SAKARYA ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ DERGİSİ
http://dergipark.gov.tr/saufenbilder

Jenerik Cordic algoritmasının FPGA'da donanımsal gerçeklenmesi

Suhap Şahin[1*], Burcu Kır Savaş[2]

07.05.2016Geliş/Received,06.10.2016Kabul/Accepted

doi: 10.16984/saufenbilder.14583

ÖZ

Trigonometrik, logaritmik, hiperbolik vb. matematiksel fonksiyonlarının donanımsal gerçeklenmesi sıklıkla kullanılmaktadır. Bu fonksiyonların donanımsal gerçeklenmesi yöntemlerden biri olan CORDIC algoritması donanım kaynağı, güç tüketimi açısından ön plana çıkmaktadır. Çalışmada sinyal işleme uygulamalarında kullanmak amacıyla döndürme ve vektörelmodlarda dairesel açı dönüşümleri kullanan Jenerik CORDIC algoritmasının FPGA'degerçeklenmesi anlatılmıştır. Uygulamada farklı iterasyon değerlerinde ve veri uzunluklarında sentez sonuçlarıyla birlikte gerçekleme sonucunda ortalama karesel hata değerleri karşılaştırmalı olarak verilmiştir. Sonuçlarda, sabit çarpan değerinde iterasyon sayısının donanımsal gerçeklemeyi etkilemediği ve sabit iterasyon değerinde çarpan değeri arttıkça çıkışta elde edilen sonuçların hata değerlerinin azaldığı gözlemlenmiştir. Gerçeklemede Xilinx firmasına ait Artix-7 FPGA XC7A100T-1CSG324C FPGA entegresi kullanılmıştır.

Anahtar Kelimeler: CORDIC, döndürme modu, vektörel mod, FPGA

Hardware iplementation of generic CORDIC algorithm on FPGA

ABSTRACT

Trigonometric, exponential, logarithmic, hyperbolic and several other mathematical functions, are frequently used in hardware implementation applications. CORDIC algorithm, which is a widely used method for hardware implementation of these functions due to its efficient space utilization and low power consumption. In this study, FPGA hardware implementation of rotation angle conversion and circular vector mode CORDIC algorithm for signal processing applications is described. The resulting mean squared error values are provided with respect to different data lengths and different iterations. In this study, the target implmentation platform is Xilinx Artix-7 FPGA platform.

Keywords: CORDIC, rotation mod, vectorial mod, FPGA

1. GİRİŞ (INTRODUCTION)

Teknolojinin gelişmesi ve kullanılan cihazların sürekli küçülmesi ile birlikte gömülü sistemler hayatımızın her alanında kullanılmaktadır. Özellikle sinyal ve görüntü işleme, iletişim sistemleri, robotik gibi çeşitli alanlarda ki uygulamaların gerçeklenmesinde kullanılan gömülü sistemler genellikle trigonometrik, logaritmik, hiperbolik gibi temel matematik fonksiyonlarına ihtiyaç duymaktadır [1][2][3][4][5][6]. Bu fonksiyonların donanımsal olarak gerçeklenmesi oldukça zor ve maliyetli olması nedeniyle FPGA, ASIC gibi mimariler üzerinde bu fonksiyonların gerçeklenebilmesi için farklı matematiksel yaklaşımlara dayalı yöntemleri geliştirilmiştir [7],[8].

* Sorumlu Yazar / Corresponding Author
1 Kocaeli Üniversitesi, Mühendislik Fakültesi Bilgisayar Mühendisliği, Kocaeli - suhapsahin@kocaeli.edu.tr
2 Kocaeli Üniversitesi, Mühendislik Fakültesi Bilgisayar Mühendisliği, Kocaeli- burcu.kir@kocaeli.edu.tr

Literatürde bu mimariler üzerinde ilgili fonksiyonların gerçeklenmesine ilişkin kullanım yoğunluğuna göre 3 donanımsal gerçekleme yöntemi ön plana çıkmaktadır [9].

İlk yöntem olan seri açılım ve polinom yaklaşımı donanımsal olarak yoğun iş yükü gerektirmektedir. İkinci yöntem olan bak-oku tablosu yaklaşımında ise uygulamaya ait veri yapısının çözünürlüğünü artırmak için yüksek miktarda hafıza birimi kullanmak gerekmektedir [10].

İlgili fonksiyonların donanımsal olarak gerçeklenmesine yönelik önerilen son yöntem olan CORDIC (COrdinateRotationalDIgitalComputer) algoritması [11], kartezyen koordinat sisteminde birim uzunluktaki bir vektörün döndürülerek vektöre ait açı, uzunluk ve yeni kartezyen koordinat bileşenlerinin hesaplanması esasına dayanan bir yaklaşımdır [12].

Hesaplamalarda iki ve ikinin katlarını iteratif olarak kullanan CORDIC algoritması, donanımsal olarak sadece öteleme işlemine ihtiyaç duymaktadır. Bundan dolayı hız ve maliyet olarak avantaj sağlamaktadır [13]. Böylece son zamanlarda literatürde sunulan matematik fonksiyonlarının donanımsal gerçeklenmesine ilişkin çalışmalarda CORDIC algoritması tercih edilmiştir [14][15][16].

Bu makalede, döndürme modda (rotationmode) ve vektörelmodda (vectoringmode) dairesel açı dönüşümü yöntemleri kullanan jenerik CORDIC algoritmasının FPGA'da donanımsal gerçeklenmesi anlatılmıştır. Çalışmada, literatürde sunulan çalışmalardan farklı olarak değişik veri uzunlukları ve iterasyon değerlerinde donanımsal gerçeklenen CORDIC algoritmasının çıkış değerlerinde oluşan ortalama karesel hata değerleri ve donanım kaynağı tüketim değerleri gösterilmiştir.

Gerçeklemede Xilinx firmasına ait Artix-7 FPGA XC7A100T-1CSG324C FPGA entegresi kullanılmıştır. Çalışmada sayı formatı olarak tam sayı formatı seçilmiştir.

2. CORDIC ALGORİTMASI (CORDIC ALGORITHM)

Trigonometrik fonksiyonların bilgisayar tarafından hesaplanabilmesi için, 1959 yılında JackVolder tarafından ortaya sürülen CORDIC algoritması, 1971 yılında J.S. Walther tarafından hiperbolik ve üstel fonksiyonlar, logaritma, karekök hesaplamaları yapabilecek şekilde geliştirilmiştir [17][18].

Denklem (1)'de;

$-\frac{\pi}{2} \le \theta \le \frac{\pi}{2}$ sınır aralığında tanımlı CORDIC algoritması genel ifadesi verilmiştir.

$$\begin{aligned} x_{i+1} &= x_i - \mu d^i y_i 2^{-i} \\ y_{i+1} &= y_i + \mu d^i x_i 2^{-i} \\ z_{i+1} &= z^i - d^i e_i \end{aligned} \quad (1)$$

Denklem (1)'de tanımlı d^i parametresi, Denklem (2)'de gösterildiği gibi z_i değerinin pozitif veya negatif olmasına göre -1 veya 1 değerini almaktadır. e_i parametresi dairesel, doğrusal ve hiperbolik hesaplamaları yapan her denklem için farklı değer almaktadır. Dairesel dönüşüm için tanımlı eşitlik Denklem (3)'de, doğrusal dönüşüm için tanımlı eşitlik Denklem (4)'de ve hiperbolik dönüşüm için kullanılan eşitlik Denklem (5)'de gösterilmiştir.

$$d_i = \begin{cases} 1, & z_i > 0 \\ -1, & z_i \le 0 \end{cases} \quad (2)$$

$$e_i = \tan^{-1}\left(2^{-i}\right) \quad (3)$$

$$e_i = 2^{-i} \quad (4)$$

$$e_i = \tanh^{-1}\left(2^{-i}\right) \quad (5)$$

Denklem (1)'de tanımlı μ parametresi yapılacak hesaplama tekniğine göre değer alan sabit parametredir. Bu değerleri **Hata! Başvuru kaynağı bulunamadı.**'de gösterilmiştir. Her dönüşüm tipi döndürme ve vektörel olmak üzere iki çözüm moduna sahiptir.

Tablo 1. Dönüşüm tipine göre μ parametresinin alacağı değerler (The values which μ parametres will take according to conversion types)

Dönüşüm Tipi	μ
Dairesel	1
Doğrusal	0
Hiperbolik	-1

2.1. Dairesel Açı Dönüşümü (Circular Angle Conversions)

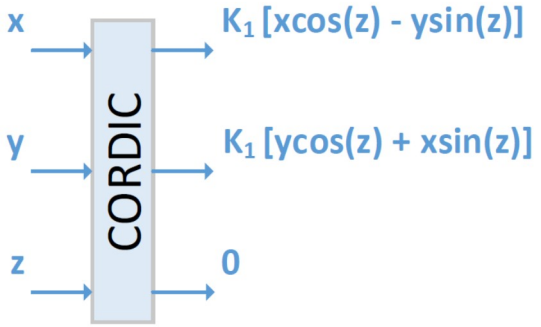
Dairesel açı dönüşüm işlemlerinde, **Hata! Başvuru kaynağı bulunamadı.**'den de görüleceği üzere Denklem (1)'de tanımlanan μ parametre değeri 1 olmaktadır. Dairesel açı dönüşüm işlemlerinde döndürme modu kullanılarak direkt olarak sinüs ve kosinüs değerleri dolaylı olarak ise tanjant ve kotanjant değerleri elde edilebilmektedir. Vektörel modda ise kartezyen

koordinat değerlerinin polar koordinat değerlerine dönüşümü yapılmaktadır.

2.1.1. Döndürme Modu (Rotation Mod)

Döndürme modunda dairesel açı dönüşüm işlemlerinde temel amaç Denklem (1)'de tanımlanan z değişkeni değerini sıfıra yaklaştırmaktadır (Denklem (6), **Hata! Başvuru kaynağı bulunamadı.**).

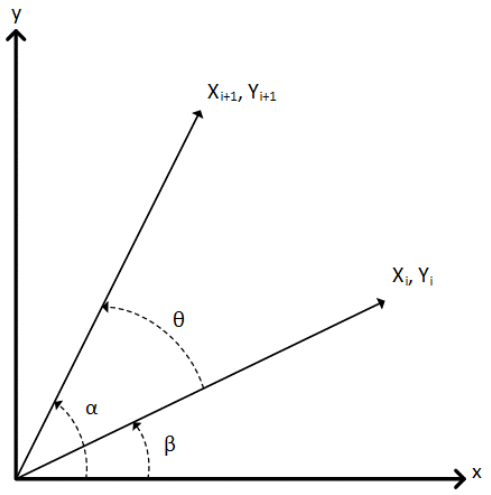
$$\begin{aligned} x_m &= K_1\left[x\cos(z) - y\sin(z)\right] \\ y_m &= K_1\left[y\cos(z) + x\sin(z)\right] \\ z_m &= 0 \end{aligned} \tag{6}$$



Şekil 1. Döndürme modunda Dairesel Açı Dönüşümü ile elde edilebilecek değerler (Values which can be obtained by circular angle rotation in rotation mode)

Döndürme modda dairesel açı dönüşümü, vektörün ***i.*** anındaki pozisyonu ile ***(i+1).*** anındaki pozisyonu arasındaki açı değeri ***θ***sıfırlanana kadar kaydırma işlemlerinin gerçekleştirilmesiyle yapılmaktadır (Şekil 1). Denklem (7)'de verilen eşitlikle döndürme modda dairesel açı dönüşüm işlemi gerçekleştirilmektedir.

$$\begin{bmatrix} X_{i+1} \\ Y_{i+1} \end{bmatrix} = \cos\theta_i \begin{bmatrix} 1 & -\tan\theta_i \\ \tan\theta_i & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \tag{7}$$



Şekil 1. Döndürme modunda Dairesel Açı Dönüşümü(Circular angle conversion in rotation mode)

i. adım ile (***i+1***). adım arasında gerçekleştirilecek döndürme işlemi açı değeri Denklem (8)'deki gibi hesaplanır. Bütün adımlardaki açıların toplamı döndürme açısı θ'yı vermelidir. Denklem (9)'de tanımlı S_n parametresi $\{-1 \quad 1\}$ değerlerini almaktadır. Bu bilgiler doğrultusunda Denklem (7)'de tanımlanan $\tan\theta_i$ değeri Denklem (10)'daki gibi ifade edilmektedir.

$$\theta_i = \arctan\left(\frac{1}{2^i}\right) \tag{8}$$

$$\sum_{n=0}^{\infty} S_n\theta_n = \theta \tag{9}$$

$$\tan\theta_i = S_i 2^{-i} \tag{10}$$

Denklem (10)'da verilen ifadeyi Denklem (6)'da yerine koyduğumuzda Denklem (11) elde edilir.

$$\begin{bmatrix} X_{i+1} \\ Y_{i+1} \end{bmatrix} = K_i \begin{bmatrix} 1 & S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \tag{11}$$

Denklem (11)'da tanımlanan K_i değişkeni Denklem (12)'de ki gibi hesaplanır.

$$K_i = \frac{1}{\sqrt{1 + 2^{-2i}}} \tag{12}$$

K_i parametresiiteratif süreç içerisinde ihmal edilebilir ve daha sonra bir ölçekleme faktörü olarak uygulanabilir (Denklem 13).

$$K(n)=\prod_{i=0}^{n-1}K_i=\prod_{i=0}^{n-1}\frac{1}{\sqrt{1+2^{-2i}}} \quad (13)$$

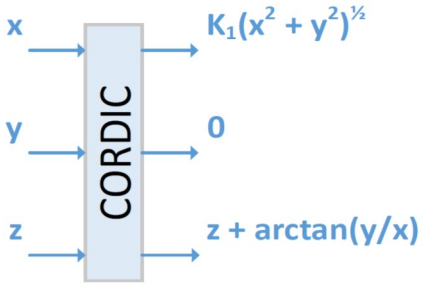
2.1.2. Vektörel Mod(Vectorial Mod)

Vektörel dairesel açı dönüşüm işlemlerinde temel amaç Denklem (1)'de tanımlanan y değişkeni değerini sıfıra yaklaştırmaktır (Denklem (14), Şekil 2). Başlangıç değerleri $x_0=1$ ve $z_0=0$ seçildiği durumda kartezyen koordinat değerlerimin polar koordinat değerlerine dönüşümü yapılmaktadır (Denklem (14).

Denklem (14)'de tanımlanan K_1 parametresi Denklem (15)'deki gibi hesaplanmaktadır.

$$\begin{aligned} x_m &= K_1\sqrt{x^2+y^2} \\ y_m &= 0 \\ z_m &= z+\arctan\left(\frac{y}{x}\right) \end{aligned} \quad (14)$$

$$K_1=\prod_{i=0}^{n-1}\sqrt{1+2^{-2i}} \quad (15)$$



Şekil 2. Vektörel Dairesel Açı Dönüşümü ile elde edilebilecek değerler(Values which can be obtained by circular angle conversion in vectoriel mode)

3.CORDIC ALGORITMASININ DONANIMSAL GERÇEKLENMESİ(HARDWARE IMPLEMENTATION OF CORDIC ALGORITHM)

Bu bölümde, Bölüm 2'de tanıtılan Döndürme ve Vektörel modda Dairesel Açı Dönüşüm işlemlerinin jenerik olarak FPGA tabanlı gerçeklenmesi anlatılmaktadır. Şekil 3'de CORDIC algoritmalarının gerçeklenmesi aşamasında kullanılan ***GENERIC*** parametreleri gösterilmektedir. ***ITERATION*** parametresi algoritmanın istenen değeri hesaplaması için yineleme sayısını ifade etmektedir. Bu değer arttıkça çıkışta elde edilecek sonuçların duyarlılığında da artış gözlemlenecektir. ***MUL_COEFF*** parametre değeri ise gerçel sayı formatında hesaplanan tablo değerlerinin tam sayı formatına dönüştürülmesinde kullanılacak 2'nin katı olan katsayı değerini (2^{MUL_COEFF}) ifade etmektedir. ***DATA_SIZE*** parametresi ise çıkışta elde edilecek verilerin uzunluğunu ifade etmektedir. Bu değerin tanımlanması sırasında, ***MUL_COEFF*** değerine $ceil\left(\log_2^{ITERATION}\right)$ değerinin eklenmesi çıkışta elde edilecek sonuçların doğruluğu açısından önerilmektedir.

```
generic(
    ITERATION : integer;
    MUL_COEFF  : integer;
    DATA SIZE  : integer );
```

Şekil 3. Jenerik CORDIC algoritmalarında kullanılan parametreler (Paramet resused in Jenerik CORDIC)

3.1 Jenerik Dairesel Açı Dönüşümünün Donanımsal Gerçeklenmesi: Döndürme Modu (Hardware Implementation of Generic Circular Angle Conversions : Rotation Mod)

Diresel açı dönüşüm işlemlerinin döndürme modda gerçekleştirilmesi aşamasında öncelikle döndürme açı değerlerinin ***ITERATION*** parametre değerine bağlı olarak bak-oku tablosu oluşturulması gerekmektedir. Dönüşüm işlemleri sırasında her iterasyonda ilgili açı değeri bak-oku tablosundan okunarak sonuç değeri hesaplanmaktadır.

FPGA tabanlı gerçeklemede döndürme açı değerlerinin tutulduğu bak-oku tablosu oluşturma işlemi için gerekli tip ve fonksiyon tanımlama VHDL kodları Şekil 4'de gösterilmiştir. Şekil 4'te ***t_Cordic_values*** tipi ***real*** verilerden oluşan ***ITERATION*** parametre değeri uzunluğunda bir bak-oku tablosu tanımlamaktadır. ***f_Calc_Angels*** fonksiyonu, Denklem (7)'yi kullanılarak ***ITERATION*** parametre değeri uzunluğunda ***t_Cordic_values*** tipinde bak-oku tablosu üretmektedir. ***ITERATION*** değeri 12 seçildiği durumda Tablo 2'deki gibi bak-oku tablosu oluşturulmaktadır.

Tablo 2. Denklem (7)'de tanımlı denklem kullanılarak bak-oku tablosu değerlerinin oluşturulması (Formation of look- read table values using equation(7))

i	Denklem (7)	i	Denklem (7)
1	0.78539816339745	7	0.01562372862048
2	0.46364760900081	8	0.00781234106010
3	0.24497866312686	9	0.00390623013197
4	0.12435499454676	10	0.00195312251648
5	0.06241880999596	11	0.00097656218956
6	0.03123983343027	12	0.00048828121119

```
typet_Cordic_values isarray(0
toITERATION - 1) ofreal;
…
functionf_Calc_Angels(ITERATION:
integer) returnt_Cordic_values is
variablev_K_angels : t_Cordic_values;
begin
forn_i in0 toITERATION - 1 loop
        v_K_angels(n_i) :=
arctan(2.0**(-1.0 * real(n_i)));
endloop;
returnv_K_angels;
endf_Calc_Angels;
```

Şekil 4. Açı değerlerinin oluşturulmasında kullanılan veri tipi ve fonksiyonun VHDL kodları (Data typeused in the formation of angle value sand VHDL codes of the function)

Denklem (12)'de tanımlanan parametre değerlerinin hesaplanması için Şekil 6'da gösterilen f_Calc_K_values fonksiyonu kullanılmaktadır. Her bir iterasyon değerleri için ölçekleme faktörü değerleri hesaplanmaktadır.

```
functionf_Calc_K_values(ITERATION:
integer) returnt_Cordic_values is
variablev_K_values : t_Cordic_values;
begin
    v_K_values(0) := 1.0 / sqrt(2.0);
forn_i in1 toITERATION - 1 loop
        v_K_values(n_i) :=
v_K_values(n_i - 1) * (1.0 / sqrt(1.0
+ 2.0**(-2.0 * real(n_i))));
endloop;
returnv_K_values;
endf_Calc_K_values;
```

Şekil 5. Her bir iterasyon değeri için ölçekleme faktörünün hesaplayan VHDL kodları (VHDL codes calculating the scaling factor for each iteration)

Şekil 4 ve Şekil 5'dan da görüleceği üzere ***f_Calc_Angels*** ve ***f_Calc_K_values*** fonksiyonlarının döndürdüğü değerler ***real*** veri tipindedir. Bu veri tipi FPGA için sentezlenebilir değildir. Bu nedenle bu değerler normalize edildikten sonra ***MUL_COEFF*** parametresi ile çarpılarak ***integer*** veri tipine ***f_Conv_Real_to_Int*** fonksiyonu ile dönüştürülmektedir (Şekil 6).

```
functionf_Conv_Real_to_Int(r_Cordic_va
lues : t_Cordic_values; ITERATION,
MUL_COEFF : integer ) returnt_Int_data
is
    variable v_Int_data : t_Int_data;
begin
forn_i in0 toITERATION - 1 loop
       v_Int_data(n_i) :=
integer(r_Cordic_values(n_i) *
real(2**MUL_COEFF));
endloop;
returnv_Int_data;
endf_Conv_Real_to_Int;
```

Şekil 6. Gerçel sayı veri tipinden tam sayı veri tipine dönüşüm işlemini gerçekleştiren VHDL kodu (VHDL code processing transformation of real number to whole number)

Örneğin 0.5 değeri ***MUL_COEFF*** parametresin değeri 8 olduğu durumda 4 değerine çevrilmektedir. ***f_Conv_Real_to_Int*** fonksiyonu ile elde edilen tam sayı değerleri ile jenerik olarak tanımlanan veri uzunluğunda işlem yapabilmek amacı ile tür dönüşümü işleminin gerçekleştirildiği ***f_Conv_Int_to_Std_Logic_Vector*** fonksiyonu **Hata! Başvuru kaynağı bulunamadı.**'de gösterilmiştir.

Jenerik CORDIC algoritmasının donanımsal gerçeklenmesine ilişkin sözde kod Şekil 8'da verilmiştir. Şekil 9'da tanımlanan 1., 2. ve 3. adımların jenerik parametrelerine bağlı olarak gerçeklenmesine ilişkin VHDL kodları Şekil 9, Şekil 10 ve **Hata! Başvuru kaynağı bulunamadı.**'de verilmiştir.

```
functionf_Conv_Int_to_Std_Logic_Vector(
r_Int_values : t_Int_data; ITERATION,
DATA_SIZE : integer)
returnt_Std_Logic_Vector_data is
   variable v_Std_Logic_Vector_data :
t_Std_Logic_Vector_data;
begin
forn_i in0 toITERATION - 1 loop
        v_Std_Logic_Vector_data(n_i):=
conv_std_logic_vector(
r_Int_values(n_i), DATA_SIZE);
endloop;
returnv_Std_Logic_Vector_data;
endf_Conv_Int_to_Std_Logic_Vector;
```

Şekil 7. Tam sayı veri tipinden std_logic_vector veri tipine dönüşüm işlemini gerçekleştiren VHDL kodu (VHDL code processing transformation of whole number to std_logic_vector number)

Şekil 9'da1. adımda tanımlanan işlemlerin gerçeklenmesine ilişkin VHDL kodları Şekil 10'da verilmiştir. Şekil 10'dan da görüleceği üzere açı değerinin pozitif veya negatif olma durumuna göre atama işlemleri yapılmaktadır.

1. Açı değerine bağlı olarak Denklem (2)'de tanımlanan işlemleri gerçekleştir (Şekil 9).
2. Denklem (10)'da tanımlan işlemleri gerçekleştir (Şekil 10).
3. Parametreleri güncelle (**Hata! Başvuru kaynağı bulunamadı.**).
4. Tüm iterasyon değerleri için işlemler koşturulduysa 5. adıma, aksi durumda 1. Adıma geç.
5. Denklem (12)'de tanımlanan işlemleri

Şekil 8. CORDIC algoritmasının donanımsal gerçeklenmesine ilişkin sözdekod (Pseude code related to the hardware realization of CORDIC algorithm)

Şekil 8'da 1. adımda tanımlanan işlemlerin gerçeklenmesine ilişkin VHDL kodları Şekil 9'da verilmiştir. Şekil 9'dan da görüleceği üzere açı değerinin pozitif veya negatif olma durumuna göre atama işlemleri yapılmaktadır.

Şekil 10'de Şekil 8'da 2. adımda tanımlanan işlemlerin gerçeklenmesine ilişkin VHDL kodları verilmiştir. Şekil 10'den de görüleceği üzere ***CALC_FACTOR*** ve ***CALC_NEW_V*** durumlarında Denklem (10)'da tanımlanan işlemler gerçeklenmektedir. ***SET_NEW_V*** değerinde güncelleme işlemleri yapılarak iterasyondaki yeni değerler, vektördeki yerlerine atanmaktadır.

```
if r_teta < 0 then
r_sigma <= (conv_std_logic_vector(-1 *
2** MUL_COEFF, DATA_SIZE));
else
r_sigma <= (conv_std_logic_vector(1 *
2** MUL_COEFF, DATA_SIZE));
endif;
```

Şekil 9.1. adımda tanımlanan işlemlerin gerçeklenmesine ilişkin VHDL kodları (VHDL codes related to realization of processes in the first step)

```
when CALC_FACTOR =>
r_factor <= r_sigma * r_POW_of_2;
    r_Cordic_Cntrl <= CALC_NEW_V;

when CALC_NEW_V =>
    r_V_new_0 <=
conv_std_logic_vector(2**MUL_COEFF,
DATA_SIZE) *  r_V_vector(0) -
    r_V_vector(1) * r_factor(MUL_COEFF +
DATA_SIZE - 1 downto MUL_COEFF);
r_V_new_1 <=
conv_std_logic_vector(2**MUL_COEFF,
DATA_SIZE) *  r_V_vector(1) +
    r_V_vector(0) * r_factor(MUL_COEFF +
DATA_SIZE - 1 downto MUL_COEFF);
r_Cordic_Cntrl <= SET_NEW_V;

when SET_NEW_V =>
   r_V_vector(0) <= r_V_new_0(MUL_COEFF
+ DATA_SIZE - 1 downto MUL_COEFF);
 r_V_vector(1) <= r_V_new_1(MUL_COEFF +
DATA_SIZE - 1 downto MUL_COEFF);
    r_teta_delta <= r_sigma * r_Angel;
```

Şekil 10. 2. adımda tanımlanan işlemlerin gerçeklenmesine ilişkin VHDL kodları (VHDL codes related to realization of processes in thesecond step)

Hata! Başvuru kaynağı bulunamadı.'de Şekil 8'da 3. adımda tanımlanan işlemlerin gerçeklenmesine ilişkin VHDL kodları verilmiştir. Bu adımda CORDIC algoritmasında kullanılan parametre güncelleme işlemleri yapılmaktadır.

3.2. Jenerik Dairesel Açı Dönüşümünün Donanımsal Gerçeklenmesi Vektörel Mod(Hardware Implementation of Generic Circular Angle Conversions : Vectorial Mod)

Diresel açı dönüşüm işlemlerinin vektörelmodda gerçekleştirilmesi aşamasında ***ITERATION***paremetre değerine bağlı olarak bak-oku tablosu oluşturulması gerekmektedir. Dönüşüm işlemleri sırasında her iterasyonda ilgili değer bak-oku tablosundan okunarak sonuç değeri hesaplanmaktadır.

```
r_teta <= r_teta -
r_teta_delta(MUL_COEFF + DATA_SIZE - 1
downto MUL_COEFF);
r_POW_of_2 <= '0' &
r_POW_of_2(DATA_SIZE - 1 downto 1);

if n_i + 2 > VALUE_SIZE then
r_Angel <= '0' & r_Angel(DATA_SIZE - 1
downto 1);
else
r_Angel <= r_Angels_SLV_data(n_i + 1);
end if;
```

Şekil 11. 3. adımda tanımlanan işlemlerin gerçeklenmesine ilişkin VHDL kodları (VHDL codes related to realization of processes in the third step)

FPGA tabanlı gerçeklemede bak-oku tablosu oluşturma işlemi için gerekli fonksiyon tanımlamaları VHDL kodları Şekil 12'da gösterilmiştir. Şekil 12'da tanımlanan ***f_Calc_tp_values*** fonksiyonunun döndürdüğü değerlerin giriş olarak verildiği ***f_Calc_atantp_values*** fonksiyonun kullanarak ***ITERATION*** parametre değeri uzunluğunda ***t_Cordic_values*** tipinde bak-oku tablosu üretmektedir.

Denklem (14)'de tanımlanan K_1 parametre değerlerinin hesaplanması için Şekil 13'da gösterilen ***f_Calc_K_values***fonksiyonu kullanılmaktadır. Her bir iterasyon değerleri için ölçekleme faktörü değerleri hesaplanmaktadır.

Şekil 14'de vektörelmodda CORDIC algoritmasının donanımsal gerçeklenmesine ait şematik gösterimi verilmiştir.

3.3. Test Sonuçları (Test Results)

Tablo 3'te sabit çarpan (MUL_COEFF = 18) ve veri uzunluğu (DATA_WIDTH = 24) değerlerinde farklı iterasyon değerleri için algoritma çıkışında elde edile ***kosinüs*** ve ***sinüs*** değerlerinin ortalama karesel hata (OKH) oranları ve FPGA üzerinde kullandığı mandal sayıları gösterilmiştir. Tablo 3'te iterasyon sayısının değişmesi algoritmanın gerçeklenmesi esnasında kullanılan alanı etkilemediği ve 20. iterasyondan sonra OKH değerleri değişmediği görülmektedir.

Tablo 4'de sabit iterasyon (ITERATION = 24) değerinde farklı çarpan değerleri için algoritma çıkışında elde edile ***kosinüs*** ve ***sinüs*** değerlerinin ortalama karesel hata (OKH) oranları ve FPGA üzerinde kullandığı mandal sayıları gösterilmiştir. Tablo 4'de çarpan değerinin artması ile alan kullanımı artmakta fakat OKH değerlerinde azalma görülmektedir.

```
functionf_Calc_tp_values(ITERATION:
integer) returnt_Cordic_values is
variablev_tp_values : t_Cordic_values;
begin
    v_tp_values(0) :=
1.0;v_tp_values(1) :=
1.0;v_tp_values(2) := 1.0;
forn_i in3 toITERATION - 1 loop
        v_tp_values(n_i) :=
v_tp_values(n_i - 1) / 2.0;
endloop;
returnv_tp_values;
endf_Calc_tp_values;
……….
functionf_Calc_atantp_values(c_tp_value
s : t_Cordic_values; ITERATION:
integer) returnt_Cordic_values is
variablev_atantp_values :
t_Cordic_values;
begin
    v_atantp_values(0) := c_PI /
4.0;v_atantp_values(1) := c_PI / 4.0;
    v_atantp_values(2) := c_PI / 4.0;
forn_i in3 toITERATION - 1 loop
        v_atantp_values(n_i) :=
arctan(c_tp_values(n_i));
endloop;
returnv_atantp_values;
endf_Calc_atantp_values;
```

Şekil 12. Bak-oku tablosu oluşturmak için kullanılan fonksiyonun VHDL kodları (VHDL codes of the function which is used to form look- uptable)

Tablo 5'de sabit çarpan (MUL_COEFF = 18) ve veri uzunluğu (DATA_WIDTH = 24) değerlerinde farklı iterasyon değerleri için algoritma çıkışında elde edile ***genlik*** ve ***açı*** değerlerinin ortalama karesel hata (OKH) oranları ve FPGA üzerinde kullandığı mandal sayıları gösterilmiştir.

Tablo 5'den iterasyon sayısının değişmesi algoritmanın gerçeklenmesi esnasında kullanılan alanı etkilemediği 20. iterasyondan sonra OKH değerleri değişmediği görülmektedir.

Tablo 6. Farklı çarpan değerlerinde kosinüs ve sinüs değerleri için ortalama karasel hata değerleri ve kullanılan mandal sayıları (ITERASYON = 24) (Root-mean square values for mean cosinus and sinus values in different multiplier values and flip-flops that are used-ITERASYON = 24)

Mul_coeff	Okh_genlık	Okh_acı	Mandal	Dsp
10	2.698429×10-8	1.812010×10-7	108	3
12	9,482901×10-9	3,605849×10-9	120	3
16	1,865137×10-10	9,353406×10-11	144	3

20	2,532061×10-13	4,06095 4×10-13	168	6
24	1,108513×10-14	1,84159 2×10-14	227	10

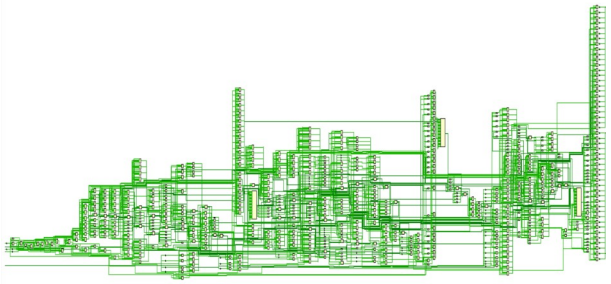
4. SONUÇLAR(CONCLUSION)

Bu çalışmada döndürme modda dairesel açı dönüşümü kullanılarak sinüs ve kosinüs değerlerinin hesaplanması ve vektörel modda dairesel açı dönüşümü kullanılarak da bir vektörün polar koordinat değerlerinin hesaplanma işlemleri FPGA tabanlı donanımsal olarak gerçeklenmiştir. Gerçeklemede iterasyon sayısının donanımsal gerçeklemede kullanılan alan tüketimini etkilemediği ve 20. iterasyon değerinden sonra OKH değerlerinin değişmediği Tablo 3 ve Tablo 5'de gösterilmiştir. Bu durumun nedeni olarak seçilen çarpan değerinin bak-oku tablosu değerlerinin tümünü uygun seviyelere ölçekleyememesinden kaynaklanmaktadır. Örneğin döndürme modda ***f_Calc_Angels*** fonsiyonunun döndürdüğü 20 iterasyon değeri 218 ile çarpma işlemi gerçekleştirdiğimizde elde edilen değer 0 olacaktır. Bu nedenle 20. iterasyon değerinden sonra gerçekleştirilecek olan iterasyonlarda yine fonksiyon 0 değerini döndürecektir. İterayon değerinin sabit tutulup çarpan değerinin artırılması ile OKH değerlerinin değişiklikler gösterdiği Tablo 4 ve **Hata! Yer işareti başvurusu geçersiz.**'da gösterilmiştir. Tablolardan da görüleceği üzere çarpan değeri arttıkça sabit iterasyon değerinde çıkışta elde edilen sonuç değerlerinde hata değerleri azalmaktadır.

'de sabit iterasyon (ITERATION = 24) değerinde farklı çarpan değerleri için algoritma çıkışında elde edile ***genlik*** ve ***açı*** değerlerinin ortalama karesel hata (OKH) oranları ve FPGA üzerinde kullandığı mandal sayıları gösterilmiştir. Tablo 4'de çarpan değerinin artması ile alan kullanımı artmakta fakat OKH değerlerinde azalma görülmektedir.

```
functionf_Calc_K_values(c_atantp_values
: t_Cordic_values; ITERATION: integer)
returnrealis
variablev_K : real;
begin
    v_K := sqrt(2.0) / 4.0;
forn_i in3 toITERATION - 1 loop
        v_K := v_K *
cos(c_atantp_values(n_i));
endloop;
returnv_K;
endf_Calc_K_values;
```

Şekil 13. İterasyon değeri için ölçekleme faktörünü hesaplayan VHDL kodları (VHDL codes calculating the scaling factor for iteration value)



Şekil 14. VektörelModda CORDIC algoritması şematik gösterimiSchematicdisplay of CORDIC algorithminVectorelmode)

Tablo 3. Farklı iterasyon değerlerinde cosinüs ve sinüs değerleri için ortalama karasel hata değerleri ve kullanılan mandal sayıları (MUL_COEFF = 18, DATA_WIDTH = 24)(Root-mean square values for mean cosinus and sinus values in different iteration values and flip-flops that are used-MUL_COEFF = 18, DATA_WIDTH = 24)

Iteratıon	Okh_cos	Okh_sın	Mandal	Dsp
12	2,335501 ×10-8	1,93478 0×10-8	247	10
16	2,346898 ×10-10	2,34689 8×10-10	247	10
20	2,474788 ×10-10	1,26875 2×10-10	247	10
24	2,474788 ×10-10	1,26875 2×10-10	247	10
28	2,474788 ×10-10	1,26875 2×10-10	247	10

Tablo 4. Farklı çarpan değerlerinde ortalama kosinüs ve sinüs değerleri için ortalama karasel hata değerleri ve kullanılan mandal sayıları (ITERASYON = 24) (Root-mean square values for mean cosinus and sinus values in different multiplier value sand flip-flops that are used-ITERASYON = 24)

Mul_coeff	Okh_cos	Okh_sın	Mandal	Dsp
10	1,951241 ×10-6	1,951241× 10-6	175	6
12	5,513225 ×10-7	5,115861× 10-7	193	6
16	2,163471 ×10-9	1,697809× 10-9	271	6
20	1,208777 ×10-11	1,208777× 10-11	325	16
24	7,503266 ×10-14	7,503266× 10-14	337	20

Tablo 5. Farklı iterasyon değerlerinde cosinüs ve sinüs değerleri için ortalama karasel hata değerleri ve kullanılan mandal sayıları (MUL_COEFF = 18, DATA_WIDTH = 24)(Root-mean square values for mean cosinus and sinus values in different iteration value sand flip-flops that are used- MUL_COEFF = 18, DATA_WIDTH = 24)

Iteratıon	Okh_genlık	Okh_acı	Mandal	Dsp
12	8.435011×10-12	1.87307 0×10-7	202	3
16	5.593260×10-12	1.73585 4×10-9	202	3
20	5.593160×10-12	8.47966 6×10-11	202	3
24	5.593160×10-12	1.20239 0×10-11	202	3

28	5.593160×10^{-12}	1.202390×10^{-11}	202	3

Tablo 6. Farklı çarpan değerlerinde kosinüs ve sinüs değerleri için ortalama karasel hata değerleri ve kullanılan mandal sayıları (ITERASYON = 24) (Root-mean square values for mean cosinus and sinus values in different multiplier values and flip-flops that are used-ITERASYON = 24)

Mul_coeff	Okh_genlık	Okh_acı	Mandal	Dsp
10	2.698429×10^{-8}	1.812010×10^{-7}	108	3
12	$9,482901\times10^{-9}$	$3,605849\times10^{-9}$	120	3
16	$1,865137\times10^{-10}$	$9,353406\times10^{-11}$	144	3
20	$2,532061\times10^{-13}$	$4,060954\times10^{-13}$	168	6
24	$1,108513\times10^{-14}$	$1,841592\times10^{-14}$	227	10

5. SONUÇLAR(CONCLUSION)

Bu çalışmada döndürme modda dairesel açı dönüşümü kullanılarak sinüs ve kosinüs değerlerinin hesaplanması ve vektörel modda dairesel açı dönüşümü kullanılarak da bir vektörün polar koordinat değerlerinin hesaplanma işlemleri FPGA tabanlı donanımsal olarak gerçeklenmiştir. Gerçeklemede iterasyon sayısının donanımsal gerçeklemede kullanılan alan tüketimini etkilemediği ve 20. iterasyon değerinden sonra OKH değerlerinin değişmediği Tablo 3 ve Tablo 5'de gösterilmiştir. Bu durumun nedeni olarak seçilen çarpan değerinin bak-oku tablosu değerlerinin tümünü uygun seviyelere ölçekleyememesinden kaynaklanmaktadır. Örneğin döndürme modda ***f_Calc_Angels*** fonsiyonunun döndürdüğü 20 iterasyon değeri 2^{18} ile çarpma işlemi gerçekleştirdiğimizde elde edilen değer 0 olacaktır. Bu nedenle 20. iterasyon değerinden sonra gerçekleştirilecek olan iterasyonlarda yine fonksiyon 0 değerini döndürecektir. İterayon değerinin sabit tutulup çarpan değerinin artırılması ile OKH değerlerinin değişiklikler gösterdiği Tablo 4 ve **Hata! Yer işareti başvurusu geçersiz.**'da gösterilmiştir. Tablolardan da görüleceği üzere çarpan değeri arttıkça sabit iterasyon değerinde çıkışta elde edilen sonuç değerlerinde hata değerleri azalmaktadır.

KAYNAKÇA(REFERENCES)

[1] S. Karthick, P. Priya ve V. S, "CORDIC Based FFT for Signal Processing System", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, cilt 1, no. 6, 2012.

[2] Y. H. Hu ve Z. Wu, "An efficient CORDIC array structure for the implementation of discrete cosine transform", IEEE Transactions on Signal Processing , cilt 43, no. 1, pp. 331 - 336, 2002.

[3] Y. H. Hu, "On the Convergence of the CORDIC Adaptive Lattice Filtering (CALF) Algorithm", IEEE TRANSACTIONS ON SIGNAL PROCESSING, cilt 46, no. 7, pp. 1861-1871, 1998.

[4] S. Sharma, P. N. Ravichandran, S. Kulkarni, V. M. ve P. Lakshminarsimahan, "Implementation of Para-CORDIC Algorithm and Its Applications in Satellite Communication", International Conference on Advances in Recent Technologies in Communication and Computing, 2009 .

[5] P. Revathi, M. N. Rao ve G. Locharla, "Architecture Design and FPGA Implementation of CORDIC Algorithm for Fingerprint Recognition Applications", Procedia Technology, cilt 6, p. 371–378, 2012.

[6] P.Karthikeyan, K.Kavaskar, P.Kirbakaran, A.Manikandan ve R. Sekar, "VLSI Implementation of Cordic Based Robot Navigation Processor", International Research Journal of Engineering and Technology, cilt 3, no. 2, 2016.

[7] J. Sujitha ve V. R. Reddy, "Implementation of Log and Exponential Function in FPGA", International Journal of Engineering Research & Technology, cilt 3, no. 11, 2014.

[8] L. Deng, C. Chakrabarti, N. Pitsianis ve X. Sun, "Automated optimization of look-up table implementation for function evaluation on FPGAs", Mathematics for Signal and Information Processing, 2009.

[9] B. Lakshmi ve A. S. Dhar, "CORDIC Architectures: A Survey", VLSI Design, cilt 2010, 2010.

[10] M. A. Çavuşlu, C. Karakuzu ve F. Karakaya, "Neural identification of dynamic systems on FPGA with improved PSO learning", Applied Soft Computing, cilt 12, no. 9, p. 2707–2718, 2012.

[11] J. Volder, "The CORDIC Trigonometric Computing Technique", IRE Trans. Electronic Computers, cilt 8, pp. 330-334, 1959.

[12] B. Kir, M. Altuncu ve S. Şahin, "FPGA based implementation of CORDIC using different number format", Technological Advances in Electrical, Electronics and Computer Engineering, 2013.

[13] J. Sanchez, A. Jimeno, H. Mora, J. Mora ve F. Pujol, "A Cordic-based Architecture for High Performance Decimal Calculations", IEEE International Symposium on Industrial Electronics, 2007.

[14] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers", IEEE Transactions on Computers, cilt 45, no. 3, p. 328–339, 1998.

[15] K. Kumar, K. Rao ve R. Durga, "FPGA Implementation of DSWG Using CORDIC Algorithm", International Journal of Innovative Research in Computer and Communication Engineering, cilt 1, no. 7, 2013.

[16] R. Mehra ve B. Kamboj, "FPGA Implementation of Pipelined CORDIC Sine Cosine Digital Wave Generator", Int. J. Comp.Tech. Appl, cilt 1, no. 1.

[17] J. S. Walther, "A unified algorithm for elementary functions",Proceedings of the AFIPS Spring Joint Computer Conference, 1971.

[18] J. S. Walther, "The story of Unified CORDIC", Journal of VLSI Signal Processing, cilt 25, no. 2, p. 107–112, 2000.