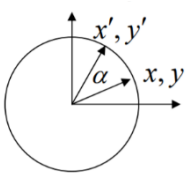
# پروژه ی پایانی FPGA

مارال رسولی جابری

سوده نیلفروشان

پاییز ۹۷

## 1. الگوریتم Cordic [1]




در این الگوریتم، برای محاسبه ی سینوس و کسینوس یک زاویه با کمک زوایایی(α) است که تانژانت های توان های صحیح از ۲ است. به عنوان مثال، $\tan(\frac{\Pi}{4})= 2^0 =1$ و $\tan(0.7854)= 2^{-1} =0.5$ .در هر زاویه ی دلخواه را میتوان به صورت مجموع این زوایا نوشت:

$$70^\circ = 1.2217 \text{ radian} = \tan^{-1}(1) + \tan^{-1}(2^{-1}) - \tan^{-1}(2^{-2}) + \tan^{-1}(2^{-3}) + \tan^{-1}(2^{-4}) - \tan^{-1}(2^{-5}) + \ldots$$

هرچه تعداد بیشتری از این زوایا کمک گرفته شود، دقت الگوریتم cordic بیشتر خواهد بود.

ایده ی کلی الگوریتم محاسبه ی سینوس (y′) و کسینوس (x′) با استفاده از یک rotate vector با زوایای از پیش تعریف شده است. به این صورت که در دایره ی مثلثاتی از نقطه ی (1,0) شروع میکند(صفر درجه) و با این زوایا به نقطه ی مورد نظر نزدیک میشود. به عنوان مثال، برای محاسبه ی سینوس $70^\circ$ (1.2217 رادیان) ، 0 را با $\frac{\Pi}{4}$ جمع میکند($\tan^{-1}(1)$). در مرحله ی دو چون زاویه ی ورودی از $\frac{\Pi}{4}$ بیشتر است، $\frac{\Pi}{4}$ را با $\tan^{-1}(2^{-1})$ جمع میکند و حاصل از زاویه ی ورودی بیشتر میشود. در مرحله ی سوم، $\tan^{-1}(2^{-2})$ را از عدد قبلی کم میکند تا به زاویه ی حاصل نزدیک شویم. در هر مرحله x′ و y′ که به ترتیب معادل تقریبی کسینوس و سینوس زاویه ی ورودی میباشند و از مقدار محاسبه شده در مرحله ی قبل و طبق فرمول های زیر به دست می آیند:

$$x' = x \cdot \cos\alpha - y \cdot \sin\alpha$$
$$y' = x \cdot \sin\alpha + y \cdot \cos\alpha$$

برای اینکه معادلات فوق با جمع و شیفت قابل پیاده سازی باشند، از cos(α) فاکتور میگیریم:

$$x' = \cos(\alpha)\,[x - y.\tan(\alpha)]$$

$$y' = \cos(\alpha)\,[y + x.\tan(\alpha)]$$

در واقع زوایای α ها همان $\tan^{-1}(1)$ و $\tan^{-1}(2^{-1})$ و $\tan^{-1}(2^{-2})$ و ... میباشند. در هر مرحله، باید این ضرایب$\cos(\alpha_i)$ در $\cos(\alpha_i)$های مراحل قبلی ضرب میشوند. لذا میتوان یه ضریب k به عنوان scaling زیر در نظر گرفت و محاسباتِ تکرار شونده را به صورت معادلات 10 ساده کرد:

[1] https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/

$$K \approx cos(45^\circ)cos(26.565^\circ) \times \cdots \times cos(0.895^\circ) = 0.6072$$

ضریب k با دقت ۱۸ بیت محاسبه شده است و در مرحله ی آخر، تنها یک بار در $y$ ضرب میشود تا sin زاویه ی ورودی به دست آید.

$$x[i+1] = x[i] - \sigma_i 2^{-i} y[i]$$
$$y[i+1] = y[i] + \sigma_i 2^{-i} x[i]$$
$$z[i+1] = z[i] - \sigma_i tan^{-1}(2^{-i})$$

رابطه ی ۱۰

---

**توضیحات مربوط به پیاده سازی VHDL**

در این پروژه ما دو ماژول cordic , sin را پیاده سازی کردیم که به توضیحی هرکدام در ادامه میپردازیم.

## ماژول cordic

این ماژول x,y,z مرحله ی i ام را به عنوان ورودی دریافت مکیند x,y,z مرحله ی بعد را تولید میکند. و داری ۴ استیت است:

**Idle : مقدار دهی اولیه**

در صورتی که start مربوط به این مرحله فعال شده باشد، ورودی ها را به Unsigned تبدیل میکند تا بتوانیم محاسبات و پردازش های مربوطه را روی ورودی ها انجام دهیم.

**Op1 : شیفت دادن**

این استیت مقدار $2^{-i}y[i]$ و $2^{-i}x[i]$ را محاسبه میکند.

ما با شیفت به راست ، این ماژول را پیاده سازی کردیم. در مرحله ی i ورودی باید به اندازه ی i به سمت راست شیفت پیدا کند، یعنی در اصل i بیت اول آن دور ریخته شود :

```
y_temp(N+1-i downto 0) <= y2_reg(N+1 downto i);
x_temp(N+1-i downto 0) <= x2_reg(N+1 downto i);
```

**Op2 : جمع/تفریق**

این استیت درواقع جمع و تفریق را بطه ۱۰ (بالا اورده شده) را محاسبه میکند . چون ما از فرمت بدون علامت استفاده کرده ایم، باید بیت علامت از محاسبات کنار گذاشته شود و magnitude و sign خروجی ها جداگانه محاسبه شوند.

بدین منظور یک If , else کلی داریم که مربوط به این است که آیا سیگما در رابطه ی ۱۰ مثبت است یا منفی. به عنوان مثال، برای سیگای مثبت ،باید x[i] با y[i] شیفت یافته مقایسه شود تا متوجه شویم کدام magnitude را باید از دیگری کم کنیم و همچنین علامتِ x[i+1] را نیز علامتِ عبارتِ بزرگتر را میگیرد . و به همین ترتیب این عملیات را برای z[i+1] , y[i+1]حساب کردیم.

مقدار $tan^{-1}(2^{-i})$ بر حسب رادیان و به فرمت fixed-point بدون علامت با دقت ۱۸ بیت در look-up-table ذخیره کردیم .در مرحله ی i ، سطر i ام look-up-table را داخل z_temp میریزیم.

برای دقت محاسبات ما دو عدد iteration_num ,generic N داریم :

مثلا اگر کاربر عدد generic N را ۵ بدهد . این look_up_table ۳+ ۵ بیت پر ارزش انرا حساب میکند(محاسبات میانی با N+3 بیت دقت انجام میشود). اگر iteration_num مقدار ۱۰ داشته باشد نشان میدهد تا سطر ۱۰ ام جدول look_up_table محاسبات تکرارشونده انجام شود.

در صورت سوال خواسته شده بود که محاسبات میانی با log(N) بیت انجام شود. به طور میانگین، کاربر زاویه ی ورودی بر حسب رادیان را در ۸ بیت به ورودی اعمال میکند (با دقت 0.03 رادیان). در نتیجه، برای تمامی سیگنال هایی که در محاسبات میانی دخالت دارند، ۳ بیت بیشتر در نظر گرفته شده تا دقت محاسبات میانی افزایش یابد. همچنین در زمان Scalling، ضرب fixed point با ماکزیمم دقت انجام میشود.

**Op3 : محاسبه ی خروجی**

این استیت مقادیر محاسبه شده و علامت های محاسبه شده را با یکدیگر concat میکند و داخل خروجی میریزد. خروجی باید به فرمت std_logic_vector تبدیل شود و سپس به استیت idle میرود.

---

**ماژول Sin** : این ماژول برای پیاده سازی محاسبات تکرار شونده به صورت pipeline است .

برای این قسمت از یک for استفاده کردیم که از ۱ تا تعداد iterationها که در واقع دقت محاسبات است پیش میرود.

مرحله اول ، خارج از for پورت مپ میشود . در این مرحله، مقادیر ثابت ۱ و ۰ و زاویه ورودی (beta) را به عنوان ورودی های x , y, z به اولین Instance از ماژول cordic پورت مپ کردیم سپس محاسبات پورت مپ های بعدی را داخل for نوشتیم .


دور اول یکبار الگوریتم کوردیک را اجرا میکند و خروجی ان وارد دور دوم میشود. در اصل، استارت هر Iteration سیگنال done مرحله ی قبلی است. در نهایت، y در مرحله ی آخر، برابر sin زاویه ی ورودی است که با ضریب K اسکیل شده است.

برای ضرب مقدار k در y ، که هر دو به صورت fixedpoint هستند ، ابتدا هردو unsigned را در هم ضرب میکنیم سپس بیت علامت را با حاصلضرب k,y ، concat میکنیم.

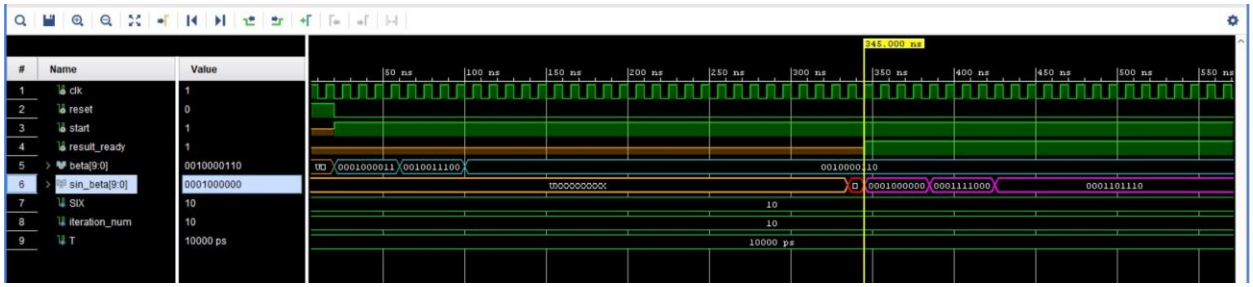
در ضرب fixed-point بین K که دارای ۲ بیت رقم صحیح و 16 بیت رقم اعشار است، و Y که دارای N-1 بیت اعشار و ۲ بیت رقم صحیح است، خروجی حاصل ضرب دارای N-1 +16 بیت رقم اعشار و۲ بیت رقم صحیح خواهد بود.(سیگنالِ حاصلِ ضرب، N+20 بیتی است) از آنجایی که خروجی ماژول sin

دارای N بیت است، Nبیت پر ارزشِ حاصل ضرب را که با فرمتِ fixed-point قرارداد شده در این پروژه مطابقت دارد را به خروجی میدهیم.

توضیح Pipline : در مرحله بعدی، زاویه بعدی وارد دور اول میشود. تا این مرحله زاویه اول وارد محاسبات دور دوم شده و زاویه دوم وارد محاسبات دور اول (که داخل ماژول کوردیک پیاده سازی شده ) میشود .و به همین ترتیب پیش میرود تا در دور N ام مقدار نهایی زاویه اول محاسبه شود و در دور N+1 ام مقدار نهایی زاویه دوم و به همین ترتیب.....

| Iteration $(i)$ | $\sigma_i$ | $x\,[i]$ | $y\,[i]$ | $z\,[i]$ |
|---|---|---|---|---|
| - | - | $1$ | $0$ | $70^\circ$ |
| $0$ | $1$ | $1$ | $1$ | $25^\circ$ |
| $1$ | $1$ | $0.5$ | $1.5$ | $-1.5651^\circ$ |

خروجی:




در این قسمت ورودی اول را زاویه ۳۰در جه دادیم که در باینری معادل با ۰۰۰۱۰۰۰۰۱۱ است(از سمت چپ بیت اول معادل علامت، بیت دوم و سوم قسمت صحیح میباشد و بقیه بیت ها مقدار اعشاری میباشد)

همانور که مشاهده میکنید خروجی اول را 0.5 یعنی ۰۰۰۱۰۰۰۰۰۰ داده

ورودی بعدی را ۷۰ و بعدی را ۶۰ درجه دادیم، همانطور که مشاهده میکنید مقدار خروجی های این دو نیز درست است.