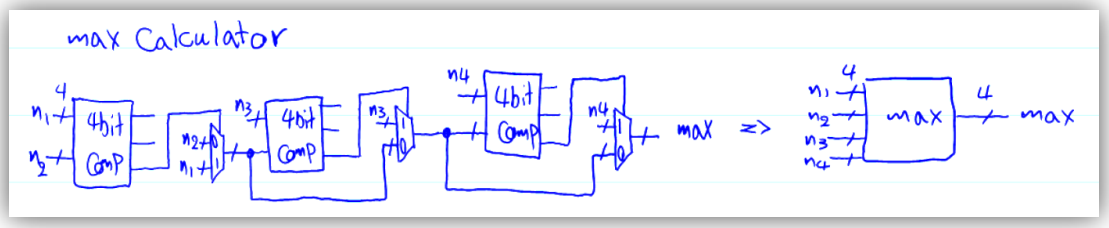
```
module mux2_4bit(input[3:0] a,b, input select, output[3:0] out);
 mux2 m1(a[3],b[3],select,out[3]);
 mux2 m2(a[2],b[2],select,out[2]);
 mux2 m3(a[1],b[1],select,out[1]);
 mux2 m4(a[0],b[0],select,out[0]);
endmodule
```



حال با بهم بستن یک مالتی پلکسر ۲ به ۱ و یک مقایسه گر می توانیم بیشینه ی دو عدد را مشخص کنیم و با بهم بستن سه عدد از این واحد ها می توان در بیشینه ی چهار عدد را مشخص کرد به طوری که نتیجه مقایسه ی هر دو عدد در مرحله بعد با عدد بعدی مقایسه شود و در نهایت با افزودن چهار صفر به ابتدا عدد به دست آمده آن را هشت بیتی می کنیم و مدار محاسبه گر بیشینه ساخته می شود:

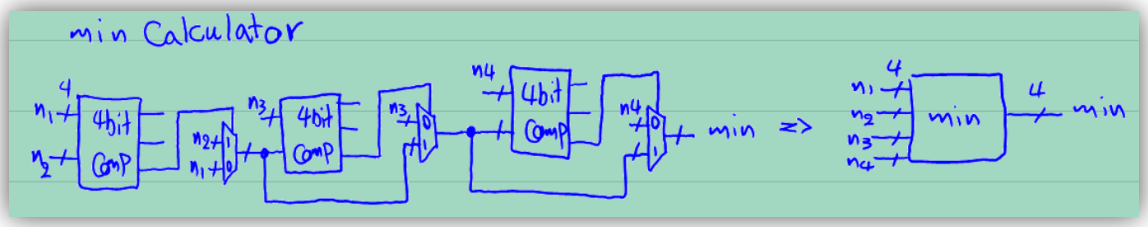
```
module max_calculator_2num(input[3:0] a,b, output[3:0] max);
 wire s;
 wire eq,ls;

 comparator_4b comp(a,b,eq,s,ls);
 mux2_4bit mux(b,a,s,max);

endmodule
```

```
module max_calculator(input[3:0] nums[1:4], output[7:0] max);
 wire[3:0] big[1:2];

 max_calculator_2num maxi1(nums[1],nums[2],big[1]);
 max_calculator_2num maxi2(nums[3],big[1],big[2]);
 max_calculator_2num maxi3(nums[4],big[2],max[3:0]);

 assign max[7:4] = 4'b0000;

endmodule
```



برای ساخت مدار محاسبه گر کمینه نیز کافیست که ورودی های مالتی پلکسر ها را جا به جا کنیم تا در هر مرحله عدد کوچکتر به مرحله ی بعد برود:

```
module min_calculator_2num(input[3:0] a,b, output[3:0] min);
 wire s;
 wire eq,ls;

 comparator_4b comp(a,b,eq,s,ls);
 mux2_4bit mux(a,b,s,min);

endmodule

module min_calculator(input[3:0] nums[1:4], output[7:0] min);
 wire[3:0] smal[1:2];

 min_calculator_2num mini1(nums[1],nums[2],smal[1]);
 min_calculator_2num mini2(nums[3],smal[1],smal[2]);
 min_calculator_2num mini3(nums[4],smal[2],min[3:0]);

 assign min[7:4] = 4'b0000;

endmodule
```

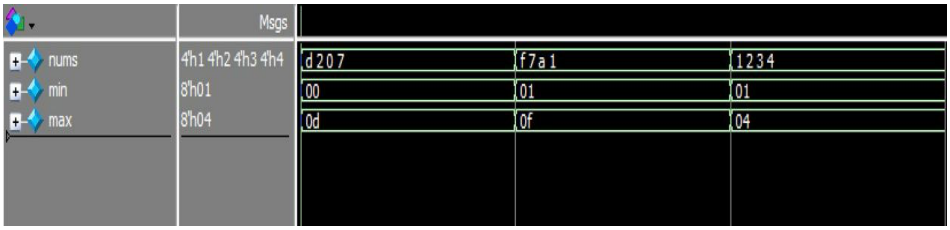


و در نهایت با اعمال تست زیر به مدار محاسبه گر بیشینه و کمینه شکل موج زیر را به دست می آوریم که صحت کار مدار را نشان می دهد:

دسته های اعداد به شکل زیر هستند

(۱۳,۲,۰,۷) – (۱۵,۷,۱۰,۱) – (۱,۲,۳,۴)

```
module min_max_TB();
 logic [3:0] nums [1:4];
 wire [7:0] max,min;

 min_calculator UUT1(nums,min);
 max_calculator UUT2(nums,max);

 initial begin
  nums = {4'd13,4'd2,4'd0,4'd7};
  #10;
  nums = {4'd15,4'd7,4'd10,4'd1};
  #10;
  nums = {4'd1,4'd2,4'd3,4'd4};
  #10;
 end
endmodule
```



بخش دوم:

برای محاسبه میانگین و واریانس دقیقا فرمول ریاضی هر کدام را به صورت رفتاری پیاده سازی می کنیم با این محدودیت که جمع کننده ها و ضرب کننده ها دو ورودی هستند. فقط باید توجه داشت از آنجا که مجاز به استفاده از تقسیم کننده نیستیم به جای عمل تقسیم بر ۴ از شیفت دادن به اندازه ۲ استفاده می کنیم که این موضوع سبب می شود تا از حاصل تقسیم مورد نظر براکت گرفته شود و رند شود همچنین در تمامی مراحل از ابتدا اعداد هشت بیتی در نظر گرفته شده اند:

```
module mean_calculator(input[3:0] nums[1:4], output[7:0] mean);
 wire[7:0] w1,w2,w3 ;

 assign w1 = nums[1] + nums[2] ;
 assign w2 = w1 + nums[3] ;
 assign w3 = w2 + nums[4] ;

 assign mean[5:0] = w3[7:2] ;
 assign mean[7:6] = {1'b0,1'b0} ;

endmodule
```

```
module var_calculator(input[3:0] nums[1:4], output[7:0] varr);
 wire[7:0] u ;
 wire[7:0] w [1:16] ;

 mean_calculator mean(nums,u);

 assign w[1] = u - nums[1] ;
 assign w[2] = u - nums[2] ;
 assign w[3] = u - nums[3] ;
 assign w[4] = u - nums[4] ;

 assign w[5] = w[1] * w[1];
 assign w[6] = w[2] * w[2];
 assign w[7] = w[3] * w[3];
 assign w[8] = w[4] * w[4];

 assign w[9] = w[5] + w[6];
 assign w[10] = w[9] + w[7];
 assign w[11] = w[10] + w[8];

 assign varr[5:0] = w[11][7:2] ;
 assign varr[7:6] = {1'b0,1'b0} ;

endmodule
```

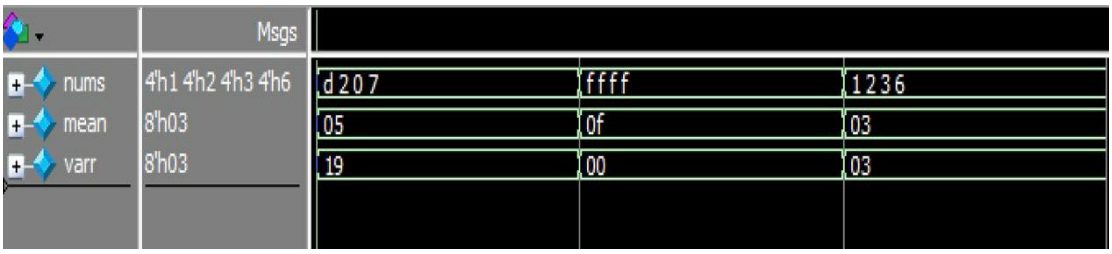
حال با استفاده از تست زیر عملکرد آن دو را می آزماییم توجه شود که انتظار داریم از مقادیر واریانس و میانگین واقعی براکت گرفته شود (دقت شود که اعداد نمایش داده شده در شکل موج به صورت hex هستند):

Mean(۱۳,۲,۰,۷) = ۵.۵ , Var(۱۳,۲,۰۷) = ۲۵.۲۵

Mean(۱۵,۱۵,۱۵,۱۵) = ۱۵ , Var(۱۵,۱۵,۱۵,۱۵) = ۰

Mean(۱,۲,۳,۶) = ۳ , Var(۱,۲,۳,۶) = ۳.۵

```
module mean_varr_TB();
 logic [3:0] nums [1:4];
 wire [7:0] mean,varr;

 mean_calculator UUT1(nums,mean);
 var_calculator UUT2(nums,varr);

 initial begin
  nums = {4'd13,4'd2,4'd0,4'd7};
  #10;
  nums = {4'd15,4'd15,4'd15,4'd15};
  #10;
  nums = {4'd1,4'd2,4'd3,4'd6};
  #10;
 end
endmodule
```



بخش سوم:

در نهایت برای پیاده سازی مدار نهایی در ابتدا نیاز به یک priority encoder است که بتواند درخواست (op) را دریافت کند و یک مالتی پلکسر ۴ به ۱ ، هشت بیتی که خروجی هر چهار مدار محاسبه گر به آن متصل شود و بر اساس کد تولید شده توسط اینکودر خروجی صحیح را به خروجی اصلی مدار بدهد همچنین یک decoder مورد نیاز است تا کد درخواست داده شده را مجددا به یک عدد تبدیل کند و مشخص کند که خروجی از چه نوعی است بیشینه، کمینه، میانگین یا واریانس. همچنین برای دیکودر و اینکورد ورودی و خروجی enable طراحی شده تا در صورتی که درخواست op صحیح نبود(۰۰۰۰) خروجی نشان دهنده ی عملیات انجام شده نیز صفر گردد:

دیکودر:

```
module decoder(input [1:0] in, output [3:0] out, input en);
 reg [0:3] Y;

 always @(in or en)
  case ({en, in})
   3'b100: Y = 4'b1000;
   3'b101: Y = 4'b0100;
   3'b110: Y = 4'b0010;
   3'b111: Y = 4'b0001;
   default: Y = 4'b0000;
  endcase

 assign out = Y ;
endmodule
```

اینکودر دارای اولویت:

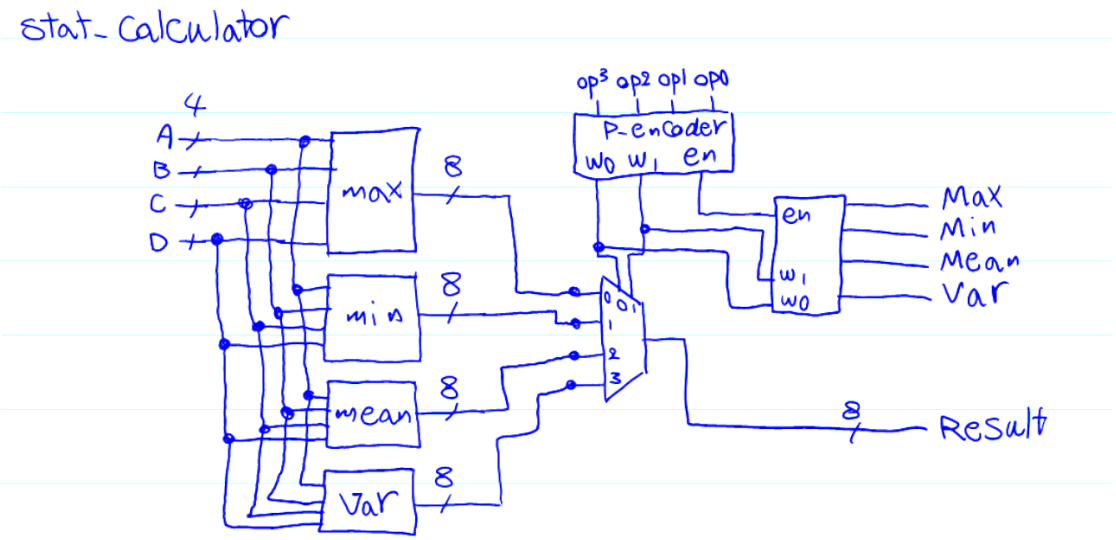
```
module priority_encoder(input [3:0] in, output [1:0] out, output en);
 reg [1:0] outr;
 reg enr;

 always @(in)
 begin
  enr = 1;
  casex(in)
   4'bxxx1: outr = 2'd0;
   4'bxx10: outr = 2'd1;
   4'bx100: outr = 2'd2;
   4'b1000: outr = 2'd3;
   default: begin
    enr = 0;
    outr = 2'bxx;
   end
  endcase
 end

 assign en = enr ;
 assign out = outr ;

endmodule
```

مالتی پلکسر ۴ به ۱ هشت بیتی:

```
module mux4_1bit(input w0, w1, w2, w3, input [1:0] s, output out);

 assign out = s[1] ? (s[0] ? w3 : w2) : (s[0] ? w1 : w0);

endmodule

module mux4(input[7:0] w0, w1, w2, w3, input [1:0] s, output[7:0] out);
 genvar i;
   generate
     for(i = 0; i < 8; i=i+1) begin: mux_stack
        mux4_1bit xx(w0[i],w1[i],w2[i],w3[i],s,out[i]);
     end
   endgenerate
endmodule
```

و در نهایت ساختار کلی stat_calculator:

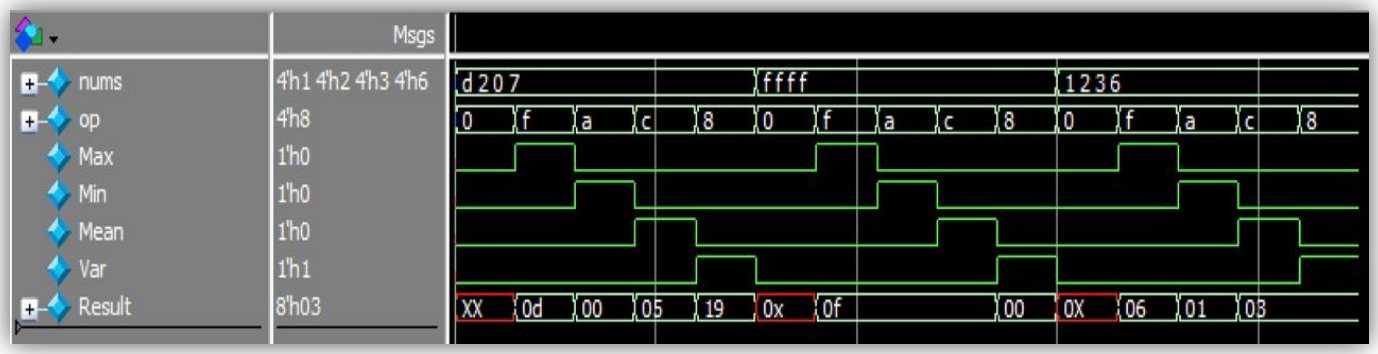
```
module stat_calculator(input[3:0] Numbers[1:4], input[3:0] OP, output Max,Min,Mean,Var, output[7:0] Result);
 wire[7:0] w1,w2,w3,w4 ;
 wire[1:0] select ;
 wire en;

 max_calculator maxC(Numbers,w1);
 min_calculator minC(Numbers,w2);
 mean_calculator meanC(Numbers,w3);
 var_calculator varC(Numbers,w4);

 priority_encoder encoderM(OP, select, en);
 decoder decoderM(select, {Max,Min,Mean,Var}, en);
 mux4 multiplexer(w1, w2, w3, w4, select, Result);

endmodule
```



و در نهایت با تست زیر که ابتدا مدار محاسبه گر را در حالت خاموش می گذارد و بعد به ترتیب تک تک عملیات ها را روی یک دسته عدد تست می کند، مدار را برای سه حالت تست می کنیم، در هر مرحله برای مشخص کردن درخواست op ورودی ای می دهیم که موضوع اولویت نیز تست شود دقت شود که این بخش نیز به علت شیفت دادن از حاصل اصلی براکت گرفته شده است:

محاسبات صحت سنجی:

- ۱,۲,۳,۶
  Max = ۶
  Min = ۱
  Mean = ۳
  Var = ۳.۵
- ۱۵,۱۵,۱۵,۱۵
  Max = ۱۵
  Min = ۱۵
  Mean = ۱۵
  Var = ۰
- ۱۳,۲,۰,۷
  Max = ۱۳
  Min = ۰
  Mean = ۵.۵
  Var = ۲۵.۲۵




```
module stat_calculator_TB();
 logic [3:0] nums [1:4];
 logic [3:0] op;
 wire [7:0] Result;
 wire Max,Min,Mean,Var;

 stat_calculator UUT(nums, op, Max, Min, Mean, Var, Result);

 initial begin
  op = 4'b0000;
  nums = {4'd13,4'd2,4'd0,4'd7};
  #3;
  op = 4'b1111;
  #3;
  op = 4'b1010;
  #3;
  op = 4'b1100;
  #3;
  op = 4'b1000;
  #3
  op = 4'b0000;
  nums = {4'd15,4'd15,4'd15,4'd15};
  #3;
  op = 4'b1111;
  #3;
  op = 4'b1010;
  #3;
  op = 4'b1100;
  #3;
  op = 4'b1000;
  #3
  op = 4'b0000;
  nums = {4'd1,4'd2,4'd3,4'd6};
  #3;
  op = 4'b1111;
  #3;
  op = 4'b1010;
  #3;
  op = 4'b1100;
  #3;
  op = 4'b1000;
  #3;
 end
endmodule
```