به نام خدا

طراحی سیستم‌های دیجیتال

پروژه پایانی ـ CORDIC

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

استاد:

جناب آقای دکتر بهاروند

نام، نام خانوادگی و شماره دانشجویی اعضای گروه:

علیرضا ایلامی ـ ۹۷۱۰۱۲۸۶

محمدمتین فتوحی ـ ۹۷۱۰۶۱۴۳

سید مهدی فقیه ـ ۹۷۱۰۶۱۹۸

علی قاسمی ـ ۹۷۱۰۶۲۰۵

امیرمهدی نامجو ـ ۹۷۱۰۷۲۱۲

محمدرضا یوسف پور ـ ۹۷۱۰۶۳۲۴

فهرست مطالب

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 بمب‌افکن 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 جمع کننده و شیفت دهنده وجود دارد و نیازی به استفاده از واحدهای پیشرفته‌تر نظیر ضرب کننده 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، مؤلفه y بردار را از بین ببرد و آن را بر محور x منطبق کند. با این کار، از طریق مؤلفه x نهایی می‌توان اندازه بردار اولیه را به دست 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 به دست 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 $\theta_1, \theta_2, \cdots, \theta_n$ دانست که $\theta = \theta_1 + \theta_2 + \cdots \theta_n$

در نتیجه می‌توان برای انجام یک دوران، آن را به چشم تعدادی دوران دیگر دانست و با نمادگذاری 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:

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

اگر عبارت شامل $\cos$ را کنار بگذاریم، درون پرانتز شاهد یک جمع (تفریق) و یک ضرب هستیم. در سخت‌افزار 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 می‌توان بررسی کرد که:

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

همان طور که در بالا دیدیم:

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

یعنی دو مقدار مثبت و منفی وجود دارد و زاویه هم می‌تواند 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، می‌توانیم 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 می‌آید:

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

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

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 می‌ماند. اولین مورد این است که بازه برد arctan در $\frac{-\pi}{2} < \theta < \frac{\pi}{2}$ است. برای این که عملیات چرخش برای زوایای بیش‌تر 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 زاویه‌ها 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 می‌کنیم:

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

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

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

را اعمال می‌کنیم و الگوریتم را با x', y', θ' ادامه می‌دهیم.
نکته دیگر مربوط به این است که دو شکل دیگر از الگوریتم CORDIC به نام‌های 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 روش‌ها 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 می‌گیرد. 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، می‌توان آن را به صورت زیر نمایش داد.

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

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

که η و ξ بر اساس 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. ضمن این که برای حالت هذلولوی، باید مراحل ۴، ۱۳، ۴۰ و... الگوریتم تکرار شوند تا روش همگرا بشود. (الگوی این اعداد $3K + 1$ است که K عدد قبلی است که نیاز به تکرار داشته است)

در نهایت لازم به ذکر است که هر چند الگوریتم چرخش، می‌تواند هر بردار اولیه‌ای را بچرخاند، اما در شکل استاندارد الگوریتم، بردار اولیه به صورت یکه و موازی محور x داده می شو. 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 نتیجه‌این چرخش‌ها اندازه و زاویه اولیه‌اش 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 زاویه‌ای 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 بر اساس 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 حالاتی 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}	yinput	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 داده‌ایم. الگوریتم‌ها بر اساس حالت Circular نوشته شده‌اند 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 بهینه‌سازی‌های پیاده‌سازی سخت‌افزاری (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**
  $d = sgn(z)$
  $x = x - d \cdot y \cdot 2^{-i}$
  $y = y + d \cdot x \cdot \cdot 2^{-i}$
  $z = z + d \cdot \arctan(2^{-i})$
  $K = K \cdot \frac{1}{\sqrt{1+2^{-2i}}}$
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**
  $d = sgn(y)$
  $x = x - d \cdot y \cdot 2^{-i}$
  $y = y + d \cdot x \cdot \cdot 2^{-i}$
  $z = z + d \cdot \arctan(2^{-i})$
  $K = K \cdot \frac{1}{\sqrt{1+2^{-2i}}}$
end
$x_{out} = \frac{x}{K}$
$z_{out} = z$

۵.۱ کاربردها

۱.۵.۱ سخت‌افزار

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 اشیا 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 سریع‌تر از CORDIC است ولی اگر امکان استفاده از ماژول ضرب کننده مجزا نباشد، CORDIC به مراتب نسبت به پیاده‌سازی نرم افزاری ضرب و سپس استفاده از ضرب، برتری دارد و با سرعت بیش‌تری امکان انجام محاسبات را فراهم می‌آورد. 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 می‌تواند مورد توجه قرار بگیرد.

۲.۵.۱ نرم افزار

در دورانی که CPU ها واحد Floating-Point مجزا نداشتند و تمامی رجیسترهای آنان، به صورت Integer بودند، شرکت‌های تولید کننده پردازنده در قالب کتابخانه‌های نرم افزاری مربوط به پیاده‌سازی IEEE Floating Point System که امکان انجام محاسبات Floating-Point را از طریق رجیسترهای Integer مهیا می‌کرد، الگوریتم CORDIC را هم به صورت نرم افزاری برای محاسبه زوایای مثلثاتی پیاده‌سازی می‌کردند، اما بعدها که واحدهای مجزای Floating-Point و محاسبات آن به صورت سخت‌افزاری 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 می‌شود. [۷]

۲ پیاده‌سازی

۱.۲ اسامی ماژول‌ها و اینترفیس‌های سیستم

۱.۱.۲ Vectoring

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 خروجی 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 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
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
```

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 خروجی 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 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_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 گرفته‌اند. به ترتیب ابتدا دیاگرام‌های بلوکی مربوط به Vectoring و سپس Rotation قرار گرفته‌اند. در صفحه اول هر کدام، ماژول‌ها و ورودی‌ـ‌خروجی‌هایشان قرار گرفته. در صفحات بعد آن ها، نمودار درختی با همین ترتیب و در صفحات بعد از آن، اتصالات درونی آن‌ها.

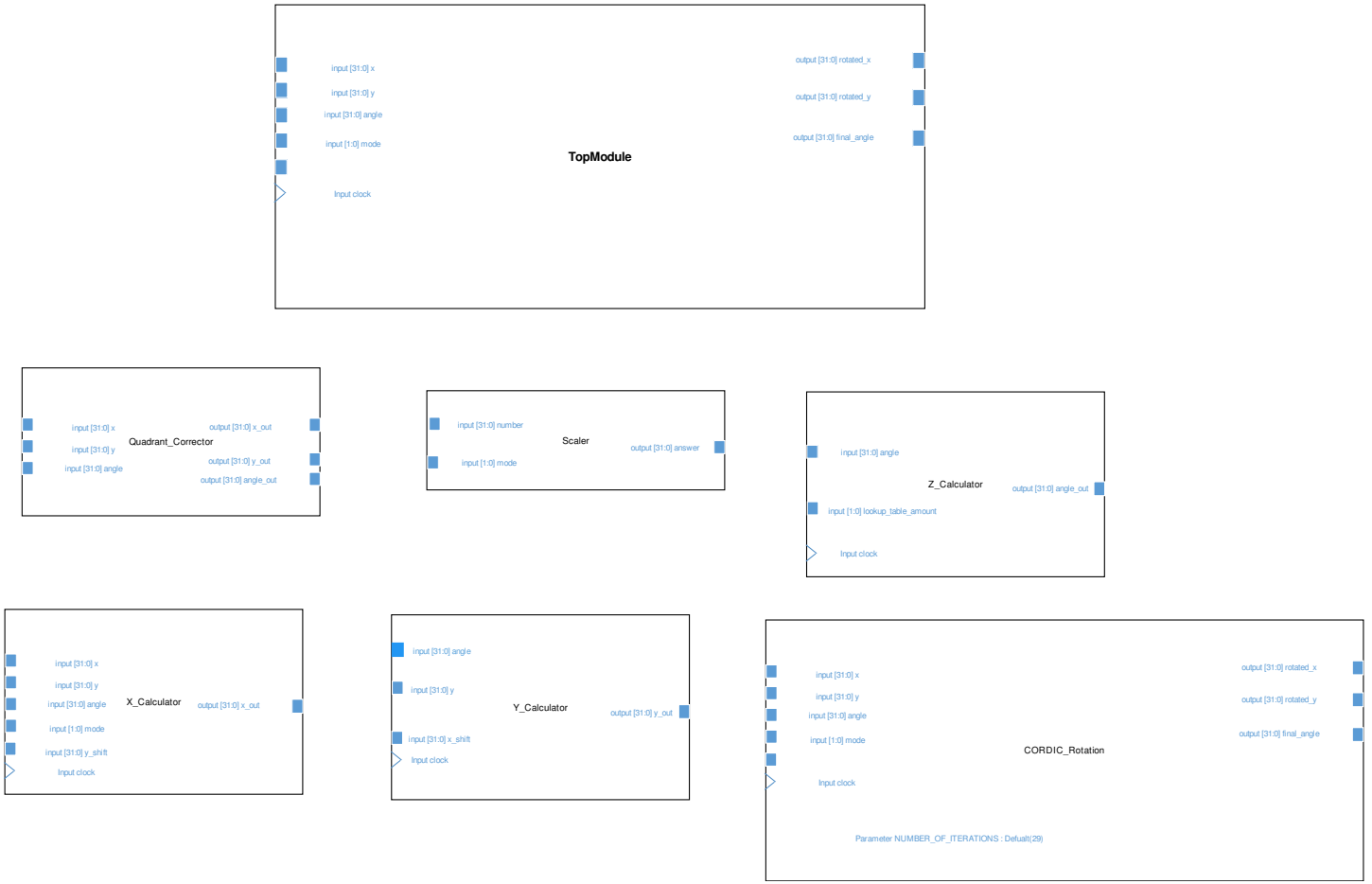
برای نمایش اتصالات درونی و از آن جایی که شکل سنتز شده کامل که شامل تعداد Iteration های زیاد باشد، بسیار بزرگ شده و در صفحه امکان نمایش درست آن وجود ندارد، یک شکل با Iteration های کم که اتصالات دقیق معلوم هستتند و یک شکل از دور برای حالتی که تمامی Iteration ها باشند، قرار گرفته است.

دیاگرام‌های مربوط به اتصالاتی که قرار گرفته اند 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

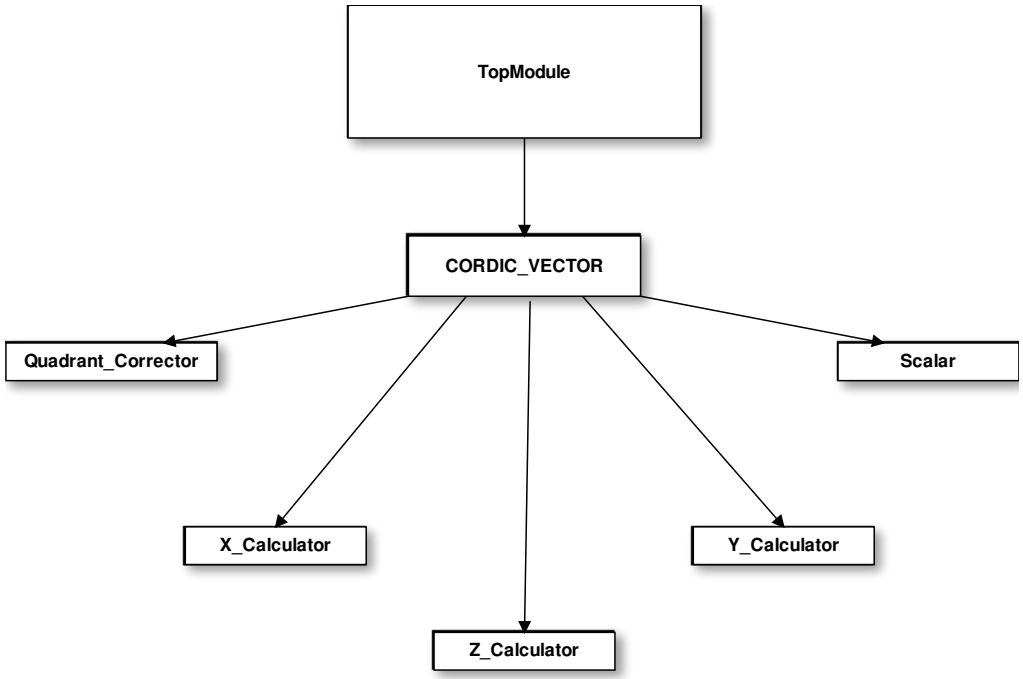
DSD Project - Page-1

DSD Project - Rotation - Page-1

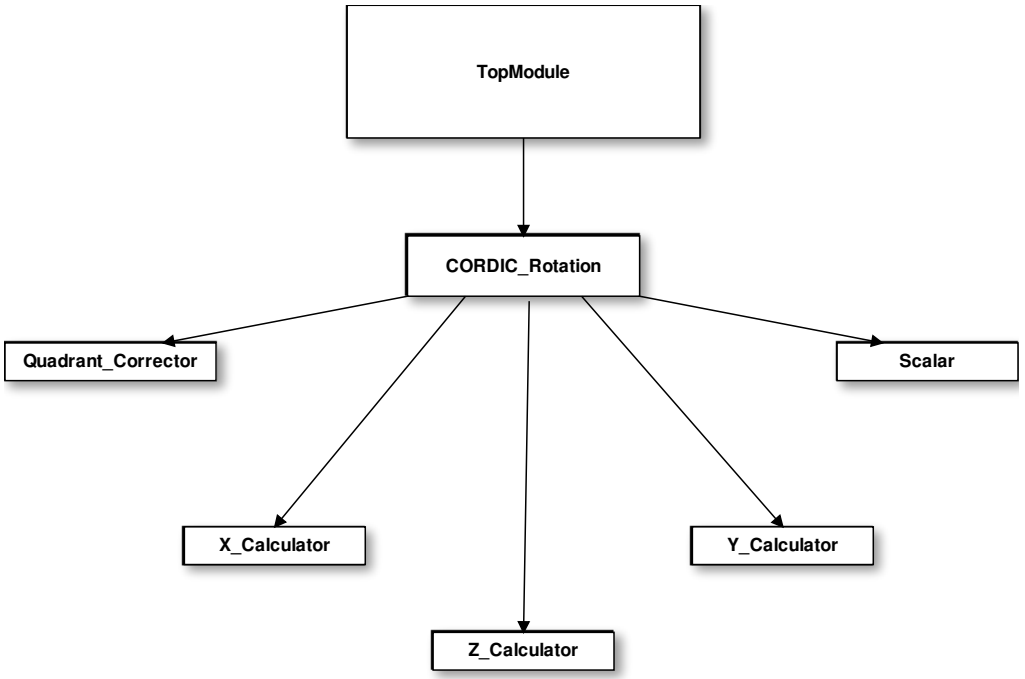


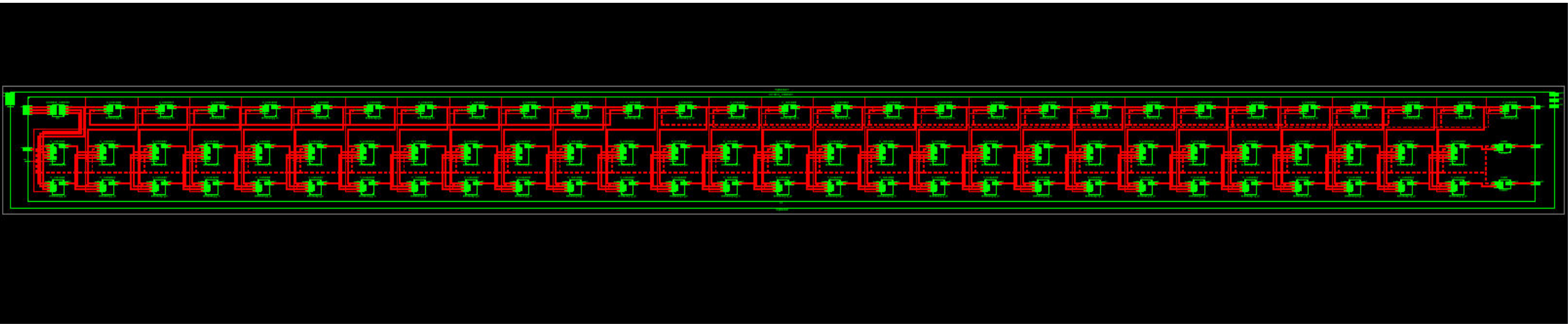
Project _ Rotation.pdf Project _ Rotation.bb

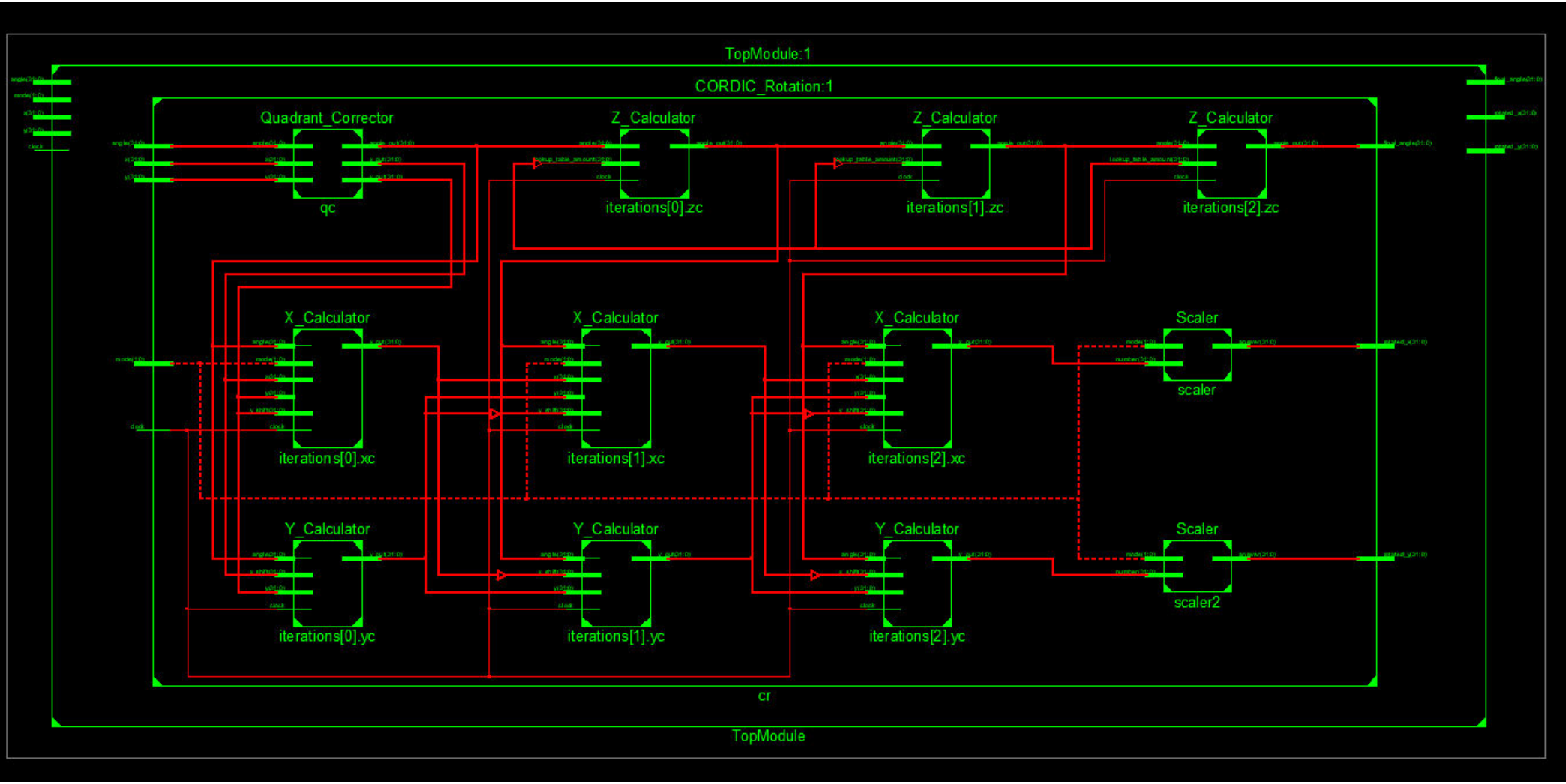
UML Design - Untitled

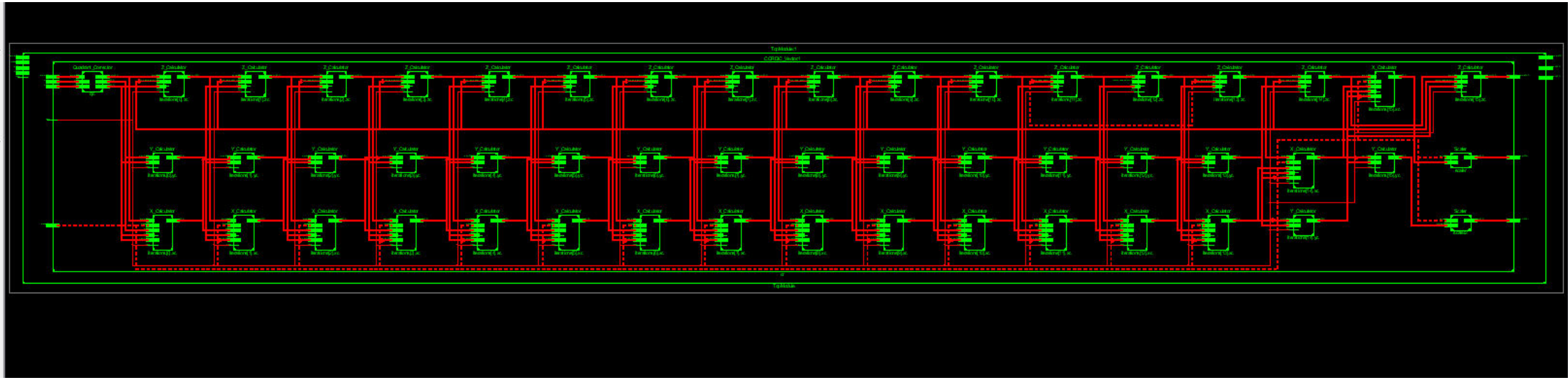


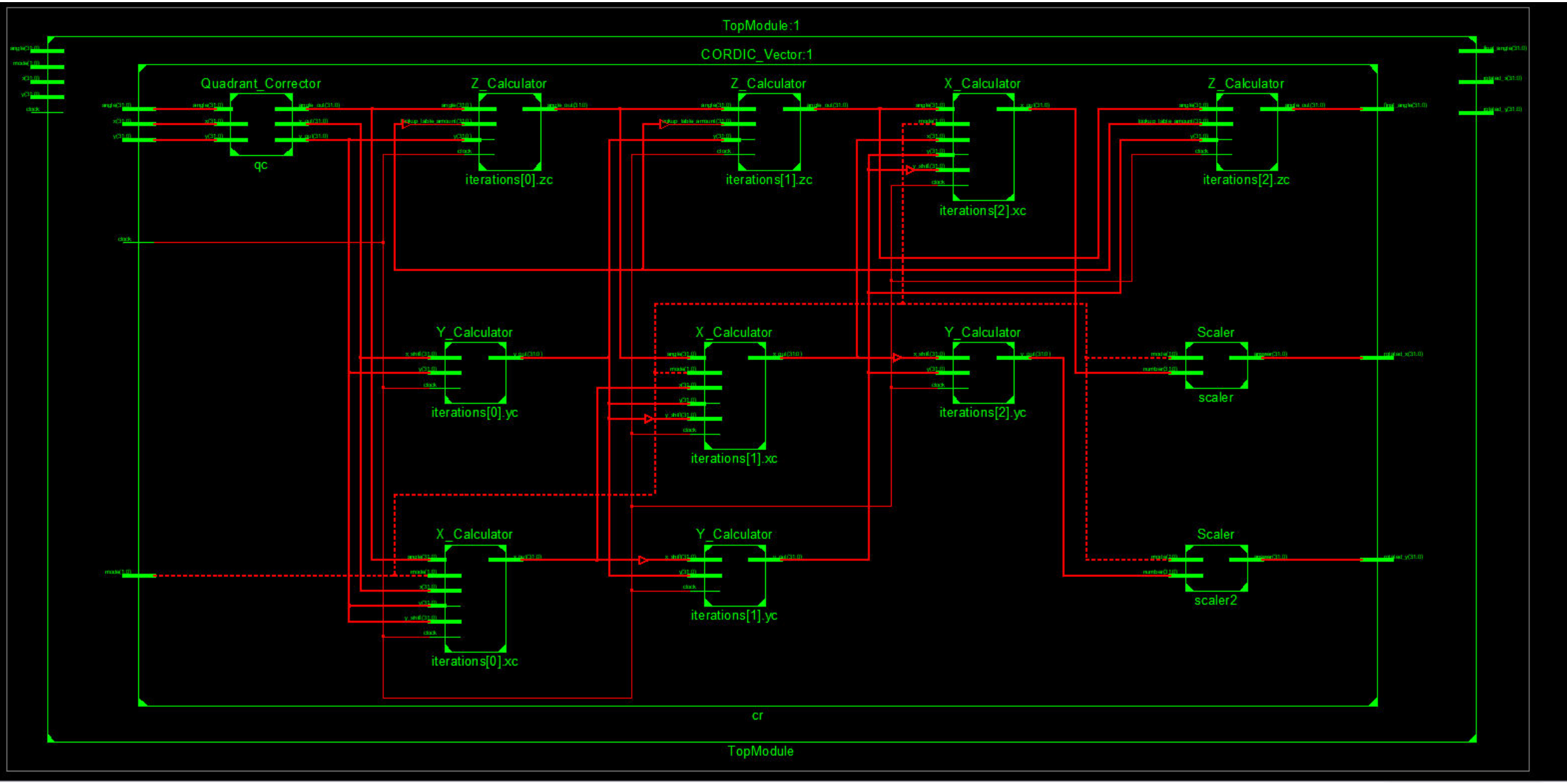
UML Design - Untitled






TopModule:1
CORDIC_Rotation:1
Quadrant_Corrector
qc
Z_Calculator
iterations[0].zc
Z_Calculator
iterations[1].zc
Z_Calculator
iterations[2].zc
X_Calculator
iterations[0].xc
X_Calculator
iterations[1].xc
X_Calculator
iterations[2].xc
Scaler
scaler
Y_Calculator
iterations[0].yc
Y_Calculator
iterations[1].yc
Y_Calculator
iterations[2].yc
Scaler
scaler2
cr
TopModule




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
clock
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

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 داده‌ایم. 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 به دست 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 به دست می آمده 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 بر اساس ثابت‌های CONSTANTS.v مشخص می‌شود. 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 می‌کنیم. 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 می‌دهیم:

```
(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 می‌کنیم، با این تفاوت که وجود 1 در عدد 1.2075 یعنی یکی از بخش‌های جمع شونده، خود عدد اصلی بدون هیچ نوع شیفت اضافی است.

ماژول بعدی X_Calculator 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,
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 فرمول‌هاست که از ماژول اصلی به عنوان ورودی وارد این ماژول می‌شود. 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 بر اساس ورودی‌های داده شده، محاسبات طبق فرمول‌های بخش ریاضی و بر اساس علامت y انجام شده و مقدار جدید x تحت نام x_out خروجی داده می‌شود.

ماژول بعدی Y_Calculator 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,
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 lookup_table_amount به آن ورودی داده می‌شود که در اصل، مقداری برابر با 2^{-i} یا $\arctan(2^{-1})$ یا $\text{arctanh}(2^{-1})$ 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 کرده‌ایم 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 صفر 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 Y_Calculator، X_Calculator و Z_Calculator ساخته‌ایم. یکسری متغیر temp هم وجود دارند که مقادیر مربوط به arctan یا arctanh و همچنین شیفت داده شده X و Y به اندازه i را که اندیس همان مرحله باشد، به عنوان وردی 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 بر اساس آن، مقدار متناسب را از بین جدول arctan یا arctanh یا 2^{-i} خروجی می‌دهد. دلیل این که از 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 پیاده‌سازی 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.

۲.۳.۲ ROTATION

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 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 بر اساس ثابت‌های CONSTANTS.v مشخص می‌شود. 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 می‌کنیم. 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 می‌دهیم:

```
(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 می‌کنیم، با این تفاوت که وجود 1 در عدد 1.2075 یعنی یکی از بخش‌های جمع شونده، خود عدد اصلی بدون هیچ نوع شیفت اضافی است.

ماژول بعدی X_Calculator 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,
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 فرمول‌هاست که از ماژول اصلی به عنوان ورودی وارد این ماژول می‌شود. 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 بر اساس ورودی‌های داده شده، محاسبات طبق فرمول‌های بخش ریاضی بر اساس 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 می‌شود.

ماژول بعدی Y_Calculator 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,
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 lookup_table_amount به آن ورودی داده می‌شود که در اصل، مقداری برابر با 2^{-i} یا $\arctan(2^{-1})$ یا $\text{arctanh}(2^{-1})$ 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_Rotation 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 = 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 کرده‌ایم 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 صفر 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 Y_Calculator، X_Calculator و Z_Calculator ساخته‌ایم. یکسری متغیر temp هم وجود دارند که مقادیر مربوط به arctan یا arctanh و همچنین شیفت داده شده X و Y به اندازه i را که اندیس همان مرحله باشد، به عنوان وردی 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 بر اساس آن، مقدار متناسب را از بین جدول arctan یا arctanh یا 2^{-i} خروجی می‌دهد. دلیل این که از 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 پیاده‌سازی 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_Rotation با اندازه پارامتر مشخص ساخته شده که در سنتز و تست بنچ مورد استفاده قرار بگیرد.

۳ تست بنچ و شبیه سازی

۱.۳ توضیحاتی در مورد Golden Model

متأسفانه ما نتوانستیم مدل طلایی را از اینترنت پیدا کنیم که هم تمام بخش‌ها (هر دو زیرپروژه و حالت‌های مختلف تمامی زوایا) را پوشش دهد و هم بدون مشکل کار کند؛ بنابراین مدل طلایی را که تمام موارد را پوشش می‌داد اما در جواب نهایی خطا داشت را انتخاب کرده و خودمان آن را تغییر دادیم تا درست کار کند.

این Golden Model از سه فایل پایتون با فرمت py. تشکیل شده است که یکی وظیفه Rotation، یکی وظیفه Vectoring و دیگری نیز وظیفه تبدیل فرمت‌های اعداد به فرمتی که در وریلاگ استفاده می‌شود به عهده دارد (تبدیل اعداد دسیمال به فرمت باینری و به فرمتی که در کد اصلی وریلاگ استفاده شده است). در نهایت مقادیر ساخته شده در فایل‌های مرتبط قرار می‌گیرند.

نحوه انجام عملیات Rotation به این صورت است که ابتدا مقادیر اولیه x و y و z (زاویه اولیه) به همراه نوع عملیات و تعداد iteration ها به تابعی داده می‌شود (تعداد iteration ها در تمامی تست‌ها ۱۵ قرار داده شده است) سپس ابتدا پیش پردازشی روی ورودی‌ها انجام می‌شود تا همانگونه که در ابتدای داک توضیح داده شده بود مقادیر x و y و z تغییر یابند تا z زاویه‌ای بین ـ۹۰ تا ۹۰ درجه باشد. در نهایت نیز z به رادیان تبدیل شده و این سه مقدار برگردانده می‌شوند.

سپس عملیات اصلی چرخش انجام می‌شود و در نهایت مقادیر K مرتبط با نوع عملیات در x و y ضرب می‌شود و سپس مقادیر خروجی به کلاس TestGenerator داده می‌شود تا فرمت آن‌ها تغییر یافته در فایل نوشته شوند. برای Vectoring نیز ترتیب توابع مشابه حالت بالاست اما عملیات داخلی برخی توابع متفاوت است که در داک و در بخش ریاضیاتی توضیحات مربوطه داده شده است.

۲.۳ توضیحاتی در مورد تست‌های ساخته شده توسط Golden Model

برای Rotation هر فایل آن که شامل RotationCircular و RotationLinear است شامل ۲۰ سطر می‌باشد که همان تعداد تست‌هایی است که انجام می‌شود، هر سطر نیز شامل ۶ داده ۳۲ بیتی باینری است که به ترتیب x و y و زاویه اولیه و x و y و زاویه باقی مانده پس از چرخش هستند برای Vectoring هر فایل آن که شامل وVectoringCircular VectoringLinear است شامل ۱۰ سطر می‌باشد که همان تعداد تست‌هایی است که انجام می‌شود، هر سطر نیز شامل ۵ داده ۳۲ بیتی باینری است که به ترتیب x و y اولیه و x و y و زاویه باقی مانده پس از پیدا کردن زاویه مورد نظر هستند (دلیل اینکه تعداد تست‌های این بخش کمتر است این است که متغیرهای ما ۲ مورد بودند برخلاف Rotation که ۳ متغیر داریم)

۳.۳ Unit Test ها

با توجه به اینکه ماژول‌های استفاده شده در دو زیر پروژه ورودی و خروجی‌های تقریباً یکسان و عملکردی مشابه دارند، البته طبیعی است که با توجه به عملکرد اندکی متفاوت این ماژول‌ها، خروجی آن‌ها با یکدیگر متفاوت باشد؛ بنابراین برای هر کدام فقط عملکرد این ماژول در یکی از زیر پروژه‌ها بررسی شده است.

۱.۳.۳ ماژول tb_X_Calculator

ماژول طراحی شده به نام tb_X_calculator.v است که در آن برای ماژول x_calculator یک سری تست نوشته‌ایم. در این ماژول ما ابتدا از X_calculator یک نمونه به نام X_Cal می‌سازیم و پس از ورودی دادن آن باید مقداردهی را شروع کنیم و در آنجا مقادیری را به ورودی‌ها می‌دهیم و بعد از اجرای هر تست به مدت ۱۰ns صبر می‌کنیم و در آخر هم خروجی را با دستور monitor نمایش می‌دهیم . پس از دادن ورودی‌ها X_cal باید مقدار x_Out را نمایش دهد پس وظیفه‌ی این ماژول این است که باید مقدار x_out را محاسبه کند به این نحو که بستگی به mode دارد که در اینجا ما ۳ mode داریم به نام‌های circular ، linear و hyperbolic که در آن متناسب با هر کدام از mode ها طرز محاسبه‌ی آن‌ها متفاوت است برای Rotation و vector ما دو فایل جداگانه داریم در ابتدا برای Rotation داریم به این گونه که برای circular ، طرز محاسبه به این گونه است که

x_out_temp <= angle[30] ? x + y_shift : x - y_shift x_out_temp

محاسبه می‌شود و در آخر در x_out ریخته می‌شود . طرز کار به این صورت است که بیت ۳۰ ام angle اگر برابر با یک باشد آنگاه مقدار x_out_reg برابر با x+y_shift می‌شود و اگر برابر با ۰ بود آنگاه مقدار x_out_reg برابر با x-y_shift می‌شود حال اگر مقدار سیگنال mode برابر با linear باشد آنگاه طبق x_out_temp <= x؛ مقدار سیگنال x_out_Reg برابر با x می‌شود

حال اگر سیگنال mode برابر با hyperbolic باشد آنگاه طبق

x_out_temp <= angle[30] ? x - y_shift : x + y_shift

اگر مقدار بیت ۳۰ ام سیگنال angle با ۱ باشد مقدار x_out_reg برابر با x-y_shift و اگر برابر با ۰ باشد آنگاه مقدار x_out_reg برابر با x+y_shift است. حال برای vector باید گفت که تنها تفاوت آن‌ها این است که برای ماژول tb_X_calculator.v ما angle نداریم (منظور از نداشتن این است که آن را مقداردهی می‌کنیم اما در ماژول اصلی هیچ کدام از کارهایمان بر حسب angle نیست) در این ماژول ما ابتدا از X_calculator یک نمونه به نام X_Cal می‌سازیم و پس از ورودی دادن آن باید مقداردهی را شروع کنیم و در آنجا مقادیری را به ورودی‌ها می‌دهیم و بعد از اجرای هر تست به مدت ۱۰ns صبر می‌کنیم و در آخر هم خروجی را با دستور monitor نمایش می‌دهیم .

۲.۳.۳ ماژول tb_Y_Calculator

این ماژول برای تست ماژول Y_Calculator در پروژه مورد استفاده قرار گرفته است؛ در ادامه توضیحات نحوه عملکرد آن در حالت Rotation قرار دارد:

در این ماژول ابتدا از Y_Calculator یک Instance به نام yc ساخته‌ایم و سپس ورودی‌ها و خروجی‌های مورد نظر را وصل کرده‌ایم. در داخل این ماژول در داخل یک initial block سه تست برای بررسی عملکرد Y_Calculator قرار دارند که با توجه به اینکه فقط یک از دو حالت جمع یا تفریق را انجام

می‌دهد همین تعداد تست کافی به نظر می‌رسد؛ در ادامه نیز در یک initial block دیگر مقادیر خروجی y_out با استفاده از دستور $monitor چاپ می‌شوند.
در بخش Rotation و در هر یک از ۳ تست گفته شده ۳ ورودی y و x_shift و angle مقداردهی شده اند (برای بخش Vector نیازی به زاویه نداریم و فقط دو متغیر دیگر را مقداردهی کرده‌ایم که این مقادیر برای هر دو حالت Rotation و Vector دقیقاً یکسان هستند) و بین هر دو تست نیز به اندازه DELAY_BETWEEN_TESTS فاصله زمانی داریم.

۳.۳.۳ ماژول tb_Z_Calculator

ماژول طراحی شده به نام tb_Z_calculator.v است که در آن برای ماژول z_calculator یک سری تست نوشته‌ایم. در این ماژول ما ابتدا از Z_calculator یک نمونه به نام Z_Cal می‌سازیم و پس از ورودی دادن آن باید مقداردهی را شروع کنیم و در آنجا مقادیری را به ورودی‌ها می‌دهیم و بعد از اجرای هر تست به مدت۱۰ns صبر می‌کنیم و در آخر هم خروجی را با دستور monitor نمایش می‌دهیم . پس از دادن ورودی‌ها Z_cal باید مقدار angle_Out را نمایش دهد پس وظیفه‌ی این ماژول این است که باید مقدار tangle_ou را محاسبه کند.
در قسمت Vector تفاوت اینجاست که یک ورودی y هم نیاز است تا به tb اضافه کنیم.

۴.۳.۳ ماژول tb_Scaler

ماژول طراحی شده به نام tb_Scaler.v است که در آن برای ماژول Scaler یک سری تست نوشته‌ایم. در این ماژول ما ابتدا از Scaler یک نمونه به نام scaler می‌سازیم و پس از ورودی دادن آن باید مقداردهی را شروع کنیم و در آنجا مقادیری را به ورودی‌ها می‌دهیم و بعد از اجرای هر تست به مدت ۱۰ns صبر می‌کنیم و در آخر هم خروجی را با دستور monitor نمایش می‌دهیم . پس از دادن ورودی‌ها scaler باید مقدار answer را نمایش دهد پس وظیفه‌ی این ماژول این است که باید مقدار answer را محاسبه کند.

۵.۳.۳ ماژول tb_Quadrant_Corrector

ابتدا باید درباره‌ی کار این ماژول گفت که الگوریتم‌های CORDIC فقط در ناحیه اول و چهارم کار می‌کنند به صورت پایه، Quadrant_Corrector بر اساس ناحیه مختصاتی اگر در ناحیه دوم یا سوم باشد، زاویه را ۹۰ درجه کم یا زیاد می‌کند و بر اساس x،y را هم می‌چرخاند (چرخش ۹۰ درجه که فقط علامت یا جای x،y عوض می‌شود و بعد از آن به ادامه‌ی برنامه می‌رود برای ادامه کار. حال درباره‌ی تست بنچ آن توضیحی را بیان می‌کنیم که در این تست بنچ ما بعد از اینک ورودی و خروجی‌ها را به صورت wire و reg تعریف کردیم از ماژول Quadrant_Corrector یک نمونه می‌سازیم و ورودی و خروجی‌های این ماژول را تعریف می‌کنیم و در بلاک initial ما مقادیری را برای x و y و angle مشخص می‌کنیم و مابین هر تست ۵ نانوثانیه delay داریم بعد از اتمام این تست‌ها خروجی‌ها را با دستور monitor نمایش می‌دهد.

۴.۳ Integration Test ها

ماژول‌های اصلی ما در پروژه، دو ماژول CORDIC_Rotation و CORDIC_Vector هستند که ماژول‌های TestBench ای که برای این دو طراحی شده اند به ترتیب tb_CORDIC_Rotation و tb_CORDIC_Vector هستند؛ این دو تست بنچ از ماژول TopModule که در آن‌ها از ماژول‌های اصلی پروژه یک Instance با پارامتر مشخص قرار گرفته است، استفاده می‌کنند. در ادامه عملیات این دو ماژول که در واقع Integration Test های ما هستند بررسی شده است:

۱.۴.۳ ماژول tb_CORDIC_Rotation

در این ماژول ۶ متغیر مربوط به حالت Circular و ۶ متغیر مربوط به حالت Linear قرار دارد که همگی آرایه‌ای ۲۰ تایی از وکتورهای ۳۲ بیتی هستند و ۳ متغیر از هر کدام مربوط به حالت اولیه و ۳ متغیر دیگر مربوط به حالت نهایی هستند.

این آرایه‌ها از روی مقادیری که از فایل‌ها خوانده می‌شوند مقداردهی می‌شوند؛ در بلاک initial ای که به همین منظور استفاده شده است ابتدا فایل مورد نظر با دستور $fopen باز شده سپس این مقادیر با استفاده از دستور $fscanf و در داخل یک for خوانده شده و در آرایه‌های گفته شده قرار می‌گیرند و همچنین خروجی‌های مورد انتظار (۳ عدد آخر هر سطر) با استفاده از دستور $display چاپ می‌شوند؛ عملیات گفته شده در داخل همین بلاک و در دو for جداگانه یک بار برای Circular و بار دیگر برای Linear انجام می‌شود.

سپس در داخل بلاک initial بعدی این مقادیر با تأخیری DELAY_BETWEEN_TESTS ثانیه‌ای (پارامتری که در ابتدای ماژول تعریف شده است) بین هر کدام و در داخل یک for به عنوان ورودی‌های ماژول CORDIC_Rotation قرار می‌گیرند (از ماژول CORDIC_Rotation در داخل این تست بنچ با نام cr یک instance ساخته شده است) عملیات بالا ابتدا برای حالت Circular انجام می‌شود و سپس با تأخیری به اندازه پارامتر CHANGE_STATE_PERIOD که در ابتدای ماژول تعریف شده است برای Linear نیز انجام می‌شود؛ در نهایت نیز در ماژول initial ای دیگر، خروجی‌های ماژول CORDIC_Rotation با دستور $monitor نشان داده می‌شوند.

برای مقایسه مقادیر خروجی کد وریلاگ و خروجی‌های اصلی که از مدل طلایی گرفته‌ایم صرفاً هر دو نمایش داده می‌شوند و می‌توان به صورت دستی این مقایسه را انجام داد که در حالت کلی اختلاف این مقادیر کمتر از ۰.۰۱ است که نشان از دقت قابل قبول کد وریلاگ دارد.

۲.۴.۳ ماژول tb_CORDIC_Vector

در این ماژول ۵ متغیر مربوط به حالت Circular و ۵ متغیر مربوط به حالت Linear قرار دارد که همگی آرایه‌ای ۱۰ تایی از وکتور های ۳۲ بیتی هستند و ۲ متغیر اول از هر کدام مربوط به حالت اولیه و ۳ متغیر دیگر مربوط به حالت نهایی هستند.

این آرایه‌ها از روی مقادیری که از فایل‌ها خوانده می‌شوند مقداردهی می‌شوند؛ در بلاک initial ای که به همین منظور استفاده شده است ابتدا فایل مورد نظر با دستور $fopen باز شده سپس این مقادیر با استفاده از دستور $fscanf و در داخل یک for خوانده شده و در آرایه‌های گفته شده قرار می‌گیرند و همچنین خروجی‌های مورد انتظار (۳ عدد آخر هر سطر) با استفاده از دستور $display چاپ می‌شوند؛ عملیات گفته شده در داخل همین بلاک و در دو for جداگانه یک بار برای Circular و بار دیگر برای Linear انجام می‌شود.

سپس در داخل بلاک initial بعدی این مقادیر با تأخیری DELAY_BETWEEN_TESTS ثانیه‌ای (پارامتری که در ابتدای ماژول تعریف شده است) بین هر کدام و در داخل یک for به عنوان ورودی‌های ماژول CORDIC_Vector قرار می‌گیرند (از ماژول CORDIC_Vector در داخل این تست بنچ با نام cr یک instance ساخته شده است) و ورودی angle آن ۰ داده می‌شود زیرا تأثیری در محاسبات و جواب نهایی ندارد.

عملیات بالا ابتدا برای حالت Circular انجام می‌شود و سپس با تأخیری به اندازه پارامتر CHANGE_STATE_PERIOD که در ابتدای ماژول تعریف شده است برای Linear نیز انجام می‌شود؛ در نهایت نیز در ماژول initial ای دیگر، خروجی‌های ماژول CORDIC_Vector با دستور monitor$ نشان داده می‌شوند.

برای مقایسه مقادیر خروجی کد وریلاگ و خروجی‌های اصلی که از مدل طلایی گرفته‌ایم صرفاً هر دو نمایش داده می‌شوند و می‌توان به صورت دستی این مقایسه را انجام داد که در حالت کلی دقت قابل قبولی دارد.

۴ سنتز

برای سنتز، از نرم افزار Xilinx ISE استفاده کردیم و در تنظیمات برنامه، سنتز را روی FPGA با مشخصات: Spartan6 XC6SLX150T و پکیج FGG484 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.

۱.۴ سنتز Rotation

نتایج اصلی سنتز به صورت زیر است. فایل‌های مربوطه نیز به طور کامل ضمیمه گزارش کار شده است:
نتایج اصلی تأخیرها و زمان‌های 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 O6 output only:          4,868
      Number using O5 output only:             94
      Number using O5 and O6:                 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%
```

۲.۴ سنتز Vectoring

نتایج اصلی سنتز به صورت زیر است. فایل‌های مربوطه نیز به طور کامل ضمیمه گزارش کار شده است:
نتایج اصلی تأخیرها 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 O6 output only:          2,903
      Number using O5 output only:             94
```

```
    Number using O5 and O6:                  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%
```

۵ نتیجه گیری

در این پروژه، ما به طور جدی و دقیق کار با زبان Verilog را تجربه کردیم و توانستیم یک پروژه سخت افزاری را با استفاده از این زبان به کمک نرم افزار Modelsim پیاده سازی کرده و شبیه‌سازی‌های اولیه را روی آن انجام بدهیم و در نهایت عملیات سنتز آن را با موفقیت بر روی نرم افزار Xilinx ISE به سرانجام برسانیم.

در این پروژه ما با الگوریتم CORDIC آشنا شدیم که در زمان‌هایی که امکان استفاده از الگوریتم‌های نرم افزاری FloatingPoint وجود نداشته باشد، می‌توان از آن که به نوعی بر پایه خواص توابع مثلثاتی و همچنین خواص درونی اعداد مختلط و شباهت‌هایی که بین توابع مثلثاتی و هذلولوی و حتی خطی وجود دارد، استفاده کرده و اقدام به محاسبه توابع مثلثاتی و هذلولوی و... کرد. به نوعی با این پروژه ما با روش‌هایی برای محاسبه توابع پیچیده بدون استفاده از روش‌های رایج نظیر بسط تیلور و مک لورن آشنا شدیم؛ روش‌هایی که پیاده سازی سخت افزاری بهینه‌تری نسبت به آن روش‌ها دارند و در نتیجه در مواردی از نظر زمانی یا سخت افزاری در مضیقه باشیم، می‌توان از آن‌ها استفاده کرد.

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.