İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

SIKIŞTIRILMIŞ BAŞVURU ÇİZELGELERİ KULLANARAK YARI-RASTGELE ERİŞİLEBİLİR İŞLEVLERİN VERİMLİ MANTIKSAL GERÇEKLEMESİ

YÜKSEK LİSANS TEZİ

Hasan ÜNLÜ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

OCAK 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

SIKIŞTIRILMIŞ BAŞVURU ÇİZELGELERİ KULLANARAK YARI-RASTGELE ERİŞİLEBİLİR İŞLEVLERİN VERİMLİ MANTIKSAL GERÇEKLEMESİ

YÜKSEK LİSANS TEZİ

Hasan ÜNLÜ
504111550

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Eşref ADALI

OCAK 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 504111550 numaralı Yüksek Lisans Öğrencisi **Hasan ÜNLÜ**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"SIKIŞTIRILMIŞ BAŞVURU ÇİZELGELERİ KULLANARAK YARI-RASTGELE ERİŞİLEBİLİR İŞLEVLERİN VERİMLİ MANTIKSAL GERÇEKLEMESİ"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Eşref ADALI** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. A. Coşkun SÖNMEZ** İstanbul Teknik Üniversitesi	.....
	Prof. Dr. Oğuz TOSUN Boğaziçi Üniversitesi	.....

Teslim Tarihi : **15 Aralık 2014**
Savunma Tarihi : **23 Ocak 2015**

Eşim İlknur ve kızım Zeynep Ela'ya,

ÖNSÖZ

Bu tez çalışmasının gerçekleşmesini sağlayan ve yardımlarını hiç esirgemeyen değerli hocalarım Prof. Dr. Eşref ADALI ve Doç. Dr. H. Fatih UĞURDAĞ'a teşekkürlerimi sunarım. Ayrıca tez çalışmamdaki katkılarından dolayı M. Akif ÖZKAN'a ve 2210-Yurt İçi Yüksek Lisans Burs Programı ile yüksek lisans çalışmamı destekleyen TÜBİTAK Bilim İnsanı Destekleme Daire Başkanlık'ına ayrıca teşekür ederim. Tez çalışmam boyunca sabır ve desteklerinden dolayı eşim İlknur, kızım Zeynep Ela, kardeşim Ömürcan, annem ve babama teşekkür ederim.

Aralık 2014

Hasan Ünlü
(Bilgisayar Mühendisi)

İÇİNDEKİLER

Sayfa

KISALTMALAR

LUT : Look-Up Table
FPGA : Field Programmable Gate Array
RAM : Random Access Memory
ROM : Read Only Memory
BÇ : Başvuru Çizelgesi (LUT)
MUX : Multiplexer
XOR : Ayrıcalıklı Veya
cLUT : Compressed LUT (Sıkıştırılmış Başvuru Çizelgesi)
CORDIC : Coordinate Rotation Digital Computer

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

SIKIŞTIRILMIŞ BAŞVURU ÇİZELGELERİ KULLANARAK YARI-RASTGELE ERİŞİLEBİLİR İŞLEVLERİN VERİMLİ MANTIKSAL GERÇEKLEMESİ

ÖZET

Cebirsel ve cebirsel olmayan fonksiyonlar (örneğin, e^x, cosx, logx..) bilimsel hesaplamalar ve sayısal işaret işleme uygulamalarında yoğun biçimde kullanılmaktadır. Bu fonksiyonları hızlı ve düşük kaynak kullanarak hesaplamak bilgisayar sistemlerinde önemli bir araştırma konusudur. Cebirsel olan fonksiyonlar polinomlar üzerinden dört işlem, kök ve üst alma işlemleri sonucu elde edilir. Cebirsel olmayan fonksiyonlar ise bu şekilde ifade edilemeyen fonksiyonlardır. Bu tür fonksiyonları hesaplayabilmek için polinomlarla ifade edilmesi gereklidir. Cebirsel olmayan fonksiyonları polinomlarla ifade edebilmek için Taylor serisi kullanılabilir.

Gerçek zamanlı olmayan uygulamalarda belirlenen bir hata değerine kadar, serinin gerekli olan elemanları hesaba katılır. Serideki eleman sayısı arttıkça, hesaplama hatası azalır ancak hesaplamanın süresi artar.

Örnek olarak ele aldığımız sinüs seri açılımına dayalı hesaplama işlemi bir programa yaptırıldığında belli bir süre almaktadır. Bu hesaplama süresi, zaman kısıtı olan gerçek zaman uygulamalarında sorunlara neden olmaktadır.

Zaman kısıtının olduğu gerçek zamanlı uygulamalarda hesaplamanın hızlı yapılabilmesi için fonksiyon değerlerinin önceden hesaplanmasına gidilir. Bu yönteme "Başvuru Çizelgesi" (BÇ) yöntemi denir.

Bir başka yöntem, ardışıl yaklaşım yöntemi fonksiyonun değerlerini hesaplamaktır. Ardışıl yaklaşım yöntemleri için önemli bir örnek Coordinate Rotation Digital Computer (CORDIC) tir. CORDIC trigonometrik fonksiyonları hesaplamak için özelleşmiş bir yöntemdir. CORDIC'te belirli rotasyon açılarının tanjant değerlerini tutan bir BÇ kullanılır. Bu değerler hesaplanması istenen açı değerine yaklaşırken rotasyonlarda kullanılan değerlerdir. Yüksek çözünürlüklü sonuç elde edilmek için adım sayısı artırılır. Sonuç olarak adım sayısının artması hesaplama süresini artırır. Bu yöntemin ana özelliği, hesaplama süresinin, hesaplamadan beklenen doğrulukla orantılı olmasıdır.

Başvuru çizelgesi tabanlı yöntemler parçalı ve toplamalı olmak üzere ikiye ayrılır: Parçalı BÇ yöntemi ve Toplamalı BÇ Yöntemi.

Parçalı BÇ yönteminde fonksiyon önceden belli parçalara ayrılır. Bu parçalar üzerinde işlem yapılır. Toplamalı BÇ yönteminde ise sadece değerler BÇ de tutulur. Herhangi bir çarpma işlemi yapılmaz, sadece tablodaki değerlerin toplamı ile sonuç üretilir.

Fonksiyon üretme yöntemleri ile ilgili çalışmalar ağırlıklı olarak başvuru çizelgesinin boyutunu küçültmeye, mantıksal devreyi küçültmeye ve kritik yolu azaltmayı hedeflemektedir.

Bu tez çalışmasında toplamalı BÇ yöntemine uygun yeni bir yöntem önerilip kendine benzer yöntemlerle karşılaştırılması yapılıp sonuçlar sunulacaktır. Tez çalışmasında bu tür fonksiyonları gerçekleyebilmek BÇ'nin kapladığı alanı küçülten ve mantıksal

devrenin gecikmesini azaltan melez bir çözüm önerilmiştir. Önerilen çözümün diğer bir üstünlüğü ise doğrudan BÇ yöntemi ile sağlanan rastgele erişim belli ölçüde sağlanmaktadır. Önerilen yöntem ile fonksiyonun ardışık elemanları arası sadece fark bilgileri tutularak bellekte %75 oranında küçülme sağlanmıştır. Önerilen çözüm Verilog-HDL dilinde yazılmıştır. Geliştirilen çözümü sentezlenme ortamı ise Xilinx Spartan-6 FPGA'dir. Çalışmamızda, örnek olarak trigonometrik fonksiyon olan sinüs hesaplaması yapılmıştır. Ayrıca ekler kısmında 16-bit giriş ve çıkış çözünürlüğüne sahip sinüs üreteci için tasarlanan çözümün Verilog-HDL kodları verilmiştir.

AREA-EFFICIENT COMPRESSED LOOK-UP TABLE IMPLEMENTATION FOR SEMI-RANDOMLY ACCESSIBLE FUNCTIONS

SUMMARY

Generating waveforms is a fundamental problem in signal processing applications. Low-latency calculation of a transcendental (i.e., non-algebraic) functions is an expensive operation, therefore many improvements have been proposed. A well-known method is to store all the pre-computed values in a piece of memory, called Look-Up Table (LUT), and directly send them as output. Although this is the fastest way of doing it, using a large memory is expensive in terms of both power and area. The state of the art methods present hybrid approaches that use efficient functions with smaller look-up tables. Usage of a LUT is indispensable.

Moreover, the LUT method is the only practical option in cases where transcendental functions need to be computed in real-time. Therefore, LUT reduction is still an active research area.

LUT based method is separated into two categories: Piecewise and Addition based LUT methods. Piecewise LUT method is also separated into two categories: approximation and interpolation method.
Approximation method calculates n^{th} order polynomial with minimum error. Coefficients of the polynomial expression are stored into LUT. Desired value is calculated using these coefficients in run-time. Product and summation blocks are used. Product determines critical path of the system.
Interpolation method everything is same as approximation. However, simple interpolation polynomials, which is linear interpolation, can be calculated in run-time. Product and summation blocks are also necessary.

Addition based LUT method only uses summation. Result is superposition of LUT elements. Disadvantages of this method is that it requires more memory space than previous methods. Our method is classified in this section.

In this thesis, we propose a new look-up table (LUT) based logic architecture for implementation of transcendental functions, which is low-latency but area efficient. Our architecture improves on the LUT approach by storing not the function values but the differences of consecutive function values. Storing just the differences offers a significant reduction in LUT size, hence reducing hardware area. A simple function generator that generates a fixed sequence of values is quite straight-forward and can easily benefit from our idea. However, what is proposed here is beyond that. Our architecture allows a certain level of random access with little loss in speed-up. Such non-fixed function generators are useful in control systems. We demonstrate our idea through a 16-bit sine generator implemented on FPGA, where the LUT size reduction is 4x.

Our novel method is called “compressed LUT” (cLUT) in this thesis. We compress the LUT by storing differences of consecutive values in cLUT instead of actual values of the function. As an example, instead of storing 16-bit values, we store 4-bit differences and then calculate actual function outputs from differences. Bit-width of

difference depends on first derivative of the target function. If first derivative is higher, difference needs to more area. Hence, memory utilization closes direct LUT method.
The cLUT method offers a significant reduction in memory for a compromise in the level of random access. This is acceptable since most applications require more or less consecutive access to a function as the independent variable of the function comes from a physical quantity that has some continuity.

The method consists of three main modules, which are address generation, memory array, and data calculation. Memory array consists of delta number of cLUT, which stores the differences. Output is the value of the signal for the given index . The previously requested output is stored in the data calculation module and used to calculate current requested output. A random output can be selected in the range of . The necessary cLUT cells are selected by the address generator based on the requested. As a result, the output is calculated by a small amount of logic and sent out in a single clock cycle. The memory array consists of delta number of cLUTs. Pre-computed differential signal values are stored in cLUTs in a special way. Subsequent data is stored in the same addresses of the parallel cLUTs. This means that if the cell of first cLUT stores the signal data indexed by , cell consists of the data indexed by . Memories are read in parallel under the control of the address generation module. The address generation module calculates the required memory accesses according to input . A limited random access capability is supported, thanks to the address generation module. Input can be smaller or bigger than the previous as the number of memory. Address generation module is composed of two main parts, which are differential address calculation unit and memory address control unit. Differential address calculation unit keeps the previous address in a register and calculates intermediate results that will be used by address control unit and data selection unit. First, previous input is subtracted from the current input so that the differential address, which may be forward or backward, is calculated. The result of subtraction is interpreted as magnitude and sign. The vector, named as *thermo*, is shifted in the direction of sign as the amount of the *Magnitude*. Finally, *thermo* and the result of shift are XORed. Output of the XOR block is used to determine the cLUT selection.

Shifting operation operates on *thermo* vector, whose size is . First , second , and last bits of thermo are always 0, while the third bits is the result of thermometer regulator block. Thermometer regulator block is used to handle the overflows that occur because of shifting operation. This block checks upper and lower side of thermo and selects the convenient thermo vector as given in Fig. 4. Memory address controller unit takes XOR outputs of differential address calculation unit and outputs necessary cLUT addresses. In order to calculate the next address of cLUTs, a conditional addition mechanism is designed.

Addition is selected by the output of the differential address calculation unit and 1, -1 or 0 added to address register, which stores the previous address. For example, current address pointer points to any location of last memory bank and the desired value is 3 steps forward from the current pointer. In this situation, address pointer should be incremented by 1 because of the current pointer location. The address register and its update mechanism is shown in Fig. 7. Data selection unit selects and sums the outputs of the cLUTs using the XOR outputs of address calculation unit. As

an example, consider a scenario that a random access is applied to 3 steps behind. Only outputs of 3 cLUTs will be summed. In this paper, we have presented a compressed look up table (cLUT) method for function generation, which provides a limited amount of random data access. A 16-bit addressable and 16-bit value sine function is implemented on Xilinx Spartan-6 FPGA using the proposed cLUT method. Memory requirement has been reduced, by the ratio of 75%, to 32kB from 128kB. The clock frequency achieved for this hardware is 87,4 MHz.

1. GİRİŞ

1.1 Tezin Amacı

Cebirsel ve cebirsel olmayan fonksiyonlar (örneğin, e^x, cosx, logx..) bilimsel hesaplamalar ve sayısal işaret işleme uygulamalarında yoğun biçimde kullanılmaktadır. Bu fonksiyonları hızlı ve düşük kaynak kullanarak hesaplamak bilgisayar sistemlerinde önemli bir araştırma konusudur. Cebirsel olan fonksiyonlar polinomlar üzerinden dört işlem, kök ve üst alma işlemleri sonucu elde edilir. Cebirsel olmayan fonksiyonlar ise bu şekilde ifade edilemeyen fonksiyonlardır. Bu tür fonksiyonları hesaplayabilmek için polinomlarla ifade edilmesi gereklidir. Örneğin cebirsel olmayan sinüs fonksiyonu Taylor serisi yardımı ile hesaplanmaktadır. Aşağıda sinüs fonksiyonunun Taylor serisine açılımı görülmektedir.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots + \mathrm{Rn} \qquad \textbf{(1.1)}$$

Gerçek zamanlı olmayan uygulamalarda belirlenen bir hata değerine kadar, serinin gerekli olan elemanları hesaba katılır. Seri açılımı yöntemiyle yapılan hesaplamada, hata değeri Rn olarak hesaplanır. Serideki eleman sayısı arttıkça, hesaplama hatası azalır ancak hesaplamanın süresi artar.

Örnek olarak ele aldığımız sinüs seri açılımına dayalı hesaplama işlemi bir programa yaptırıldığında belli bir süre almaktadır. Bu hesaplama süresi, zaman kısıtı olan gerçek zaman uygulamalarında sorunlara neden olmaktadır. Tezimizin amacı bu tür program yardımıyla hesaplamalardan kaynaklanan zaman kaybını gidermek üzere donanım çözümlerinin geliştirilmesi üzerinedir.

Zaman kısıtının olduğu gerçek zamanlı uygulamalarda hesaplamanın hızlı yapılabilmesi için fonksiyon değerlerinin önceden hesaplanmasına gidilir [1, 2]. Bu yönteme "Başvuru Çizelgesi" (BÇ) yöntemi denir.

Bir başka yöntem, ardışıl yaklaşım yöntemi fonksiyonun değerlerini hesaplamaktır [3].

Ardışıl yaklaşım yöntemleri için önemli bir örnek Coordinate Rotation Digital Computer (CORDIC) tir. CORDIC trigonometrik fonksiyonları hesaplamak için özelleşmiş bir yöntemdir. CORDIC'te belirli rotasyon açılarının tanjant değerlerini tutan bir BÇ kullanılır [4, 5]. Bu değerler hesaplanması istenen açı değerine yaklaşırken rotasyonlarda kullanılan değerlerdir. Yüksek çözünürlüklü sonuç elde edilmek için adım sayısı artırılır. Sonuç olarak adım sayısının artması hesaplama süresini artırır. Bu yöntemin ana özelliği, hesaplama süresinin, hesaplamadan beklenen doğrulukla orantılı olmasıdır.

Başvuru çizelgesi tabanlı yöntemler parçalı ve toplamalı olmak üzere ikiye ayrılır:

1. Parçalı BÇ yöntemi

2. Toplamalı BÇ yöntemi

Parçalı BÇ yönteminde fonksiyon önceden belli parçalara ayrılır. Bu parçalar üzerinde işlem yapılır. Parçalı BÇ kendi içinde iki türe ayrılır: [6]

1. Yaklaşıklık yöntemi

2. Ara değerleme yöntemi

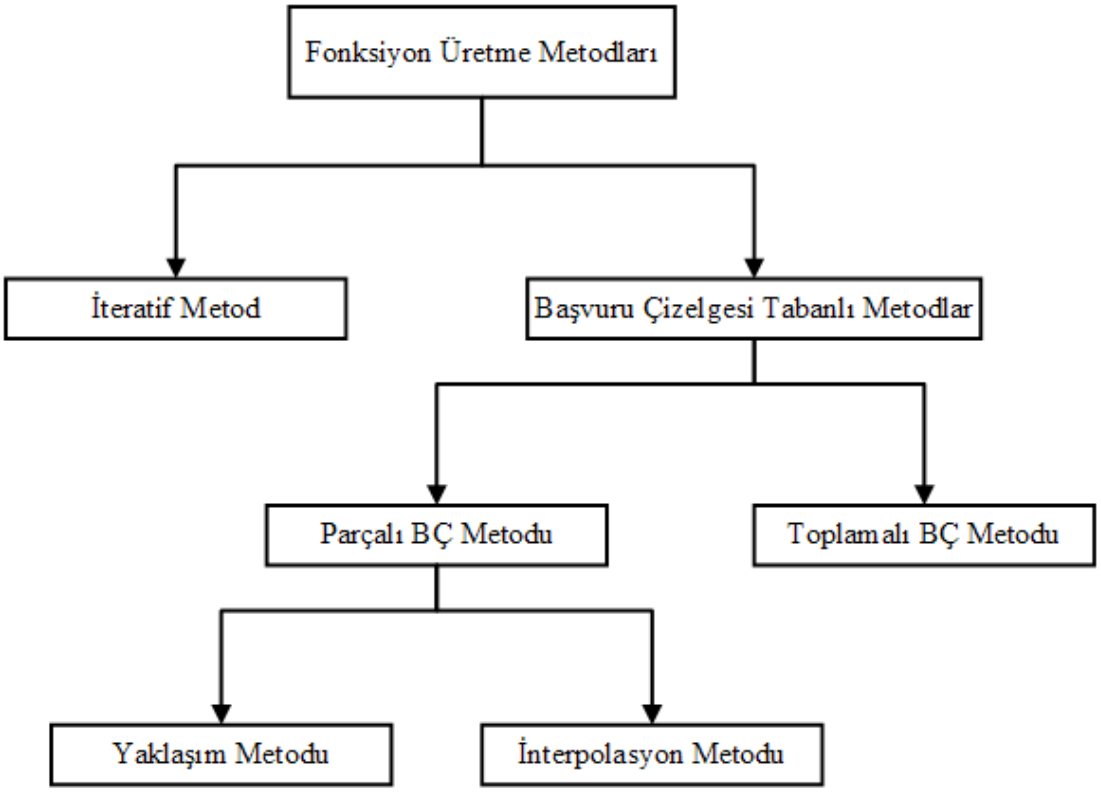
Yaklaşıklık yönteminde parçalara ayrılan her kısım için elde edilen polinomların katsayıları BÇ'de tutulur [7]. İstenen değer hangi parçadaysa onun fonksiyonu hesaplanır. Çarpma ve toplama bloklarına ihtiyacı vardır.

Ara değerleme yönteminde ise değeri bilinen fonksiyon değerleri kullanılarak ara değerler hesaplanır. Çalışma zamanında belirlenen noktalardan geçen n. derece polinomun katsayıları hesaplanır. Sonra istenen değer bu polinom aracılığıyla hesaplanıp sonuç elde edilir. Doğrusal ara değerler yöntemiyle başarılı sonuçlar elde edilebilir ancak polinom derecesi arttıkça fonksiyonun değerini hesaplama süresi uzamaktadır.

Toplamalı BÇ yönteminde ise sadece değerler BÇ de tutulur [8]. Herhangi bir çarpma işlemi yapılmaz sadece tablodaki değerlerin toplamı ile sonuç üretilir.

Fonksiyon üretme yöntemleri ile ilgili yöntemler topluca Şekil 1.1'de gösterilmiştir. Fonksiyon üretme yöntemleri ile ilgili çalışmalar ağırlıklı olarak başvuru çizelgesinin

boyutunu küçültmeye, mantıksal devreyi küçültmeye ve kritik yolu azaltmayı hedeflemektedir [9, 10, 11].



Şekil 1.1 : Fonksiyon üretme yöntemleri [1].

Bu tez çalışmasında toplamalı BÇ yöntemine uygun yeni bir yöntem önerilip kendine benzer yöntemlerle karşılaştırılması yapılıp sonuçlar sunulacaktır. Tez çalışmasında bu tür fonksiyonları gerçekleyebilmek BÇ'nin kapladığı alanı küçülten ve mantıksal devrenin gecikmesini azaltan melez bir çözüm önerilmiştir. Önerilen çözümün diğer bir üstünlüğü ise doğrudan BÇ yöntemi ile sağlanan rastgele erişim belli ölçüde sağlanmaktadır. Önerilen çözüm Verilog-HDL dilinde yazılmıştır. Geliştirilen çözümü sentezlenme ortamı ise Xilinx Spartan-6 FPGA'dir. Çalışmamızda, örnek olarak trigonometrik fonksiyon olan sinüs hesaplaması yapılmıştır. Ayrıca ekler kısmında 16-bit giriş ve çıkış çözünürlüğüne sahip sinüs üreteci için tasarlanan çözümün Verilog-HDL kodları verilmiştir.

2. BENZER ÇALIŞMALAR

2.1 Doğrudan Başvuru Çizelgesi Yöntemi

Bu yöntem f(x) fonksiyonun belli aralıklarla hesaplanmış her değerini bir BÇ üzerinde yani bellekte tutar. Hesaplanması istenen x değeri temel sıralı adresleme yöntemi ile bellekten çekilir. x değerinin bellekte bulunduğu yerin adresi, çözünürlük artırıldığında daha uzak olacaktır. Ancak bu durum bellekteki veriye ulaşmada gecikmeye neden olmaz.

Bellek içerisinde verilerin yer alışı Şekil 2.1'de gösterilmektedir. Bu yöntem kapladığı alan açısından en kötü yöntemdir. Hiçbir optimizasyon veya sıkıştırma yoktur. Bunun yanında bu sistem hız açısından en hızlı olandır ve hesaplanan değere erişim, değerin büyüklüğü ve çözünürlük ile ilişkili değildir. Sadece çözünürlük artırıldığında çizelgenin boyutu çözünürlüğe bağlı olarak olarak artmaktadır.

Adres	İçerik
x_1	$f(x_1)$
x_2	$f(x_2)$
x_3	$f(x_3)$
…	…
x_n	$f(x_n)$

Şekil 2.1 : Doğrudan yöntem için bellek örneği.

Bu yöntem için bellekten başka mantıksal bir devreye ihtiyaç yoktur. Çalışma boyunca referans alınan sinüs fonksiyonu için 16-bit giriş ve çıkış çözünürlüğünde kaplanan bellek alanı 2^{16}*16 bit=128 KByte olmaktadır. Bu sistem aynı zaman gerçek zamanlı programlarda da kullanılmaktadır. Programlama dilinde basit bir dizi tanımlamasıyla yapılabilir. Bu tasarım FPGA üzerinde BÇ blok bellekler üzerinde gerçekleştirilmektedir.

2.2 Fark Değerlerini Kullanan Başvuru Çizelgesi Yöntemi

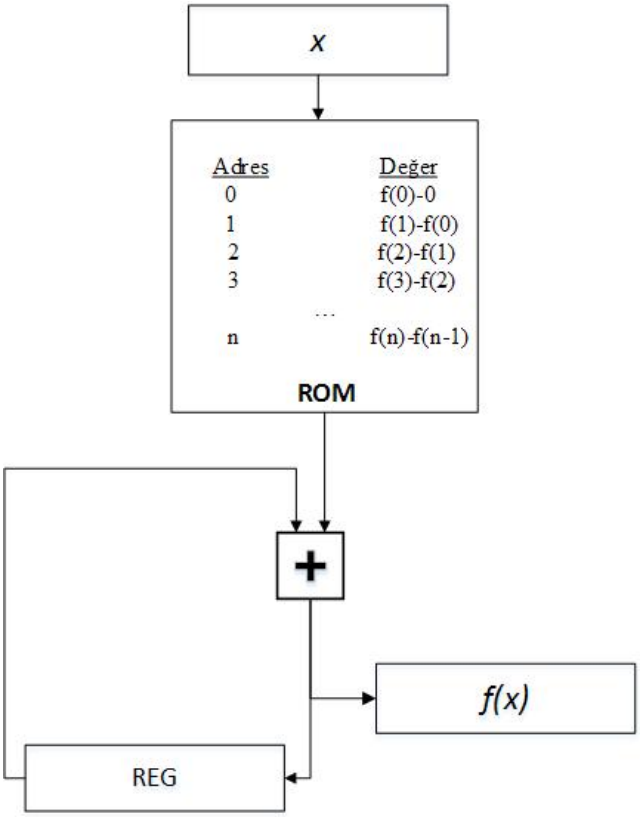
Araştırmalarımız sırasında, bu yönteme dayalı çözümlere rastlanmamıştır. Dolayısıyla, bu çalışma kapsamında geliştirilen çözümle sadece yöntem olarak karşılaştırılması yapılmıştır.

Fark değerlerini kullanan BÇ yöntemi ile Doğrudan Başvuru Çizelgesi Yöntemi arasındaki temel fark, fark değerlerini kullanan BÇ yönteminin sadece ardışık fark değerlerini tutmasıdır. Böylece tutulan verinin boyutu azaltılmış olmaktadır. Veri boyutunun azalmasının en önemli sebebi fonksiyonların ardışık değerleri arasındaki farkın daha az bit kullanılarak ifade edilmesidir. Fark değerleri tutan başvuru çizelgesine "Compressed Look-Up Table-Sıkıştırılmış Başvuru Çizelgesi" (cLUT) ismini verdik. Kaplanan alan fonksiyonun birinci türevine bağlı olarak çok düşebilmektedir. Bu yöntemin düzenli çalışabilmesi için sistemde sürekli bir sonraki veya bir önceki değere erişilmesi gerekir, aksi takdirde doğru sonuç üretilmez. Örneğin çözünürlük 0,1 derece olsun. Devre sin(0,5)'i hesaplar. Ardından sin(0,7)'yi hesaplamak istese bunu yapamaz çünkü sin(0,6) değerine ihtiyaç vardır. Önce sin(0,6)'yı hesaplamalı ardından sin(0,7)'ye geçmelidir. Bunlara ek olarak başlangıç değeri sürekli sabit bir adres değerinden başlamak zorundadır sin(0) gibi. Verilerin bellekte saklanışı Şekil 2.2'de gösterilmektedir. Bu yöntem için bellekten başka bir de toplama devresine ihtiyaç vardır. Çalışma boyunca referans alınan sinüs fonksiyonu için 16-bit giriş ve çıkış çözünürlüğünde kaplanan yer 2^{16}*4 bit=32 KByte olmaktadır. 2^{16} örnekten oluşan sinüs değerinin ardışık iki değeri arasındaki fark en fazla 4 bit olmaktadır. MATLAB ile hesaplanıp değerler incelenmiştir. Bellek gereksiniminin azalması fonksiyonun birinci türevinin değişimi ile ilgilidir.

Adres	İçerik
x_1	$f(x_1)$-0
x_2	$f(x_2)$-$f(x_1)$
x_3	$f(x_3)$-$f(x_2)$
…	…
x_n	$f(x_n)$-$f(x_{n-1})$

Şekil 2.2 : Fark değerlerini kullanan yöntem için bellek örneği.

Türev değeri yüksek olan fonksiyonlarda kaplanan alan ilk yönteme yaklaşmaktadır. Şekil 2.2'den görüleceği gibi teleskopik toplam söz konusudur ve rastgele erişim yapma imkanı yoktur. Devrenin hızını iki faktör belirlemektedir. Bunlardan biri hesaplanmak istenen değerin güncel değere uzaklığıdır. İkincisi ise devrenin saat frekansıdır. Devrenin saat frekansını devre üzerindeki en uzun gecikme belirler. Şekil 2.3'te fark değerlerini kullanan BÇ yönteminin genel yapısı görülmektedir. Buradan görüldüğü gibi devrenin gecikmesi toplama devresinin gecikmesine eşittir.

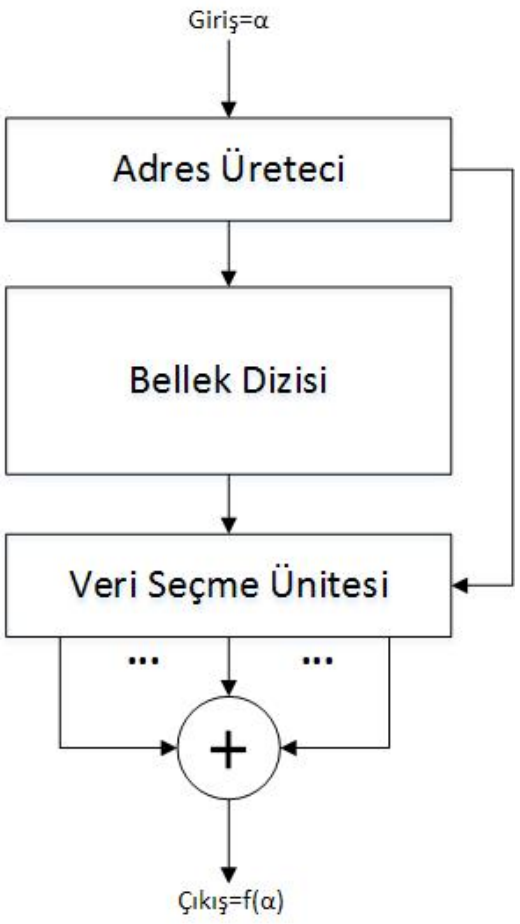


Şekil 2.3 : Fark değerlerini kullanan yöntem için genel yapı.

3. ÖNERİLEN SİSTEM

3.1 Genel Yapı

Önerilen sistemin genel yapısı üç ana bloktan oluşmaktadır. Bunlar; adres üreteci, bellek dizisi ve veri seçme ünitesidir. Şekil 3.1'de genel yapı görülmektedir. Bellek dizisi kısmında fark değerlerini tutan sıkıştırılmış başvuru çizelgesi (cLUT) bulunmaktadır. Bellek dizisi diye ifade edilmesinin sebebi paralel olarak adreslenebilen RAM veya ROM dizilerini ifade etmektedir. Veri seçme ünitesi de adres üretecinin ürettiği seçme verilerini kullanarak sadece ilgili belleklerin çıkışını seçer. Seçilen çıkışlar toplanarak sonuç üretilir.



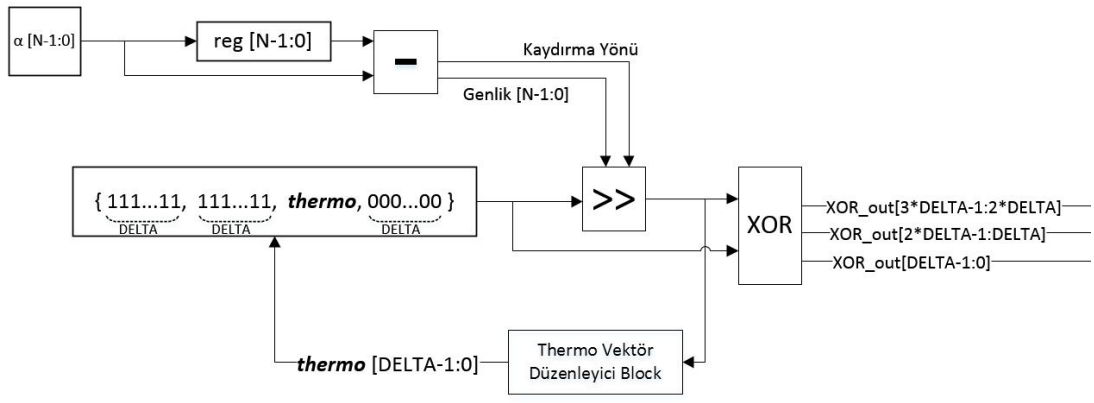
Şekil 3.1 : Genel yapı.

Hesaplanması istenen giriş değeri geldiğinde adres üreteci bir önceki girişi kullanarak güncel değerden ne kadar uzakta olduğunu hesaplar. Uzaklık eğer rastgelelik sınırı içindeyse her bir bellek dizisi için adresler üretilir ve bir saat süresinde bunlar toplanarak sonuç hesaplanır.

Uzaklık rastgelelik sınırı dışındaysa en büyük rastgelelik adımı ile ilerlenerek istenen değere yaklaşılmaya çalışılır.

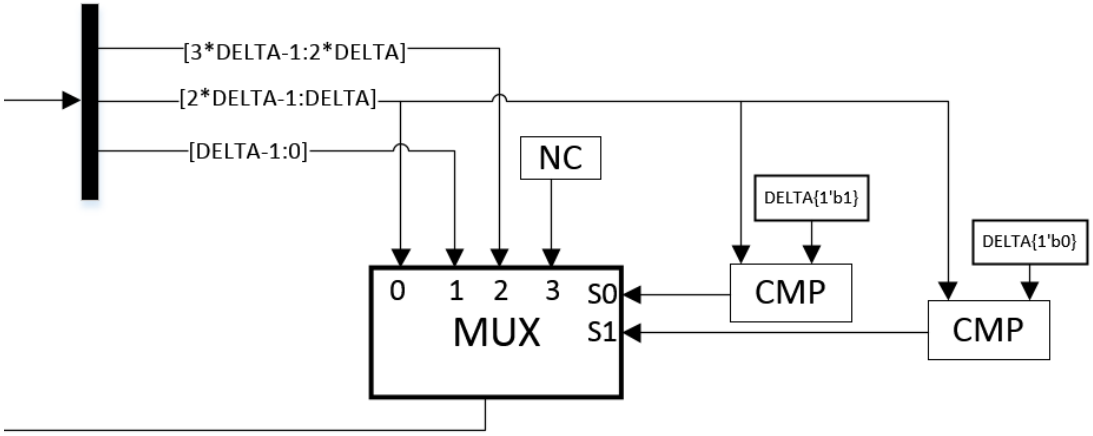
3.2 Adres Üreteci

Adres üretecinde sıcaklık vektörü (Termo Vector) kullanılmaktadır. Bu vektör çubuk gösterim yapısına benzemektedir. Kolay anlaşılması için örneğin 8 bitlik bir sıcaklık vektörü ile 5 ifade edilmek istenirse 11111000 olarak, 3 ifade edilmek istenirse 11100000 şeklinde yazılır. Adres üreteci sıcaklık vektörünü kullanarak bir önceki ile şimdiki durum arasındaki fark kadar öteleme işlemi yapmaktadır. Öteleme işleminin yönü fark ifadesinin işaretine bağlıdır. Şekil 3.2'de adres üretecinin yapısı görülmektedir. Sıcaklık vektörü Şekil 3.2'de ***thermo*** şeklinde belirtilmiştir. Sıcaklık vektöründeki ötelemeden kaynaklanacak hataları engellemek için başı ve sonu 1 ve 0 ile doldurulmuştur. Öteleme sonucu bazı durumlarda anlamsız kalabilmektedir. Bunu engellemek için sıcaklık vektör düzenleyici bloğu tasarlanmıştır. **DELTA** ifadesi paralel RAM veya ROM bloğu sayısını, başka bir ifadeyle en fazla rastgele erişim miktarını belirtmektedir. YADA (XOR) işlemi ile bir önceki ve şuanda elde edilen sıcaklık vektörünün arasındaki değişim bulunur. Bu değişim ile hangi bellek bloklarının seçileceği ve nasıl adresleneceği belirlenir.



Şekil 3.2 : Adres üreteci.

Sıcaklık vektör düzenleyici blok ötelenen ifadenin içerisindeki sıcaklık vektörünü bulmaya yardımcı olur. Şekil 3.3'te sıcaklık vektör düzenleyici blok yer almaktadır. **CMP** karşılaştırma devresini **NC** ise bağlı değil anlamına gelmektedir. Ötelenen ifade 3 parçaya ayrılır. Taşma gerçekleşmişse bu durum karşılaştırıcı ile belirlenip ötelenen ifadenin ilk, orta veya son **DELTA** adet biti yeni sıcaklık vektörünü oluşturur.

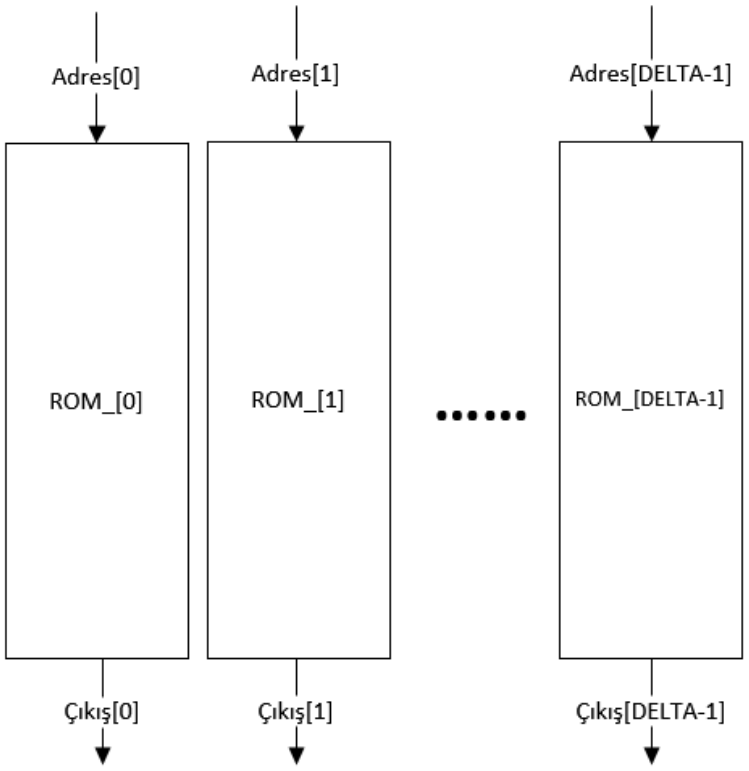


Şekil 3.3 : Sıcaklık vektör düzenleyici devre.

3.3 Bellek Dizisi

Bellek dizisi rastgelelik miktarı olan **DELTA** kadar paralel RAM veya ROM içermektedir. Şekil 3.4'te bellek dizisi görülmektedir. Bellekler Şekil 2.2'de gösterilen şekildeki gibi fark bilgilerini tutmaktadır. Değerlerin bellekler üzerindeki dizilişi sıralı değildir. Belleklerin aynı satırları ardışıl elemanları tutmaktadır. Satır sonlarında ise değerler en baştaki bellek üzerinden devam eder. Bu şekilde saklanarak bir saat işaretinde ardışık **DELTA** değerin okunması sağlanır. Eğer aynı bellek üzerinde ardışık elemanlar içerilseydi, rastgele erişim yapılamazdı.

Rastgelelik oranı artırılmak istenirse paralel bellek sayısı artırılmalıdır. Bellek dizisindeki her bir ROM veya RAM adres üretecinden gelen değişim bilgisi ile adreslenir. Adresleme işlemi güncel değerin işaret ettiği yere bağlı olarak her bellek bir önceki veya bir sonraki elemanını adresleyebilir. Adresleme tamamlandıktan sonra en son kalınan adres değeri bir saklayıcıda tutulur.



Şekil 3.4 : Bellek dizisi.

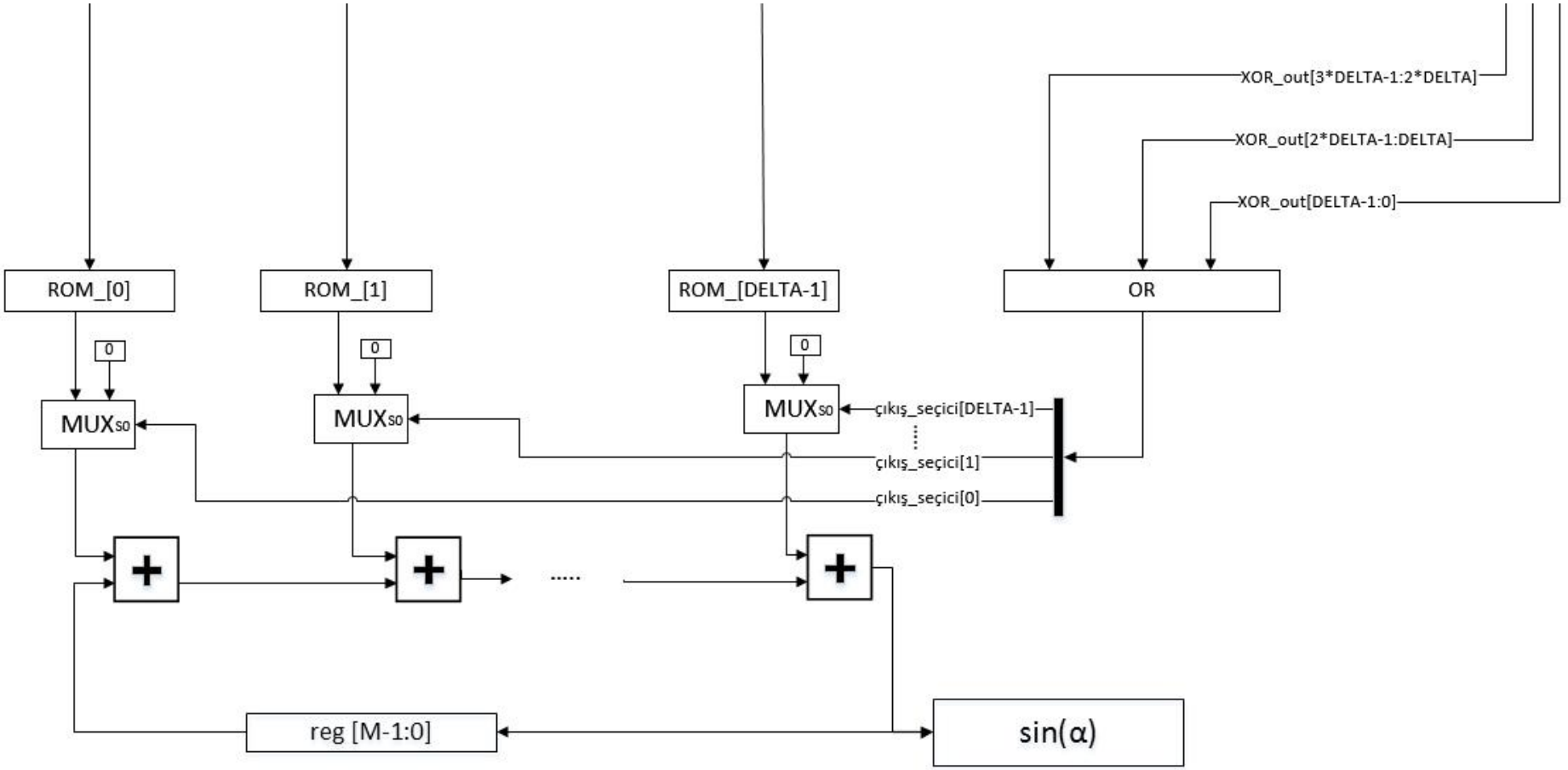
3.4 Veri Seçme Ünitesi

Veri seçme ünitesi Şekil 3.5'te görülmektedir. Veri seçme ünitesi adres üretecinden elde ettiği değişim bilgisinden faydalanarak hangi bellek çıkışlarını seçeceğini belirler. İlk denemelerimizde buraya çarpma blokları yerleştirilmiştir. Ancak gecikmesi çok yüksek çıkmıştır. Çarpma işlemi olarak 0 ile çarpma ve 1 ile çarpma uygulandığı için **MUX** ile daha verimli bir gerçekleştirme yapılmıştır.

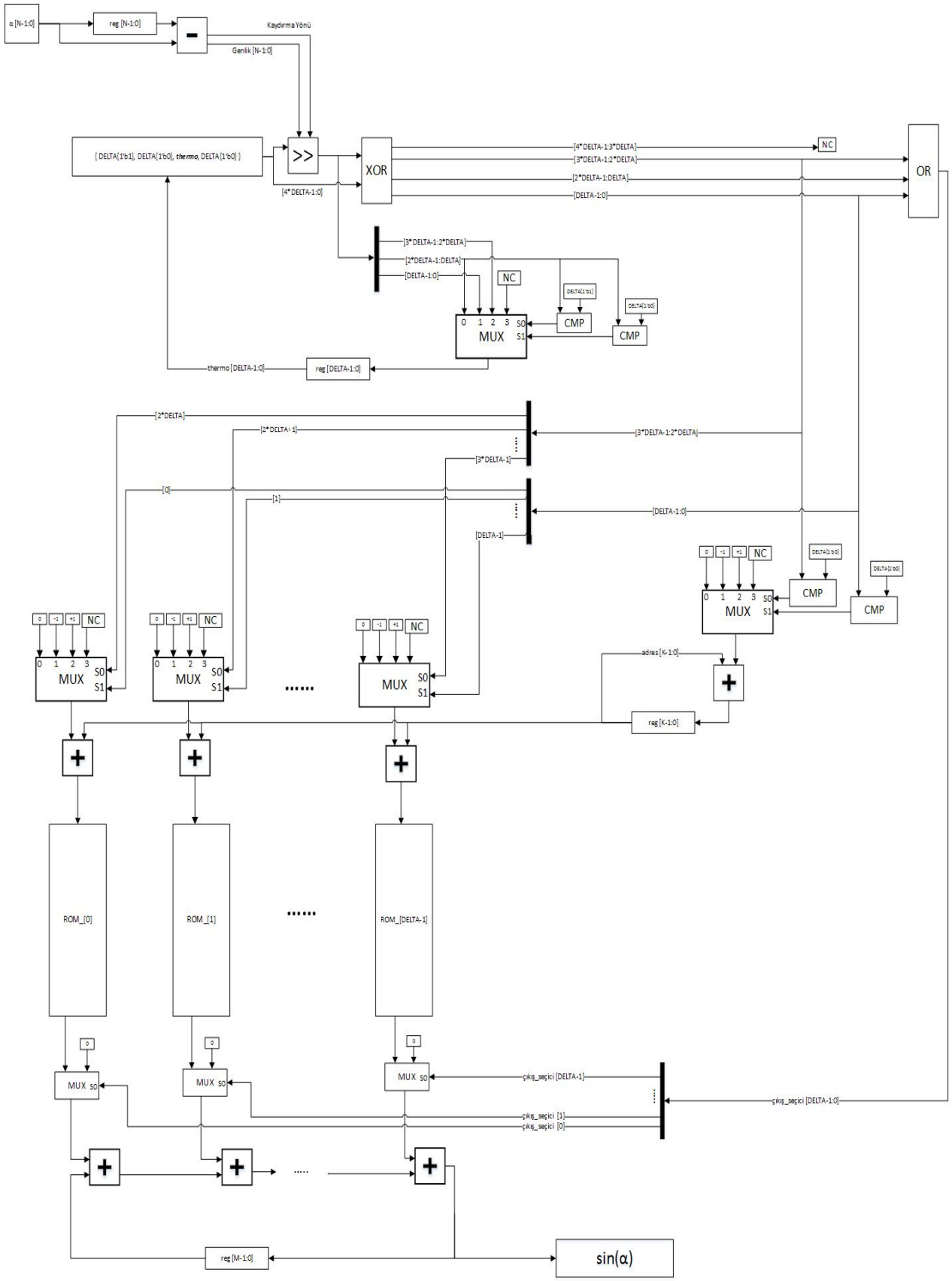
Belirlenen çıkışların seçilmesi **MUX** ile gerçekleşir. Eğer ilgili bellek çıkışı seçilecekse bellek çıkışı **MUX** çıkışına aktarılır. Diğer durumda ise 0 dışarı çıkarılır. Seçilen bu çıkışlar art arda bağlı toplama blokları ile toplanır. Aynı zamanda toplama işleminde bir önceki değerde toplanarak güncel değer elde edilir. Güncel değer yine bir saklayıcıda saklanır. Çünkü bir sonraki hesaplamada kullanılacaktır. Veri seçme ünitesi devrenin en yavaş kısmıdır. Burada art arda bağlı toplama blokları kullanılmaktadır. Sistemin rastgelelik miktarı artırılmak istenirse devresel gecikme artacak ve maksimum çalışma frekansı düşecektir.

3.5 Sistemin Birleşmesi

Şekil 3.6'da birleşmiş sistem görülmektedir. Adres üreteci üst kısımda XOR ifadesinin çıkışına kadar olan kısmı oluşturmaktadır. Bellek ve üzeride adres değerlerini belirleyen kısım bellek dizisi. En son olarakta bellek çıkışını seçip toplama yapan very seçme ünitesi görülmektedir. Verilen tasarım genel bir rastgelelik miktarı olan olan **DELTA** ile belirlenmiştir. İstenen **DELTA** ve bit genişliklerine göre tasarıma ait Verilog-HDL kodlarını üreten C kodu yazılmıştır.



Şekil 3.5 : Veri seçme ünitesi.



Şekil 3.6 : Tüm sistemin detaylı şeması.

4. SONUÇ VE KARŞILAŞTIRMA

Karşılaştırma yapılırken referans bir sinyal ve sabit giriş-çıkış çözünürlükleri kullanılmıştır. Önerdiğimiz sistem toplamalı BÇ yöntemleri çatısı altında değerlendirilmektedir. Karşılaştırma işlemi bu yönteme benzer olan doğrudan BÇ ve fark tutan BÇ yöntemi ile yapılmıştır. 16-bit giriş-çıkış çözünürlüğü ve 8 adımlık rastgelelik miktarı olan sistemin Verilog-HDL kodları üretilmiştir. Bellek dizilimini oluşturmak için MATLAB üzerinde sinüs işareti 2^{16} parçaya ayrılıp ardışık değerler arasındaki farklar kullanılmıştır.

Belirlenen referans değerler için karşılaştırmalı sonuçlar Çizelge 4.1'de verilmiştir. Doğrudan BÇ donanımsal sadece RAM veya ROM'dan oluşmaktadır. Bellek boyutu kolaylıkla hesaplanır; 2^{16}x16 bit= 128 KByte olarak bulunur. Bu yöntem istenen her değere rastgele erişim sağlayabilmektedir. Fark tutan BÇ ile bellek miktarı %75 azaltılmış, 32 KByte'e düşürmüştür. Ancak rastgelelik desteklenmemektedir.

Önerilen sistem ise hem kısmi rastgelelik sağlaması ve bellek miktarını %75 azaltarak avantaj sağlamıştır. MATLAB ortamında üretilen bellek içerikleri Xilinx ISE programında bellek içeriklerine bağlanarak simülasyon yapılmıştır.

Çizelge 4.1 : Yöntemlerin karşılaştırılması.

Yöntem	Doğrudan BÇ	Fark Tutan BÇ	Önerilen Yöntem
Rastgelelik	Var	Yok	Kısmi
Bellek	128 KByte	32 KByte	32 KByte

Önerilen sistem lojik devre karmaşıklığı olarak diğer iki yönteme göre daha fazla alan kaplamaktadır. Bunun miktarını ölçebilmek için tasarlanan sistem Spartan-6 FPGA üzerinde sentezlenmiştir. Lojik kullanım detayları Şekil 4.1'de verilmiştir. Lojik sistemin en yüksek çalışma frekansı 87,4 MHz olarak elde edilmiştir. Bu

çalışma frekansının kısıtı veri seçme ünitesindeki toplama bloklarıdır. Önerilen bu tasarıma yapılan araştırmalarda rastlanmamıştır. Bundan sonraki çalışmalarda olarak BÇ boyutunu trigonometrik fonksiyonlar için ¼'ine düşürmek amaçlanmaktadır. Çünkü sinüs, cosinüs gibi fonksiyonların 0-90 derece aralığını işaret değiştirerek tekrar etmektedir.

Buna ek olarak fark işaretinden oluşan BÇ oluşturulurken fonksiyonun birinci türev karakteristiğine göre analiz yapılarak fark değerini en az miktarda bit ile ifade edilmesi amaçlanmaktadır.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	45	54576	0%
Number of Slice LUTs	352	27288	1%
Number of fully used LUT-FF pairs	39	358	10%
Number of bonded IOBs	23	218	10%
Number of BUFG/BUFGCTRLs	1	16	6%

Şekil 4.1 : Spartan-6 FPGA üzerinde lojik kullanım.

KAYNAKLAR

[1] **Low, J.Y.L., C.C. Jong (2013)** "A memory-efficient tables-and-additions method for accurate computation of elementary functions", IEEE Trans. on Computers Vol. 62, No. 5, pp. 858-872.

[2] **Hassler, H., N. Takagi (1995)** "Function evaluation by table look-up and addition", Proc. of IEEE Symp. on Computer Arithmetic, pp. 10-16.

[3] **S.-F. Hsiao, P.-C. Wei, and C.-P. Lin (2008)** "An Automatic Hardware Generator for Special Arithmetic Functions Using Various ROMBased Approximation Approaches," Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '08), pp. 468-471.

[4] **J.E. Volder (1959)** "The CORDIC Trigonometric Computing Technique," IRE Trans. Electronic Computers, vol. EC-8, pp. 330-334.

[5] **T. Lang and E. Antelo (2005)** "High-Throughput CORDIC-Based Geometry Operations for 3D Computer Graphics," IEEE Trans Computers, vol. 54, no. 3, pp. 347-361, Mar.

[6] **D.-U. Lee, R.C.C. Cheung, W. Luk, and J.D. Villasenor (2008)** "Hardware Implementation Trade-offs of Polynomial Approximations and Interpolations," IEEE Trans. Computers, vol. 57, no. 5, pp. 686-701.

[7] **G.M. Philips and P.J. Taylor (1996)** Theory and Applications of Numerical Analysis. Acedemic Press.

[8] **W.F. Wong and E. Goto (1995)** "Fast Evaluation of the Elementary Functions in Single Precision," IEEE Trans. Computers, vol. 44, no. 3, pp. 453-457, Mar.

[9] **Jeng, S.-S., H.-C. Lin, C.-H. Lin (2012)** "A novel ROM compression architecture for DDFS utilizing the parabolic approximation of equi-section division", IEEE Trans. on Ultrasonics, Ferroelectrics and Frequency Control, Vol. 59, No. 12, pp. 2603-2612.

[10] **Lygouras, J.N. (1999)** "Memory reduction in look-up tables for fast symmetric function generators", IEEE Trans. on Instrumentation and Measurement, Vol. 48, No. 6, pp. 1254-1258.

[11] **Jung, J-H. (2010)** "Data reduction method of sine look-up tables in microprocessor's memory storage", Electronics Letters, Vol. 46, No. 25, pp. 1656-1658.

EKLER

EK A: Kodlar

EK A

```
//-------------------------------------------------------
//Bu devre sistemin alt bloklarını birleştiren devredir.
//Aynı zamanda bu kısım bellek adresleme kısmını içerir.
//-------------------------------------------------------
`timescale 1ns / 1ps
 module LUT_delta #(       parameter INADDRWIDTH=16,
                     parameter OUTVALWIDTH=16,
                     parameter DELTA=8,
                     parameter DELTAWIDTH=3,
                     parameter DATAWIDTH=4,
                     parameter ADDRWIDTH=13)
                (alfa,
                reset,
                clk,
                outval,
                enable
     );

input  [INADDRWIDTH-1:0] alfa; //0-2PI derece 0
                     //2^INADDRWIDTH arasına karsı düşer.
 input reset; //reset
 input clk; //saat işareti
 input enable; //izin girişi
 output wire  [OUTVALWIDTH-1:0] outval; //sinüs sonucu
 //0-1 değerleri 0-2^(OUTVALWIDTH)-1 arasına karşı düşer.

 reg   [OUTVALWIDTH-1:0]  outval_prev;
 //önceki çıkış değeri
 reg   [INADDRWIDTH-1:0]  alfa_prev;
 //önceki açı değeri
 wire  [DELTAWIDTH:  0]   delta;
 //fark değeri
 wire  [DELTAWIDTH-1:0]   mag_delta;
 wire sign;
 reg   [DELTA-1:0] thermo_prev;
 wire  [DELTA-1:0] thermo;
 //thermometre vektörü
 wire  [DELTA-1:0] out_select;
 assign delta = alfa-alfa_prev;
 //önceki değerle olan farkı hesaplayan devre

 assign sign = delta [DELTAWIDTH];
 assign mag_delta = sign ? ~delta : delta;
```

```
 wire  [3*DELTA-1:0] addrselect;

selector #(.DELTA(DELTA), .DELTAWIDTH(DELTAWIDTH) )
selector_1(
          .in(thermo_prev),
          .shift(mag_delta),
          .out(out_select),
          .next_in(thermo),
          .addrselect(addrselect),
          .direction(sign) //Şekil 3.2 deki fark devresi
                           //çıkışı
 );


 wire   [ADDRWIDTH-1:0] romaddr  [0 : DELTA-1];
 wire   [DATAWIDTH-1:0] out  [0:DELTA-1];


//fark değerlerini tutan ROM dizisi
ROM #( .ADDRWIDTH(ADDRWIDTH), .DATAWIDTH(DATAWIDTH),
.ROMFILE("ROM0.mif") ) rom_0(.addr(romaddr [0]) ,
.out(out [0]));
ROM #( .ADDRWIDTH(ADDRWIDTH), .DATAWIDTH(DATAWIDTH),
.ROMFILE("ROM1.mif") ) rom_1(.addr(romaddr [1]) ,
.out(out [1]));
ROM #( .ADDRWIDTH(ADDRWIDTH), .DATAWIDTH(DATAWIDTH),
.ROMFILE("ROM2.mif") ) rom_2(.addr(romaddr [2]) ,
.out(out [2]));
ROM #( .ADDRWIDTH(ADDRWIDTH), .DATAWIDTH(DATAWIDTH),
.ROMFILE("ROM3.mif") ) rom_3(.addr(romaddr [3]) ,
.out(out [3]));
ROM #( .ADDRWIDTH(ADDRWIDTH), .DATAWIDTH(DATAWIDTH),
.ROMFILE("ROM4.mif") ) rom_4(.addr(romaddr [4]) ,
.out(out [4]));
ROM #( .ADDRWIDTH(ADDRWIDTH), .DATAWIDTH(DATAWIDTH),
.ROMFILE("ROM5.mif") ) rom_5(.addr(romaddr [5]) ,
.out(out [5]));
ROM #( .ADDRWIDTH(ADDRWIDTH), .DATAWIDTH(DATAWIDTH),
.ROMFILE("ROM6.mif") ) rom_6(.addr(romaddr [6]) ,
.out(out [6]));
ROM #( .ADDRWIDTH(ADDRWIDTH), .DATAWIDTH(DATAWIDTH),
.ROMFILE("ROM7.mif") ) rom_7(.addr(romaddr [7]) ,
.out(out [7]));
```

```
//Bellek adreslemeyi yapan kısım
//burada her bellek elemanı için adresin +1, -1 veya 0
//olup olmayacağı belirlenir.

 reg   [ADDRWIDTH-1:0] prev_address;
 wire  [ADDRWIDTH-1:0] address;


assign romaddr [0] = ( addrselect [0+2*DELTA] ) ?
prev_address-1'b1 :
                 ( addrselect [0] )          ?
prev_address+1'b1 :
                  prev_address;

assign romaddr [1] = ( addrselect [1+2*DELTA] ) ?
prev_address-1'b1 :
                 ( addrselect [1] )          ?
prev_address+1'b1 :
                  prev_address;

assign romaddr [2] = ( addrselect [2+2*DELTA] ) ?
prev_address-1'b1 :
                 ( addrselect [2] )          ?
prev_address+1'b1 :
                  prev_address;

assign romaddr [3] = ( addrselect [3+2*DELTA] ) ?
prev_address-1'b1 :
                 ( addrselect [3] )          ?
prev_address+1'b1 :
                  prev_address;

assign romaddr [4] = ( addrselect [4+2*DELTA] ) ?
prev_address-1'b1 :
                 ( addrselect [4] )          ?
prev_address+1'b1 :
                  prev_address;

assign romaddr [5] = ( addrselect [5+2*DELTA] ) ?
prev_address-1'b1 :
```

```
                ( addrselect [5] )          ?
prev_address+1'b1 :
                 prev_address;

assign romaddr [6] = ( addrselect [6+2*DELTA] ) ?
prev_address-1'b1 :
                ( addrselect [6] )          ?
prev_address+1'b1 :
                 prev_address;

assign romaddr [7] = ( addrselect [7+2*DELTA] ) ?
prev_address-1'b1 :
                ( addrselect [7] )          ?
prev_address+1'b1 :
                 prev_address;

assign address = (addrselect [3*DELTA-
1:2*DELTA]!={DELTA{1'b0}} ) ?   prev_address-1'b1 :
          (addrselect [DELTA-1  :      0]!={DELTA{1'b0}}
) ?   prev_address+1'b1 :
          prev_address;


wire   [DATAWIDTH-1:0] out_sel  [0:DELTA-1];

//iyileştirme yapılan kısım çarpma bloklarına göre çok
daha az gecikme sağlandı.
assign out_sel [0]=out [0] ? out_select [0] : 1'b0;
assign out_sel [1]=out [1] ? out_select [1] : 1'b0;
assign out_sel [2]=out [2] ? out_select [2] : 1'b0;
assign out_sel [3]=out [3] ? out_select [3] : 1'b0;
assign out_sel [4]=out [4] ? out_select [4] : 1'b0;
assign out_sel [5]=out [5] ? out_select [5] : 1'b0;
assign out_sel [6]=out [6] ? out_select [6] : 1'b0;
assign out_sel [7]=out [7] ? out_select [7] : 1'b0;

assign outval=outval_prev+out_sel [0]+out_sel [1]+out_sel
[2]+out_sel [3]+out_sel [4]+out_sel [5]+out_sel
[6]+out_sel [7];



 always@(posedge clk) begin
```

```
     if(reset)begin
          outval_prev<=0;
          alfa_prev<=10;
          thermo_prev<=2**(DELTA-1);
          prev_address<=7;


     end else begin
          if(enable) begin
               alfa_prev<=alfa;
               thermo_prev<=thermo;
               prev_address<=address;
               outval_prev<=outval;
          end
     end
 end
endmodule


//----------------------------------------------------------
//Bu devre sistemin adres üretecindeki thermo vektör ile
//olan hesaplamaları düzenler aynı zamanda exor çıkışı
//ile çıkış seçmede kullanılan sinyalleri üretir.
//Aynı zamanda bu kısım bellek adresleme kısmını içerir.
//----------------------------------------------------------

module selector #( parameter DELTA=16, parameter
DELTAWIDTH=4 ) (
          in,
          shift,
          out,
          next_in,
          addrselect,
          direction //if direction is 0, left - else
right shift is performed
    );


input wire  [DELTA-1:0] in;
input wire  [DELTAWIDTH-1:0] shift;
input wire direction;

output wire  [DELTA-1:0] out;
output wire  [DELTA-1:0] next_in;
```

```
output wire  [3*DELTA-1:0] addrselect;

wire  [4*DELTA-1:0] temp;
wire  [4*DELTA-1:0] cur;
wire  [4*DELTA-1:0] tmp_xor;

assign addrselect = tmp_xor [3*DELTA-1:0];

//öne 1 arkaya 0 ekleme işlemi burada yapılmaktadır.
assign cur = {
{DELTA{1'b1}},{DELTA{1'b1}},in,{DELTA{1'b0}} };

assign temp = direction ? cur << (shift+1'b1) : cur >>
shift;

assign tmp_xor = temp ^ cur;


assign out = tmp_xor [3*DELTA-1:2*DELTA] | tmp_xor
[2*DELTA-1:DELTA] | tmp_xor [DELTA-1:0];


//bu kısım thermo vektör düzenleyici bloğu
assign next_in =    (temp [2*DELTA-
1:DELTA]=={DELTA{1'b0}} ) ?   temp [3*DELTA-1:2*DELTA] :
                              (temp [2*DELTA-
1:DELTA]=={DELTA{1'b1}} ) ?   temp [DELTA-1:0] :
                              (temp [2*DELTA-1:DELTA]);


endmodule
```

ÖZGEÇMİŞ

Ad Soyad : **Hasan ÜNLÜ**

Doğum Yeri ve Tarihi : **Niğde, 1988**

E-Posta : **unluh@itu.edu.tr, hasanunlu9@gmail.com**

ÖĞRENİM DURUMU:

- **Lisans** **:** 2011, İstanbul Teknik Üniversitesi, Elektronik Mühendisliği
- **Lisans** **:** 2012, İstanbul Teknik Üniversitesi, Bilgisayar Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

- Elektronik Mühendisliği bölümü birincilik ödülü.
- Bilgisayar Mühendisliği bölümü birincilik ödülü.
- İstanbul Teknik Üniversitesi Bilgisayar ve Bilişim Fakültesi Araştırma Görevlisi 2012-…
- Intel Open Labs İstanbul 2014-…

TEZDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- H. Unlu, M.A. Ozkan, H.F. Ugurdag, E. Adali (2014), Area-Efficient Look-Up Tables for Semi-Randomly Accessible Functions, *Instrumentation, Measurement, Circuits and Systems (IMCAS 14)*, December 15-17, 2014 Istanbul, Turkey.

DİĞER YAYINLAR, SUNUMLAR VE PATENTLER:

- H. Unlu, B. Ors, G. Saldamli (2011), RFID Distance Bounding Protocol Implementation on an FPGA, *7th International Conference on Electrical and Electronics Engineering (ELECO 11),* December 14, 2011, Bursa, Turkey