Summery

چکیده:

در این پروژه به بررسی و محاسبه ی توابع مختلف با استفاد از الگوریتم CORDIC می پردازیم و سعی می کنیم ان ها را به ازای مقادیر مختلف محاسبه کنیم و فرق بین CORDIC و بسط تیلور را در اخر مقایسه می کنیم

مقدمه:

هدف از انجام این پروژه یادگیری الگوریتم CORDIC و نوشتن ان به زبان Verilog است. در این پروژه در Modelsim با استفاده از Verilog توابع مختلف را پیاده سازی کرده و ان ها را بررسی می کنیم.

مراحل انجام تمرین:

در این پروژه ابتدا سعی می کنیم توابع Sin , Cos را با CORDIC محاسبه کنیم سپس تابع قدر مطلق را محاسبه کرده و سپس تابع جذر را به دو روش CORDIC و Tylor series محاسبه می کنیم و ان ها را با هم مقایسه می کنیم

قسمت اول:

می خواهیم تابع Trigonometry را با CORDIC پیاده سازی کنیم

این تابع ورودی Theta را گرفته و مقادیر Sin , Cos ان را محاسبه می کند برای این کار یک سیگنال ورودی Valid و دو سیگنال Done , Busy نیز استفاده شده است.

طبق CORDIC در مد گردشی مقدار z را به 0 می رسانیم و مقادیر Sin , Cos را بدست می اوریم

برای این الگریتم ابتدا نیاز به مقادیر ثابت Tan-1 ها داریم که در عکس امده اند.

```
// Alphas
parameter [15:0] Alpha0 = 16'd8192; // 45 = tan-1(2**0)
parameter [15:0] Alpha1 = 16'd4836; // tan-1(2**-1)
parameter [15:0] Alpha2 = 16'd2555; // tan-1(2**-2)
parameter [15:0] Alpha3 = 16'd1297; // tan-1(2**-3)
parameter [15:0] Alpha4 = 16'd651; // tan-1(2**-4)
parameter [15:0] Alpha5 = 16'd326; // tan-1(2**-5)
parameter [15:0] Alpha6 = 16'd163; // tan-1(2**-6)
parameter [15:0] Alpha7 = 16'd81; // tan-1(2**-7)
parameter [15:0] Alpha8 = 16'd41; // tan-1(2**-8)
parameter [15:0] Alpha9 = 16'd20; // tan-1(2**-9)
parameter [15:0] Alpha[9:0] = '{Alpha9, Alpha8, Alpha7,
```

همان طور که دیده می شود این مقادیر را به باینری تبدیل کرده ایم.

سپس با پیدا کردن نا حیه ی Theta مقادیر اولیه مانند شکل داده می شود

```
region1:
begin
  s_x0 = zero;
  s_y0 = one;
  s_z0 = i_Theta;
end
region2:
begin
  s_x0 = one;
  s_y0 = zero;
  s_z0 = {2'b00, i_Theta[13:0]}; // -p/2
end
region3:
begin
  s_x0 = (16'b0 - one);
  s_y0 = zero;
  s_z0 = {2'b11, i_Theta[13:0]}; // +p/2
end
region4:
begin
  s_x0 = zero;
  s_y0 = one;
  s_z0 = i_Theta;
end
```

و سپس در هر Loop مقدار Sin , Cos جدید که برابر مقدار قبلی جمع/تفریق با مقدار ان شیفت یافته به تعداد با محاسبه می شود

```
Calculate:
begin
  o_Busy <= 1'b1;
  s_Count_Times <= s_Count_Times + 4'd1;
  if(s_Theta[15])
  begin
    s_Theta <= s_Theta + Alpha[s_Count_Times];
    s_sin <= s_sin - ({{16{s_cos[15]}}, s_cos} >> s_Count_Times);
    s_cos <= s_cos + ({{16{s_sin[15]}}, s_sin} >> s_Count_Times);
  end
  else
  begin
    s_Theta <= s_Theta - Alpha[s_Count_Times];
    s_sin <= s_sin + ({{16{s_cos[15]}}, s_cos} >> s_Count_Times);
    s_cos <= s_cos - ({{16{s_sin[15]}}, s_sin} >> s_Count_Times);
  end
end
```

و سپس تست مربوط به ان را می نویسیم

Summery

```
wire signed [15:0] COS , o_Sin ;
wire o_Done,o_Busy;
Trigonometry UUT
  (
    i_clock,
    i_Reset,
    i_Theta,
    i_Valid,
    o_Sin,
    o_Cos,
    o_Busy,
    o_Done
  );

initial i_clock=0;
always #20 i_clock=~i_clock;

initial begin
  i_Reset = 1;
  #30
  i_Reset = 0;
  #30
  i_Theta = 16'b1;
  i_Valid = 1;
  #500
  $stop;
end
endmodule
```

Summery

```
initial begin
  i_Reset = 1;
  #30
  i_Reset = 0;
  #30
  i_Theta = 16'd1;
  i_Valid = 1;
  #30
  i_Theta = 16'd90;
  i_Valid = 1;
  #30
  i_Theta = 16'd120;
  i_Valid = 1;
  #30
  i_Theta = 16'd150;
  i_Valid = 1;
  #30
  i_Theta = 16'd150;
  i_Valid = 1;
  #30
  i_Theta = 16'd180;
  i_Valid = 1;
  #30
  i_Theta = 16'd60;
  i_Valid = 1;
  #30
  i_Theta = 16'd30;
  i_Valid = 1;
```

و مقادیر بدست امده را می بینیم

قسمت دوم:

در این قسمت می خواهیم تابع قدر مطلق را پیاده سازی کنیم

تابع قدر مطلق به صورت

$$X0 = A\,, Y0 = B \qquad |\mathrm{A} + \mathrm{jB}| = \mathrm{K1}\sqrt{y^2 + x^2}$$

با استفاده از الگریتم *CORDIC* پیاده سازی می شود در این قسمت از *Pipeline* با Throughput=*1* نیز استفاده شده است و ضریب *K* برابر 1.62 در نظر گرفته شده است

برای این کار ابتدا به مقادیر *Tan-1* ها مانند قبل نیاز داریم

```
parameter [21:0] Alpha0 = 21'd8192; // 45 = tan-1(2**0)
parameter [21:0] Alpha1 = 21'd4836; // tan-1(2**-1)
parameter [21:0] Alpha2 = 21'd2555; // tan-1(2**-2)
parameter [21:0] Alpha3 = 21'd1297; // tan-1(2**-3)
parameter [21:0] Alpha[3:0] = '{Alpha3, Alpha2, Alpha1, Alpha0};
```

سپس باید با توجه به مقدار *A , B* ناحیه ی کارمان را پیدا کنیم

```
case(s_place)
  region1:
  begin
    s_x0 = i_A;
    s_y0 = i_B;
    s_z0 = 0;
  end
  region2:
  begin
    s_x0 = i_B;
    s_y0 = -i_A;
    s_z0 = 90;
  end
  region3:
  begin
    s_x0 = -i_B;
    s_y0 = i_A;
    s_z0 = -90;
  end
  region4:
  begin
    s_x0 = i_B;
    s_y0 = i_A;
    s_z0 = 0;
  end
endcase
end
```

Summery

بعد از پیدا کردن ناحیه مقادیر اولیه ی مناسب را قرار می دهیم

سپس مانند قسمت قبل در هر *iteration* مقدار درست x, y, z را پیدا می کنیم و برای کل تابع یک کنترلر دسرت می نویسیم

```
idle:
begin
  s_Count_Times <= 4'd0;
  s_x <= s_x0;
  s_y <= s_y0;
  s_z <= s_z0;
end
Calculate:
begin
  s_Count_Times <= s_Count_Times + 4'd1;
  if(s_y[21])
  begin
    s_z <= s_z - Alpha[s_Count_Times];
    s_x <= s_x - (s_y >> s_Count_Times);
    s_y <= s_y + (s_x >> s_Count_Times);
  end
  else
  begin
    s_z <= s_z - Alpha[s_Count_Times];
    s_x <= s_x - (s_y >> s_Count_Times);
    s_y <= s_y + (s_x >> s_Count_Times);
  end
end
Ending:
begin
  o_ABS <= 1.62 * s_x;
  o_Z <= s_z;
```

و در اخر خروجی را برابر قرار داده و مقدار ان را بدست می اوریم

برای بررسی ان تست بنچ را می نویسیم

```
reg i_clock;
reg i_Reset;
reg [21:0] i_A,i_B;
wire signed [21:0] o_ABS , o_Z ;

Absolute UT
  (
    i_clock,
    i_Reset,
    i_A,
    i_B,
    o_ABS,
    o_Z,
  );

initial i_clock=0;
always #20 i_clock=~i_clock;

initial begin
  i_Reset = 1;
  #30
  i_Reset = 0;
  #30
  i_A = 21'd9;
  i_B = 21'd16;
  #500
  $stop;
end
```

قسمت سوم:

در این قسمت می خواهیم تابع جذر را پیاده سازی کنیم

تابع قدر مطلق به صورت

$$\sqrt{W} = \sqrt{y^2 - x^2} \qquad X0 = W + 0.25\,, Y0 = W + 0.25$$

با استفاده از الگریتم *CORDIC* پیاده سازی می شود در این قسمت از *Pipeline* با Throughput=*1* نیز استفاده شده است

این تابع به صورت مستقیم از *CORDIC* قابل محاسبه نیست ول می توان با تغییر کوچکی در مقدار اولیه های تابع قدر مطلق ان را بدست اورد

برای این کار مانند قبل عمل می کنیم

ابتدا مقادیر *Tan-1* ها را ذخیره می کنیم

```
// Alphas
parameter [21:0] Alpha0 = 21'd8192; // 45 = tan-1(2**0)
parameter [21:0] Alpha1 = 21'd4836; // tan-1(2**-1)
parameter [21:0] Alpha2 = 21'd2555; // tan-1(2**-2)
parameter [21:0] Alpha3 = 21'd1297; // tan-1(2**-3)
parameter [21:0] Alpha[3:0] = '{Alpha3, Alpha2, Alpha1, Alpha0};
```

سپس برای پیدا کردن ناحیه ی کار چون *x* باید از *Y* بزرگ تر باشد پس بر خلاف قسمت قبل فقط 1 ناحیه داریم که در شکل زیر اورده شده است

```
always@(*)
begin
    begin
      s_x0 = W + 0.25;
      s_y0 = W - 0.25;
      s_z0 = 0;
    end
end
```

سپس برای هر *Iteration* مقادیر مورد نظر را با استفاده از الگوریتم پیدا کرده در مواقع نیاز ان را شیفت داده و یا از مقادیر ثابت *Alpha* استفاده می کنیم و برای ان کنترلر درست را می نویسیم

Summery

```
always@(posedge i_clock)
begin
   case(s_ps)
   idle:
   begin
      s_Count_Times <= 4'd0;
      s_x <= s_x0;
      s_y <= s_y0;
      s_z <= s_z0;
   end
   Calculate:
   begin
      s_Count_Times <= s_Count_Times + 4'd1;
      if(s_y[21])
      begin
         s_z <= s_z - Alpha[s_Count_Times];
         s_x <= s_x - (s_y >> s_Count_Times);
         s_y <= s_y + (s_x >> s_Count_Times);
      end
      else
      begin
         s_z <= s_z - Alpha[s_Count_Times];
         s_x <= s_x - (s_y >> s_Count_Times);
         s_y <= s_y + (s_x >> s_Count_Times);
      end
   end
```

سپس مقدار خروجی را پیدا می کنیم

```
begin
   o_Q <= s_x;
end
```

برای بررسی تابع مورد نظر تست بنچ ان را درست می کنیم

```
module Absulote_tb();
reg i_clock;
reg i_Reset;
reg [21:0] i_W;
wire signed [21:0] o_Q;

SQRT UUTT
   (
      i_clock,
      i_Reset,
      i_W,
      o_Q,
   );

initial i_clock=0;
always #20 i_clock=~i_clock;

initial begin
   i_Reset = 1;
   #30
   i_Reset = 0;
   #30
   i_W = 21'd9;
   #500
   $stop;
end
endmodule
```

قسمت چهار:

در این قسمت می خواهیم همان تابع جذر را که با الگوریتم CORDIC ساختیم را با بسط تیلور ساخته و با ان مقایسه کنیم

برای این کار ابتدا datapath و controller ان را ساخته سپس به هم وصل کرده و تست بنچ ان را می نویسیم

برای datapath ان داریم:

```
input[1:0] SelectM;
output [21:0] o_Q;

wire [21:0] s_Xout,s_Mout,s_Aout,s_Rout;

Register Wreg
(
  i_W,
  s_Xout,
  enW,
  clock,
  reset
);

assign s_Mout = (SelectM == 2'b00) ? 1 : (SelectM == 2'b01) ? i_W << 1 : (SelectM == 2'b10) ? i_W << 3 : (SelectM == 2'b11)

assign s_Aout = s_Rout + s_Mout;

Register Qreg
(
  s_Aout,
  s_Rout,
  enQ,
  clock,
  reset
);

assign o_Q = s_Rout;
```

که در ان Register ماژول چداکانه ای است.

سپس control ان را درست می کنیم پس داریم:

```
always@(ps) begin
  enW=1'b0;enQ=1'b0;SelectM=2'b0;
  case(ps)

    Idle: begin
      enW <= 1'b1;
    end

    First: begin
      SelectM <= 2'b00;
    end

    Second: begin
      SelectM <= 2'b01;
    end

    Third: begin
      SelectM <= 2'b10;
    end

    Fourth: begin
      SelectM <= 2'b11;
      enQ <= 1'b1;
    end

  endcase
end
```

سپس datapath و controllerرا به هم وصل می کنیم:

```
  input i_clock,i_reset;
  input[21:0] i_W;
  output [21:0] o_Q;

  wire s_enW,s_enQ;
  wire[1:0] s_SelectM;

  T_Square DP
  (
    i_clock,
    i_reset,
    s_enW,
    s_enQ,
    s_SelectM,
    i_W,
    o_Q
  );

  Controller Cnt
  (
    s_SelectM,
    s_enW,
    s_enQ,
    i_clock,
    i_reset
  ) ;

endmodule
```

و سپس تست بنچ ان را درست می کنیم:

```
Tseries UT
  (
    i_clock,
    i_reset,
    i_W,
    o_Q
  );

initial i_clock=0;
always #20 i_clock=~i_clock;

initial begin
  i_Reset = 1;
  #30
  i_Reset = 0;
  #30
  i_W = 21'd9;
    #30
  i_W = 21'd16;
    #30
  i_W = 21'd25;
    #30
  i_W = 21'd36;
    #30
  i_W = 21'd49;
    #30
  i_W = 21'd64;
```

و خروجی های ان را با خروجی های مربوط به الگریتم *CORDIC* مقایسه می کنیم:

معایب محاسبه به روش تیلور ان است که ما فقط 4 جمله از ان را در نظر گرفتیم در حالی که این یک تقریب است ولی در *CORDIC* این مقدار به مقدار اصلی بسیار نزدیک تر است و همین طور در روش تیلور *Area* بیشتر و *Component* های بیشتری استفاده شده است که باعث کند تر عمل کردن ان می شود