

به نام خدا



طراحی سیستم‌های دیجیتال

پروژه پایانی - CORDIC

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

استاد:

جناب آقای دکتر بهاروند

نام، نام خانوادگی و شماره دانشجویی اعضای گروه:

علیرضا ایلامی - ۹۷۱۰۱۲۸۶

محمدمتین فتوحی - ۹۷۱۰۶۱۴۳

سید مهدی فقیه - ۹۷۱۰۶۱۹۸

علی قاسمی - ۹۷۱۰۶۲۰۵

امیرمهدی نامجو - ۹۷۱۰۷۲۱۲

محمد رضا یوسف پور - ۹۷۱۰۶۳۲۴

فهرست مطالب

۲	۱	مقدمه
۲	۱.۱	معرفی اجمالی و تاریخچه
۲	۲.۱	هدف الگوریتم
۳	۳.۱	پایه ریاضی
۳	۱.۳.۱	حالت Rotation
۶	۲.۳.۱	حالت Vectoring
۸	۴.۱	شکل دقیق الگوریتم
۱۰	۵.۱	کاربردها
۱۰	۱.۵.۱	سخت افزار
۱۰	۲.۵.۱	نرم افزار
۱۱	۲	پیاده سازی
۱۱	۱.۲	اسامی ماژول ها و اینترفیس های سیستم
۱۱	۱.۱.۲	Vectoring
۱۴	۲.۱.۲	Rotation
۱۷	۲.۲	دیاگرام های بلوکی
۲۶	۳.۲	شرح وظایف ماژول ها
۲۷	۱.۳.۲	Vectoring
۳۲	۲.۳.۲	ROTATION
۳۷	۳	سنتز
۳۷	۱.۳	سنتز Rotation
۳۸	۲.۳	سنتز Vectoring
۴۰		مراجع

۱ مقدمه

۱.۱ معرفی اجمالی و تاریخچه

هدف اصلی ما در این پروژه، طراحی یک واحد CORDIC است. CORDIC مخفف واژه COordinate Rotation DIgital Computer به معنی کامپیوتر دیجیتال چرخش مختصاتی است. این الگوریتم اولین بار در سال ۱۹۵۶ توسط آقای جک ولدر (Jack E. Volder)، که یک مهندس اویونیک (مهندس الکترونیک مرتبط به صنعت هوانوردی) بود، برای سیستم مسیریابی بمب افکن B-۵۸ ابداع شد. هر چند بعضی از ایده‌های استفاده شده در این روش، حتی در قرن هفدهم ذکر شده بودند. ویژگی اصلی این الگوریتم هم این است که امکان پیاده‌سازی آن با عناصر ساده جمع‌کننده و شیفتر دهنده وجود دارد و نیازی به استفاده از واحدهای پیشرفته‌تر نظیر ضرب‌کننده وجود ندارد. همچنین باید توجه کرد که در زمان ساخت و توسعه این الگوریتم، کامپیوتر واژه‌ای برای اشاره به دستگاه محاسبه‌گر بوده است و کامپیوتر با آن مفهوم سیستم متشکل از پردازنده و حافظه و مفاهیم مطرح شده توسط تورینگ، هر چند وجود داشت، اما هنوز کاربرد آن گسترده نشده بود. از این رو نباید لغت کامپیوتر در نام این الگوریتم را با تعریف امروزی آن اشتباه گرفت. [۱] [۲]

۲.۱ هدف الگوریتم

الگوریتم CORDIC در شکل ساده و رایج خود، الگوریتمی برای محاسبه مقادیر توابع مختلف علی‌الخصوص توابع مثلثاتی و هذلولوی است. از الگوریتم CORDIC در مواردی برای محاسبه توابع لگاریتمی، جذر گرفتن و موارد نظیر این هم استفاده می‌شود اما شکل اولیه آن مبتنی بر همان توابع مثلثاتی است. در اصل الگوریتم اصلی که در این جا قصد پیاده‌سازی آن را داریم، دو حالت عملکردی مختلف دارد. حالت Rotation و حالت Vectoring. در حالت Rotation، یک بردار همراستا با محور x و همچنین یک زاویه به الگوریتم داده شده و این الگوریتم، در اثر چرخش‌های متوالی در مراحل پشت سرهم، بردار اولیه را می‌چرخاند تا در نهایت برآیند تمامی این چرخش‌ها، با دقت مشخصی برابر با زاویه داده شده به برنامه بشود. در حالت Vectoring، یک بردار دلخواه داده می‌شود و الگوریتم سعی می‌کند در اثر چرخش‌های متوالی و منظم، مولفه y بردار را از بین ببرد و آن را بر محور x منطبق کند. با این کار، از طریق مولفه x نهایی می‌توان اندازه بردار اولیه را بدست آورد و همچنین با بررسی روند چرخش‌های انجام شده، می‌توان به زاویه بردار اولیه هم پی برد. [۲] [۳] [۴]

۳.۱ پایه ریاضی

۱.۳.۱ حالت Rotation

ابتدا باید به نحوه مدل کردن چرخش یک نقطه در دستگاه مختصات بپردازیم. اگر نقطه $v = (x, y)$ را به اندازه زاویه θ به صورت پادساعتگرد (در جهت مثلثاتی) دوران بدهیم، نقطه $v' = (x', y')$ به صورت زیر بدست می‌آید:

$$x' = x \cos(\theta) - y \sin(\theta)$$

$$y' = x \sin(\theta) + y \cos(\theta)$$

از طرف دیگر، یک دوران به اندازه θ را می‌توان مجموعی از n دوران دیگر $\theta_1, \theta_2, \dots, \theta_n$ دانست که $\theta = \theta_1 + \theta_2 + \dots + \theta_n$

در نتیجه می‌توان برای انجام یک دوران، آن را به چشم تعدادی دوران دیگر دانست و با نماد گذاری نقطه شروع به شکل $v_0 = (x_0, y_0)$ و نتیجه هر کدام از n دوران به صورت $v_i = (x_i, y_i)$ داریم:

$$x_{i+1} = x_i \cos(\theta_{i+1}) - y_i \sin(\theta_{i+1})$$

$$y_{i+1} = x_i \sin(\theta_{i+1}) + y_i \cos(\theta_{i+1})$$

از طرف دیگر، با فاکتور گرفتن از $\cos(\theta_{i+1})$ در عبارات بالا داریم:

$$x_{i+1} = \cos(\theta_{i+1})(x_i - y_i \tan(\theta_{i+1}))$$

$$y_{i+1} = \cos(\theta_{i+1})(x_i \tan(\theta_{i+1}) + y_i)$$

اگر عبارت شامل \cos را کنار بگذاریم، درون پرانتز شاهد یک جمع (تفریق) و یک ضرب هستیم. در سخت افزار ضرب به حالت کلی، هزینه نسبتاً بالایی دارد؛ اما الگوریتم CORDIC سعی می‌کند با ایده هوشمندانه ضرب در توان‌های 2 این مشکل را برطرف کند. زوایایی که قرار است برای چرخش‌ها انتخاب بشوند، باید به این صورت باشند که:

$$\tan(\theta_{i+1}) = \pm 2^{-i}$$

از آن جایی که عملیات‌های ضرب یا تقسیم بر توان‌های 2 در یک سیستم دیجیتالی به راحتی از طریق شیفت دادن قابل انجام است، بدون نیاز به استفاده از ضرب کننده، می‌توانیم عملیات‌ها را انجام بدهیم. نکته مهمی که باقی می‌ماند، کسینوس‌هایی است که در هر مرحله در حال ضرب شدن هستند. ابتدا باید توجه کرد که:

$$\cos(\arctan(2^{-i})) = \frac{1}{\sqrt{1 + 2^{-i}}}$$

در نتیجه، در اصل بسته به این که قرار است این محاسبات تا چند مرحله ادامه پیدا کنند، عدد ضرب شده نهایی به صورت:

$$K = \prod_i \frac{1}{\sqrt{1 + 2^{-2i}}}$$

خواهد بود.

جدول محاسبات مربوط به K در زیر آمده است:

i	2^{-i}	$\arctan(2^{-i})$	$\cos(\arctan(2^{-i}))$	$\prod_i \cos(\arctan(2^{-i}))$
0	1.0000	0.7854	0.7071	0.7071
1	0.5000	0.4636	0.8944	0.6325
2	0.2500	0.2450	0.9701	0.6136
3	0.1250	0.1244	0.9923	0.6088
4	0.0625	0.0624	0.9981	0.6076
5	0.0312	0.0312	0.9995	0.6074
6	0.0156	0.0156	0.9999	0.6073
7	0.0078	0.0078	1.0000	0.6073
8	0.0039	0.0039	1.0000	0.6073
9	0.0020	0.0020	1.0000	0.6073
10	0.0010	0.0010	1.0000	0.6073
11	0.0005	0.0005	1.0000	0.6073
12	0.0002	0.0002	1.0000	0.6073
13	0.0001	0.0001	1.0000	0.6073
14	0.0001	0.0001	1.0000	0.6073
15	0.0000	0.0000	1.0000	0.6073

همان طور که مشخص است، با دقت چهار رقم اعشار سری مذکور به 0.6073 همگراست. با کمک نرم افزارهای ریاضیاتی نظیر Mathematica نیز می توان بررسی کرد که:

$$\lim_{n \rightarrow \infty} \prod_{i=0}^n \frac{1}{\sqrt{1 + 2^{-2i}}} \approx 0.607253 \approx 0.6073$$

همان طور که در بالا دیدیم:

$$\tan(\theta_{i+1}) = \pm 2^{-i}$$

یعنی دو مقدار مثبت و منفی وجود دارد و زاویه هم می تواند به دو شکل مختلف مثبت و منفی باشد. برای این که بدانیم در هر مرحله، باید مقدار مثبت زاویه را اعمال کنیم یا مقدار منفی آن را باید به این توجه کنیم که ما در این جا با زوایایی با مقادیر مشخص سر و کار داریم و در ابتدا هم یک زاویه به عنوان زاویه هدف در نظر داریم که هر بار با اجرای یکی از مراحل، می توانیم زاویه هدف را کاهش داده و به مقدار جدید آپدیت کنیم. حال اگر مقدار زاویه هدف باقی مانده مثبت باشد، باید با کم کردن یک زاویه مثبت، آن را به صفر نزدیک کنیم و اگر منفی باشد، با کم کردن یک زاویه منفی، آن را به صفر نزدیک نماییم. در نتیجه فرمول کلی الگوریتم به صورت زیر در می آید:

$$x_{i+1} = x_i - d_i \cdot y_i \cdot 2^{-i}$$

$$\begin{aligned} y_{i+1} &= y_i + d_i x_i 2^{-i} \\ \theta_{i+1} &= \theta_i - d_i \arctan(2^{-i}) \end{aligned}$$

که در آن اگر z_i نامنفی باشد، مقدار d_i برابر ۱ و در غیر این صورت برابر -1 است. چند نکته در این جا باقی می ماند. اولین مورد این است که بازه برد \arctan در $\frac{\pi}{2} < \theta < \frac{-\pi}{2}$ است. برای این که عملیات چرخش برای زوایای بیش تر از $\frac{\pi}{2}$ و کمتر از $\frac{-\pi}{2}$ هم به خوبی جواب بدهد، باید دوران ناشی از $\frac{\pi}{2}$ ای که این زاویه ها اضافه یا کم دارند را در همان ابتدا اعمال کنیم. برای این کار، اگر $\theta > \frac{\pi}{2}$ باشد، در همان ابتدا تبدیل زیر را اعمال می کنیم:

$$x' = -y, y' = x, \theta' = \theta - \frac{\pi}{2}$$

و اگر $\theta < \frac{-\pi}{2}$ باشد، در همان ابتدا تبدیل:

$$x' = y, y' = -x, \theta' = \theta + \frac{\pi}{2}$$

را اعمال می کنیم و الگوریتم را با x', y', θ' ادامه می دهیم. نکته دیگر مربوط به این است که دو شکل دیگر از الگوریتم CORDIC به نام های Hyperbolic و Linear یعنی هذلولوی و خطی وجود دارند و شکلی که در بالا آن را توصیف کردیم، شکل Circular یا دایروی آن بود. منطق کلی این روش ها مشابه بالا است، با این تفاوت که در شکل هذلولوی، از توابع هذلولوی استفاده شده و در شکل خطی، عملاً به جای $\arctan(2^{-i})$ خود 2^{-i} قرار می گیرد. به جز این، این دو روش تنها در منفی یا مثبت بودن یکسری از عبارات با هم تفاوت دارند که به صورت خلاصه و به شکل متحدالشکل، می توان آن را به صورت زیر نمایش داد.

$$x_{i+1} = x_i - \eta \cdot d_i \cdot y_i \cdot 2^{-i}$$

$$\begin{aligned} y_{i+1} &= y_i + d_i x_i 2^{-i} \\ \theta_{i+1} &= \theta_i - d_i \cdot \xi_i \end{aligned}$$

که η و ξ براساس جدول زیر هستند:

Mode	η	ξ
Linear	0	2^{-i}
Circular	1	$\tan^{-1}(2^{-i})$
Hyperbolic	-1	$\tanh^{-1}(2^{-i})$

در نهایت هم بعد از اعمال همه مراحل، باید x و y نهایی در ضریب K که بالا نوشته شد، ضرب بشوند. این ضریب برای حالت هذلولوی، برابر $K = 1.2075$ و برای حالت خطی $K = 1$ است.

در نهایت لازم به ذکر است که هر چند الگوریتم چرخش، می‌تواند هر بردار اولیه ای را بچرخاند، اما در شکل استاندارد الگوریتم، بردار اولیه به صورت یک و موازی محور x داده می‌شود. در دو جدول زیر، این که ورودی‌ها و خروجی‌های هر کدام از این حالات در وضعیت استاندارد چه هستند، نمایش داده شده است:

جدول ورودی‌های استاندارد:

Mode	x_{input}	y_{input}	θ_{input}
Linear	x	0	θ
Circular	1	0	θ
Hyperbolic	1	0	θ

جدول خروجی‌های استاندارد:

Mode	x_{output}	y_{output}	θ_{output}
Linear	x	$x \times \theta$	0
Circular	$\cos(\theta)$	$\sin(\theta)$	0
Hyperbolic	$\cosh(\theta)$	$\sinh(\theta)$	0

۲.۳.۱ حالت Vectoring

پایه و اساس حالت Vectoring از نظر ریاضیاتی مشابه حالت Rotation است. با این تفاوت که در حالت Rotation ما قصد داشتیم یک بردار را به اندازه زاویه مشخص شده بچرخانیم، اما این جا قصد داریم یک بردار را طوری بچرخانیم که روی محور x منطبق شده و در نتیجه این چرخش‌ها اندازه و زاویه اولیه اش با محور x را پیدا کنیم. از نظر ریاضیاتی، همه مواردی که در بخش قبل گفته شده، در این جا هم برقرار است. فقط در این جا باید توجه کرد که در ابتدای کار، ما زاویه ای در اختیار نداریم و در عوض باید با استفاده از x و y داده شده، این زاویه را پیدا کنیم. مشابه بخش قبل، اساس کلی این الگوریتم هم براساس زوایا و بردارهای ناحیه اول و چهارم مختصات است. در نتیجه، اضافه کردن زوایا، بر این اساس انجام می‌شود که آیا y بردار فعلی، مثبت است یا منفی، اگر منفی باشد، با افزایش زاویه و اگر مثبت باشد، با کاهش زاویه رو به رو هستیم. در اصل تفاوت اصلی این الگوریتم با الگوریتم بخش قبل، در مقدار d_i است که به شکل زیر مشخص می‌شود:

$$d_i = \begin{cases} 1 & \text{if } y_i < 0 \\ -1 & \text{if } y_i \geq 0 \end{cases}$$

معادلات اصلی الگوریتم، همان معادلات قبلی است:

$$x_{i+1} = x_i - \eta \cdot d_i \cdot y_i \cdot 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$\theta_{i+1} = \theta_i - d_i \cdot \xi_i$$

در حالتی که ورودی و خروجی در ناحیه اول یا چهارم نباشند، تبدیل‌هایی مانند تبدیل‌های بخش قبل صورت می‌گیرد، با این تفاوت که در این جا، معیار ما مثبت یا منفی بودن x و y است. اگر x منفی و y مثبت باشد یعنی در ناحیه دوم هستیم و تبدیل زیر اعمال می‌شود:

$$x' = y, y' = -x, \theta' = \frac{\pi}{2}$$

و اگر x منفی و y هم منفی باشد، یعنی در ناحیه سوم هستیم و تبدیل زیر اعمال می‌شود:

$$x' = -y, y' = x, \theta' = \frac{-\pi}{2}$$

توجه داشته باشید که در این حالت، در ابتدا $\theta = 0$ است، چون چیزی در مورد زاویه نمی‌دانیم و هدف پیدا کردن زاویه است. بعد از انجام تبدیلات، الگوریتم را با استفاده از x', y', θ' پی می‌گیریم. جدول ورودی و خروجی‌های استاندارد این حالت به صورت زیر است:

جدول ورودی‌های استاندارد:

Mode	x_{input}	y_{input}	θ_{input}
Linear	x	y	0
Circular	x	y	0
Hyperbolic	x	y	0

جدول خروجی‌های استاندارد:

Mode	x_{output}	y_{output}	θ_{output}
Linear	x	0	$\frac{y}{x}$
Circular	$\sqrt{x^2 + y^2}$	0	$\tan^{-1} \frac{y}{x}$
Hyperbolic	$\sqrt{x^2 - y^2}$	0	$\tanh^{-1} \frac{y}{x}$

منابع استفاده شده برای بخش مربوط به پایه ریاضی: [۳] [۴] [۵]

۴.۱ شکل دقیق الگوریتم

در الگوریتم‌های زیر، θ را با z نمایش داده ایم. الگوریتم‌ها براساس حالت Circular نوشته شده اند ولی تنها تفاوت دو حالت دیگر، همان طور که در بالا هم گفته شده، در علامت مثبت و منفی یکی از عبارات و همچنین استفاده از تابعی متفاوت با \arctan است. ضمناً بهینه سازی‌های پیاده سازی سخت افزاری (نظیر این که عملاً از یک K ثابت استفاده خواهیم کرد) در زیر اعمال نشده و شکل کلی الگوریتم نوشته شده است. [۴]

Algorithm 1: CORDIC Rotation

```

input :  $x_{in}, y_{in}, z_{in}$ :angle,  $n$ :number-of-iterations
output:  $x_{out}, y_{out}$ 
if  $!(-\frac{\pi}{2} < z_{in} < \frac{\pi}{2})$  then
  if  $z_{in} > \frac{\pi}{2}$  then
     $x = -y_{in}$ 
     $y = x_{in}$ 
     $z = z_{in} - \frac{\pi}{2}$ 
  else
     $x = y_{in}$ 
     $y = -x_{in}$ 
     $z = z_{in} + \frac{\pi}{2}$ 
  end
else
   $x = x_{in}$ 
   $y = y_{in}$ 
   $z = z_{in}$ 
end
 $K = 1$ 
for  $i \leftarrow 0$  to  $n$  do
   $d = \text{sgn}(z)$ 
   $x = x - d \cdot y \cdot 2^{-i}$ 
   $y = y + d \cdot x \cdot 2^{-i}$ 
   $z = z + d \cdot \arctan(2^{-i})$ 
   $K = K \cdot \frac{1}{\sqrt{1+2^{-2i}}}$ 
end
 $x_{out} = \frac{x}{K}$ 
 $y_{out} = \frac{y}{K}$ 

```

Algorithm 2: CORDIC Vectoring

input : x_{in} , y_{in} , z_{in} :angle, n :number-of-iterations**output:** x_{out} , z_{out} **if** $x_{in} < 0$ **then** **if** $y_{in} \geq 0$ **then** $x = y_{in}$ $y = -x_{in}$ $z = +\frac{\pi}{2}$ **else** $x = -y_{in}$ $y = x_{in}$ $z = -\frac{\pi}{2}$ **end****else** $x = x_{in}$ $y = y_{in}$ $z = 0$ **end** $K = 1$ **for** $i \leftarrow 0$ **to** n **do** $d = \text{sgn}(y)$ $x = x - d \cdot y \cdot 2^{-i}$ $y = y + d \cdot x \cdot 2^{-i}$ $z = z + d \cdot \arctan(2^{-i})$ $K = K \cdot \frac{1}{\sqrt{1+2^{-2i}}}$ **end** $x_{out} = \frac{x}{K}$ $z_{out} = z$

۵.۱ کاربردها

۱.۵.۱ سخت افزار

یکی از اولین کاربردهای اصلی و واقعی الگوریتم CORDIC، استفاده از آن در سیستم مسیریابی و ناوبری ماه‌نورد ناسا در پروژه Apollo در حدود سال‌های ۱۹۷۱ و ۱۹۷۲ میلادی بوده است که از آن برای سیستم‌های محاسبه زاویه افقی نسبت به اشیاء و همچنین محاسبه فاصله تا سفینه اصلی استفاده شده است. [۶]

به عنوان کاربرد عمومی، CORDIC در سیستم‌هایی که نیاز به محاسبه توابع مثلثاتی داشته باشند، ولی به دلایلی نظیر هزینه، امکان استفاده از ضرب کننده در آن‌ها وجود نداشته باشد، کاربرد به سزایی دارد. در صورتی که بتوان از ضرب کننده استفاده کرد، محاسبه این توابع به کمک بسط تیلور و مک لورن آن‌ها در ترکیب با Lookup Table معمولاً سریع تر از CORDIC است ولی اگر امکان استفاده از ماژول ضرب کننده مجزا نباشد، CORDIC به مراتب نسبت به پیاده سازی نرم افزاری ضرب و سپس استفاده از ضرب برتری دارد و با سرعت بیش تری امکان انجام محاسبات را فراهم می آورد. همچنین در مواقعی که سعی بر کمینه کردن تعداد گیت‌های استفاده شده باشد و از این رو بخواهیم از ضرب کننده استفاده نکنیم (مثلاً در یک سیستم پیاده شده روی FPGA استفاده از CORDIC می تواند مورد توجه قرار بگیرد).

۲.۵.۱ نرم افزار

در دورانی که CPU‌ها واحد Floating-Point مجزا نداشتند و تمامی رجیسترهای آنان، به صورت Integer بودند، شرکت‌های تولید کننده پردازنده در قالب کتابخانه‌های نرم افزاری مربوط به پیاده سازی IEEE Floating Point System که امکان انجام محاسبات Floating-Point را از طریق رجیسترهای Integer مهیا می کرد، الگوریتم CORDIC را هم به صورت نرم افزاری برای محاسبه زوایای مثلثاتی پیاده سازی می کردند، اما بعدها که واحدهای مجزای Floating-Point و محاسبات آن به صورت سخت افزاری به پردازنده‌ها استفاده شد، استفاده از CORDIC به این شیوه منسوخ شد و تنها در سیستم‌هایی که از نظر زمانی و Real-Time بودن محدودیت‌های ویژه‌ای دارند، از آن استفاده می شود. [۷]

۲ پیاده سازی

۱.۲ اسامی ماژول ها و اینترفیس های سیستم

۱.۱.۲ Vectoring

ماژول Quadrant_Corrector
ورودی های آن موارد زیر هستند:

```
input [31:0] x,  
input [31:0] y,  
input [31:0] angle,
```

و خروجی های آن:

```
output reg [31:0] x_out,  
output reg [31:0] y_out,  
output reg [31:0] angle_out
```

ماژول Scaler
ورودی های آن موارد زیر هستند:

```
input [31:0] number,  
input [1:0] mode
```

و خروجی آن:

```
output [31:0] answer
```

ماژول X_Calculator
ورودی های آن موارد زیر هستند:

```
input [31:0] x,  
input [31:0] y,  
input [31:0] angle,  
input [1:0] mode,  
input [31:0] y_shift,  
input clock
```

و خروجی آن:

```
output [31:0] x_out
```

ماژول Y_Calculator
ورودی‌های آن موارد زیر هستند:

```
input [31:0] x,  
input [31:0] y,  
input [31:0] angle,  
input [31:0] x_shift,  
input clock
```

و خروجی آن:

```
output [31:0] y_out
```

ماژول Z_Calculator
ورودی‌های آن موارد زیر هستند:

```
input [31:0] angle,  
input [31:0] y,  
input [31:0] lookup_table_amount,  
input clock,
```

و خروجی آن:

```
output wire [31:0] angle_out
```

ماژول اصلی CORDIC_Vector
ورودی‌های آن:

```
input signed [31:0] x,  
input signed [31:0] y,  
input signed [31:0] angle,  
input [1:0] mode
```

و خروجی‌های آن:

```
output wire signed [31:0] rotated_x,  
output wire signed [31:0] rotated_y,  
output wire signed [31:0] final_angle
```

پارامتر برنامه هم:

```
parameter NUMBER_OF_ITERATIONS = 17;
```

ماژول TopModule
ورودی های آن:

```
input signed [31:0] x,  
input signed [31:0] y,  
input signed [31:0] angle,  
input [1:0] mode,
```

و خروجی های آن:

```
output wire signed [31:0] rotated_x,  
output wire signed [31:0] rotated_y,  
output wire signed [31:0] final_angle
```

Rotation ۲.۱.۲

ماژول Quadrant_Corrector
ورودی‌های آن موارد زیر هستند:

```
input [31:0] x,  
input [31:0] y,  
input [31:0] angle,
```

و خروجی‌های آن:

```
output reg [31:0] x_out,  
output reg [31:0] y_out,  
output reg [31:0] angle_out
```

ماژول Scaler
ورودی‌های آن موارد زیر هستند:

```
input [31:0] number,  
input [1:0] mode
```

و خروجی‌های آن:

```
output [31:0] answer
```

ماژول X_Calculator
ورودی‌های آن موارد زیر هستند:

```
input [31:0] x,  
input [31:0] y,  
input [31:0] angle,  
input [1:0] mode,  
input [31:0] y_shift,  
input clock
```

و خروجی‌های آن:

```
output [31:0] x_out
```

ماژول Y_Calculator
ورودی‌های آن موارد زیر هستند:

```
input [31:0] x,  
input [31:0] y,  
input [31:0] angle,  
input [31:0] x_shift,  
input clock
```

و خروجی آن:

```
output [31:0] y_out
```

ماژول Z_Calculator
ورودی‌های آن موارد زیر هستند:

```
input [31:0] angle,  
input [31:0] y,  
input [31:0] lookup_table_amount,  
input clock,
```

و خروجی آن:

```
output wire [31:0] angle_out
```

ماژول اصلی CORDIC_Rotation

```
input signed [31:0] x,  
input signed [31:0] y,  
input signed [31:0] angle,  
input [1:0] mode
```

و خروجی‌های آن:

```
output wire signed [31:0] rotated_x,  
output wire signed [31:0] rotated_y,  
output wire signed [31:0] final_angle
```

پارامتر برنامه هم:


```
parameter NUMBER_OF_ITERATIONS = 29;
```

ماژول TopModule
ورودی‌های آن:

```
input signed [31:0] x,  
input signed [31:0] y,  
input signed [31:0] angle,  
input [1:0] mode,
```

و خروجی‌های آن:

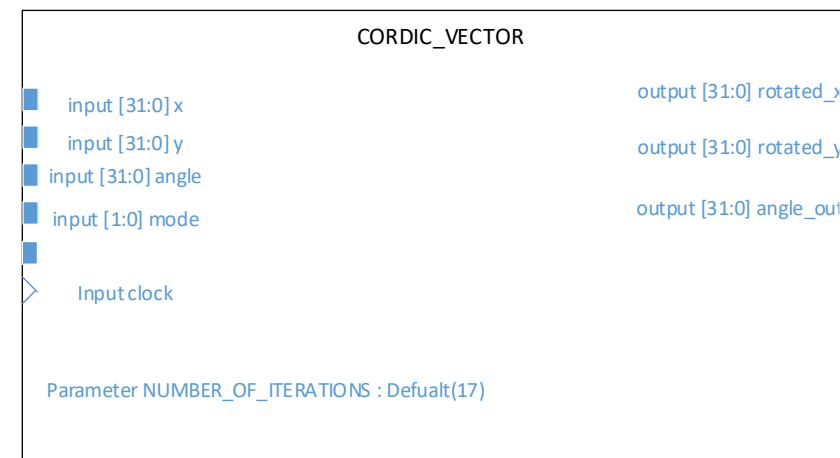
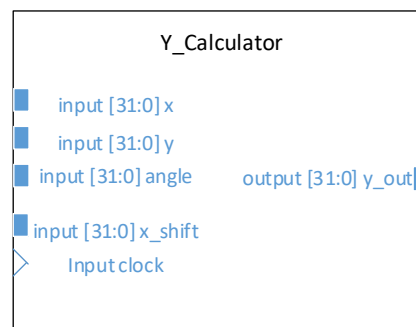
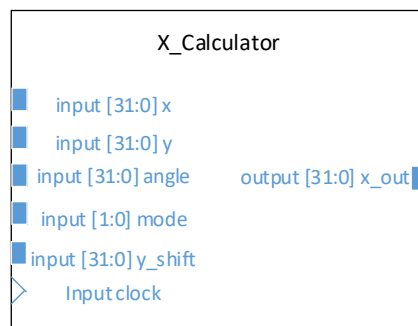
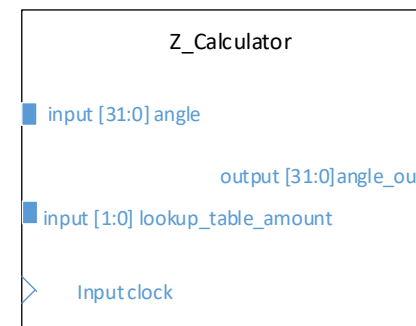
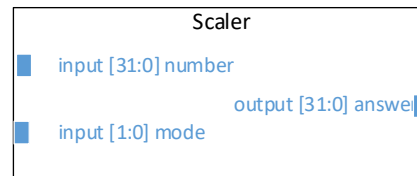
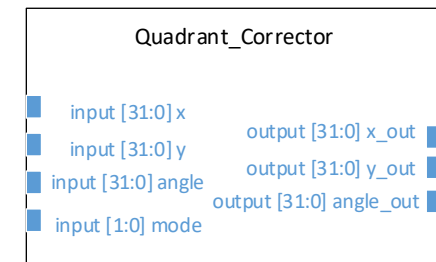
```
output wire signed [31:0] rotated_x,  
output wire signed [31:0] rotated_y,  
output wire signed [31:0] final_angle
```

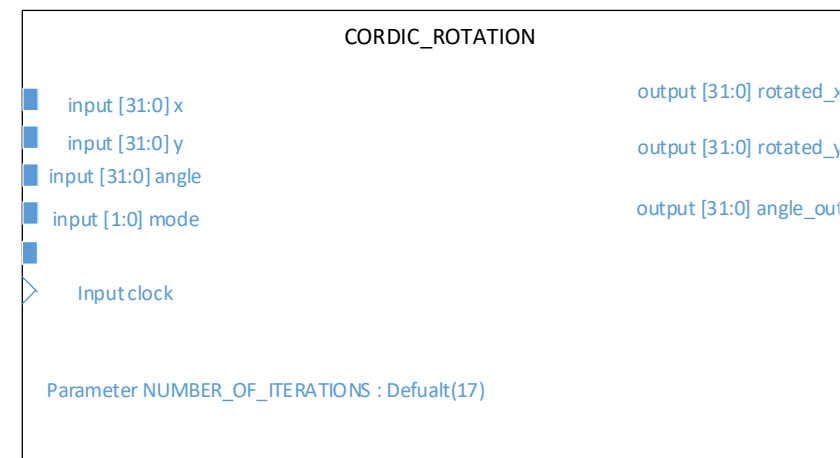
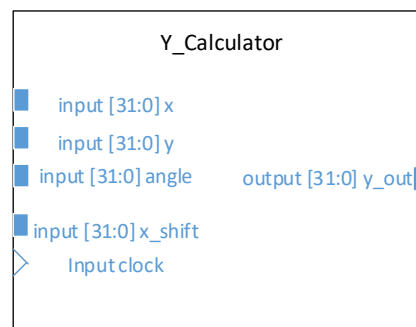
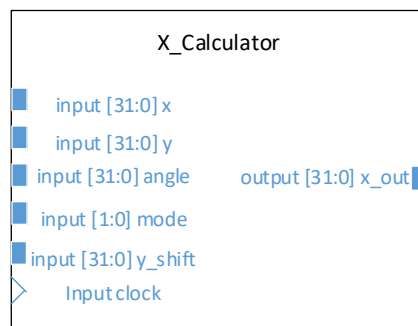
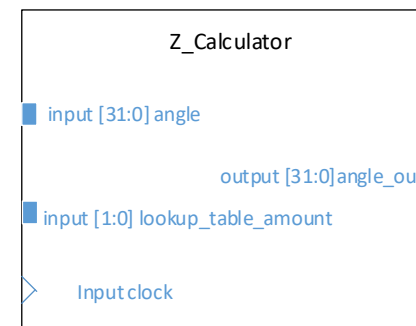
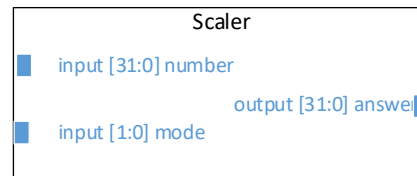
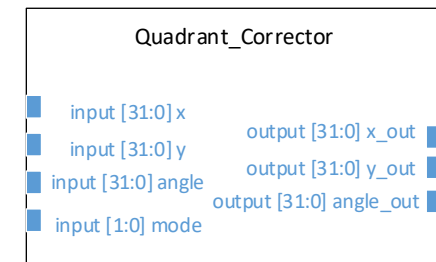
۲.۲ دیاگرام‌های بلوکی

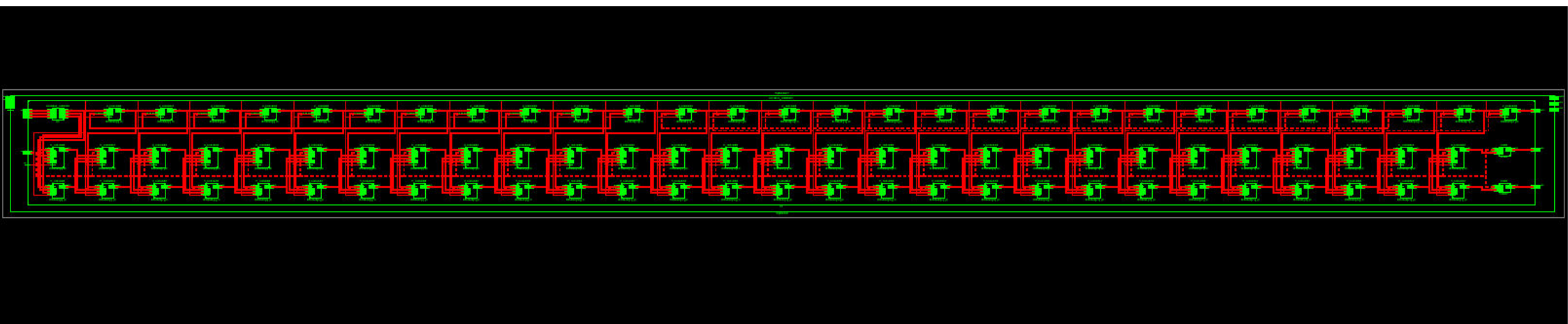
دیاگرام‌های بلوکی از صفحه بعد قرار گرفته اند. به ترتیب ابتدا دیاگرام‌های بلوکی مربوط به Vectoring و سپس Rotation قرار گرفته اند. در صفحه اول هر کدام، ماژول‌ها و ورودی-خروجی‌هایشان قرار گرفته و صفحه بعدی، اتصالات درونی آن‌ها.

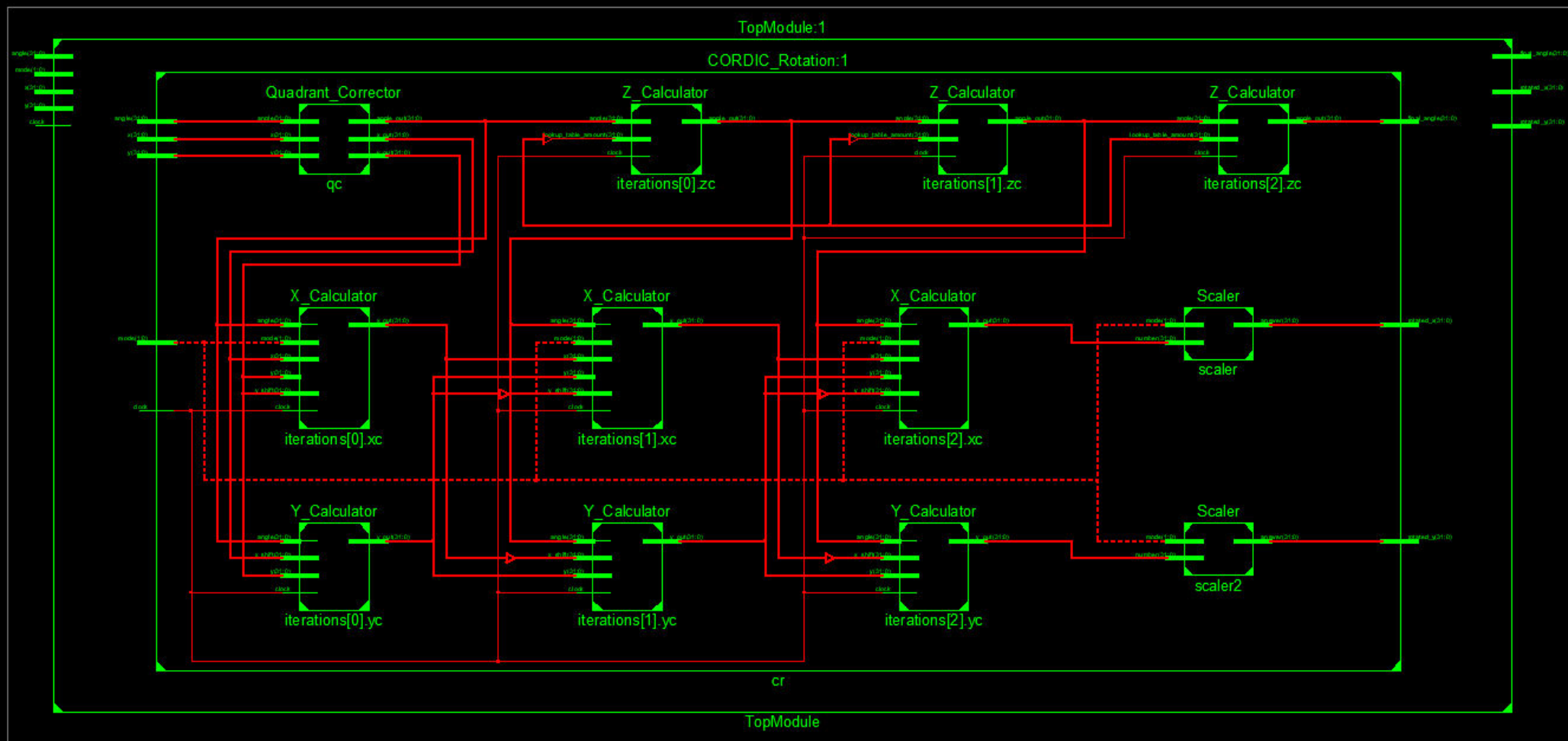
برای نمایش اتصالات درونی و از آن جایی که شکل سنتز شده کامل که شامل تعداد Iteration‌های زیاد باشد، بسیار بزرگ شده و در صفحه امکان نمایش درست آن وجود ندارد، یک شکل با Iteration‌های کم که اتصالات دقیق معلوم هستند و یک شکل از دور برای حالتی که تمامی Iteration‌ها باشند، قرار گرفته است. دیاگرام‌های کلی قرار گرفته به ترتیب زیر هستند:

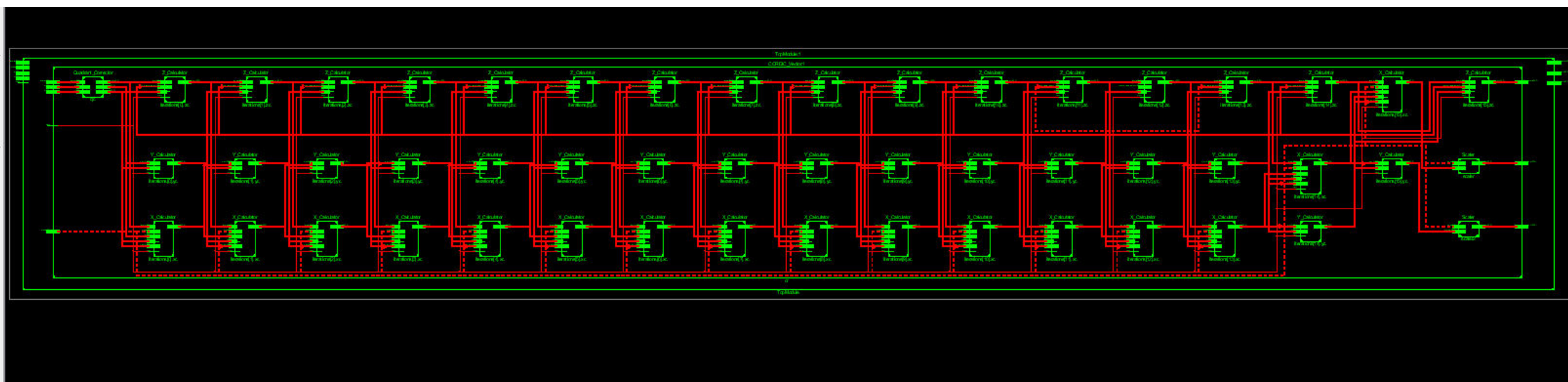
- حالت Rotation با تعداد Iteration کامل
- حالت Rotation با تعداد Iteration کم
- حالت Vectoring با تعداد Iteration کامل
- حالت Vectoring با تعداد Iteration کم











TopModule:1

CORDIC_Vector:1

Quadrant_Corrector

qc

Z_Calculator

iterations[0].zc

Z_Calculator

iterations[1].zc

X_Calculator

iterations[2].xc

Z_Calculator

iterations[2].zc

Y_Calculator

iterations[0].yc

X_Calculator

iterations[1].xc

Y_Calculator

iterations[2].yc

Scaler

scaler

X_Calculator

iterations[0].xc

Y_Calculator

iterations[1].yc

Scaler

scaler2

cr

TopModule

angle[0].m

angle[1].m

angle[2].m

angle[3].m

angle[4].m

angle[5].m

angle[6].m

angle[7].m

angle[8].m

angle[9].m

angle[10].m

angle[11].m

angle[12].m

angle[13].m

angle[14].m

angle[15].m

angle[16].m

angle[17].m

angle[18].m

angle[19].m

angle[20].m

angle[21].m

angle[22].m

angle[23].m

angle[24].m

angle[25].m

angle[26].m

angle[27].m

angle[28].m

angle[29].m

angle[30].m

angle[31].m

angle[32].m

angle[33].m

angle[34].m

angle[35].m

angle[36].m

angle[37].m

angle[38].m

angle[39].m

angle[40].m

angle[41].m

angle[42].m

angle[43].m

angle[44].m

angle[45].m

angle[46].m

angle[47].m

angle[48].m

angle[49].m

angle[50].m

angle[51].m

angle[52].m

angle[53].m

angle[54].m

angle[55].m

angle[56].m

angle[57].m

angle[58].m

angle[59].m

angle[60].m

angle[61].m

angle[62].m

angle[63].m

angle[64].m

angle[65].m

angle[66].m

angle[67].m

angle[68].m

angle[69].m

angle[70].m

angle[71].m

angle[72].m

angle[73].m

angle[74].m

angle[75].m

angle[76].m

angle[77].m

angle[78].m

angle[79].m

angle[80].m

angle[81].m

angle[82].m

angle[83].m

angle[84].m

angle[85].m

angle[86].m

angle[87].m

angle[88].m

angle[89].m

angle[90].m

angle[91].m

angle[92].m

angle[93].m

angle[94].m

angle[95].m

angle[96].m

angle[97].m

angle[98].m

angle[99].m

angle[100].m

angle[101].m

angle[102].m

angle[103].m

angle[104].m

angle[105].m

angle[106].m

angle[107].m

angle[108].m

angle[109].m

angle[110].m

angle[111].m

angle[112].m

angle[113].m

angle[114].m

angle[115].m

angle[116].m

angle[117].m

angle[118].m

angle[119].m

angle[120].m

angle[121].m

angle[122].m

angle[123].m

angle[124].m

angle[125].m

angle[126].m

angle[127].m

angle[128].m

angle[129].m

angle[130].m

angle[131].m

angle[132].m

angle[133].m

angle[134].m

angle[135].m

angle[136].m

angle[137].m

angle[138].m

angle[139].m

angle[140].m

angle[141].m

angle[142].m

angle[143].m

angle[144].m

angle[145].m

angle[146].m

angle[147].m

angle[148].m

angle[149].m

angle[150].m

angle[151].m

angle[152].m

angle[153].m

angle[154].m

angle[155].m

angle[156].m

angle[157].m

angle[158].m

angle[159].m

angle[160].m

angle[161].m

angle[162].m

angle[163].m

angle[164].m

angle[165].m

angle[166].m

angle[167].m

angle[168].m

angle[169].m

angle[170].m

angle[171].m

angle[172].m

angle[173].m

angle[174].m

angle[175].m

angle[176].m

angle[177].m

angle[178].m

angle[179].m

angle[180].m

angle[181].m

angle[182].m

angle[183].m

angle[184].m

angle[185].m

angle[186].m

angle[187].m

angle[188].m

angle[189].m

angle[190].m

angle[191].m

angle[192].m

angle[193].m

angle[194].m

angle[195].m

angle[196].m

angle[197].m

angle[198].m

angle[199].m

angle[200].m

angle[201].m

angle[202].m

angle[203].m

angle[204].m

angle[205].m

angle[206].m

angle[207].m

angle[208].m

angle[209].m

angle[210].m

angle[211].m

angle[212].m

angle[213].m

angle[214].m

angle[215].m

angle[216].m

angle[217].m

angle[218].m

angle[219].m

angle[220].m

angle[221].m

angle[222].m

angle[223].m

angle[224].m

angle[225].m

angle[226].m

angle[227].m

angle[228].m

angle[229].m

angle[230].m

angle[231].m

angle[232].m

angle[233].m

angle[234].m

angle[235].m

angle[236].m

angle[237].m

angle[238].m

angle[239].m

angle[240].m

angle[241].m

angle[242].m

angle[243].m

angle[244].m

angle[245].m

angle[246].m

angle[247].m

angle[248].m

angle[249].m

angle[250].m

angle[251].m

angle[252].m

angle[253].m

angle[254].m

angle[255].m

angle[256].m

angle[257].m

angle[258].m

angle[259].m

angle[260].m

angle[261].m

angle[262].m

angle[263].m

angle[264].m

angle[265].m

angle[266].m

angle[267].m

angle[268].m

angle[269].m

angle[270].m

angle[271].m

angle[272].m

angle[273].m

angle[274].m

۳.۲ شرح وظایف ماژول‌ها

در فایل‌های قرار داده شده، دو پوشه جدا داریم. یکی Rotation و دیگری Vectoring که هر کدام مختص به پروژه مربوط به خود هستند. البته ساختار بسیاری از ماژول‌های آنان، مشابه است اما از آن جایی که دو پروژه مجزا بودند، ما هم آنان را به صورت دو پوشه و پروژه مجزا قرار داده ایم. اما پیش از توضیح در مورد خود ماژول‌ها، باید در مورد نحوه انکود کردن اعداد به صورت باینری توضیح داده بشود. برای انکود کردن زاویه‌ها، آن‌ها را بر 360 تقسیم کرده و سپس در 2^{31} ضرب می‌کنیم و عدد حاصل را به صورت 32 بیتی نمایش می‌دهیم. یعنی با فرمول زیر:

$$\frac{\text{Angle}}{360} \times 2^{31} \rightarrow 32 \text{ bit binary number}$$

مثلاً زوایای 45 درجه، 135 درجه، 225 درجه و 315 درجه، به شکل زیر می‌شوند:

```
45 deg: 32'b00010000000000000000000000000000
135 deg: 32'b00110000000000000000000000000000
225 deg: 32'b01010000000000000000000000000000
315 deg: 32'b01110000000000000000000000000000
```

برای تبدیل برعکس آن هم عدد باینری را به صورت یک Integer در نظر می‌گیریم و عدد بدست آمده را بر 2^{31} تقسیم کرده و ضربدر 360 می‌کنیم. دلیل این که ضرب را در 2^{31} انجام دادیم و نه 2^{32} ، این است که در حالت Linear نیاز به تبدیل عدد 2^0 به مقیاس انکود شده در بالا بدست می‌آمده است و اگر در 2^{32} ضرب انجام می‌شد، با مشکل Overflow رو به رو می‌شدیم. اعداد x و y هم به شکل Fixed-Point با 12 رقم صحیح و 20 رقم اعشاری انکود شده‌اند. در نتیجه برای تبدیل یک عدد به حالت نمایش داده شده 32 بیتی باید از فرمول زیر استفاده کرد:

$$\text{Number} \times 2^{20} \rightarrow 32 \text{ bit binary number}$$

و برای تبدیل برعکس هم باید عدد باینری 32 بیتی گرفته شده را به صورت Integer در نظر گرفته و بر 2^{20} تقسیم کرد.

مثلاً عدد 120، 2 و 170.5 در این انکودینگ به شکل زیر نمایش داده می‌شود.

```
120: 32'b000001111000\_00000000000000000000
2: 32'b000000000010\_00000000000000000000
170.5: 32'b000010101010\_10000000000000000000
```

Vectoring ۱.۳.۲

قبل از اشاره به ماژول‌های اصلی پروژه، می‌بایست به فایل ثوابت یا CONSTANTS.v اشاره کنیم. در این فایل از طریق 'define' سه ثابت به صورت زیر تعریف شده است:

```
`define CIRCULAR 2'b01
`define LINEAR 2'b00
`define HYPERBOLIC 2'b11
```

این فایل از طریق 'include' در سایر ماژول‌ها قرار گرفته است. ماژول‌ها را در ادامه به ترتیب Bottom-Up شرح می‌دهیم. اولین ماژول Quadrant_Corrector است که در فایل هم نام خودش قرار دارد. ورودی‌های آن موارد زیر هستند:

```
input [31:0] x,
input [31:0] y,
input [31:0] angle,
```

و خروجی‌های آن:

```
output reg [31:0] x_out,
output reg [31:0] y_out,
output reg [31:0] angle_out
```

وظیفه این ماژول این است که اگر x و y ورودی در ناحیه اول یا چهارم قرار نداشتند، از طریق تبدیل‌هایی که در بخش ریاضی نوشتیم، آن‌ها را به ناحیه اول یا چهارم منتقل کرده و برای حالتی که در ناحیه دوم باشند، زاویه اولیه خروجی را 90 درجه و در حالتی که در ناحیه سوم باشند، زاویه اولیه خروجی را 270 درجه (معادل 90- درجه) قرار بدهد.

ماژول بعدی Scaler است که در فایل هم نام خودش قرار دارد. ورودی‌های آن موارد زیر هستند:

```
input [31:0] number,
input [1:0] mode
```

و خروجی آن:

```
output [31:0] answer
```

است. ورودی mode بیانگر این است که وضعیت برنامه در حالت CIRCULAR یا HYPERBOLIC یا LINEAR قرار دارد و براساس ثابت های CONSTANTS.v مشخص می شود. وظیفه کلی این ماژول این است که عدد داده شده را متناسب با حالت داده شده در ضریب K مناسب ضرب کند. برای حالت CIRCULAR این K برابر 0.6073 برای HYPERBOLIC برابر 1.2075 و در حالت LINEAR برابر 1 است. برای این که در ضرب کردن، از ماژول ضرب کننده استفاده نکنیم، برای ضرب در این اعداد از ترکیب شیفت و جمع استفاده می کنیم. به عنوان مثال

$$0.6073 \approx 2^{-1} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-7} + 2^{-10} + 2^{-11} + 2^{-12} + 2^{-13}$$

است در نتیجه، این جمع را به این شکل انجام می دهیم:

```
(number >>> 1) + (number >>> 4) + (number >>> 5)
+ (number >>> 7) + (number >>> 8) + (number >>> 10) +
(number >>> 11) + (number >>> 12) + (number >>> 13);
```

در مورد 1.2075 هم روش مشابهی اعمال می کنیم، با این تفاوت که وجود 1 در عدد 1.2075 یعنی یکی از بخش های جمع شونده، خود عدد اصلی بدون هیچ نوع شیفت اضافی است.

ماژول بعدی X_Calculator است که در فایل هم نام خودش قرار دارد. ورودی های آن موارد زیر هستند:

```
input [31:0] x,
input [31:0] y,
input [31:0] angle,
input [1:0] mode,
input [31:0] y_shift,
input clock
```

و خروجی آن:

```
output [31:0] x_out
```

سه ورودی اول که مشخص هستند. ورودی چهارم مربوط به همین است که در حالت CIRCULAR هستیم یا HYPERBOLIC یا LINEAR. ورودی پنجم که y_shift است، در اصل همان مقدار $y_i \times 2^{-i}$ در فرمول هاست که از ماژول اصلی به عنوان ورودی وارد این ماژول می شود. در نهایت هم clock را داریم که برای اجرای مرحله به مرحله سیستم به صورت Pipeline، وجود کلاک که باعث ذخیره شدن و باقی ماندن مقادیر در رجیستر ها بشود، ضروری است. درون این ماژول براساس ورودی های داده شده، محاسبات طبق فرمول های بخش ریاضی و براساس علامت y انجام شده و مقدار جدید x تحت نام x_out خروجی داده می شود.

ماژول بعدی Y_Calculator است که در فایل هم نام خودش قرار دارد. ورودی های آن موارد زیر هستند:

```
input [31:0] x,
input [31:0] y,
input [31:0] angle,
input [31:0] x_shift,
input clock
```

و خروجی آن:

```
output [31:0] y_out
```

این ماژول از نظر ورودی‌ها کاملاً مشابه X_Calculator است، با این تفاوت که از آن جایی که نیازی به جز این که در این جا نیازی به دانستن mode نداریم و به عنوان ورودی داده نشده است. هدف این ماژول هم انجام محاسبات مربوط به y است.

ماژول بعدی Z_Calculator است که در فایل هم نام خودش قرار دارد.
ورودی‌های آن موارد زیر هستند:

```
input [31:0] angle,
input [31:0] y,
input [31:0] lookup_table_amount,
input clock,
```

و خروجی آن:

```
output wire [31:0] angle_out
```

این ماژول هم از نظر عملکردی، مشابه دو ماژول قبلی است. تنها نکته این جاست که یک مقدار تحت نام $\arctan(2^{-1})$ یا 2^{-i} مقدار برابر با 2^{-i} در اصل، مقدار $\arctanh(2^{-1})$ است که از طریق یک تابع در ماژول اصلی تولید شده و به بسته به مود کلی سیستم، به عنوان ورودی به این ماژول که محاسبه کننده Z یا همان $angle$ و زاویه بعدی است، داده می‌شود.

فایل ماژول اصلی این پروژه، CORDIC_Vector نام دارد. ورودی‌های آن موارد زیر هستند:

```
input signed [31:0] x,
input signed [31:0] y,
input signed [31:0] angle,
input [1:0] mode
```

و خروجی‌های آن:

```
output wire signed [31:0] rotated_x,
output wire signed [31:0] rotated_y,
output wire signed [31:0] final_angle
```

هستند. دلیل این که ۳۲ بیتی در نظر گرفتیم، این بود که در حالت ۱۶ بیتی، برای بسیاری از ورودی‌ها خروجی برنامه دقت کافی را نداشت. پارامتر برنامه هم:

```
parameter NUMBER_OF_ITERATIONS = 17;
```

است. مقدار پیش فرض ۱۷ از این رو قرار گرفته است که بهترین نتیجه‌ای که در تست‌های مختلف دریافت کردیم، برای ورودی و خروجی ۳۲ بیتی، در حالت Vectoring مقدار ۱۷، نتیجه مناسبی ارائه می‌داد و مقادیر بیش‌تر بعضاً منجر به ایجاد Overflow در محاسبات شده و مقادیر کمتر هم دقت را کاهش می‌دادند.

ابتدا به تعداد Iteration های برنامه، یک آرایه از Vector های ۳۲ بیتی Wire برای ایجاد ارتباطات میان خروجی $x, y, angle$ مراحل مختلف ایجاد کرده ایم که x_prime , y_prime و $rotated_angles$ نام دارند.

در ابتدا، یک Instance از ماژول Quadrant_Corrector ساخته شده که وضعیت زاویه و ورودی‌های اصلی برنامه را مشخص کند و مقادیر خروجی آن، به مقادیر اندیس x_prime , y_prime و $rotated_angles$ وصل می‌شوند.

سپس از طریق Generate به تعداد Iteration هایی که از طریق پارامتر تعریف شده است، instance از ماژول $X_Calculator$, $Y_Calculator$ و $Z_Calculator$ ساخته ایم. یکسری متغیر $temp$ هم وجود دارند که مقادیر مربوط به \arctan یا $\operatorname{arctanh}$ و همچنین شیفت داده شده X و Y به اندازه i را که اندیس همان مرحله باشد، به عنوان ورودی به این ماژول‌ها متصل کنند.

بعد از بخش Generate، دو Instance از Scaler ساخته شده که خروجی‌های مرحله آخر ماژول‌ها که یعنی اندیس $NUMBER_OF_ITERATIONS - 1$ ام x_prime , y_prime به آن‌ها وصل شده تا متناسب با mode با ضریب K مشخص Scale بشوند.

در نهایت این خروجی‌ها به عنوان خروجی اصلی برنامه داده شده‌اند.

در انتهای این کد، یک تابع به نام Lookup وجود دارد که Index و mode را ورودی گرفته و براساس آن، مقدار متناسب را از بین جدول \arctan یا $\operatorname{arctanh}$ یا 2^{-i} خروجی می‌دهد. دلیل این که از این تابع استفاده کردیم، این است که در اصل مقادیر این تابع، به عنوان یکسری ثابت که از طریق Mux انتخاب می‌شوند، در حین Instantiate شدن ماژول‌های درون Generate به آن‌ها داده می‌شوند و نیازی نیست که از یک ROM جداگانه استفاده کنیم. زیرا در صورت استفاده از ROM باید امکان خوانده شدن همزمان حدود ۳۲ مقدار مختلف را در بدترین حالت برای آن فراهم می‌کردیم تا همه مراحل سیستم بتوانند به صورت Pipeline و مستقل از هم کار بکنند که هزینه اجرایی بالایی داشت و از این رو آن را به صورت تابع پیاده سازی کردیم.

در نهایت یک Top Module با نام TopModule داریم که ورودی‌های آن به صورت زیر است:

```
input signed [31:0] x,  
input signed [31:0] y,  
input signed [31:0] angle,  
input [1:0] mode,
```

و خروجی‌های آن:

```
output wire signed [31:0] rotated_x,  
output wire signed [31:0] rotated_y,  
output wire signed [31:0] final_angle
```

در این ماژول فقط یک Instance از CORDIC_Vector با اندازه پارامتر مشخص ساخته شده که در سنتز و تست بنچ مورد استفاده قرار بگیرد.

ROTATION ۲.۳.۲

بخش زیادی از ماژول‌های استفاده شده در این بخش، مشابه بخش قبل است اما برای کامل بودن گزارش، توضیحات آن‌ها مجدداً مانند بخش قبل آورده شده است. قبل از اشاره به ماژول‌های اصلی پروژه، می‌بایست به فایل ثوابت یا CONSTANTS.v اشاره کنیم. در این فایل از طریق 'define' سه ثابت به صورت زیر تعریف شده است:

```
`define CIRCULAR 2'b01
`define LINEAR 2'b00
`define HYPERBOLIC 2'b11
```

این فایل از طریق 'include' در سایر ماژول‌ها قرار گرفته است. ماژول‌ها را در ادامه به ترتیب Bottom-Up شرح می‌دهیم. اولین ماژول Quadrant_Corrector است که در فایل هم نام خودش قرار دارد. ورودی‌های آن موارد زیر هستند:

```
input [31:0] x,
input [31:0] y,
input [31:0] angle,
```

و خروجی‌های آن:

```
output reg [31:0] x_out,
output reg [31:0] y_out,
output reg [31:0] angle_out
```

وظیفه این ماژول این است که اگر زاویه داده شده در بازه $-\frac{\pi}{2} < angle < \frac{\pi}{2}$ یا به بیان دیگر، $270 < angle \leq 360$ یا $0 \leq angle < 90$ قرار نداشت، یعنی تبدیل متناسب با ناحیه اول و چهارم قرار نداشت، از طریق تبدیل‌هایی که در بخش ریاضی نوشتیم، آن‌ها را به ناحیه اول یا چهارم منتقل کرده و برای حالتی که در ناحیه دوم باشند، زاویه اولیه خروجی را 90 درجه کم کرده و در حالتی که در ناحیه سوم باشند، زاویه اولیه خروجی را 90 درجه افزایش بدهد (به بیان دیگر 270 درجه کم کند) تا در ناحیه چهارم قرار بگیرد و متناسب با آن تغییرات را روی x و y هم اعمال کند.

ماژول بعدی Scaler است که در فایل هم نام خودش قرار دارد. ورودی‌های آن موارد زیر هستند:

```
input [31:0] number,
input [1:0] mode
```

و خروجی‌های آن:

output [31:0] answer

است. ورودی mode بیانگر این است که وضعیت برنامه در حالت CIRCULAR یا HYPERBOLIC یا LINEAR قرار دارد و براساس ثابت های CONSTANTS.v مشخص می شود. وظیفه کلی این ماژول این است که عدد داده شده را متناسب با حالت داده شده در ضرب K مناسب ضرب کند. برای حالت CIRCULAR این K برابر 0.6073 برای HYPERBOLIC برابر 1.2075 و در حالت LINEAR برابر 1 است. برای این که در ضرب کردن، از ماژول ضرب کننده استفاده نکنیم، برای ضرب در این اعداد از ترکیب شیفت و جمع استفاده می کنیم. به عنوان مثال

$$0.6073 \approx 2^{-1} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-7} + 2^{-10} + 2^{-11} + 2^{-12} + 2^{-13}$$

است در نتیجه، این جمع را به این شکل انجام می دهیم:

```
(number >>> 1) +(number >>>4) + (number>>>5)
+ (number>>>7) + (number>>>8) + (number>>>10) +
(number>>>11) + (number>>>12) + (number>>>13);
```

در مورد 1.2075 هم روش مشابهی اعمال می کنیم، با این تفاوت که وجود 1 در عدد 1.2075 یعنی یکی از بخش های جمع شونده، خود عدد اصلی بدون هیچ نوع شیفت اضافی است.

ماژول بعدی X_Calculator است که در فایل هم نام خودش قرار دارد.
ورودی های آن موارد زیر هستند:

```
input [31:0] x,
input [31:0] y,
input [31:0] angle,
input [1:0] mode,
input [31:0] y_shift,
input clock
```

و خروجی آن:

output [31:0] x_out

سه ورودی اول که مشخص هستند. ورودی چهارم مربوط به همین است که در حالت CIRCULAR هستیم یا HYPERBOLIC یا LINEAR. ورودی پنجم که y_shift است، در اصل همان مقدار $y_i \times 2^{-i}$ در فرمول هاست که از ماژول اصلی به عنوان ورودی وارد این ماژول می شود. در نهایت هم clock را داریم که برای اجرای مرحله به مرحله سیستم به صورت Pipeline، وجود کلاک که باعث ذخیره شدن و باقی ماندن مقادیر در رجیسترها بشود، ضروری است.

درون این ماژول براساس ورودی های داده شده، محاسبات طبق فرمول های بخش ریاضی براساس علامت $angle$ که با توجه به فرمت خاصی که برای انکود کردن انتخاب کردیم، در بیت 31 ام آن (اندیس 30) قرار دارد، انجام شده و مقدار جدید x تحت نام x_out خروجی داده می شود.

ماژول بعدی Y_Calculator است که در فایل هم نام خودش قرار دارد.
ورودی های آن موارد زیر هستند:

```
input [31:0] x,
input [31:0] y,
input [31:0] angle,
input [31:0] x_shift,
input clock
```

و خروجی آن:

```
output [31:0] y_out
```

این ماژول از نظر ورودی ها کاملاً مشابه X_Calculator است، با این تفاوت که از آن جایی که نیازی به جز این که در این جا نیازی به دانستن mode نداریم و به عنوان ورودی داده نشده است. هدف این ماژول هم انجام محاسبات مربوط به y است.

ماژول بعدی Z_Calculator است که در فایل هم نام خودش قرار دارد.
ورودی های آن موارد زیر هستند:

```
input [31:0] angle,
input [31:0] y,
input [31:0] lookup_table_amount,
input clock,
```

و خروجی آن:

```
output wire [31:0] angle_out
```

این ماژول هم از نظر عملکردی، مشابه دو ماژول قبلی است. تنها نکته این جاست که یک مقدار تحت نام `lookup_table_amount` به آن ورودی داده می شود که در اصل، مقداری برابر با 2^{-i} یا $\arctan(2^{-1})$ یا $\operatorname{arctanh}(2^{-1})$ است که از طریق یک تابع در ماژول اصلی تولید شده و به بسته به مود کلی سیستم، به عنوان ورودی به این ماژول که محاسبه کننده Z یا همان $angle$ و زاویه بعدی است، داده می شود.

فایل ماژول اصلی این پروژه، CORDIC_Rotation نام دارد. ورودی های آن موارد زیر هستند:

```
input signed [31:0] x,
input signed [31:0] y,
input signed [31:0] angle,
input [1:0] mode
```

و خروجی های آن:

```
output wire signed [31:0] rotated_x,
output wire signed [31:0] rotated_y,
output wire signed [31:0] final_angle
```

هستند. دلیل این که ۳۲ بیتی در نظر گرفتیم، این بود که در حالت ۱۶ بیتی، برای بسیاری از ورودی ها خروجی برنامه دقت کافی را نداشت. پارامتر برنامه هم:

```
parameter NUMBER_OF_ITERATIONS = 29;
```

ابتدا به تعداد Iteration های برنامه، یک آرایه از Vector های ۳۲ بیتی Wire برای ایجاد ارتباطات میان خروجی $x, y, angle$ مراحل مختلف ایجاد کرده ایم که x_prime, y_prime و $rotated_angles$ نام دارند.

در ابتدا، یک Instance از ماژول Quadrant_Corrector ساخته شده که وضعیت زاویه و ورودی های اصلی برنامه را مشخص کند و مقادیر خروجی آن، به مقادیر اندیس صفر x_prime, y_prime و $rotated_angles$ وصل می شوند.

سپس از طریق Generate به تعداد Iteration هایی که از طریق پارامتر تعریف شده است، instance از ماژول $X_Calculator, Y_Calculator$ و $Z_Calculator$ ساخته ایم. یکسری متغیر temp هم وجود دارند که مقادیر مربوط به \arctan یا $\operatorname{arctanh}$ و همچنین شیفت داده شده X و Y به اندازه i را که اندیس همان مرحله باشد، به عنوان وردی به این ماژول ها متصل کنند.

بعد از بخش Generate، دو Instance از Scaler ساخته شده که خروجی های مرحله آخر ماژول ها که یعنی اندیس $NUMBER_OF_ITERATIONS - 1$ ام x_prime, y_prime به آن ها وصل شده تا متناسب با mode با ضریب K مشخص Scale بشوند.

در نهایت این خروجی ها به عنوان خروجی اصلی برنامه داده شده اند. در انتهای این کد، یک تابع به نام Lookup وجود دارد که Index و mode را ورودی گرفته و براساس آن، مقدار متناسب را از بین جدول \arctan یا $\operatorname{arctanh}$ یا 2^{-i} خروجی می دهد. دلیل این که از تابع استفاده کردیم، این است که در اصل مقادیر این تابع، به عنوان یکسری ثابت که از طریق Mux انتخاب می شوند، در حین Instantiate شدن ماژول های درون Generate به آن ها داده می شوند و نیازی نیست که از یک ROM جداگانه استفاده کنیم. زیرا در صورت استفاده از ROM باید امکان خوانده شدن همزمان حدود ۳۲ مقدار مختلف را در بدترین حالت برای آن فراهم می کردیم تا همه مراحل سیستم بتوانند به صورت Pipeline و مستقل از هم کار بکنند که هزینه اجرایی بالایی داشت و از این رو آن را به صورت تابع پیاده سازی کردیم.

در نهایت یک Top Module با نام TopModule داریم که ورودی های آن به صورت زیر است:

```
input signed [31:0] x,
input signed [31:0] y,
input signed [31:0] angle,
input [1:0] mode,
```

و خروجی های آن:

```
output wire signed [31:0] rotated_x,  
output wire signed [31:0] rotated_y,  
output wire signed [31:0] final_angle
```

در این ماژول فقط یک Instance از CORDIC_Rotation با اندازه پارامتر مشخص ساخته شده که در سنتز و تست بنچ مورد استفاده قرار بگیرد.

۳ سنتز

برای سنتز، از نرم افزار Xilinx ISE استفاده کردیم و در تنظیمات برنامه، سنتز را روی FPGA با مشخصات: Spartan6 XC6SLX150T و پکیج FG484 انجام دادیم. سنتز پروژه اول و دوم به صورت جداگانه انجام گرفت. برای سنتز در هر پروژه، ماژول TopModule به عنوان Top Module انتخاب شده است.

۱.۳ سنتز Rotation

نتایج اصلی سنتز به صورت زیر است. فایل های مربوطه نیز به طور کامل ضمیمه گزارش کار شده است: نتایج اصلی تاخیرها و زمان های مربوط به کلاک:

Minimum period: 3.960ns (Maximum Frequency: 252.534MHz)
 Minimum input arrival time before clock: 6.375ns
 Maximum output required time after clock: 11.079ns
 Maximum combinational path delay: 9.252ns

نتایج اصلی مربوط به FlipFlop ها و LUT ها و میزان استفاده از Slice های FPGA و به طور کلی، Utilization و مساحت مورد استفاده:

Slice Logic Utilization:

Number of Slice Registers:	2,926 out of 184,304	1%
Number used as Flip Flops:	2,700	
Number used as Latches:	0	
Number used as Latch-thrus:	0	
Number used as AND/OR logics:	226	
Number of Slice LUTs:	5,205 out of 92,152	5%
Number used as logic:	5,198 out of 92,152	5%
Number using 06 output only:	4,868	
Number using 05 output only:	94	
Number using 05 and 06:	236	
Number used as ROM:	0	
Number used as Memory:	0 out of 21,680	0%
Number used exclusively as route-thrus:	7	
Number with same-slice register load:	0	
Number with same-slice carry load:	7	
Number with other load:	0	

Slice Logic Distribution:

Number of occupied Slices:	1,369 out of	23,038	5%
Number of MUXCYs used:	4,160 out of	46,076	9%
Number of LUT Flip Flop pairs used:	5,214		
Number with an unused Flip Flop:	2,289 out of	5,214	43%
Number with an unused LUT:	9 out of	5,214	1%
Number of fully used LUT-FF pairs:	2,916 out of	5,214	55%
Number of unique control sets:	2		
Number of slice register sites lost to control set restrictions:	4 out of	184,304	1%

۲.۳ سنتر Vectoring

نتایج اصلی سنتر به صورت زیر است. فایل‌های مربوطه نیز به طور کامل ضمیمه گزارش کار شده است:
نتایج اصلی تاخیرها و زمان‌های مربوط به کلاک:

Minimum period: 4.003ns (Maximum Frequency: 249.799MHz)

Minimum input arrival time before clock: 7.763ns

Maximum output required time after clock: 10.936ns

Maximum combinational path delay: 9.147ns

نتایج اصلی مربوط به FlipFlop ها و LUT ها و میزان استفاده از Slice های FPGA و به طور کلی، Utilization و مساحت مورد استفاده:

Slice Logic Utilization:

Number of Slice Registers:	1,752 out of	184,304	1%
Number used as Flip Flops:	1,526		
Number used as Latches:	0		
Number used as Latch-thrus:	0		
Number used as AND/OR logics:	226		
Number of Slice LUTs:	3,271 out of	92,152	3%
Number used as logic:	3,265 out of	92,152	3%
Number using 06 output only:	2,903		
Number using 05 output only:	94		

Number using 05 and 06:	268			
Number used as ROM:	0			
Number used as Memory:	0 out of	21,680	0%	
Number used exclusively as route-thrus:	6			
Number with same-slice register load:	0			
Number with same-slice carry load:	6			
Number with other load:	0			
Slice Logic Distribution:				
Number of occupied Slices:	866 out of	23,038	3%	
Number of MUXCYs used:	2,624 out of	46,076	5%	
Number of LUT Flip Flop pairs used:	3,276			
Number with an unused Flip Flop:	1,524 out of	3,276	46%	
Number with an unused LUT:	5 out of	3,276	1%	
Number of fully used LUT-FF pairs:	1,747 out of	3,276	53%	
Number of unique control sets:	2			
Number of slice register sites lost to control set restrictions:	2 out of	184,304	1%	

مراجع

- [1] Volder, Jack. The birth of cordic. *VLSI Signal Processing*, 25:101–105, 06 2000.
- [2] Volder, Jack E. The cordic computing technique. in *IRE-AIEE-ACM '59 (Western)*, 1959.
- [3] Boppana, Lakshmi and Dhar, Anindya. Cordic architectures: a survey. *VLSI Design*, 2010, 03 2010.
- [4] Andraka, Ray. A survey of cordic algorithms for fpga based computers. *ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA*, 12 2001.
- [5] Valls, Javier, Kuhlmann, Martin, and Parhi, Keshab. Evaluation of cordic algorithms for fpga design. *VLSI Signal Processing*, 32:207–222, 11 2002.
- [6] Heffron, W. and Piana, F. The navigation system of the lunar roving vehicle. 01 1971.
- [7] Wikipedia. Cordic, 2020.