# Haversine Formülü Kullanılarak Yapılan Eğim Aralığı Hesabının Alan Optimizasyonu ile FPGA Gerçeklemesi

# Area Optimized FPGA Implementation of Slant Range Calculation Using Haversine Formula

*Yazarlar Gizlenmiştir*

***Özetçe*—Bu çalışmada iki nokta arasındaki eğim aralığı (slant range) hesabının Xilinx Zynq serisi FPGA üzerinde implementasyonu gösterilmiştir. Hesaplama, çok yoğun sinyal işleme (DSP) blokları içermemesine rağmen %28 hafıza (LUT) ve %14 DSP bloğu kullanmaktadır. Zaman bölmeli çoklama (TDM) tekniğiyle birden fazla işlem aynı donanım blokları (resource sharing) üzerinde çalıştırılmıştır. Matematiksel hesaplamalar aynı donanım üzerinde sıralı bir şekilde koşturulmuştur. Bu sayede LUT kullanımı %25'e, DSP bloğu kullanımı ise %5'e düşürülmüştür. Kaynakların kullanımında %20'ye yakın bir azalma sağlanmıştır. FPGA'lerin çok yüksek hızlara çıkabilme özelliği sebebiyle ortaya çıkan latency artışı, hesaplamanın gerçek zamanlı olmasını engellememektedir. Sabit noktalı işlemlerden (fixed point) kaynaklı kesim ve yuvarlama hataları %1 in altında tutulmuştur.**

***Anahtar Kelimeler — Sahada programlanabilir kapı elemanları, Haversine formülü, Eğim aralığı, Zaman-bölmeli çoklama, Kaynak paylaşımı.***

***Abstract*— In this study, the implementation of slant range calculation between two points on Xilinx Zynq series FPGA is shown. Although the calculation does not include very dense signal processing (DSP) blocks, it utilizes 28% memory (LUT) and 14% DSP blocks. With the time-division multiplexing (TDM) technique, more than one process was run on the same hardware blocks (resource sharing). Mathematical calculations were performed sequentially on the same hardware. In this way, the use of LUT was reduced to 25% and the use of DSP block was reduced to 5%. A reduction of nearly 20% has been achieved in the use of resources. The latency increase due to the ability of FPGAs to reach very high speeds does not prevent the calculation to be real-time. Truncation and rounding errors caused by fixed-point operations were kept below 1%.**

***Keywords — FPGA, Haversine equation, Slant range, Time-division multiplexing, Resource sharing***

## I. GİRİŞ

Eğim aralığı(Slant range) aynı seviyede olmayan iki nokta arasındaki görüş hattı mesafesidir. Bu bazen havadaki hedefin kara radar antenine göre mesafesini veya bazen hava radar antenin yerdeki hedefe olan uzaklığını gösterir. Bu mesafe bulunurken yerin küresel geometrisi de hesaba katılır. Yerin geometrisini kullanan eşitliklerden birisi de Haversine formülüdür[1,2,3]. Kullanılan formülasyonlarda çarpma, trigonometrik fonksiyonlar, bölme ve karekök alma gibi matematiksel işlemler yer almakta ve FPGA kaynaklarını çok fazla kullanarak kaynak sorunu doğurmaktadır.

Kaynak paylaşımı optimizasyonları ile gerekli alanın küçültülmesi gerekmektedir. Bu teknikler son yıllarda sıkça kullanılmaktadır. Örneğin, kaynak paylaşımının yapılacak işleme ve FPGA mimarisine bağlılığı gösterilmiş [4] ve cyclone2 üzerinde %16, stratix4 üzerinde %12 alan kaynak düşüşü sağlanmıştır. ALU tasarımı gibi bazı işlemlerde kaynak kullanımı %66 gibi yüksek bir oranda azaltılmıştır [5]. Yine bir başka çalışmada FIR filtre %20 daha az alan kullanımı ile gerçekleştirilmiştir[6].

Bu çalışmanın 2.bölümünde formülasyonlar verilmiş ve 3. ve 4. Bölümlerde hava aracının takip ettiği rotaya göre eğim aralığı hesaplanmıştır. Hangi FPGA kaynaklarının sıralı şekilde paylaşıldığı, kaynak kullanımındaki düşüş gösterilmiş sabit noktalı hesap hataları gösterilmiştir.

## II. FORMULASYON

### A. Haversine Denklemi

Haversine denklemi, iki noktanın boylam ve enlemlerinden büyük daire mesafelerini hesaplayarak uzaklık değerini vermektedir. Kullanılan formül(1) de gösterilmiştir.

```
function[d]= haversine(lat1,lat2,long1,long2)
rad_cons= pi/180;
R= 6371000;
lat_1= (rad_cons*(lat1));
lat_2= (rad_cons*(lat2));
long_1= (rad_cons*(long1));
long_2= (rad_cons*(long2));
dlat= lat_1-lat_2;
dlong= long_1-long_2;
a1= sin(dlat/2)^2;
a2= cos(lat_1);
a3= cos(lat_2);
a4= sin(dlong/2)^2;
a= a1+a2*a3*a4;
c1= sqrt(a);
c2= sqrt(1-a);
c3= atan2(c1,c2);
c= 2*c3;
d= R*c                                                    (1)
```

### B. *Eğim aralığı*

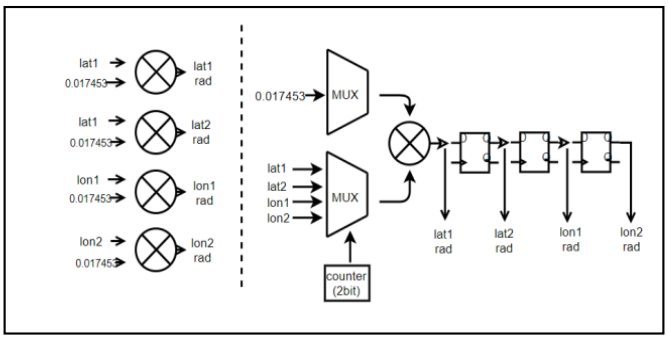
Haversine denkleminden gelen izdüşüm mesafesi ile iki noktanın yüksekleri de hesaba katılarak eğim aralığı hesabı yapılmalıdır. Kullanılan formül (2) de gösterilmiştir.

$$s\_range = \sqrt{[d^2 + (h_1 - h_2)^2]} \qquad (2)$$

d: Haversine mesafesi
$h_1$:radar yüksekliği
$h_2$:hedef yüksekliği

### C. *Kaynak Paylaşım Optimizasyonu*

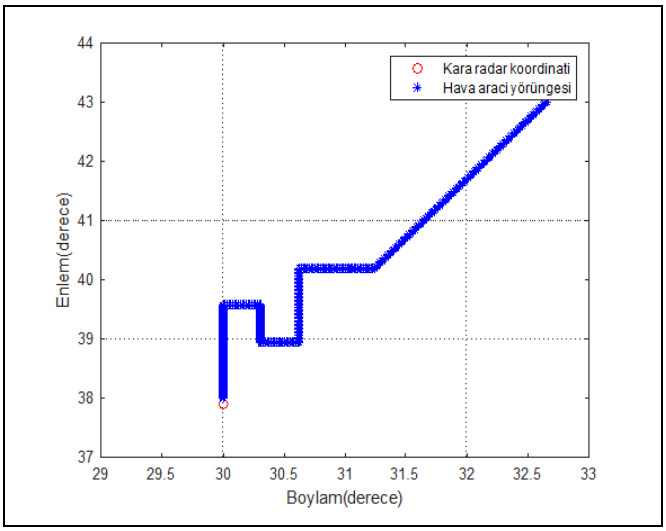
Kaynak paylaşımı kavramına ait bir örnek Şekil1'de verilmiştir. Sol tarafta derece-radyan çevrimi 4 çarpma bloğuyla yapılmakta iken sağ tarafta çarpma bloğunun tekrarlı kullanımı sayesinde 1 çarpma bloğu kullanılmış fakat işlem 4 kat yavaş yapılmıştır. Çarpma bloğu çıkışında kayan yazmaç (shift register) kullanılarak işlem 4 saat saykılında gerçekleşmiştir.




Şekil. 1.Kaynak paylaşımı

## III. SENARYO

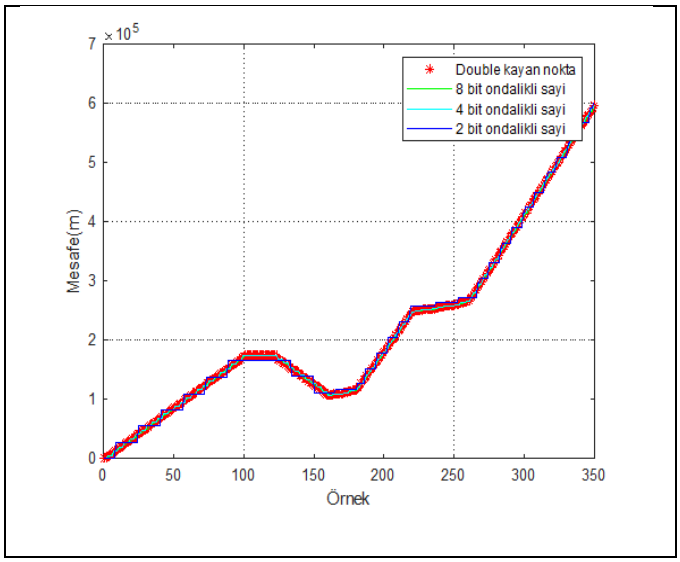
Şekil 2'de sabit kara radarı ve hareketli hava aracının izlediği yol enlem-boylam haritasında gösterilmektedir. Kara radarı, 38 derece kuzey enlemi ve 30 derece doğu boylamına sabitlenmiştir. Şekil üzerinde görülebilmesi için radar koordinatlarına 0.1 derece kaçıklık verilmiştir. Hava aracı, 38-43 kuzey enlemleri ve 30-32.65 doğu boylamları arasında Şekil 2'de gösterilen rotayı takip ederek radardan uzaklaşmaktadır. Haversine ark boyu yaklaşık 600 km'dir. Hava aracı sabit irtifada 10 km yükseklikte seyretmektedir. Hava aracının yerden yüksekliği enlem-boylam farkının yanında küçük olduğundan eğim aralığı da yaklaşık 600 km hesaplanmaktadır.



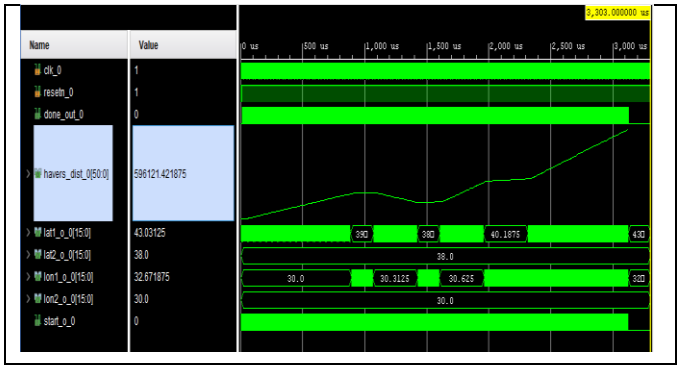

Şekil. 2.Hareket yörüngesi

## IV. IMPLEMENTASYON

Koordinat bilgisinin veri tipi yapılacak tüm matematiksel işlemlerin sonuçlarının veri tipini etkilemesi yanında kullanılacak FPGA kaynak sayısını da etkilemektedir. Şekil 3'de hedef koordinatlarının sabit noktalı temsil edilmesi durumunda hesaplanan mesafeler gösterilmiştir. Sabit noktalı koordinat bilgisinde 2 bit ondalık kullanılınca yaklaşık %3 hata, 4 bit kullanılınca %1 ve 8 bit olması halinde % 0.01'den az hata alınmıştır. Gelen koordinat bilgisinin çözünürlüğü yaklaşık 0.015 derece kabul edilmiştir. Hataları minimumda tutabilmek için işlemler 8 bit ondalık kullanılarak yapılmıştır.




Şekil. 3. Hedef koordinatlarının farklı ondalık sayılarla temsil edilmesi




Şekil. 4.Vivado simülasyon çıktısı

Şekil4, Vivado 2018.3 üzerinde yapılan simülasyon sonucunda mesafe hesabını göstermektedir. Ayrıca kodların Vivado'da sentezi ve implementasyonu yapılmıştır.

Sabit noktalı hesap yöntemi doğruluk, donanım maliyeti, hız ve donanım limitleri gibi faktörler gözönüne alınarak seçilir. Literatürde sinüs-kosinüs hesabı için başlıca yöntemler look-up tablosu, taylor serisi açılımı[*] ve CORDIC algoritmasıdır. Karekök hesabı için geri yüklenemeyen(non-restroning) algoritması[*] ve ikiye bölme(bisection) nümerik yöntemi[*] CORDIC dışında sıklıkla kullanılır. Arktanjant hesabı için Chebyshev polinomu yaklaşımı kullanılmaktadır[*]. Yine basitleştirilmiş look-up tablosu üzerinden gidip en yakın komşu doğrusal interpolasyon ile hesaplama mümkündür. Fakat bu her iki yöntemde çarpma ve bölme operasyonları gerektirmektedir. Bu çalışmada genlik hesabı dahil tüm işlemler CORDIC ile yapılmış, gerekli doğruluk elde edilmiş ve matematiksel işlemler için farklı algoritmalar kullanılmadığından sonuca hızlı gidilmeye çalışılmıştır. Ayrıca donanım maliyeti azaltılmıştır. DSP48 Xilinx tarafından geliştirilen kompleks hesaplama bloğudur[*]. Bu makro blok çarpma işlemlerinde kullanılarak performans artışı sağlamaktadır.

CORDIC yöntemi, çarpma operatörü kullanmaz ve sadece kaydırma, toplama işlemleriyle sonuca yakınsamasıyla en etkili algoritmalardan biridir[*]. Trigonometrik değerler (sinüs, kosinüs, arktanjant) ve karekök alma işlemleri CORDIC algoritması ile iteratif bir şekilde bulunmuştur. Sinüs-kosinüs hesabı algoritmanın rotasyon modunda, arktanjant hesabı vektör modunda ve karekök hesabı hiperbolik vektör modunda yapılmıştır. İterasyon sayıları bağıl hatalara göre bulunmuş ve aşağıdaki Tablo 1'de iterasyon sayıları verilmiştir. Karekök alma yöntemi [0.5-2) aralığında çalıştığından giriş değeri normalize edilmiştir. Sinüs-kosinüs işlemi (-pi/2,pi/2) aralığında çalıştığından kuadrant düzeltme uygulanmıştır.

Bağıl hata hesabında kullanılan formül (3) de verilmiştir. Sabit noktadan kaynaklı bağıl hata yüzdeleri de Tablo1'de gösterilmiştir. Hatayı düşük tutabilmek için iterasyon sayıları yüksek tutulmuştur.

$$e = \text{Yüzde Hata} = \frac{|(\text{sabit noktalı değeri} - \text{kayan noktalı değeri})|}{\text{kayan noktalı değeri}} \quad (3)$$

TABLO I. CORDIC SABİT NOKTA BAĞIL HATASI

| CORDIC MODU | İterasyon sayısı | Bağıl hata |
|---|---|---|
| Sinüs-kosinüs | 30 | % 0.5 |
| Arktanjant | 30 | 0 |
| Karekök alma | 20 | % 0.001 |
| Genlik hesabı | 30 | 0 |

CORDIC algoritmasında kullanılan işlem formül(4)'de gösterilmiştir. Sırasıyla sinüs-kosinüs, karekök ve arktanjant için iterasyon sayısı sonsuza giderken hesaplamanın yakınsadığı değerler (5),(6) ve (7) ile verilmiştir.

CORDIC algoritması

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

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

$$z_{i+1} = z_i \pm d_i * \text{atanh}(2^{-i}) \quad (4)$$

Sinüs-kosinüs

$$z_N = 0$$

$$x_N = A_N(x_0 cosz_0 - y_0 sinz_0)$$

$$y_N = A_N(y_0 cosz_0 + x_0 sinz_0) \quad (5)$$

Karekök alma

$$x_N = A_N\sqrt{({x_0}^2 - {y_0}^2)}$$

$$y_N \approx 0$$

$$z_N \approx z_0 + \text{atanh}(x_0/y_0) \quad (6)$$

Arktanjant

$$x_N = A_N\sqrt{({x_0}^2 + {y_0}^2)}$$

$$y_N \approx 0$$

$$z_N = z_0 + \text{atanh}(y_0/x_0) \quad (7)$$

TABLO II. YÜKSEK SEVİYE KAYNAK KULLANIMI

| | ***Paylaşımsız*** | ***Paylaşımlı*** |
|---|---|---|
| Çarpma | 9 | 3 |
| CORDIC sinüs-kosinüs | 4 | 1 |
| CORDIC arktanjant | 2 | 1 |
| CORDIC karekök alma | 1 | 1 |
| CORDIC genlik bulma | 1 | 1 |
| Gecikme (latency) | 100 | 220 |
| DSP48 | 11 | 4 |
| Slice LUT | 4905 | 3200 |

Kaynak kullanımı tablo2 de verilmiştir. DSP48 bloğu çarpma işlemi için kullanılmaktadır. Paylaşımlı halde bu sayı 9 dan 3'e düşmüştür. CORDIC bloklarından sadece aynı fonkisyonlu olanlar paylaşılmıştır.

## V. SONUÇ

Bu çalışmada, kaynak paylaşımı yönteminin CORDIC algoritması ve çarpma operatörlerine uygulanmasıyla kaynak maliyetinin etkin bir şekilde azaltıldığı gösterilmiştir. Tasarımda hiçbir hazır fikri mülkiyet (IP Core) kullanılmamış ve kodlama VHDL dilinde durum makinesi mantığıyla yapılmıştır. Kaynak paylaşımı yanında sabit nokta aritmetiğinden kaynaklı hatalarda %1'in altında tutulmuştur. İlerleyen çalışmalarda VHDL kodları pipeline mimarisiyle yazılacak ve aynı seviyeli blokların paylaşımı yerine tüm hesaplama boyunca aynı amaçlı kullanılan operatörlerin tek blok kullanılarak paylaşımı yapılarak hem kaynak kullanımı daha da düşürülecek hem de zaman gecikmesi (latency) pipeline mimarisinden ötürü azaltılacaktır. Ayrıca yazılan CORDIC algoritması Xilinx firmasının geliştirdiği IP Core'lar ile kaynak kullanımı ve performans açısından kıyaslanacaktır.

## KAYNAKLAR


[1] Smith, J. O. and Abel, J. S., ``Bark and ERB Bilinear Transforms", *IEEE Trans. Speech and Audio Proc*., 7(6):697-708, 1999.

[2] Lee, K.-F., *Automatic Speech Recognition: The Development of the SPHINX SYSTEM*, Kluwer Academic Publishers, Boston, 1989.

[3] Rudnicky, A. I., Polifroni, Thayer, E H., and Brennan, R. A. "Interactive problem solving with speech", *J. Acoust. Soc. Amer., Vol. 84, 1988, p S213(A).*