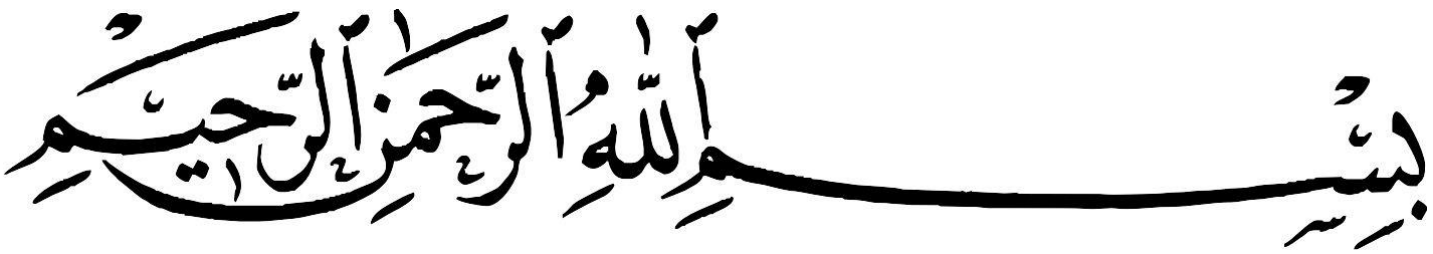

گزارش کار آزمایشگاه FPGA

آزمایش 6

نام استاد : دکتر ثامنی

ایمان استوار9332366

علیرضا رحمان ستایش9332679

صورت مسئله :

Moving Average Filter :

در این آزمایش قصد داریم الگوریتم cordic در مد circular vectoring پیاده سازی کنیم .

الگوریتم cordicروشی برای یک سری محاسبات از طریق shif & add که محاسبات کمی لازم دارد پیشنهاد شده است .

ازین الگوریتم ما به صورت piplineویا hardware reusing و استفاده از fsm میتوانیم استفاده کنیم .

ما در اینجا به صورت fsm پیاده کردیم .

طرح (پیاده سازی)

ورودی و خروجی های های ماژول :

ماژولی تحت عنوان cordic نوشته با پارامتر های زیر :

```
module Cordic#(parameter answidth=32))
  input clk ,
```

کلاک ورودی

```
      input en,
```

پایه فعال ساز جهت اعلام قرار داشتن ورودی مناسب و شروع محاسبات

```
      input [7:0]DipSw,
```

4 بیت اول ورودی x و4 بیت بعدی ورودی y

```
      output [7:0]segment1pin,
```

x خروجی

```
      output [7:0]segment2pin
```

y خروجی

```
      ;(
```

رجیستر های داخلی ماژول:

```
      reg signed [answidth-1:0] x , y , z;
```

این 3 regدر حقیقت همان 3 رجیستر اصلی که محاسبات کوردیک روی انها انجام میشود .

reg [4:0]count=8'b0;

این رجیستر کنترلی جهت مدیریت stateهای fsm استفاده میشود .

reg enable=0;

این رجیستر کنترلی جهت فعال شدن fsm استفاده میشود .

سایر wireها

wire den;

دیبانس شده کلید

wire [4:0]address;

منفی علامت Y یا همان d در محاسبه کوردیک

wire signed [1:0]sign;

assign sign = (y >= 0)? -'d1 : +'d1;

xm ymحاصلضرب x y درsigned و xms , ymsشیفت یافته xm , ymمتناسب مرحله fsm است.

wire signed[answidth-1:0] xms,yms , xm ,ym;

assign xm=x*sign;

assign ym=y*sign;

ansx ,ansy , ansz : scaleشده جواب یعنی x y zمیباشد (ضریب kکوردیک 5/8 محاسبه شده و که اینجا اعمال شده است)

wire signed [24:0] ansx ,ansy , ansz;

assign ansx=(x*5)>>>20;

assign ansy=y>>>17;

assign ansz=(z*1000)>>>17;

ادرس اعمالی به ماژول atang

assign address=count;

خروجی ماژول atang

wire signed[15:0]atan;

خروجی های دیکود شده جهت 7segmentها

```
wire [3:0]seg1 , seg2;
```

پروسه های : initial

به طور کلی 17بیت اعشار و بقیه بیت ها صحیح برای محاسبات fix point در نظر گرفته شده

```
initial begin
        x='d4<<<17;
        y='d3<<<17;
        z='d0<<<17;
end
```

پروسه های : always

```
always @ (posedge clk)begin
        if (~den)
            enable <=1;
        if(enable)begin
```

پس از هربار اتمام پروسس در صورت فعال بودن fsm داده جدید را از dipswitch برو رجیستر های x, y می ریزد .

```
        if (count==0)begin
            x<=(DipSw[3:0])<<<17;
            y<=(DipSw[7:4])<<<17;
            count<=count+1;
        end
```

پیاده سازی الگوریتم این مد از کوردیک در هر مرحله بر روی x y z

```
                else if(count<'d16)begin
                        count<=count+1;
                        x<=x-yms;
                        y<=y+xms;
                        z<=z-sign*atan;
                end
                else begin
                        enable<=0;
                        count<=0;
                end
        end
end
```

ماژول shift register

```
module ShiftR#(parameter width=10)(
```

تعداد شیفت ورودی

```
        input [4:0]Nshift ,
```

عدد ورودی

```
         input signed[width-1:0]in,
```

عدد خروجی شیفت یافته

```
         output signed[width+10:0]out
         );

 assign out =  in>>>Nshift;
```

به صورت زیر instanceمیگیریم :

```
ShiftR  #(.width(answidth))xs(
   .Nshift(count-1),
   .in(xm),
   .out(xms)
   );

ShiftR #(.width(answidth)) ys (
   .Nshift(count-1),
   .in(ym),
   .out(yms)
   );
```

ماژول Debouncer

در پروژه های قبلی توضیح داده شده و دلیل استفاده این ماژول دیبانس کردن کلید ورودی جهت ریست کردن fsm

```
module Debouncer(input in ,output reg out ,input clk);
	reg [15:0] cnt;
	initial begin
	//out<=in;
	out<=0;
	cnt<=0;
	end
always @ (posedge clk)begin
	if(out !=in)
		cnt<=cnt+1;
	else
		cnt<=0;
if(cnt==16'hffff)
	out<=in;
end



endmodule
```

به صورت زیر instanceمیگیریم :

```
	Debouncer decouncer (
   .in(en),
   .out(den),
   .clk(clk)
   );
```

ماژول Block Rom

محتویات خانه های این حافظه مقادیر خاصی از arctang جهت محاسبات مرحله های مختلف fsm

نحوه instance گرفتن :

atang your_instance_name)

. clka(clk), // input clka

. addra(address), // input [4 : 0] addra

. douta(atan) // output [15 : 0] douta

;(

ماژول 7segment decoder

module Decoder_7)

input wire[3 : 0] in,

output reg [7 : 0] out

; (

//abcde fg = fd [6] , d [5] , d [4] , d [3] , d [2] , d [1] , d [0] g

//a c t i v e low

always @(*) begin

out <= 7'b0000_001;

case (in)

'4b0000 : out <= 7'b0000_001;

'4b0001 : out <= 7'b1001_111;

'4b0010 : out <= 7'b0010_010;

'4b0011 : out <= 7'b0000_110;

'4b0100 : out <= 7'b1001_100;

'4b0101 : out <= 7'b0100_100;

```
'4b0110 : out <= 7'b0100_000;
'4b0111 : out <= 7'b0001_111;
'4b1000 : out <= 7'b0000_000;
'4b1001 : out <= 7'b0000_100;
endcase
end
endmodule
```

و نحوه inctanceگرفتن :

```
        Decoder_7 dec)
.   in(seg1) ,
.   out(segment1pin)
;(

        Decoder_7 dec2)
.   in(seg2) ,
.   out(segment2pin)
;(
```

به پیوست پروژه فایل متلب شبیه سازی جهت یافت تنظیمات دلخواه و مد های مختلف الگوریتم و همچنین ضرایب ثابت قرار داده شده است .