İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

STOKASTİK HESAPLAMADA HATA ORANLARINI AZALTMAK İÇİN YENİ YÖNTEMLER

YÜKSEK LİSANS TEZİ

Serter YAVUZ

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

MAYIS 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

STOKASTİK HESAPLAMADA HATA ORANLARINI AZALTMAK İÇİN YENİ YÖNTEMLER

YÜKSEK LİSANS TEZİ

Serter YAVUZ
(504101224)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Yrd. Doç. Dr. Mustafa ALTUN

MAYIS 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 504101224 numaralı Yüksek Lisans Öğrencisi **Serter YAVUZ**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"STOKASTİK HESAPLAMADA HATA ORANLARINI AZALTMAK İÇİN YENİ YÖNTEMLER"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Yrd. Doç. Dr. Mustafa ALTUN** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Doç. Dr. Berna ÖRS YALÇIN** İstanbul Teknik Üniversitesi	.....
	Yrd. Doç. Dr. İndrit MYDERRİZİ Doğuş Üniversitesi	.....

Teslim Tarihi : **4 Mayıs 2015**
Savunma Tarihi : **29 Mayıs 2015**

Anneme ve babama,

ÖNSÖZ

"Stokastik Hesaplamada Hata Oranlarını Azaltmak İçin Yeni Yöntemler" adlı tez İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektronik Mühendisliği Yüksek Lisans Programı'nda yüksek lisans tez çalışması olarak araştırılmıştır.
Tüm bilgi ve deneyimleriyle, sabrıyla ve hoşgörüsüyle bana yardımcı olan danışman hocam sayın Yrd. Doç. Dr. Mustafa ALTUN'a sonsuz teşekkürlerimi sunuyorum.

Nisan 2015

Serter YAVUZ
(Elektronik Mühendisi)

İÇİNDEKİLER

Sayfa

ÖNSÖZ ..... **vii**
İÇİNDEKİLER ..... **ix**
KISALTMALAR ..... **xi**
ÇİZELGE LİSTESİ ..... **xiii**
ŞEKİL LİSTESİ ..... **xv**
ÖZET ..... **xvii**
SUMMARY ..... **xix**
1. GİRİŞ **1**
1.1 Stokastik Hesaplama 1
1.2 Hata Oranları 4
1.3 Stokastik Hesaplamada Bağımlılığın Etkileri 6
2. İYİLEŞTİRİLMİŞ BİT DİZİLERİ **9**
2.1 Amaç 9
2.2 Rastgele Bit Atama Yöntemi 9
2.3 Rastgele Bit Karıştırma Yöntemi 10
2.3.1 Giriş bit dizisi p_1=1/2, p_2=1/2 için VE kapı sonuçları 10
2.3.2 Giriş bit dizisi p_1=1/2, p_2=[0,1] için VE kapı sonuçları 11
2.3.3 Giriş bit dizisi p_1=[0,1], p_2=[0,1] için VE hata denklemi 13
2.3.4 Giriş bit dizisi p_1=1/2, p_2=1/2 için VEYA kapı sonuçları 15
2.3.5 Giriş bit dizisi p_1=1/2, p_2=[0,1] için VEYA kapı sonuçları 16
2.3.6 Giriş bit dizisi p_1=[0,1], p_2=[0,1] için VEYA hata denklemi 17
2.3.7 Giriş bit dizisi p_1=1/2, p_2=1/2 için A. VEYA kapı sonuçları 19
2.3.8 Giriş bit dizisi p_1=1/2, p_2=[0,1] için A. VEYA kapı sonuçları 20
2.3.9 Giriş bit dizisi p1=[0,1], p2=[0,1] için A. VEYA hata denklemi 22
2.4 Deneysel Sonuçlar 23
2.4.1 Giriş bit dizisi p_1=1/2, p_2=1/2 için RBAY-RBKY karşılaştırma 23
2.4.2 64 bitlik giriş dizileri p_1=(0,1], p_2=(0,1] iken VE kapı sonuçları 26
2.4.3 64 bitlik giriş dizileri p_1=[0,1], p_2=(0,1] iken VEYA kapı sonuçları 27
2.4.4 64 bitlik giriş dizileri p_1=[0,1], p_2=(0,1] iken A. VEYA kapı sonuçları 28
2.4.5 Çoklayıcı ile karşılaştırma 29
2.4.6 Üretilen olasılık değerleri üzerinden karşılaştırma 30
3. İYİLEŞTİRİLMİŞ DEVRE SİSTEMİ **33**
3.1 0.5 ile Hatasız Çarpma Yapan Devre Bloğu Gerçekleme 34
3.2 Herhangi Bir Olasılık Değerini Hatasız Üretme 35
3.2.1 Stokastik anahtarlamalı devreler 35
3.2.2 Stokastik anahtarlamalı devre benzetimi 37
3.3 Bit Bit Kaydırma Yöntemi 37
3.4 Dizi Klonlama Yöntemi 40
4. SONUÇ VE ÖNERİLER **41**
4.1 Çalışmanın Uygulama Alanları 41

KAYNAKLAR **45**
EKLER **47**
ÖZGEÇMİŞ **53**

KISALTMALAR

SH	**:** Stokastik Hesaplama
RBAY	**:** Rastgele Bit Atama Yöntemi
RBKY	**:** Rastgele Bit Karıştırma Yöntemi
A. VEYA	**:** Ayrıcalıklı Veya
bkz	**:** Bakınız

ÇİZELGE LİSTESİ

Sayfa

Çizelge 1.1 : VE kapısının doğruluk tablosu. 5
Çizelge 2.1 : p_1 = 0.5 için RBAY ile üretilen 8 bitlik örnek bir dizi. 10
Çizelge 2.2 : p_1 = 6/8 için katsayılar tablosu. 14
Çizelge 2.3 : p_1 = 3/8 için katsayılar tablosu. 14
Çizelge 2.4 : p_1 = 1/4 için katsayılar tablosu. 14
Çizelge 2.5 : p_1 = 6/8 için katsayılar tablosu. 18
Çizelge 2.6 : p1 = 2/4 için katsayılar tablosu. 18
Çizelge 2.7 : p_1 = 6/8 için katsayılar tablosu. 22
Çizelge 2.8 : p_1 = 2/4 için katsayılar tablosu. 22
Çizelge 2.9 : Üç lojik kapı için değişen bit uzunları ile elde edilen hata değerleri. . 24
Çizelge 2.10 : Çoklayıcı ile (giriş dizileri 64 bit) iki yöntem için hata oranları. 30
Çizelge 2.11 : RBKY-RBAY üretilen olasılık değerleri üzerinden karşılaştırma. ... 31
Çizelge 3.1 : Stokastik anahtarlamalı devre benzetimi..... 37

ŞEKİL LİSTESİ

Sayfa

Şekil 1.1 : Aritmetik işlemleri stokastik gerçekleme: (a) Çarpma, (b) Ölçekli toplama.. 1
Şekil 1.2 : VE kapısı ile stokastik işlem örneği. 2
Şekil 1.3 : VEYA kapısı ile stokastik işlem örneği. 2
Şekil 1.4 : A. VEYA kapısı ile stokastik işlem örneği. 3
Şekil 1.5 : Dönüştürücüler: (a) ikiliden stokastiğe; (b) stokastikten ikiliye. 3
Şekil 1.6 : 4 bitlik stokastik üreteç örneği [11]. 4
Şekil 1.7 : p_1,p_2=1/2 iken VE kapısı için hata oranları-bit uzunluğu grafiği. 4
Şekil 1.8 : Bağımlı girişlerle elde edilen çıkışlar. 6
Şekil 2.1 : VE kapısı hata-bit sayısı grafiği. 11
Şekil 2.2 : 4 bit için VE kapısı hata-p_2 grafiği. 12
Şekil 2.3 : 8 bit için VE kapısı hata-p_2 grafiği. 12
Şekil 2.4 : 16 bit için VE kapısı hata-p2 grafiği. 12
Şekil 2.5 : 32 bit için VE kapısı hata-p2 grafiği. 13
Şekil 2.6 : VEYA kapısı hata-bit sayısı grafiği. 15
Şekil 2.7 : 4 bit için VEYA kapısı hata-p_2 grafiği. 16
Şekil 2.8 : 8 bit için VEYA kapısı hata-p_2 grafiği. 16
Şekil 2.9 : 16 bit için VEYA kapısı hata-p_2 grafiği. 17
Şekil 2.10 : 32 bit için VEYA kapısı hata-p_2 grafiği. 17
Şekil 2.11 : A.VEYA kapısı hata-bit sayısı grafiği. 19
Şekil 2.12 : 4 bit için A.VEYA kapısı hata-p_2 grafiği. 20
Şekil 2.13 : 8 bit için A.VEYA kapısı hata-p2 grafiği. 20
Şekil 2.14 : 16 bit için A.VEYA kapısı hata-p_2 grafiği. 21
Şekil 2.15 : 32 bit için A.VEYA kapısı hata-p_2 grafiği. 21
Şekil 2.16 : VE kapısı için RBAY- RBKY bit uzunluğu-hata oranı grafiği. 24
Şekil 2.17 : VEYA kapısı için RBAY- RBKY bit uzunluğu-hata oranı grafiği. 25
Şekil 2.18 : A. VEYA kapısı için RBAY- RBKY bit uzunluğu-hata oranı grafiği. 25
Şekil 2.19 : RBAY VE kapısı p_2-p_1-hata oranı grafiği.. 26
Şekil 2.20 : RBKY VE kapısı p_2-p_1-hata oranı grafiği.. 26
Şekil 2.21 : RBAY VEYA kapısı p_2-p_1-hata oranı grafiği.. 27
Şekil 2.22 : RBKY VEYA kapısı p_2-p_1-hata oranı grafiği.. 27
Şekil 2.23 : RBAY A. VEYA kapısı p_2-p_1-hata oranı grafiği.. 28
Şekil 2.24 : RBKY A. VEYA kapısı p_2-p_1-hata oranı grafiği.. 28
Şekil 2.25 : Çoklayıcı devresi birinci tasarım. 29
Şekil 2.26 : Çoklayıcı devresi ikinci tasarım. 29
Şekil 2.27 : Lojik kapılarla olasılıklar elde etme. 30
Şekil 2.28 : Şekil 2.26'dan seçilen iki devre. 31
Şekil 3.1 : Girişler ve çıkışlar kullanılan devre yapısından bağımsız olarak Gauss dağılımına sahiptir. 33
Şekil 3.2 : 0.5 ile hatasız çarpma yapan devre bloğu. 34
Şekil 3.3 : Birleştirme bloğunun gerçeklenmesi. 34
Şekil 3.4 : VEYA kapısı ile blok gerçekleme. 35
Şekil 3.5 : Olasılıksal anahtarların seri ve paralel bağlanması [18]. 36
Şekil 3.6 : 11/16 = 0.1011 olasılık değerinin gerçeklenmesi [18]. 36
Şekil 3.7 : Olasılık gerçekleme ağacı. 38
Şekil 3.8 : Bit bit kaydırma yönteminin algoritması. 39

Şekil 3.9 : Bit bit kaydırma yönteminin teorik uygulaması. 39
Şekil 3.10 : Bit bit kaydırma yönteminin gerçeklenmesi..... 40
Şekil 3.11 : Dizi klonlama yönteminin algoritması. 40
Şekil 4.1 : Gama düzelticiler: (a) Stokastik; (b) Geleneksel [6]. 42
Şekil 4.2 : İki yöntem arasında hata tolerans karşılaştırması [10]. 42
Şekil 4.3 : LDPC kodlarının çalışmamıza uygulanabilecek kısmı [24]. 43

STOKASTİK HESAPLAMADA HATA ORANLARINI AZALTMAK İÇİN YENİ YÖNTEMLER

ÖZET

Günümüzde hesaplama yapması için tasarlanan devreler küçük boyut ve düşük güç tüketimi gibi konuları dikkate alır. Üretim sürecindeki hatalar olasılıksal terimlerle ifade edilebilir. Bu nedenle bu hatalarla ilgili geleneksel olmayan yöntemler tercih edilmeye başlanmıştır. Stokastik hesaplama, bu geleneksel olmayan yöntemlerden biridir.

Stokastik hesaplama, rastgele bit dizileriyle zamanda kesintisiz değerleri temsil eden teknikler topluluğudur ve ilk olarak 1953 yılında John von Neumann tarafından hazırlanan bir makale ile takdim edilmiştir. Bu kesintisiz değerler olasılıklar olarak değerlendirilir. Dolayısıyla geçerli aralık [0,1] aralığıdır. Stokastik hesaplama devrelerini gerçeklemek kolaydır (çarpma için tek bir VE kapısı). Diğer bir deyişle düşük maliyetli gerçeklemeler yapılabilmektedir.

İlave olarak, tek bir bit değeri değişimi stokastik sayının temsil ettiği değerde küçük bir değişime neden olacaktır. Bir bit değeri değişimi gerçekleşirse temsil edilen değerin paydası 1 artacak veya azalacaktır.

Stokastik hesaplamanın sahip olduğu etkileyici avantajların yanısıra dezavantajları da vardır. Bunlar temelde süre ve hata oranlarıdır. Aynı olasılık değerleri farklı permütasyonlarla da elde edilebileceği için beklenen sonuçlar her zaman elde edilemeyebilir. Bu durum hatalı bir değer elde edilmesine yol açabilir. Kısa bit dizileri için bu hata oranları oldukça yüksektir. Hata oranlarını azaltmak için, dizilerin uzunluklarının arttırılması gerekmektedir. Ancak, uzun bit dizileri işlem süresini arttırmaktadır. Stokastik hesaplama alanındaki araştırmalar hata oranlarını göz ardı etmektedir. Yüksek hata oranları stokastik hesaplamanın uygulama alanlarını daraltmaktadır. Bu problem bu tezi yazmada temel motivasyonumuz olmuştur.

Çarpma ve ölçekli toplama stokastik hesaplamanın temel aritmetik işlemleridir. Çarpma giriş dizisinin uzunluğundan bağımsız olarak tek bir VE kapısı ile gerçeklenebilmektedir. Geleneksel yöntem ile uzun bit dizilerini çarpmak için fazla sayıda eleman içeren bir devre kullanılmalıdır.

[0,1] kapalı aralığında kalma zorunluluğu olduğu için toplama işlemi stokastik hesaplamada kullanılamaz. Onun yerine ölçekli toplama tanımlanmıştır. Ölçekli toplama yapmak için çoklayıcı kullanılır.

Stokastik devrelerde çıkış deterministiktir. Çünkü çıkıştaki değer, çıkış dizisinin içerdiği lojik 1'ler sayılarak elde edilir. Ancak, rastgele bit atama yönteminde girişler olasılıksaldır. Bu yüzden hata oranları bu yöntemde yüksek çıkmaktadır. Hata oranlarını azaltmak amacıyla girişlerin de deterministik olduğu rastgele bit karıştırma yöntemi önerilmiştir.

Geleneksel stokastik hesaplamada Bernoulli dağılımına sahip bit dizileri birbirinden bağımsızdır. Bağımsız bit dizileri ile girişlerin tüm kombinasyonları için hatasız çıkış elde etmek mümkün değildir. Eğer giriş dizileri bağımlı hale gelirse beklenen çıkış değeri de değişir. Bağımlılık kullanılarak hatasız çıkışlar elde etmek mümkündür.

Bir girişin değili ikinci giriş olarak uygulanırsa çıkışta 0 değeri elde edilir. Ayrıca iki girişe de aynı dizi uygulanırsa, çıkışta da aynı dizi elde edilir. Ancak çıkışta elde edilmesi beklenen değerler elde edilen değerler değildir. Bu bağımlılığın sonucudur.

Bu tezde, ilk bölümde hata oranlarını azaltmak amacıyla, adını rastgele bit karıştırma yöntemi koyduğumuz yeni bir yöntem önerdik. Bu yöntem, stokastik hesaplamada geleneksel olarak kullanılan rastgele bit üreteçleri ile oluşturulan bit dizileri ile (bu yönteme de rastgele bit atama yöntemi adını koyduk), lojik kapılar ve bunlardan oluşan devreler kullanılarak hata oranları üzerinden karşılaştırılmıştır. İki yöntemin arasındaki fark giriş dizileri aynı değerleri temsil etse de dizilerin fiziksel olarak farklılık göstermesidir. Karşılaştırma yapmak için çeşitli giriş kombinasyonları ile hatayı veren iki ve üç boyutlu grafikler çıkarılmış, bit dizilerinin değişik uzunlukları dikkate alınmış, lojik kapılar için genel hata denklemleri oluşturulmuş, yazılımlar yardımıyla oluşturulan algoritmalar gerçeklenmiştir.

Elde edilen veriler ışığında, önerilen rastgele bit karıştırma yöntemi, temel lojik kapılar için geleneksel yöntemden daha iyi hata oranları vermiştir.

Bağımlılık kullanılarak hatasız çıkışlar elde etmek mümkündür sonucuna vardıktan sonra bağımlılığı kullanarak hatasız çıkış değerleri elde etmeye çalıştık. Bağımlılığı kullandığımız devre yapılarında rastgele bit karıştırma yöntemi kullanılmalıdır. Rastgele bit atama yöntemi ile hatasız sonuçlar elde edilemez.

Giriş dizilerini iyileştirme çalışmalarından sonra devre yapısını iyileştirmeye çalıştık. 3. kısımda 0.5 ile hatasız çarpma yapan bir blok ve hatasız çarpma için iki yöntem önerdik. 0.5 ile çarpma bloğunu herhangi bir olasılık değerini hatasız elde etmek için uyarladık. Önerdiğimiz iki yönteme de bit bit kaydırma ve dizi klonlama yöntemi adını verdik. Bilgisayar programları ile yaptığımız simülasyonlarda önerdiğimiz bloğun ve yöntemlerin hatasız sonuçlar verdiğini doğruladık.

4. kısımda, çalışmamız sonucunda elde ettiğimiz sonuçları yorumladık ve gelecekte uygulanabileceği çalışma alanlarından da bahsettik.

NEW METHODS FOR REDUCING ERROR RATES IN STOCHASTIC COMPUTING

SUMMARY

Today circuits which are designed on the purpose of computing, care about small size, low power consumption and high reliability. Manufacturing process's defects can be described in probabilistic terms. For this reason unconventinal methods which are related with this defects are being preferred. Stochastic computing is one of these unconventional methods.

Stochastic computing was first presented in a paper written by John von Neumann in 1953. In stochastic computing, numbers stand for probability values (with or without error) are represented by random bit streams. Therefore, the available range is [0,1] interval. It is easy to realise the circuits of stochastic computing (e.g. one AND gate for multiplication). In other words low cost implementations can be realised .

Moreover, a single bit flip will cause a small change in the value of the stochastic numbers. The value of the nominator will be increased or decreased by 1 if a single bit flip happens.

Along the great advantages, stochastic computing has disadvantages. These are mainly time and error rates. Different combination of streams represent same values and this yields error rates in stochasting computing. These error rates are extremely high for short bit streams. To reduce error rates, longer bit streams should be used. But these longer streams extend the operation time. Research about stochastic computing in literature has not mentioned about error rates before. The high error rates narrows down the application field of stochastic computing. This problem becomes our motivation to write this thesis.

Complicated stream operations can be performed with simple circuits in stochastic computing. Multiplication and scaled addition are basic arithmetic operation of stochasting computing. Multiplication can be realized with one AND gate independent from the input bit stream length. The circuit may be extremely large to multiply long bit streams with the conventional method.

Due to the interval [0,1], addition cannot be performed in stochastic computing. Instead, scaled addition that guarantees to remain in the interval is described. MUX is used for scaled addition.

The output in stochastic circuits is deterministic because it is obtained by counting 1s in the output stream. However, the inputs in the random bit assigning method are probabilistic. Therefore, error rates appear to be high in this method. In order to reduce the error rates, random bit shuffling method whose inputs are aslo deterministic was researched.

In conventinal SC, bit streams which have Bernoulli distribution are independent from each other. With independent streams it is not possible to obtain error free outputs for all combination of inputs. If input streams become dependent, expected result for classical SC will change. By using dependency it is possible to obtain error free outputs.

If the inverse of an input is applied as second input, 0 value will be obtained at output. It is expected that z = p * (1-p), but due to dependency this expectation fails.

Morever, if the first input is applied also as a second input, the same input will be obtained at output. It acts like a buffer. It is expected that z = p * p, but due to dependency this expectation fails.

In the first part of this thesis, we offer a method based on improving the generation of bit streams. In Section 2 detailed information about our method and the conventional one is given. This method has been compared with the bit sequence generated by random bit generators (we named it random bit assigning method) traditionally used in stochastic computing by using logic gates and circuits in terms of error rates. In order to compare, 2D and 3D graphs that shows the error were drawn by realising different combinations of two inputs, different length of bit streams were taken into consideration, general error equations for logic gates were obtained and algorithms were realised with the aid of softwares.

In order to generate bit streams in SC, random bit generators are used. We named this bit stream generation technic as random bit assign method (RBAM). Let desired input probability value for RBAM be p_1 and let random number generator generates values between [0-1] with binomial distribution. If the generated value is smaller than p_1, logic 1 is added to the stream. If the generated value is bigger than p_1 logic 0 is added to the stream. Therefore, there exists difference between the desired input value and the obtained input value.

Let the input desired value for random bit shuffling method (RBSM) be p_1. In this method, depending on the desired probability value (p_1) and the length of the bit stream (n), a bit stream containing $n*p_1$ 1s is generated. Fisher-Yates shuffling algorithm, adapted to the computers by Durstenfeld is applied. In this way, it is guaranteed that the desired and the generated input values are probabilistically same. With this approach output error rates will be downgraded.

The difference between two methods is that the input streams shows differences physically although input bit streams represent same values.

We worked on basic logic gates to make comparison between two methods. The results were obtained by fixing the two inputs, changing the inputs between [0, 1] interval and using different bit length.

We benefit from the Monte Carlo analysis to obtain the equations. This analysis is a generic name given to obtaining the solution of the problem by using random numbers.

In the light of obtained data, the random bit shuffling method that is recommended, has given better error rates that traditional method for main logic gates.

After the research on generating the input stream for reducing the error rates, we started working on circuitry. In Section 3 we show our works which are error free multiplication by 0.5 block, obtaining any probability value without an error, and two method for error free multiplication. We present a block for multiplication with 0.5 without an error. The block can also be used in generating probabilities. We also present new error free multiplication methods that we named it bitwise shifting method and stream cloning method. In the light of obtained data from computer programs, our block and method that is recommended, has given zero error rates.

First, by keeping the input values, generated with RBSM, and expected output value same we tried to make changes with logic gates. During this study, we could not get the results we want with different circuits. The inputs had Gaussian distribution and

output stream was having Gaussian distribution independent from the circuit we used. Therefore we could not obtain a change in error rates.

Our research shows that with independency it is not possible to obtain error free circuitry. After facing this result, we decide to use dependency to obtain error free circuitry as a next step.

In order to obtain error free circuitry, RBSM must be used. There is no way to get result without an error with RBAM.

We suggest an error free block for multiplication by 0.5. With this block, the input stream is multiplied by 0.5 and zero error at output stream is achieved. However if the length of the input stream is n, the length of the output stream will be 2n because of merging operation. We design a circuit for this merge operation. Same block is working for OR gate. AND gates should substitute with OR gates. At the output 0.5 + 0.5 * p is obtained. OR gates can be used to obtain probability values.

We adapt the block realizing the error free multiplication with 0.5 to Switch Circuits in order to obtain any probability value. Inputs are 0.5 (0.1_2=1/2). In order to shift the binary value one digit to right and add "0", the circuit block based on AND gate is used (0.01_2=1/4). In order to shift the binary value one digit to right and add "1", the circuit block based on OR gate is used (0.11_2=3/4). By cascading the blocks, any desired probability value can be obtained without error.

At a later stage, we suggest Bitwise Shifting Method for multiplication. In this method one of the two inputs is kept, while shifting one bit of other input to the left. Shift operation is applied "the length of the input stream-1" times (e.g. if the input is 8 bit, shift operation will be applied 7 times). Bit streams obtained from the output gate are merged. This method will be implemented by changing the structure of "Binary to Stochastic Converter" part. We aim to get the manipulated stream we want with this philosophy. Also just one AND gate will be used. We are aware that randomness is destroyed with changing the generation of the input streams. We intend to multiply two probability values with using less transistors.

We suggest another method for multiplication. In this method, cloning is the main idea. The philosophy of RBSM constitutes a basis for this method. We must generate the probability value without an error. One of the input is cloned up to the number of bits (p_1=1/4 (0100010001000100)). The other input is re-prepared based on p_2.

In Section 4, we discuss the results which we obtained during our work. Also future works that we will apply our methods/circuits are considered.

1. GİRİŞ

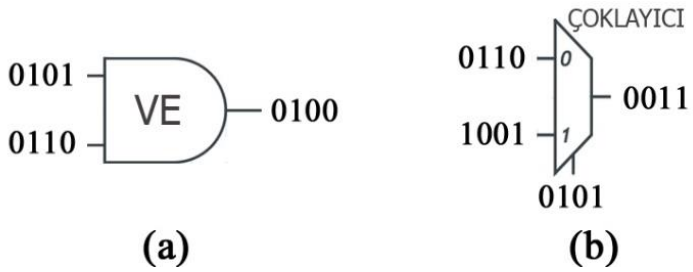
Stokastik hesaplama (SH) [1], rastgele bit dizileriyle zamanda kesintisiz değerleri temsil eden teknikler topluluğudur ve ilk olarak 1953 yılında John von Neumann tarafından hazırlanan bir makale ile takdim edilmiştir [2]. Bu kesintisiz değerler olasılıklar olarak değerlendirilir. Dolayısıyla geçerli aralık [0,1] aralığıdır. Örneğin, 16 bit uzunluklu bir dizi 12 adet lojik 1 içeriyorsa, bu dizi 1'lerin dizideki konumundan bağımsız olarak p = 0.75'i temsil eder. Örneğin (0000111111111111) ve (0101111110111110). Farklı bit dizisi uzunluklarıyla da aynı olasılık elde edilebilir.

1.1 Stokastik Hesaplama

Diziler üzerinde karmaşık hesaplamalar basit devrelerle bit bit işlemlerle yapılabilir. Çarpma ve ölçeklenmiş toplama gibi temel işlemlerin yanısıra SH, bölme ve karekök alma [3], matris işlemleri [4] ve polinom aritmetiğine uygulanabilir [5]. Aritmetik işlemlerin yanısıra görüntü işleme [6], hata düzeltme kodları [7] ve yapay sinir ağları tasarımı [8] alanlarında da uygulamaları vardır.

Çarpma işlemi giriş dizisinin uzunluğundan bağımsız olarak Şekil 1.1'deki gibi tek bir lojik VE kapısıyla gerçeklenebilmektedir [9]. Klasik yöntemle çarpma yaparken bit dizisi uzadıkça devre de çok büyük hale gelmektedir.

Olasılık değerleri [0,1] aralığında olduğu için toplama işlemi SH'de yapılamaz. Toplama işleminin yerini tanımlı aralıkta kalmayı garanti eden ölçekli toplama tanımlanmıştır. Ölçekli toplama için Şekil 1.1'deki gibi çoklayıcı kullanılmaktadır.



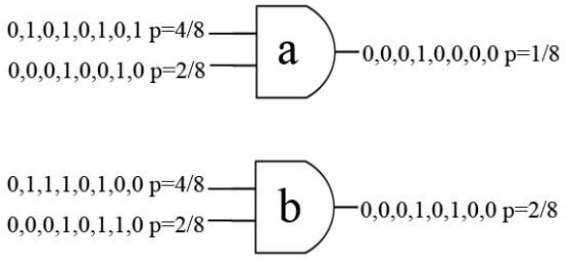
Şekil 1.1 : Aritmetik işlemleri stokastik gerçekleme:
(a) Çarpma, (b) Ölçekli toplama.

İki adet girişte bulunan bit dizisindeki 1 sayısına bağlı olarak temsil edilen olasılıklar sırasıyla p_1 ve p_2 olsun. Çarpma işlemi için çıkışta elde edilmesi beklenen olasılık

$$z_b = p_1 . p_2 \quad \textbf{(1.1)}$$

'dir. Ancak girişe gelen aynı olasılık değerleri farklı permütasyonlarla elde edilebileceği için çıkışta beklenen sonuç her zaman elde edilemeyebilir. Bu noktada hata oranı devreye girer.

Şekil 1.2'de a ile gösterilen VE kapısı ile beklenen sonuç elde edilebilirken b ile gösterilen kapıda sonuç hatalı elde edilmiştir.



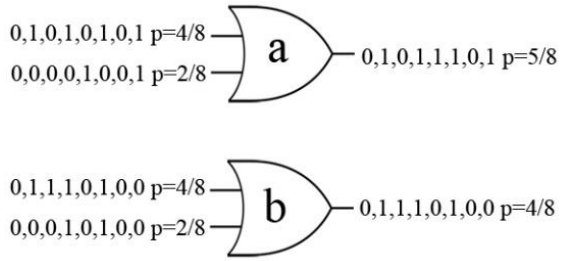
Şekil 1.2 : VE kapısı ile stokastik işlem örneği.

VEYA kapısı için iki adet girişte bulunan bit dizisindeki 1 sayısına bağlı olarak temsil edilen olasılıklar sırasıyla p_1 ve p_2 olsun. Çıkışta elde edilmesi beklenen olasılık

$$z_b = 1 - [(1 - p_1) \cdot (1 - p_2)] \quad \textbf{(1.2)}$$

'dir.

Şekil 1.3'te a ile gösterilen VEYA kapısı ile beklenen sonuç elde edilebilirken b ile gösterilen kapıda sonuç hatalı elde edilmiştir.



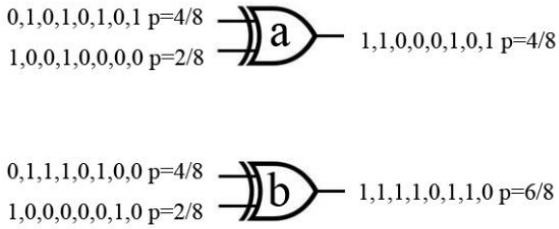
Şekil 1.3 : VEYA kapısı ile stokastik işlem örneği.

A. VEYA kapısı için iki adet girişte bulunan bit dizisindeki 1 sayısına bağlı olarak temsil edilen olasılıklar sırasıyla p_1 ve p_2 olsun. Çıkışta elde edilmesi beklenen olasılık

$$z_b = p_1 \cdot (1 - p_2) + p_2 \cdot (1 - p_1) \tag{1.3}$$

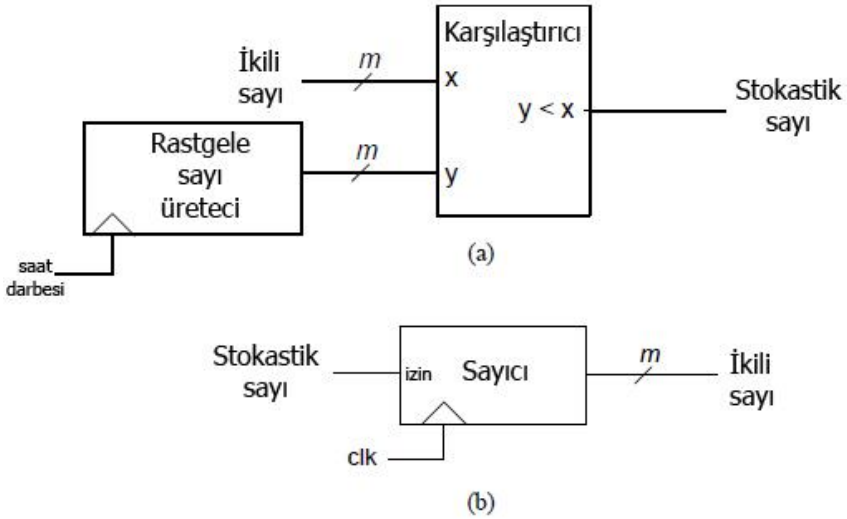
'dir.

Şekil 1.4'te a ile gösterilen A. VEYA kapısı ile beklenen sonuç elde edilebilirken b ile gösterilen kapıda sonuç hatalı elde edilmiştir.



Şekil 1.4 : A. VEYA kapısı ile stokastik işlem örneği.

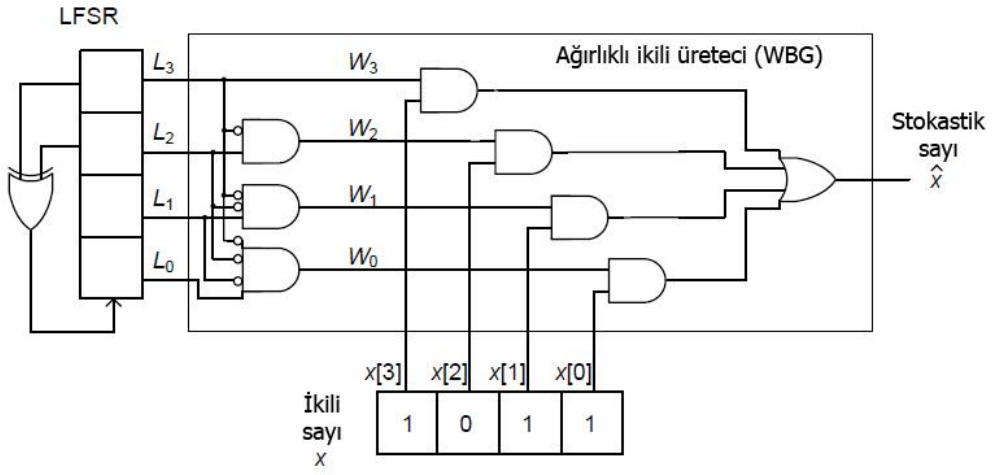
SH'nin diğer temel elemanları, stokastik sayılardan ikili sayılara ve ikili sayılardan stokastik sayılara dönüştürücülerdir [10]. Şekil 1.5'te [10] bu dönüştürücüler görülmektedir.



Şekil 1.5 : Dönüştürücüler: (a) ikiliden stokastiğe; (b) stokastikten ikiliye.

İkiliden stokastiğe dönüştürücüler stokastik dizi üreteci olarak ta adlandırılır. Çalışma mantığı Kısım 2.2'de detaylı olarak anlatılacaktır. Stokastik sayıyı ikiliye çevirmek daha kolaydır. Çıkıştaki dizinin içerdiği 1 sayısı sayılarak ikili sayı elde edilir.

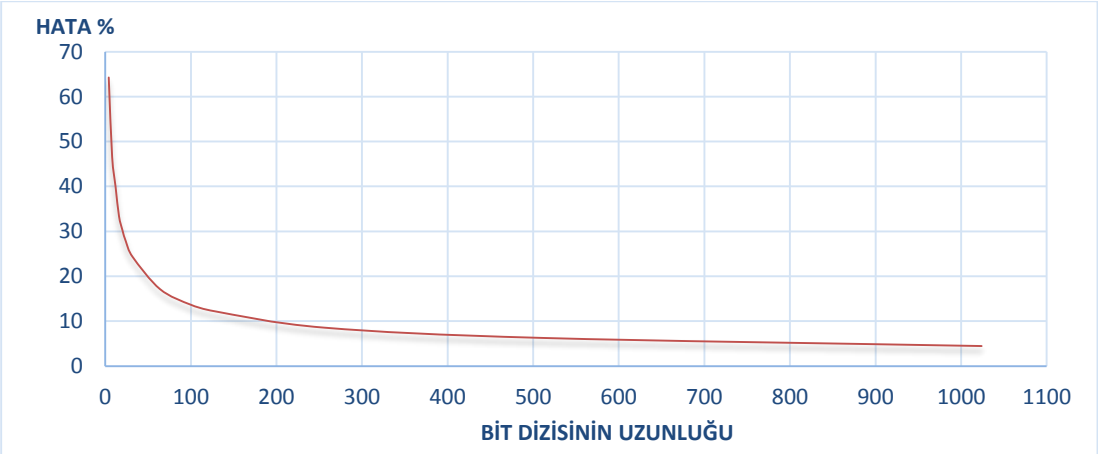
Stokastik sayı üretmek için Şekil 1.6'daki gibi Lineer Geri Beslemeli Ötelemeli Kaydedici (Linear Feedback Shift Register-LFSR) kullanılabilir [11].



Şekil 1.6 : 4 bitlik stokastik üreteç örneği [11].

1.2 Hata Oranları

Bit dizilerinin farklı kombinasyonları aynı olasılık değerini temsil eder ve bu durum SH'de sonuçların belirli bir hata oranı ile elde edilmesine neden olur. Bu hata oranları kısa bit dizileri için oldukça yüksektir. Şekil 1.7'de görüldüğü gibi hata oranlarını azaltmak amacıyla uzun bit dizileri kulanılmalıdır. Ancak bu uzun bit dizileri işlem süresini uzatmaktadır. Bu nedenle klasik SH'nin uygulama alanları dardır.



Şekil 1.7 : p_1, p_2=1/2 iken VE kapısı için hata oranları-bit uzunluğu grafiği.

Hata oranı **(1.4)**'teki gibi hesaplanır.

$$\varepsilon = \frac{\left|z - z_b\right|}{z_b} \tag{1.4}$$

ε hata oranını, z çıkışta elde edilen olasılık değerini, z_b ise beklenen çıkış değerini gösterir. Beklenen değer kullanılan lojik kapılara göre değişiklik gösterir.

Çizelge 1.1'de VE kapısı için verilen doğruluk tablosundan çıkışta lojik 1 elde etmek için iki girişin de lojik 1 olması gerektiği görülmektedir. İlk girişin lojik 1 olma olasılığı p_1, ikinci girişin lojik 1 olma olasılığı p_2 olsun. VE kapısının/Çarpma işleminin beklenen değeri (**1.5**)'teki gibi hesaplanır.

$$z_b = p_1 \cdot p_2 \tag{1.5}$$

Çizelge 1.1 : VE kapısının doğruluk tablosu.

X_1	X_2	Z
0	0	0
0	1	0
1	0	0
1	1	1

Benzer şekilde çoklayıcının çıkışındaki beklenen değer z_b

$$z_b = p_1 \cdot p_s + p_2 \cdot (1 - p_s) \tag{1.6}$$

'dir. p_1 ve p_2 girişlerin temsil ettiği olasılık değerini, p_s seçme girişine gelen dizinin temsil ettiği olasılık değerini gösterir. VEYA kapısının çıkışında beklenen değer z_b

$$z_b = p_1 + p_2 \cdot (1 - p_1) \tag{1.7}$$

'dir. p_1 ve p_2 girişlerin temsil ettiği olasılık değerleridir. A. VEYA kapısının çıkışında beklenen değer z_b

$$z_b = p_1 \cdot (1 - p_2) + p_2 \cdot (1 - p_1) \tag{1.8}$$

'dir.

Stokastik devrelerde çıkış deterministiktir; çünkü çıkış, çıkış dizisinin içerdiği 1'ler sayılarak elde edilir. Ancak geleneksel yöntem olan RBAY'de girişler olasılıksaldır. Bu nedenle bu yöntemde hata oranları yüksek çıkmaktadır. Hata oranlarını azaltmak için girişlerin de deterministik olduğu RBKY'yi önerdik ve analiz ettik.

1.3 Stokastik Hesaplamada Bağımlılığın Etkileri

Geleneksel SH'de Bernoulli dağılımına sahip girişler birbirinden bağımsızdır. Bağımsız giriş dizileri ile girişlerin tüm kombinasyonları için hatasız çıkışlar elde etmek mümkün değildir. Eğer giriş dizileri bağımlı hale getirilirse, klasik SH için beklenen değerler değişecektir. Bağımlılık kullanılarak hatasız çıkışlar elde etmek mümkündür.

Eğer bir girişin değili, ikinci giriş olarak uygulanırsa, Şekil 1.8'de görüldüğü gibi çıkışta 0 değeri elde edilir. Çıkıştaki beklenen değer z_b ise

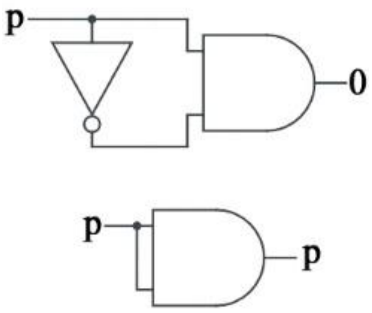
$$z_b = p \cdot (1 - p) \tag{1.9}$$

'dir, ancak bağımlılık nedeniyle bu beklenen değer elde edilememektedir.

İlave olarak, eğer bir giriş aynı anda iki girişe de uygulanırsa, Şekil 1.8'de görüldüğü gibi çıkışta girişteki aynı olasılık değeri elde edilir. Devre, ara bellek (buffer) gibi davranır. Çıkıştaki beklenen değer z_b ise,

$$z_b = p \cdot p \tag{1.10}$$

'dir, ancak bağımlılık nedeniyle bu beklenen değer elde edilememektedir.



Şekil 1.8 : Bağımlı girişlerle elde edilen çıkışlar.

Stokastik devreler geleneksel donanım gerçeklemelerinden daha az alana ihtiyaç duyar [5]. Öte yandan, çıkışlar belirli bir hata oranı ile edilir. Bu yüzden, uygulama alanları kısıtlıdır (görüntü işleme, düşük yoğunlu parite kontrol kodları). Daha önce SH hakkında yapılan çalışmalar hata oranlarını göz ardı etmiştir.

Yaptığımız çalışmada ilk olarak giriş bit dizilerinin üretimini iyileştirmek için bir yöntem önerdik. 2. Bölümde önerdiğimiz yöntem ve geleneksel olarak kullanılan yöntem hakkında detaylı bilgi verilecektir. İki yöntem arasında yapılan karşılaştırma analizleri ele alınacaktır.

Giriş dizilerinin iyileştirilmesinden sonra, hatasız işlemler yapabilmek amacıyla bağımlılığı kullanarak devre sistemini iyileştirmeye yönelik bir çalışma yürüttük. 3. bölümde 0.5 ile hatasız çarpma bloğu, herhangi bir olasılık değerini hatasız elde etme ve hatasız çarpma yapmak için önerdiğimiz iki yöntem gösterilecektir.

4. bölümde çalışmamız sırasında elde ettiğimiz sonuçlar tartışılacaktır. Önerilen yöntemlerin/devrelerin gelecekte hangi alanlarda uygulanabileceği de belirtilecektir.

2. İYİLEŞTİRİLMİŞ BİT DİZİLERİ

2.1 Amaç

Stokastik devrelerde çıkış, dizideki 1'ler sayılarak elde edildiği için deterministiktir. Ancak rastgele bit atama yönteminde girişler ise olasılıksaldır. Bu nedenle hata oranları yüksek çıkmaktadır. Hata oranlarını azaltmak amacıyla girişin de deterministik olduğu rastgele bit karıştırma yöntemi araştırılmıştır, detaylı bir analizi yapılmıştır.

Bu iki yöntem arasındaki fark, girişe gelen zamanda sürekli bit dizilerinin temsil ettiği değerler aynı olmasına rağmen dizilerin fiziksel olarak farklılık göstermesidir.

Tezin bu bölümünde ilk olarak karşılaştırılan yöntemlerle ilgili ayrıntılı bilgiler verilecek, karşılaştırma analizleri yapılacak, örnek bir devre uygulaması gerçekleştirilecek, çalışmanın elde edilen sonuçlarından bahsedilecektir.

2.2 Rastgele Bit Atama Yöntemi

SH'de bit dizisi oluşturmak için stokastik bit üreteçleri kullanılır [9]. Bu şekilde dizi oluşturma yöntemine rastgele bit atama yöntemi (RBAY) adını koyduk. RBAY için girişte istenen olasılık değeri p_1 olsun. Rastgele sayı üreteci de [0-1] arası binom dağılıma sahip değerler üretsin. Üretilen sayı p_1'den küçük ise girişteki diziye lojik 1 değeri, p_1'den büyük ise girişteki diziye lojik 0 değeri atanır. Bu sebepten girişte istenen değer ile elde edilen arasında farklar oluşmaktadır.

Çizelge 2.1'deki örnekte üretilen 8 bitlik dizide üç adet lojik 1, beş adet lojik 0 bulunmaktadır. Bu nedenle girişte elde edilen dizi $p_e = 0.375$ olmaktadır. Çıkışta elde edilecek değerde de hata oluşacaktır. Devreler karmaşıklaştıkça hata oranları da kabul edilebilir seviyelerin üzerine çıkacaktır.

Çizelge 2.1 : $p_1 = 0.5$ için RBAY ile üretilen 8 bitlik örnek bir dizi.

Üreteçte Elde Edilen Sayı	Bit Dizisi
0.89967	0
0.45847	1
0.81537	0
0.20818	1
0.71289	0
0.43615	1
0.64214	0
0.78744	0

2.3 Rastgele Bit Karıştırma Yöntemi

Adını koyup, SH için önerdiğimiz rastgele bit karıştırma yöntemi (RBKY) [12] için girişte istenen olasılık değeri p_1 olsun. Bu yöntemde istenen olasılık değeri (p_1) ve bit dizisi uzunluğuna (n) göre $n*p_1$ adet 1 içeren dizi oluşturulur ve Fisher–Yates karıştırma algoritmasının [13] Durstenfeld tarafından bilgisayar kullanımına uyarlanan versiyonu [14] uygulanır. Bu nedenle girişte istenen değerle, elde edilen değerin olasılıksal olarak aynı olması garanti edilir. Bu sayede elde edilecek çıkıştaki hata oranlarında bir düşüş elde edilmiş olacaktır.

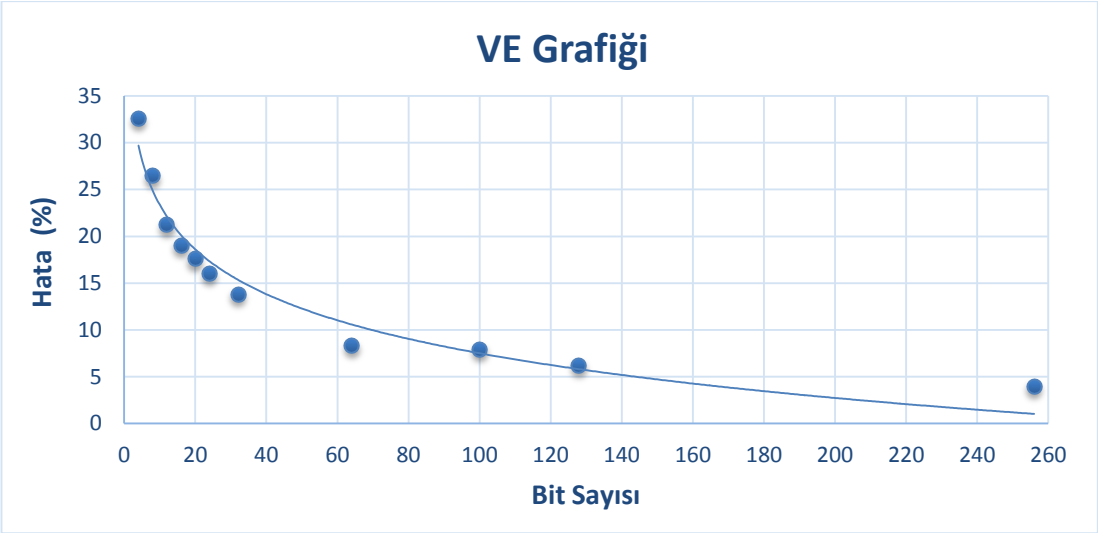
Deneysel aşamada ilk olarak RBKY için giriş olasılık değerlerini sabit tutarak lojik kapılarda hata oranlarını tespit edip, denklemler çıkarttık.

İkinci aşamada bir girişi sabit tutarak, ikinci giriş [0-1] aralığında değişirken çeşitli bit uzunlukları için hata oranlarını belirledik.

Üçüncü aşamada iki giriş te [0-1] aralığında değişirken çeşitli bit uzunlukları için hata denklemlerini çıkarttık.

2.3.1 Giriş bit dizisi $p_1=1/2$, $p_2=1/2$ için VE kapı sonuçları

Bu analizde giriş olasılıkları sabit tutularak farklı bit uzunlukları için VE lojik kapısı için Şekil 2.1'deki grafik elde edilmiştir.



Şekil 2.1 : VE kapısı hata-bit sayısı grafiği.

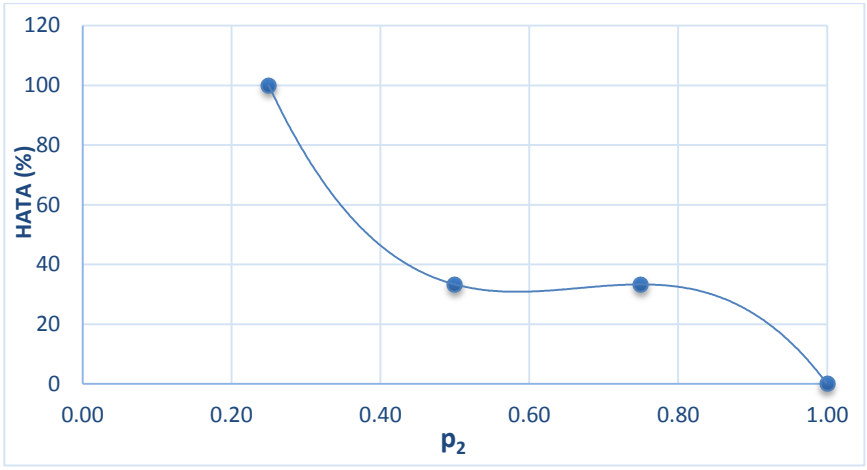
VE kapısı için rastgele bit karıştırma yöntemi ile elde edilen sonuçlar gösteriyor ki, bit dizisinin uzunluğu arttıkça hata oranı düşmektedir. 64 bitten sonrası için hata oranı %10'un altında olduğu için çalışılabilecek bit uzunluğu da bu noktadan sonrasıdır.

$p_1=p_2=1/2$ ve x=bit dizisinin uzunluğu iken RBKY için VE kapısı genel hata denklemini çıkarttık **(2.1)**.

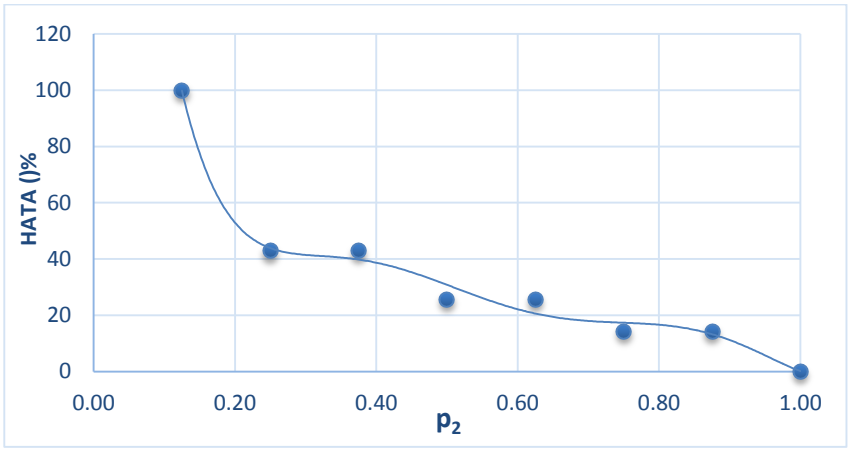
$$Hata(x) = \frac{2.\sum_{s=0}^{\frac{x}{4}-1} \binom{x/2}{s} \cdot \binom{x/2}{\frac{x}{2}-s} \cdot \binom{\frac{x}{4}-s}{x/4}}{\binom{x}{x/2}} \qquad \textbf{(2.2)}$$

2.3.2 Giriş bit dizisi $p_1=1/2$, $p_2=[0,1]$ için VE kapı sonuçları

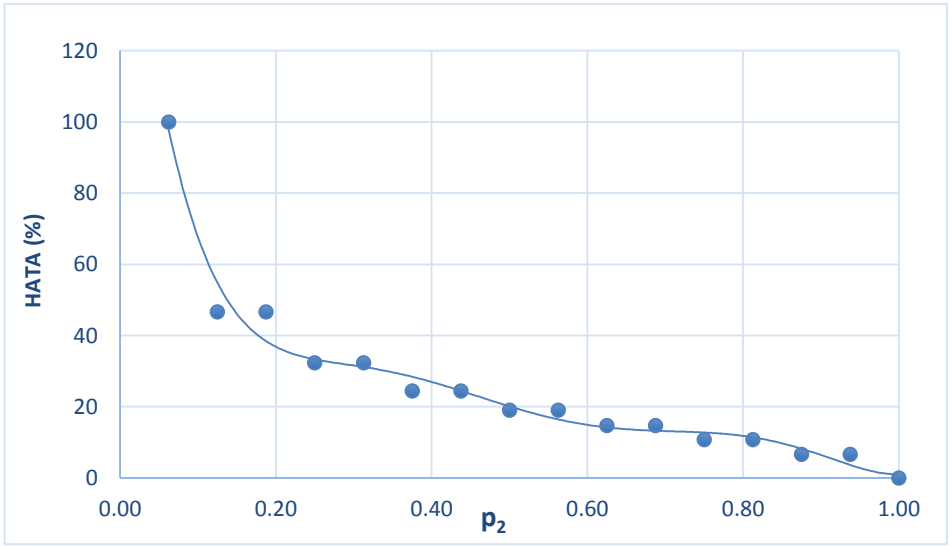
Bu analizde bir girişi sabit tutarak, ikinci giriş [0-1] aralığında değişirken çeşitli bit uzunlukları için VE lojik kapısı için Şekil 2.2, Şekil 2.3, Şekil 2.4 ve Şekil 2.5'teki grafikler elde edilmiştir.



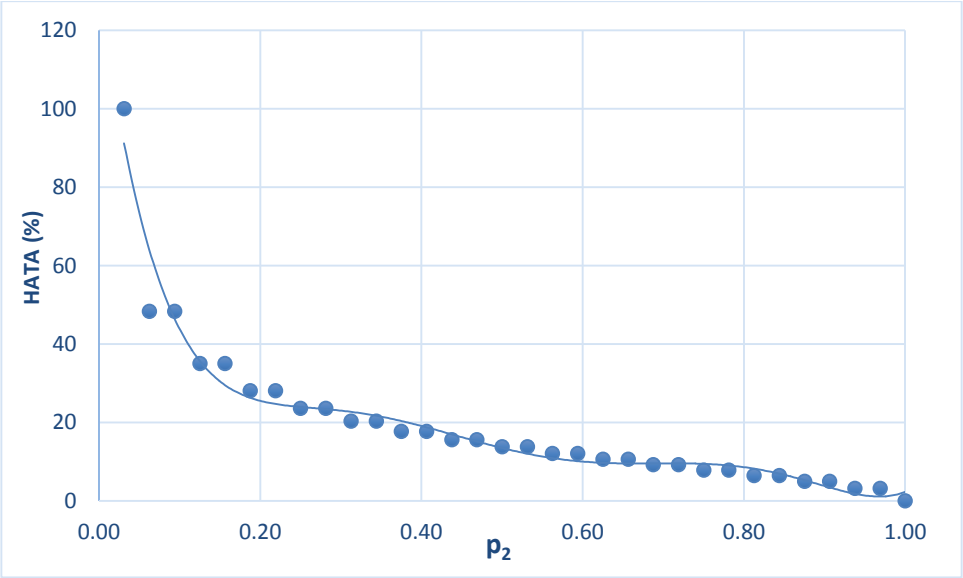
Şekil 2.2 : 4 bit için VE kapısı hata-p_2 grafiği.



Şekil 2.3 : 8 bit için VE kapısı hata-p_2 grafiği.



Şekil 2.4 : 16 bit için VE kapısı hata-p_2 grafiği.



Şekil 2.5 : 32 bit için VE kapısı hata-p_2 grafiği.

Sonuçlar VE Kapısı için rastgele bit karıştırma yöntemi ile elde edilmiştir. İlk dizinin olasılık değerini sabit tutup, ikinci diziyi [0,1] aralığında değiştirerek elde edilen grafikler ilk sonuçlarla paralellik göstermektedir. Bit uzunluğu arttıkça hata oranı azalmaktadır. Olasılığı değişen ikinci dizi p=1/2'den daha büyük olduğu zaman hata oranları düşük çıkmaktadır.

2.3.3 Giriş bit dizisi p_1=[0,1], p_2=[0,1] için VE hata denklemi

Denklemleri elde etmek için Nicholas Constantine Metropolis tarafından bulunan, modern versiyonu Stanislaw Ulam tarafından günümüze ulaştırılan Monte Carlo analizinden [15] faydalandık. Bu analiz, problemlerin çözümünü rastgele sayılar kullanarak elde etmeye verilen genel bir isimdir.

Bu denklemi çıkarmaktaki amaç; 1000 çevrimlik Monte Carlo analizi yapmadan, [0,1] aralığındaki olasılık giriş değerlerini temsil eden herhangi bir bit dizisi uzunluğu için hata oranlarını hesaplamaktır.

Oluşturmaya çalıştığımız VE kapısı hata denklemi için ilk olarak bir olasılık giriş değeri alınıp, diğer girişin [0,1] aralığında alabileceği olasılık değeri değiştirilerek, tüm olası çıkış değerlerinin katsayıları Çizelge 2.2, Çizelge 2.3 ve Çizelge 2.4'teki gibi bulundu.

Çizelge 2.2 : $p_1 = 6/8$ için katsayılar tablosu.

(p_1) - (p_2)	a_0=0/8	a_1=1/8	a_2=2/8	a_3=3/8	a_4=4/8	a_5=5/8	a_6=6/8
(6/8) - (0/8)	1	0	0	0	0	0	0
(6/8) - (1/8)	2	6	0	0	0	0	0
(6/8) - (2/8)	1	12	15	0	0	0	0
(6/8) - (3/8)	0	6	30	20	0	0	0
(6/8) - (4/8)	0	0	15	40	15	0	0
(6/8) - (5/8)	0	0	0	20	30	6	0
(6/8) - (6/8)	0	0	0	0	15	12	1
(6/8) - (7/8)	0	0	0	0	0	6	2
(6/8) - (8/8)	0	0	0	0	0	0	1

Çizelge 2.3 : $p_1 = 3/8$ için katsayılar tablosu.

(p_1) - (p_2)	a_0=0/8	a_1=1/8	a_2=2/8	a_3=3/8
(3/8) - (0/8)	1	0	0	0
(3/8) - (1/8)	5	3	0	0
(3/8) - (2/8)	10	15	3	0
(3/8) - (3/8)	10	30	15	1
(3/8) - (4/8)	5	30	30	5
(3/8) - (5/8)	1	15	30	10
(3/8) - (6/8)	0	3	15	10
(3/8) - (7/8)	0	0	3	5
(3/8) - (8/8)	0	0	0	1

Çizelge 2.4 : $p_1 = 1/4$ için katsayılar tablosu.

(p_1) - (p_2)	a_0=0/4	a_1=1/4
(1/4) - (0/4)	1	0
(1/4) - (1/4)	3	1
(1/4) - (2/4)	3	3
(1/4) - (3/4)	1	3
(1/4) - (4/4)	0	1

Elde edilen katsayılar tablosu gösteriyor ki, katsayılar Pascal üçgeninin versiyonu şeklindedir.

p_1=[0,1], p_2=[0,1] bit dizisi için VE hata denklemi; n bit dizisinin uzunluğu, p_1 ve p_2 giriş olasılık değerleri iken

$$Hata(x) = \frac{\sum_{i=\left\{\begin{matrix} 0, p_1+p_2 \le 1 \\ n.(p_1+p_2-1), p_1+p_2>1 \end{matrix}\right\}}^{n.\min(p_1,p_2)} \frac{\left|\frac{i}{n} - p_1 . p_2\right|}{p_1 . p_2} \cdot \binom{(1-p_1).n}{(n.p_2)-i} \cdot \binom{n.p_1}{i}}{\binom{n}{n.p_2}} \qquad \textbf{(2.2)}$$

şeklindedir. i={0,1, ..., n}'dir.

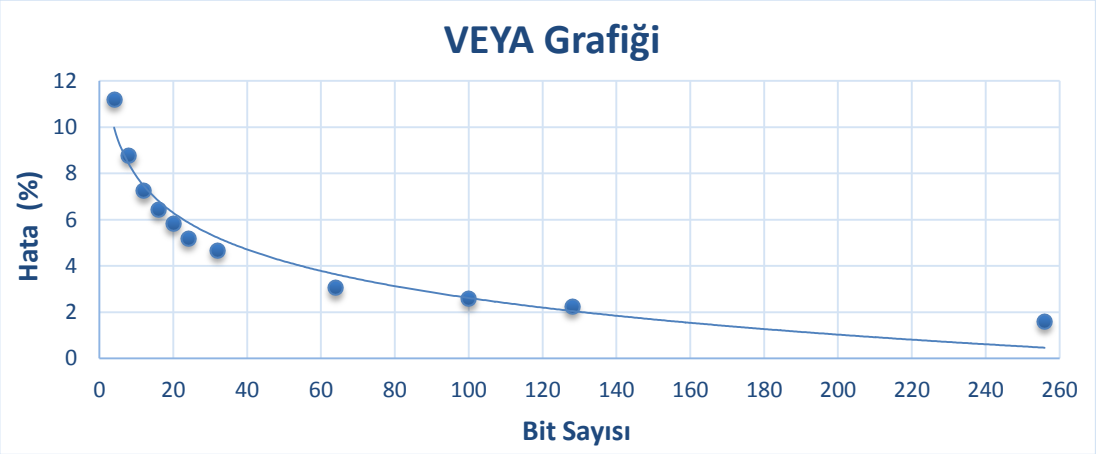
Elde edilen çıkışın gelme olasılığı ise

$$p_{\left(\frac{i}{n}\right)} = \frac{\binom{(1-p_1).n}{(n.p_2)-i}.\binom{n.p_1}{i}}{\binom{n}{n.p_2}} \qquad \textbf{(2.3)}$$

şeklindedir.

2.3.4 Giriş bit dizisi p_1=1/2, p_2=1/2 için VEYA kapı sonuçları

Bu analizde giriş olasılıkları sabit tutularak farklı bit uzunlukları ile VEYA lojik kapısı için Şekil 2.6'daki grafik elde edilmiştir.



Şekil 2.6 : VEYA kapısı hata-bit sayısı grafiği.

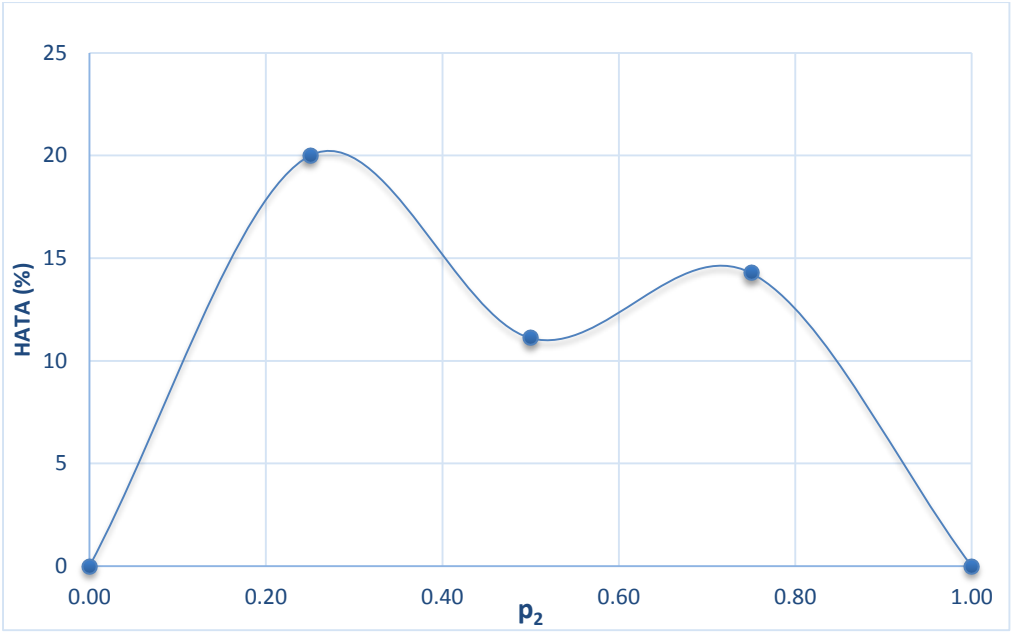
VEYA kapısı için RBKY ile elde edilen sonuçlar gösteriyor ki, bit dizisinin uzunluğu arttıkça hata oranı düşmektedir. 4 bitten sonrası için hata oranı %10'un altında olduğu için çalışılabilecek bit uzunluğu da bu noktadan sonrasıdır. VE kapısı ile aradaki fark VEYA için beklenen hatanın hesabından kaynaklanmaktadır.

p_1=p_2=1/2 ve x=bit dizisinin uzunluğu iken RBKY için VEYA kapısı genel hata denklemi çıkartıldı **(2.4)**. VE - VEYA arasında beklenen değerdeki 1/3'lük fark, formülden daha rahat görülmektedir.

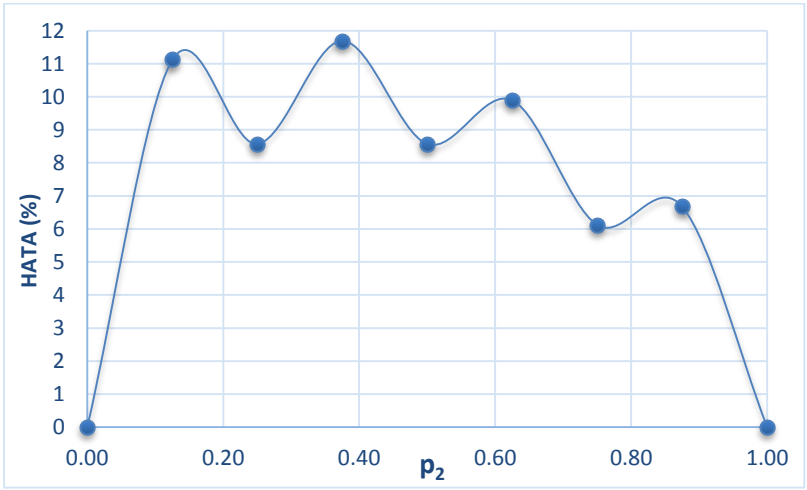
$$Hata(x) = \frac{2.\sum_{s=0}^{\frac{x}{4}-1}\binom{x/2}{s}.\binom{x/2}{\frac{x}{2}-s}.\binom{\frac{x}{4}-s}{3x/4}}{\binom{x}{x/2}} \qquad \textbf{(2.4)}$$

2.3.5 Giriş bit dizisi $p_1=1/2$, $p_2=[0,1]$ için VEYA kapı sonuçları

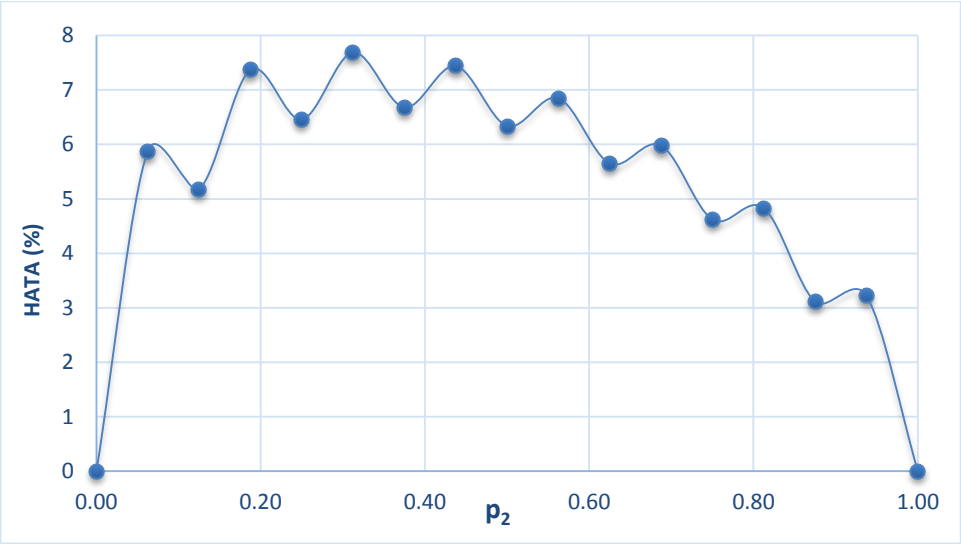
Bu analizde bir girişi sabit tutarak, ikinci giriş [0-1] aralığında değişirken çeşitli bit uzunlukları ile VEYA lojik kapısı için Şekil 2.7, Şekil 2.8, Şekil 2.9 ve Şekil 2.10'daki grafikler elde edilmiştir.



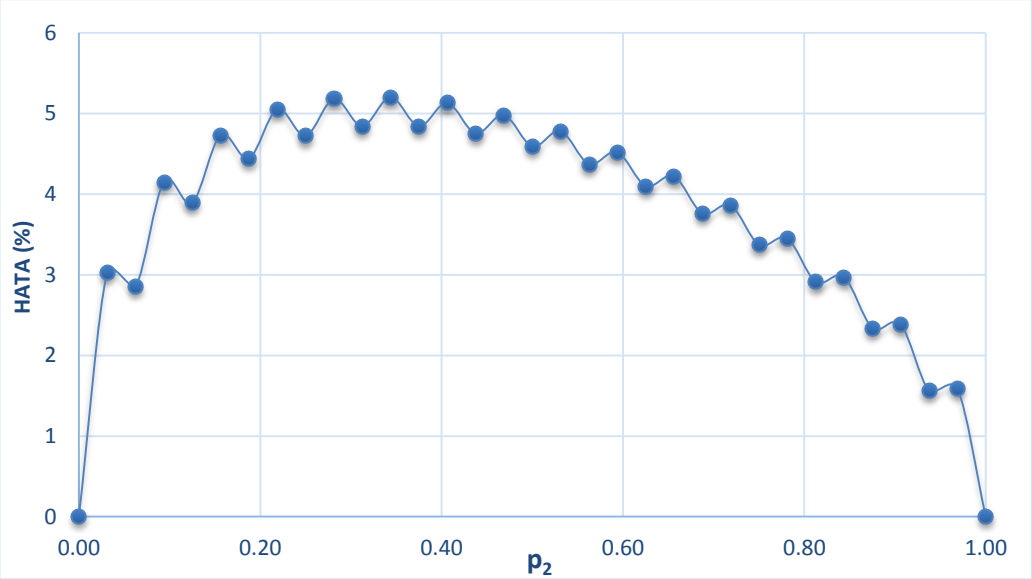
Şekil 2.7 : 4 bit için VEYA kapısı hata-p_2 grafiği.



Şekil 2.8 : 8 bit için VEYA kapısı hata-p_2 grafiği.



Şekil 2.9 : 16 bit için VEYA kapısı hata-p_2 grafiği.



Şekil 2.10 : 32 bit için VEYA kapısı hata-p_2 grafiği.

Sonuçlar VEYA kapısı için RBKY ile elde edilmiştir. İlk dizi olasılığını sabit tutup, ikinci diziyi [0,1] aralığında değiştirerek elde edilen grafikler ilk sonuçlarla paralellik göstermektedir. Bit uzunluğu arttıkça hata oranı azalmaktadır. Ancak VE grafiğinden farklı olarak ikinci dizinin olasılığının arttırılması üstel bir azalış vermemektedir, grafik yarım çember şeklini almaktadır.

2.3.6 Giriş bit dizisi p_1=[0,1], p_2=[0,1] için VEYA hata denklemi

Denklemleri elde etmek için Monte Carlo analizinden [15] faydalandık. Bu analiz, problemlerin çözümünü rastgele sayılar kullanarak elde etmeye verilen genel bir isimdir.

Bu denklemi çıkarmaktaki amaç 1000 çevrimlik Monte Carlo analizi yapmadan [0,1] aralığındaki olasılık giriş değerleri için herhangi bit dizisi uzunluğu için hata oranlarını hesaplamaktır.

VEYA kapısı için oluşturmaya çalıştığımız hata denklemi için ilk olarak bir olasılık giriş değeri alınıp diğer giriş [0,1] aralığında alabileceği olasılık değeri değiştirilerek tüm olası çıkış değerlerinin katsayıları Çizelge 2.5 ve Çizelge 2.6'daki gibi bulundu.

Çizelge 2.5 : $p_1 = 6/8$ için katsayılar tablosu.

(p_1) - (p_2)	$a_6=6/8$	$a_7=7/8$	$a_8=8/8$
(6/8) - (0/8)	1	0	0
(6/8) - (1/8)	6	2	0
(6/8) - (2/8)	15	12	1
(6/8) - (3/8)	20	30	6
(6/8) - (4/8)	15	40	15
(6/8) - (5/8)	6	30	20
(6/8) - (6/8)	1	12	15
(6/8) - (7/8)	0	2	6
(6/8) - (8/8)	0	0	1

Çizelge 2.6 : $p_1 = 2/4$ için katsayılar tablosu.

(p_1) - (p_2)	$a_2=2/4$	$a_3=3/4$	$a_4=4/4$
(2/4) - (0/4)	1	0	0
(2/4) - (1/4)	2	2	0
(2/4) - (2/4)	1	4	1
(2/4) - (3/4)	0	2	2
(2/4) - (4/4)	0	0	1

Elde edilen katsayılar tablosu gösteriyor ki, katsayılar Pascal üçgeninin versiyonu şeklindedir.

p_1=[0,1], p_2=[0,1] bit dizisi için VEYA hata denklemi; n bit dizisinin uzunluğu, p_1 ve p_2 giriş olasılık değerleri iken

$$\frac{\sum_{i=n.\max(p_1,p_2)}^{n.\min(p_1+p_2,1)} \frac{\left|\frac{i}{n}-\left(1-\left[(1-p_1).(1-p_2)\right]\right)\right|}{1-\left[(1-p_1).(1-p_2)\right]} \cdot \binom{n.p_1}{i-(n.p_2)} \cdot \binom{n.(1-p_1)}{i-(n.p_1)}}{\binom{n}{n.p_2}} \qquad \textbf{(2.5)}$$

şeklindedir. i={0,1, ..., n}'dir.

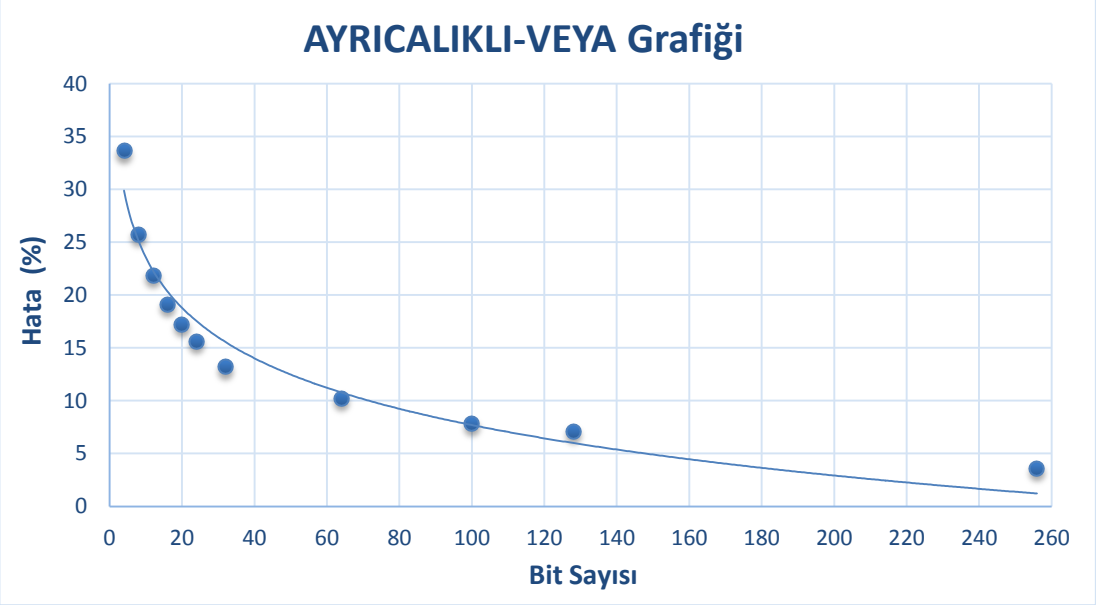
Elde edilen çıkışın gelme olasılığı ise

$$p_{\left(\frac{i}{n}\right)} = \frac{\binom{n.p_1}{i-(n.p_2)} \cdot \binom{n.(1-p_1)}{i-(n.p_1)}}{\binom{n}{n.p_2}} \qquad \textbf{(2.6)}$$

şeklindedir.

2.3.7 Giriş bit dizisi p_1=1/2, p_2=1/2 için A. VEYA kapı sonuçları

Bu analizde giriş olasılıkları sabit tutularak farklı bit uzunlukları için A. VEYA lojik kapısı için Şekil 2.11'deki grafik elde edilmiştir.



Şekil 2.11 : A. VEYA kapısı hata-bit sayısı grafiği.

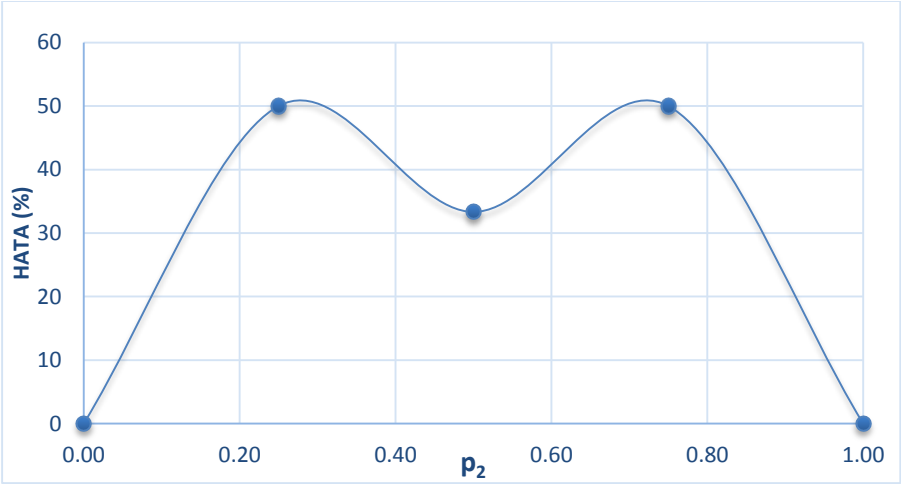
A. VEYA kapısı için RBKY ile elde edilen sonuçlar gösteriyor ki, bit dizisinin uzunluğu arttıkça hata oranı düşmektedir. 64 bitten sonrası için hata oranı %10'un altında olduğu için çalışılabilecek bit uzunluğu da bu noktadan sonrasıdır. Hata oranları VE kapısıyla paralellik göstermektedir.

p_1=p_2=1/2 ve x=bit dizisinin uzunluğu iken RBKY için A. VEYA kapısı genel hata denklemini çıkarttık **(2.1)**.

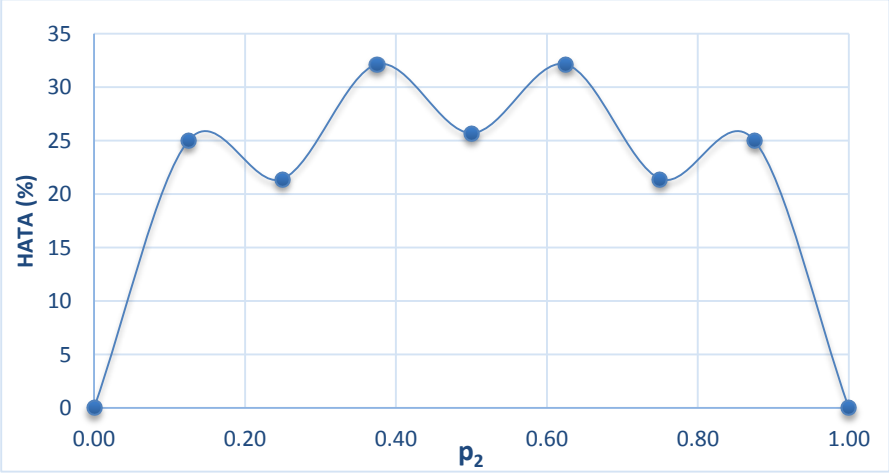
$$Hata(x) = \frac{2.\sum_{s=0}^{\frac{x}{4}-1} \binom{x/2}{s} \cdot \binom{x/2}{\frac{x}{2}-s} \cdot \binom{\frac{x}{4}-s}{x/4}}{\binom{x}{x/2}} \quad \textbf{(2.7)}$$

2.3.8 Giriş bit dizisi p_1=1/2, p_2=[0,1] için A. VEYA kapı sonuçları

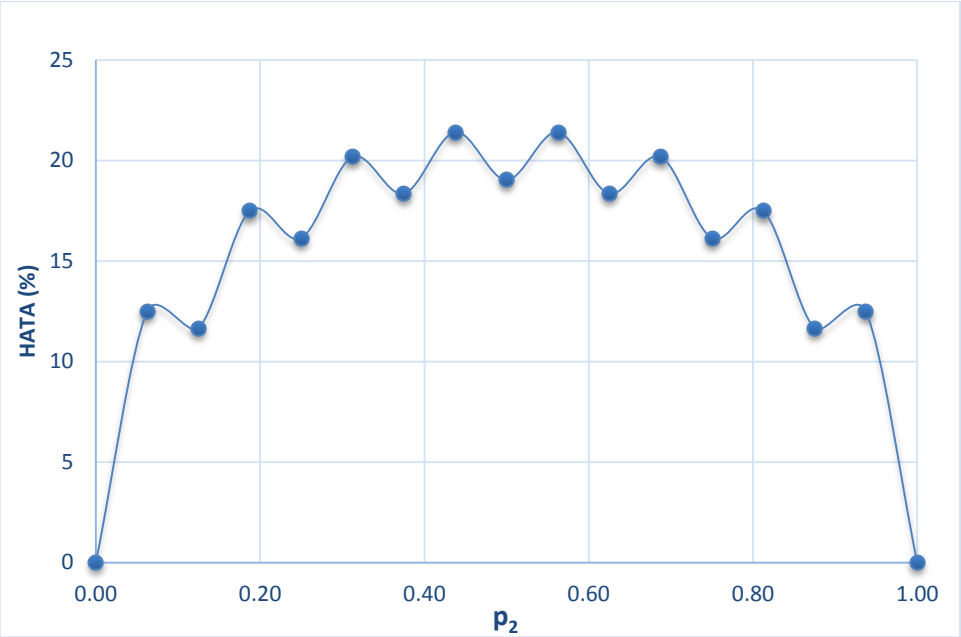
Bu analizde girişlerden biri sabit tutularak, ikinci giriş [0-1] aralığında değişirken çeşitli bit uzunlukları için A. VEYA lojik kapısı için Şekil 2.12, Şekil 2.13, Şekil 2.14 ve Şekil 2.15'teki grafikler elde edilmiştir.



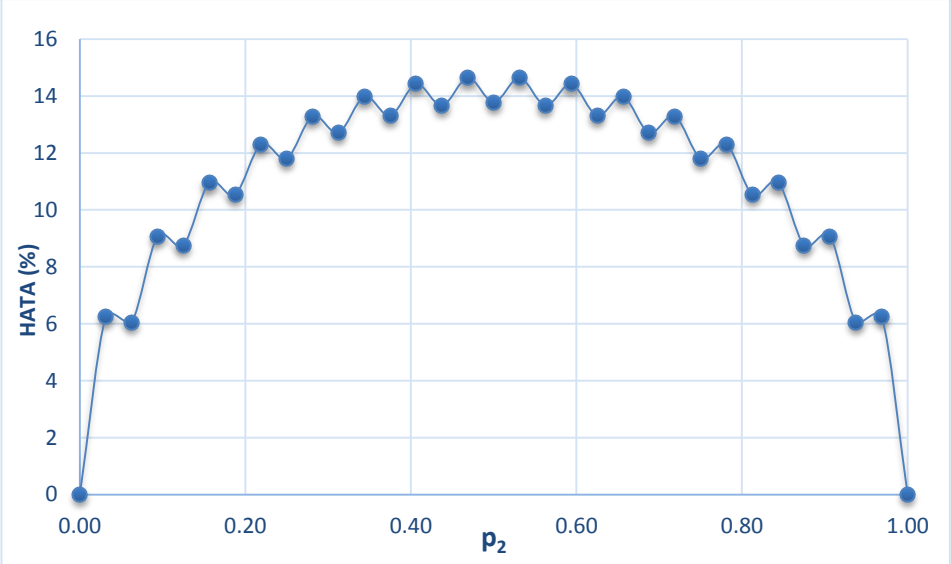
Şekil 2.12 : 4 bit için A. VEYA kapısı hata-p_2 grafiği.



Şekil 2.13 : 8 bit için A. VEYA kapısı hata-p_2 grafiği.



Şekil 2.14 : 16 bit için A. VEYA kapısı hata-p_2 grafiği.



Şekil 2.15 : 32 bit için A. VEYA kapısı hata-p_2 grafiği.

Sonuçlar A. VEYA Kapısı için RBKY ile elde edilmiştir. İlk dizi olasılığını sabit tutup, ikinci diziyi [0,1] aralığında değiştirerek elde edilen grafikler ilk sonuçlarla paralellik göstermektedir. Bit uzunluğu arttıkça hata oranı azalmaktadır. Ancak VE grafiğinden farklı olarak ikinci dizinin olasılığının arttırılması üstel bir azalış vermemektedir, grafik yarım çember şeklini almaktadır.

2.3.9 Giriş bit dizisi p_1=[0,1], p_2=[0,1] için A. VEYA hata denklemi

Denklemleri elde etmek için Monte Carlo analizinden [15] faydalandık. Bu analiz, problemlerin çözümünü rastgele sayılar kullanarak elde etmeye verilen genel bir isimdir.

Bu denklemi çıkarmaktaki amaç; 1000 çevrimlik Monte Carlo analizi yapmadan, [0,1] aralığındaki olasılık giriş değerleri ile herhangi bit dizisi uzunluğu için hata oranlarını hesaplamaktır.

A. VEYA kapısı için oluşturmaya çalıştığımız hata denklemi için ilk olarak bir olasılık giriş değeri alınıp, diğer giriş [0,1] aralığında alabileceği olasılık değeri değiştirilerek tüm olası çıkış değerlerinin katsayıları Çizelge 2.7 ve Çizelge 2.8'deki gibi bulundu.

Çizelge 2.7 : p_1 = 6/8 için katsayılar tablosu.

(p_1) - (p_2)	a_0=0/8	a_1=1/8	a_2=2/8	a_3=3/8	a_4=4/8	a_5=5/8	a_6=6/8	a_7=7/8	a_8=8/8
(6/8) - (0/8)	0	0	0	0	0	0	1	0	0
(6/8) - (1/8)	0	0	0	0	0	6	0	2	0
(6/8) - (2/8)	0	0	0	0	15	0	12	0	1
(6/8) - (3/8)	0	0	0	20	0	30	0	6	0
(6/8) - (4/8)	0	0	15	0	40	0	15	0	0
(6/8) - (5/8)	0	6	0	30	0	20	0	0	0
(6/8) - (6/8)	1	0	12	0	15	0	0	0	0
(6/8) - (7/8)	0	2	0	6	0	0	0	0	0
(6/8) - (8/8)	0	0	1	0	0	0	0	0	0

Çizelge 2.8 : p_1 = 2/4 için katsayılar tablosu.

(p_1) - (p_2)	a_0=0/4	a_1=1/4	a_2=2/4	a_3=3/4	a_4=4/4
(2/4) - (0/4)	0	0	1	0	0
(2/4) - (1/4)	0	2	0	2	0
(2/4) - (2/4)	1	0	4	0	1
(2/4) - (3/4)	0	2	0	2	0
(2/4) - (4/4)	0	0	1	0	0

Elde edilen katsayılar tablosu gösteriyor ki, katsayılar Pascal üçgeninin versiyonu şeklindedir.

p_1=[0,1], p_2=[0,1] bit dizisi için A. VEYA hata denklemi; n bit dizisinin uzunluğu, p_1 ve p_2 giriş olasılık değerleri ve $n * |p_2 - p_1|$ çift iken

$$\frac{\sum_{i=\frac{n.|p_2-p_1|}{2}}^{\frac{n.\min(p_1+p_2,1-p_1+1-p_2)}{2}} \frac{\left|\frac{2i}{n} - (p_1.(1-p_2) + p_2.(1-p_1))\right|}{p_1.(1-p_2) + p_2.(1-p_1)} \cdot \binom{n.p_1}{\left(\frac{p_1+p_2}{2}\right).n-i} \cdot \binom{(1-p_1).n}{n.p_2-\left[\left(\frac{p_1+p_2}{2}\right).n-i\right]}}{\binom{n}{n.p_2}} \quad \textbf{(2.8)}$$

şeklindedir. i={0,1, ..., n}'dir.

Elde edilen çıkışın gelme olasılığı ise

$$p_{\left(\frac{i}{n}\right)} = \frac{\left(\begin{matrix} n.\,p_1 \\ \left(\frac{p_1+p_2}{2}\right).n-i \end{matrix}\right).\left(\begin{matrix} n.(1-p_1) \\ (n.\,p_2)-\left[\left(\frac{p_1+p_2}{2}\right).n-i\right] \end{matrix}\right)}{\binom{n}{n.\,p_2}} \tag{2.9}$$

şeklindedir.

p_1=[0,1], p_2=[0,1] bit dizisi için A. VEYA hata denklemi; n bit dizisinin uzunluğu, p_1 ve p_2 giriş olasılık değerleri ve $n * |p_2 - p_1|$ tek iken

$$\frac{\sum_{i=\frac{(n.|p_2-p_1|)+1}{2}}^{\frac{[n.\min(p_1+p_2,1-p_1+1-p_2)]+1}{2}} \frac{\left|\frac{2i-1}{n}-(p_1.(1-p_2)+p_2.(1-p_1))\right|}{p_1.(1-p_2)+p_2.(1-p_1)} x \binom{n.p_1}{\frac{(p_1+p_2).n+1}{2}-i} x \binom{(1-p_1).n}{n.p_2-[\frac{(p_1+p_2).n+1}{2}-i]}}{\binom{n}{n.p_2}} \tag{2.10}$$

şeklindedir. i={0,1, ..., n}'dir. Elde edilen çıkışın gelme olasılığı ise

$$p_{\left(\frac{i}{n}\right)} = \frac{\left(\begin{matrix} n.\,p_1 \\ \frac{(p_1+p_2).n+1}{2}-i \end{matrix}\right).\left(\begin{matrix} n.(1-p_1) \\ (n.\,p_2)-\left[\frac{(p_1+p_2).n+1}{2}-i\right] \end{matrix}\right)}{\binom{n}{n.\,p_2}} \tag{2.11}$$

şeklindedir.

2.4 Deneysel Sonuçlar

RBKY için lojik kapılarla yapılan analizlerden sonra iki yöntem, benzer süreçlerle karşılaştırılmıştır.

2.4.1 Giriş bit dizisi p_1=1/2, p_2=1/2 için RBAY-RBKY karşılaştırma

İki yöntemi karşılaştırmak için ilk olarak, girişler ½ değerinde sabit tutularak, üç lojik kapı için değişen bit uzunlukları ile hata değerleri % olarak Çizelge 2.9'da elde edilmiştir.

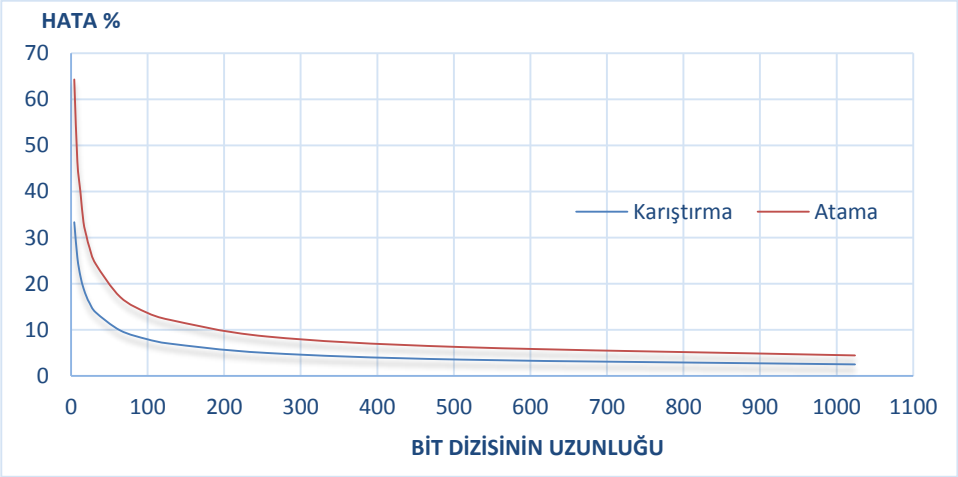
Çizelge 2.9'da görüldüğü üzere RBKY, VE ve VEYA kapıları için daha iyi sonuç vermiştir. A. VEYA kapısı birbirine oldukça yakın sonuçlar vermektedir. Bit uzunluğunun artışı beklendiği üzere hata oranlarını azaltmış, iki yöntem arasındaki

hata oranları arasındaki fark ta azalmıştır. Beklenen, sonsuzda iki yöntemin hata oranlarının oturmasıdır.

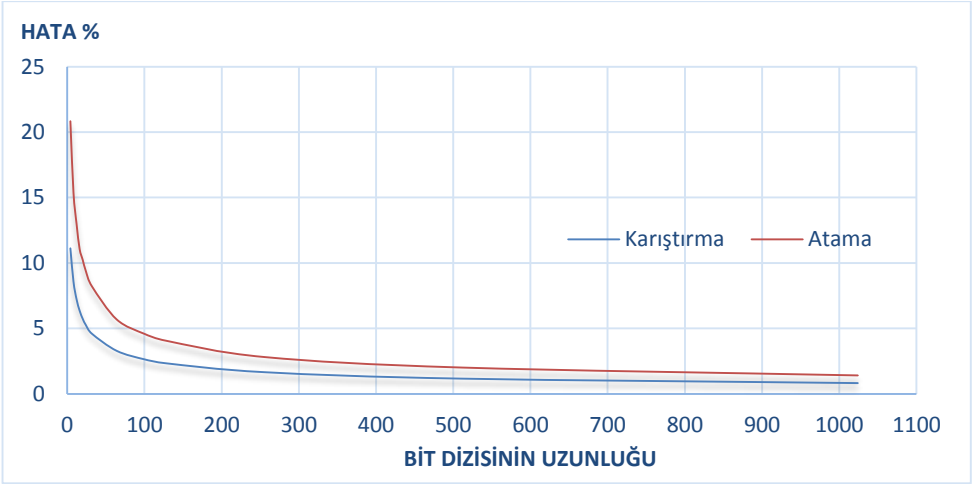
Çizelge 2.9 : Üç lojik kapı için değişen bit uzunları ile elde edilen hata değerleri (%).

BiT	RBAY			RBKY		
	VE	VEYA	A.VEYA	VE	VEYA	A.VEYA
4	64.3	20.82	37.55	33.33	11.11	33.33
8	46.58	15.38	27.1	25.71	8.57	25.71
12	39.92	12.98	22.52	21.64	7.22	21.64
16	33.38	11.05	19.4	19.04	6.35	19.04
20	30.28	10.22	17.68	17.19	5.73	17.19
24	27.75	9.38	16.35	15.79	5.26	15.79
32	24.28	8.2	14.02	13.78	4.59	13.78
64	17.1	5.68	9.65	9.86	3.29	9.86
100	13.62	4.58	7.88	7.92	2.64	7.92
128	12.12	4.05	7.08	7.01	2.34	7.01
256	8.52	2.8	4.98	4.97	1.66	4.97
512	6.22	2	3.55	3.52	1.17	3.52
1024	4.42	1.4	2.52	2.49	0.83	2.49

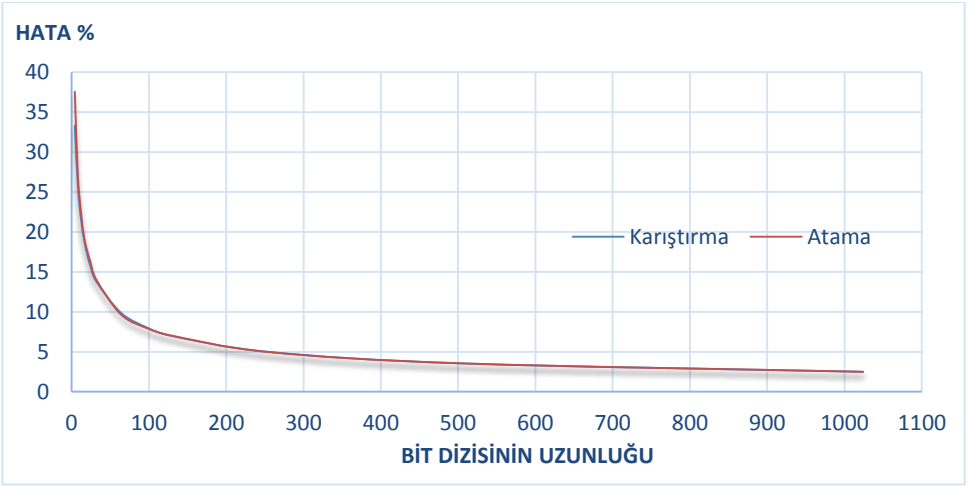
Şekil 2.16, Şekil 2.17 ve Şekil 2.18'deki grafiklerde iki yöntemin hata oranları aynı grafikte çizdirilerek, hata oranlarının daha kolay gözlemlenmesi sağlanmıştır. A. VEYA kapısı yakın hata oranları nedeniyle iki grafik çakışmaktadır.



Şekil 2.16 : VE kapısı için RBAY- RBKY bit uzunluğu-hata oranı grafiği.



Şekil 2.17 : VEYA kapısı için RBAY-RBKY bit uzunluğu-hata oranı grafiği.

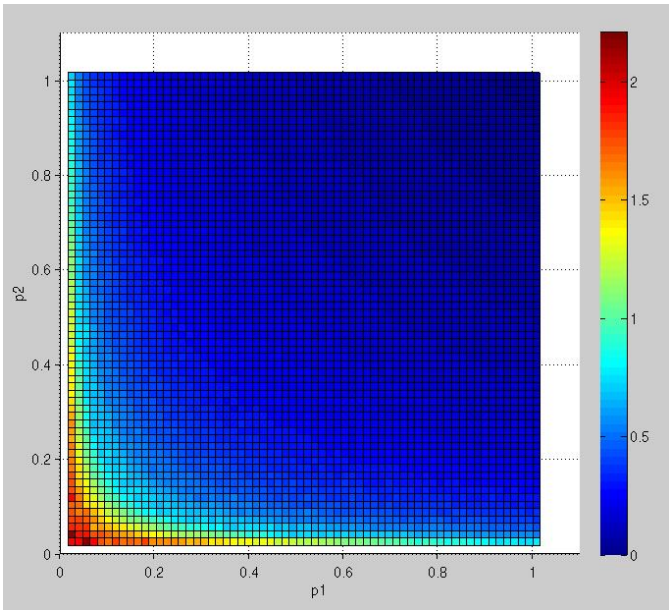


Şekil 2.18 : A. VEYA için RBAY-RBKY bit uzunluğu-hata oranı grafiği.

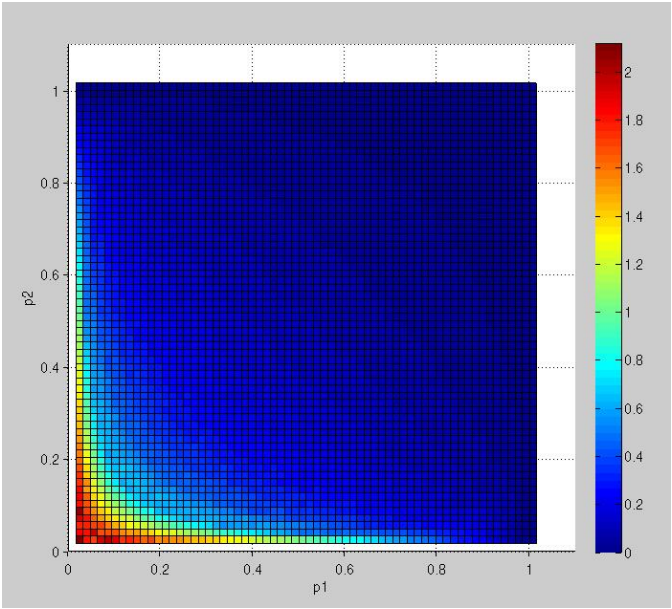
Yöntem karşılaştırma işleminde bir sonraki adım olarak olasılık giriş değerleri [0,1] aralığında değişirken hata oranlarını gösteren 3 boyutlu grafikler elde edildi. Bu grafiklerin elde edilmesindeki birinci amaç; iki olasılık değeri verildiğinde grafikten hata oranını okumak, ikinci amaç ise kabul edilebilir hata oranı verildiğinde girişlerin çalışma aralıklarını bulabilmektir.

2.4.2 64 bitlik giriş dizileri p_1=(0,1], p_2=(0,1] iken VE kapı sonuçları

Sonuçları elde etmek için Şekil 2.19 ve Şekil 2.20'deki grafikler çizdirilmiştir.



Şekil 2.19 : RBAY VE kapısı p_2-p_1-hata oranı grafiği.

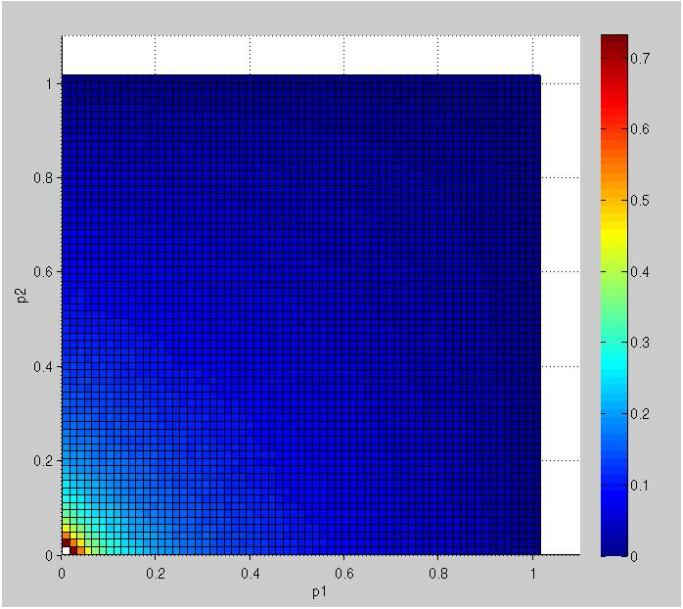


Şekil 2.20 : RBKY VE kapısı p_2-p_1-hata oranı grafiği.

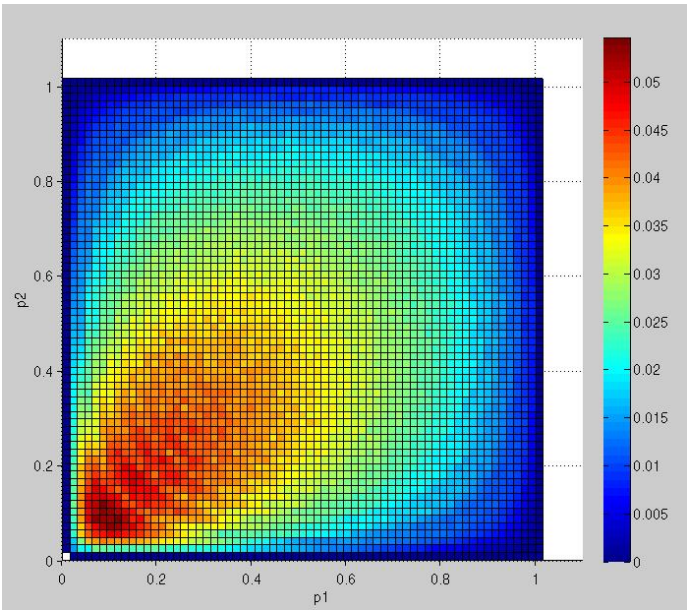
VE için p_1 veya p_2 değeri 0 olamaz. Çünkü paydada yer alacak beklenen değer de "0" olacağından hata hesaplanamaz. Grafik köşegene göre simetriktir. Çünkü girişlerin yer değiştirmesi sonucu değiştirmemektedir. VE için ideal çalışma aralığı iki girişinde 0,5 ve üzeri değerde olmasıdır. Karelerin temsil ettiği hata oranlarının ortalaması alındığında RBAY için ortalama hata oranı % 30.11, RBKY için ise %19.31'dir. RBKY için hata oranları bakımından sonuçlar daha iyidir.

2.4.3 64 bitlik giriş dizileri p_1=[0,1], p_2=(0,1] iken VEYA kapı sonuçları

Sonuçları elde etmek için Şekil 2.21 ve Şekil 2.22'deki grafikler çizdirilmiştir.



Şekil 2.21 : RBAY VEYA kapısı p_2-p_1-hata oranı grafiği.

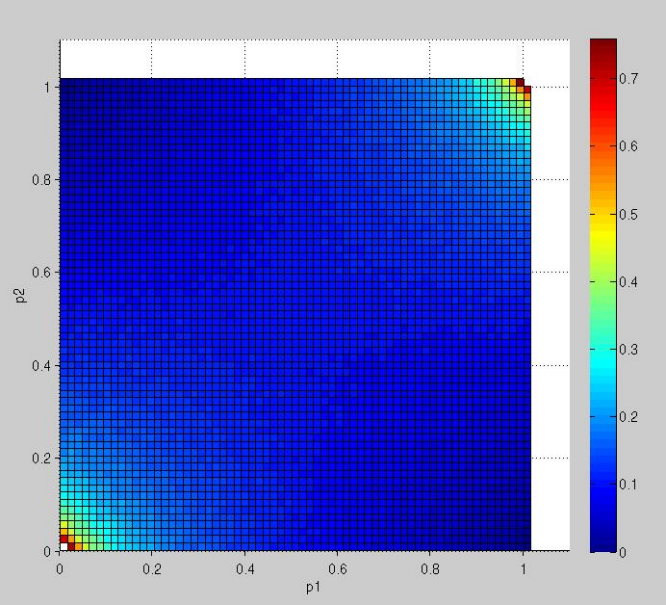


Şekil 2.22 : RBKY VEYA kapısı p_2-p_1-hata oranı grafiği.

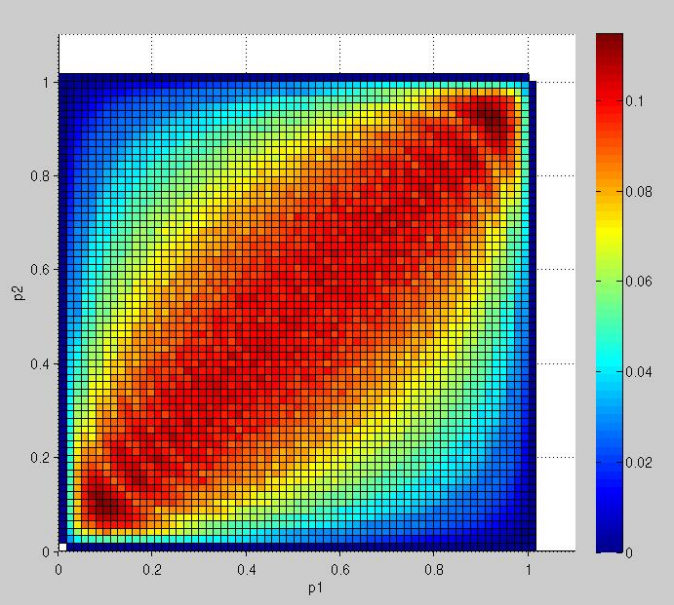
VEYA için p_1 ve p_2 değeri aynı anda 0 olamaz. Çünkü paydada yer alacak beklenen değer de 0 olacağından hata hesaplanamaz. Grafik köşegene göre simetriktir. Çünkü girişlerin yer değiştirmesi sonucu değiştirmemektedir. VE kapısına göre hata oranları oldukça düşüktür. VEYA için ideal çalışma aralığı iki girişinde 0,3 ve üzeri değerde olmasıdır. Karelerin temsil ettiği hata oranlarının ortalaması alındığında RBAY için ortalama hata oranı % 6.07, RBKY için ise %2.24'dir.

2.4.4 64 bitlik giriş dizileri p_1=[0,1], p_2=[0,1] iken A. VEYA kapı sonuçları

Sonuçları elde etmek için Şekil 2.23 ve Şekil 2.24'teki grafikler çizdirilmiştir



Şekil 2.23 : RBAY A. VEYA kapısı p_2-p_1-hata oranı grafiği.



Şekil 2.24 : RBKY A. VEYA kapısı p_2-p_1-hata oranı grafiği.

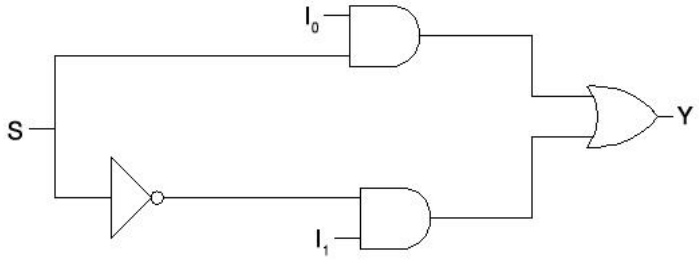
A. VEYA için p_1 ve p_2 değeri aynı anda 0 veya 1 olamaz. Çünkü paydada yer alacak beklenen değer de 0 olacağından hata hesaplanamaz. Grafik her iki köşegene göre de simetriktir. Çünkü girişlerin yer değiştirmesi ve girişlerden birisinin değilini alarak işlem yapılması da sonucu değiştirmemektedir. VE kapısına göre hata oranları düşük, VEYA kapısına göre daha yüksektir. A. VEYA için ideal çalışma aralığı iki girişinde 0,3 ve üzeri değerde olmasıdır, ancak iki giriş aynı anda köşegen üzerinde değerler

almamalıdır. Karelerin temsil ettiği hata oranlarının ortalaması alındığında RBAY için ortalama hata oranı % 10.94, RBKY için ise %6.58'dir.

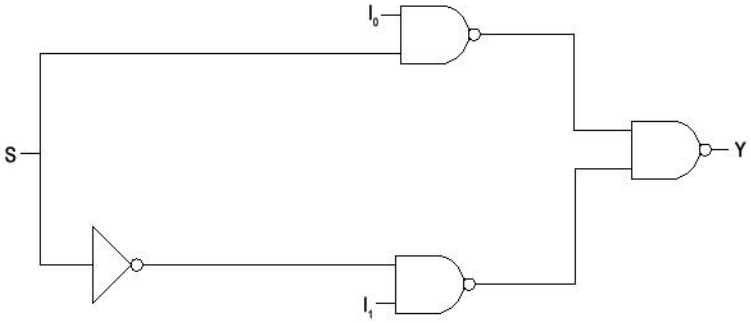
2.4.5 Çoklayıcı ile karşılaştırma

Yöntemler arasında karşılaştırmanın bir sonraki aşamasında lojik kapı uygulamalarından çoklayıcı kullanılmıştır. Çoklayıcı bir veya daha fazla giriş hattından ikili bilgiyi seçen ve bunu tek çıkış hattına bağlayan bir kombinezonsal devredir [16]. Stokastik hesaplamada ölçekli toplayıcı olarak kullanılmaktadır.

İki yöntem arasında karşılaştırma yaparken Şekil 2.25 ve Şekil 2.26'daki iki farklı çoklayıcı tasarımı için 64 bitlik diziler seçilmiştir. Girişlerin yer değiştirmesi sonucu değiştirmemektedir. Tüm karşılaştırmalarda yöntemin algoritmasına göre $p_S = 1/2$ alınmıştır.



Şekil 2.25 : Çoklayıcı devresi birinci tasarım [16].



Şekil 2.26 : Çoklayıcı devresi ikinci tasarım.

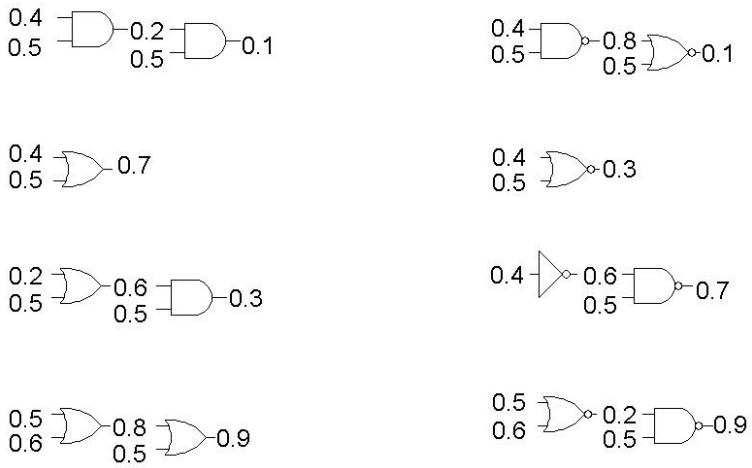
Çizelge 2.10'da görüldüğü üzere RBKY, RBAY'dan hata oranları bakımından daha iyi sonuçlar vermektedir. Sonuçlara bakıldığında iki farklı tasarım arasında, RBAY açısından büyük bir hata oranı farkı yoktur, ancak RBKY ile bazı girişler için ikinci tasarım daha iyi sonuçlar vermiştir.

Çizelge 2.10 : Çoklayıcı ile (giriş dizileri 64 bit) iki yöntem için hata oranları.

		Birinci Tasarım		İkinci Tasarım	
Girişler		RBKY	RBAY	RBKY	RBAY
$p_S = 0.5$	$p_1= 0.25, p_0= 0.25$	% 13.73	% 18.48	% 6.67	% 19.28
	$p_1= 0.25, p_0= 0.5$	% 12.08	% 16.09	% 9.09	% 15.65
	$p_1= 0.25, p_0= 0.75$	% 10.99	% 14.34	% 10.34	% 13.97
	$p_1= 0.5, p_0= 0.5$	% 15.06	% 16.4	% 14.29	% 16.65
	$p_1= 0.5, p_0= 0.75$	% 17.45	% 18.42	% 17.65	% 17.9
	$p_1= 0.75, p_0= 0.75$	% 23.04	% 23.65	% 23.08	% 23.04
$p_S = 0.25$	$p_1= 0.25, p_0= 0.25$	% 12.87	% 18.73	% 6.67	% 19.07
	$p_1= 0.25, p_0= 0.5$	% 27.51	% 27.88	% 27.27	% 28.47
	$p_1= 0.25, p_0= 0.75$	% 38.02	% 38.12	% 37.43	% 37.48
	$p_1= 0.5, p_0= 0.5$	% 14.39	% 16.6	% 14.29	% 17.12
	$p_1= 0.5, p_0= 0.75$	% 29.27	% 29.49	% 29.41	% 29.58
	$p_1= 0.75, p_0= 0.75$	% 23.07	% 23.27	% 23.08	% 23.07
$p_S = 0.75$	$p_1= 0.25, p_0= 0.25$	% 12.37	% 18.39	% 6.67	% 18.65
	$p_1= 0.25, p_0= 0.5$	% 11.38	% 15	% 11.17	% 15.81
	$p_1= 0.25, p_0= 0.75$	% 17.01	% 18.06	% 17.21	% 17.85
	$p_1= 0.5, p_0= 0.5$	% 14.78	% 17.29	% 14.29	% 16.59
	$p_1= 0.5, p_0= 0.75$	% 7.56	% 10.53	% 5.88	% 10.99
	$p_1= 0.75, p_0= 0.75$	% 23.09	% 23.12	% 23.08	% 23.09

2.4.6 Üretilen olasılık değerleri üzerinden karşılaştırma

İstenilen olasılık değeri farklı lojik kapılar ve kombinasyonları kullanılarak elde edilmeye çalışıldı. Bu çalışmadaki amaç ara bir süreçte lazım olabilecek bir olasılık değerini üretmeye gerek kalmadan çıkışlardaki olasılık değerleri ile Şekil 2.27'de görüldüğü gibi elde edebilmektir. 0.4 ve 0.5 olasılık değerlerinin elimizde olduğu varsayıldı [17].



Şekil 2.27 : Lojik kapılarla olasılıklar elde etme.

İki yöntem 100 ve 1000 bitlik giriş dizileri altında Şekil 2.28'deki iki farklı devreyle elde edildi.

Şekil 2.28 : Şekil 2.27'den seçilen iki devre.

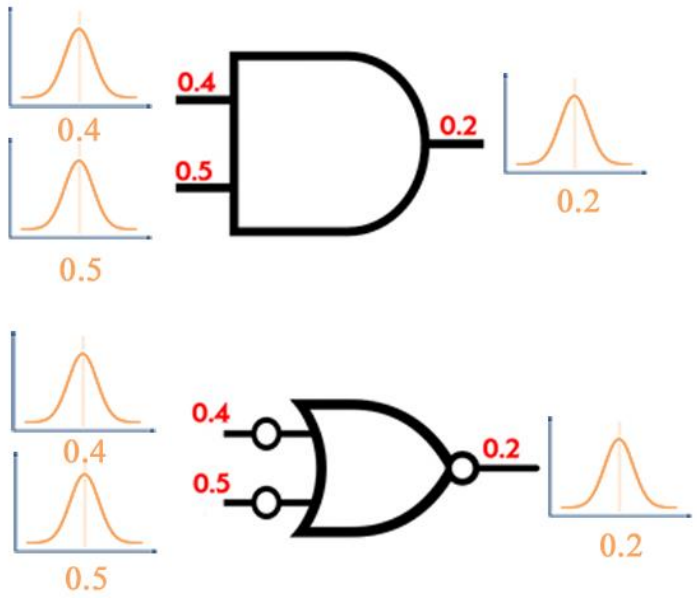
Çizelge 2.11'de görüldüğü üzere olasılık değerleri elde etmek için de kullandığımız iki yöntemden RBKY daha önce de olduğu gibi RBAY'dan hata oranları bakımından daha iyi sonuçlar vermiştir.

Çizelge 2.11 : RBKY-RBAY üretilen olasılık değerleri üzerinden karşılaştırma.

Giriş Bit	Devre No	RBAY	RBKY
100	1	% 24.58	% 18.15
	2	% 23.65	% 17.98
1000	1	% 7.58	% 5.54
	2	% 7.56	% 5.78

3. İYİLEŞTİRİLMİŞ DEVRE SİSTEMİ

Tezin ikinci kısmında hata oranlarını azaltmak için SH'de kullanılan devre sistemlerini iyileştirme çalışması yapıldı. İlk olarak RBKY ile üretilen giriş değerleri ve beklenen çıkış değeri sabit tutulup, devre yapısı değiştirilerek daha düşük hata oranları elde edilmeye çalışıldı. Bu çalışma sırasında farklı devre yapılarıyla istediğimiz daha düşük hata oranlarını elde edemedik. Girişler Gauss dağılımına sahipti ve kullandığımız devre yapısından bağımsız olarak çıkışlar da Şekil 3.1'deki gibi Gauss dağılımına sahip oluyordu. Bu yüzden istenen hata oranı değeri elde edilemiyordu.

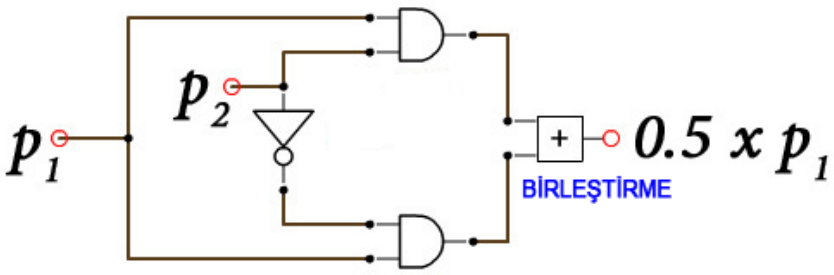


Şekil 3.1 : Girişler ve çıkışlar, kullanılan devre yapısından bağımsız olarak Gauss dağılımına sahiptir.

Yaptığımız araştırmalar gösteriyor ki, bağımsız giriş dizleriyle hatasız devre sistemi elde etmek mümkün değildir. Bu sonuca ulaştıktan sonra, bir sonraki aşama olarak bağımlı girişleri kullanarak hatasız devre sistemi elde etme üzerine çalıştık. Hatasız devre sistemi elde etmek için RBKY kullanılmak zorundadır. RBAY ile hatasız sonuçlar elde etmek mümkün değildir.

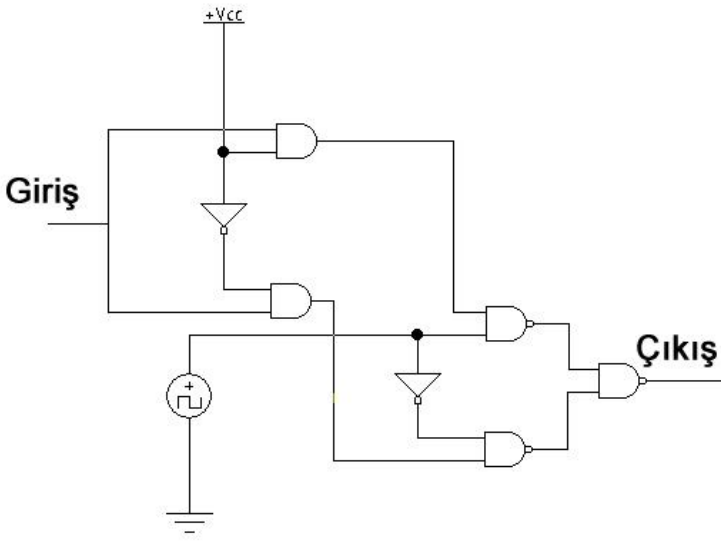
3.1 0.5 ile Hatasız Çarpma Yapan Devre Bloğu Gerçekleme

Bağımlılık kullanılarak gerçeklenen devre sistemlerinde, ilk olarak bir girişin 0.5 olduğu durumda hatasız çarpma bloğu gerçeklenmeye çalışıldı. Şekil 3.2'deki devre bu çarpma işlemi için önerildi.



Şekil 3.2 : 0.5 ile hatasız çarpma yapan devre bloğu.

Bu devre bloğu ile p_1 giriş dizisi 0.5 ile çarpılmaktadır ve çıkış sıfır hata ile elde edilmektedir. Ancak giriş dizisinin uzunluğu n ise, birleştirme işlemi nedeniyle çıkış dizisinin uzunluğu 2n olmaktadır. Şekil 3.2'de gösterilen birleştirme bloğu iki ayrı VE kapısından elde edilen dizileri tek bir dizi haline getirir. Bu bloğun tasarımı için bir adet çoklayıcı kullanılmıştır. Çoklayıcının seçme girişleri saat darbesine bağlanarak dizilerin sırayla çıkış hattına verilmesi sağlanmıştır. Saat darbesinin frekansı 2f ise, VE kapılarının girişine gelen bit dizilerinin frekansı f olmalıdır. Bu şekilde gerçeklemenin bir diğer dezavantajı bit dizilerinin frekansının çıkışta iki katına çıkmasıdır. Şekil 3.3'te gösterilen devrenin simülasyonu, Electronic Workbench 5.12 programı ile yapılmıştır.

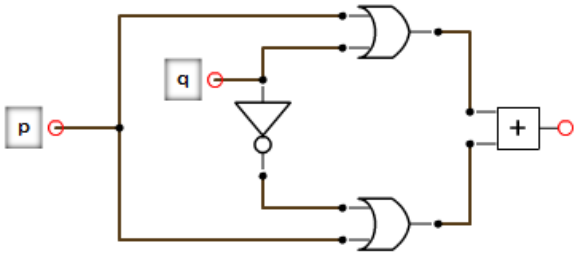


Şekil 3.3 : Birleştirme bloğunun gerçeklenmesi.

Aynı devre bloğu VEYA kapısı ile de hatasız çalışmaktadır (Şekil 3.4). VE kapılarının VEYA kapıları ile yer değiştirmesi yeterlidir. Çıkışta **(3.3)** olasılık değeri elde edilir.

$$0.5+0.5.\ p \tag{3.3}$$

VEYA kapıları olasılık değerleri elde etmek için Şekil 3.4'teki gibi kullanılabilir.



Şekil 3.4 : VEYA kapısı ile blok gerçekleme.

3.2 Herhangi Bir Olasılık Değerini Hatasız Üretme

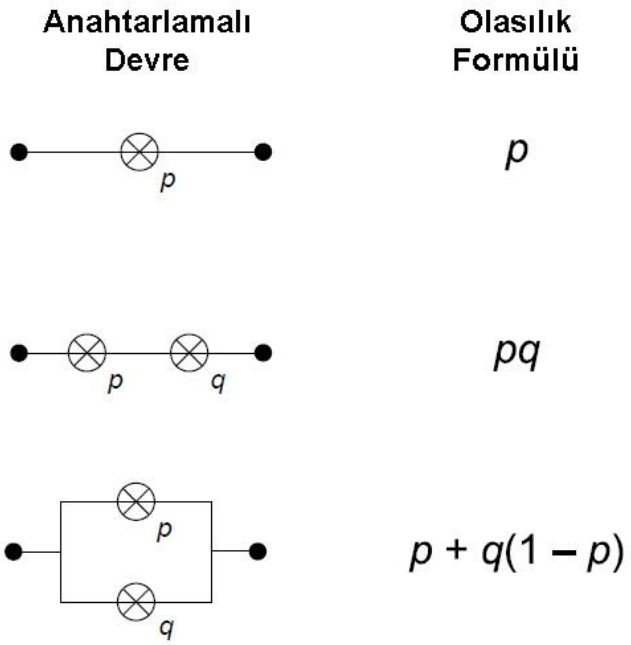
İstenilen bir olasılık değerini elde etmek için literatürde çeşitli yöntemler mevcuttur. Bunlardan birisi kombinezonsal devrelerle [17] diğeri de stokastik anahtarlamalı devrelerle [18] olasılık değerini üretmektir. Bağımlılık kullanılarak herhangi bir olasılık değerini hatasız elde etmek için gerçeklediğimiz hatasız sonuç veren blokları Stokastik Anahtarlamalı Devreler'e [18] uyarladık.

3.2.1 Stokastik anahtarlamalı devreler

Claude Shannon 1937'deki Yüksek Lisans Tezi'nde herhangi bir Boole fonksiyonunun anahtarlamalı röle devreleri ile gerçeklenebildiğini göstermiştir [19]. Klasik anahtarların herbiri olasılıksal anahtarlarla değiştirilerek stokastik anahtarlamalı devreler elde edilir [18]. İstenilen olasılık değerini temsil eden ikili tabanda herhangi bir basit kesir için optimal boyutta devre sentezleme algoritması sunulmuştur [18].

Olasılıksal anahtarların kapalı olma olasılığı sırasıyla p ve q olsun. Bu anahtarlardan bir tanesinin kullanılması durumunda anahtarlamalı devrenin çıkışında p veya q elde edilecektir. İki olasılıksal anahtar seri bağlandığında, devrenin çıkış vermesi için ikisininde kapalı olması gerekir. Bu durumda anahtarlamalı devrenin çıkışında p.q elde edilir. İki olasılıksal anahtar paralel bağlandığında, devrenin çıkış vermesi için anahtarlardan sadece birinin kapalı olması veya ikisininde aynı anda kapalı olması

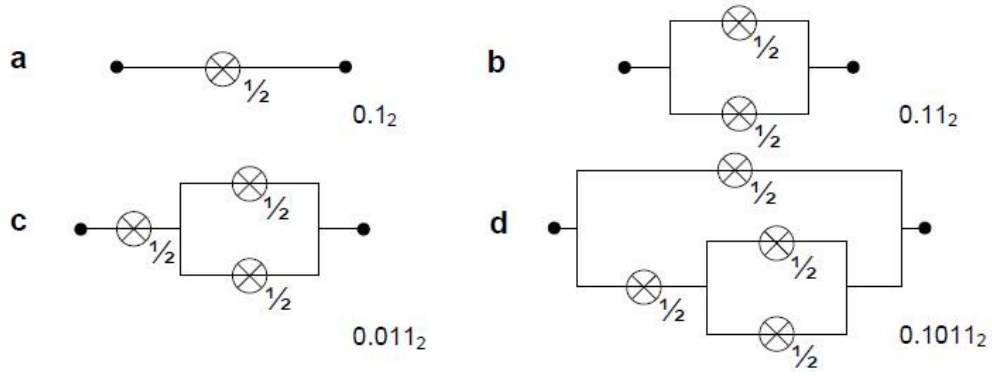
gerekir. Bu durumda anahtarlamalı devrenin çıkışında p+q.(1-p) elde edilir (bkz Şekil 3.5).



Şekil 3.5 : Olasılıksal anahtarların seri ve paralel bağlanması [18].

Seri ve paralel bağlama işlemleri birlikte yapılarak istenilen olasılık değeri elde edilebilmektedir. Kapalı olma olasılığı ½ olan anahtarları seri bağlamak ikili gösterimde olasılık değerinin noktadan sonraki kısmını sağa kaydırır ve başa 0 eklerken, paralel bağlamak noktadan sonraki kısmı sağa kaydırır ve başa 1 ekler.

11/16 = 0.1011 değerini gerçeklemek için kapalı olma olasılığı ½ olan anahtarlar kullanalım. İlk önce iki anahtar paralel bağlanmalı, sonrasında paralel bağlı anahtarlara seri bağlı bir anahtar eklenmeli ve en son olarak ta bu anahtarlara paralel bir anahtar bağlanmalıdır (bkz Şekil 3.6) [19].

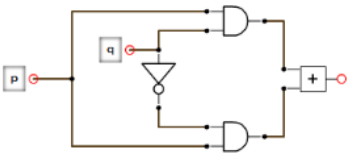
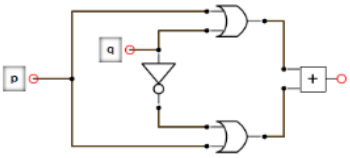


Şekil 3.6 : 11/16 = 0.1011 olasılık değerinin gerçeklenmesi [18].

3.2.2 Stokastik anahtarlamalı devre benzetimi

3.1'de gerçeklenen devre blokları ikili tabanda herhangi bir olasılık değerini elde etmek için Çizelge 3.1'de görüldüğü üzere stokastik anahtarlamalı devrelere uyarlandı.

Çizelge 3.1 : Stokastik anahtarlamalı devre benzetimi.

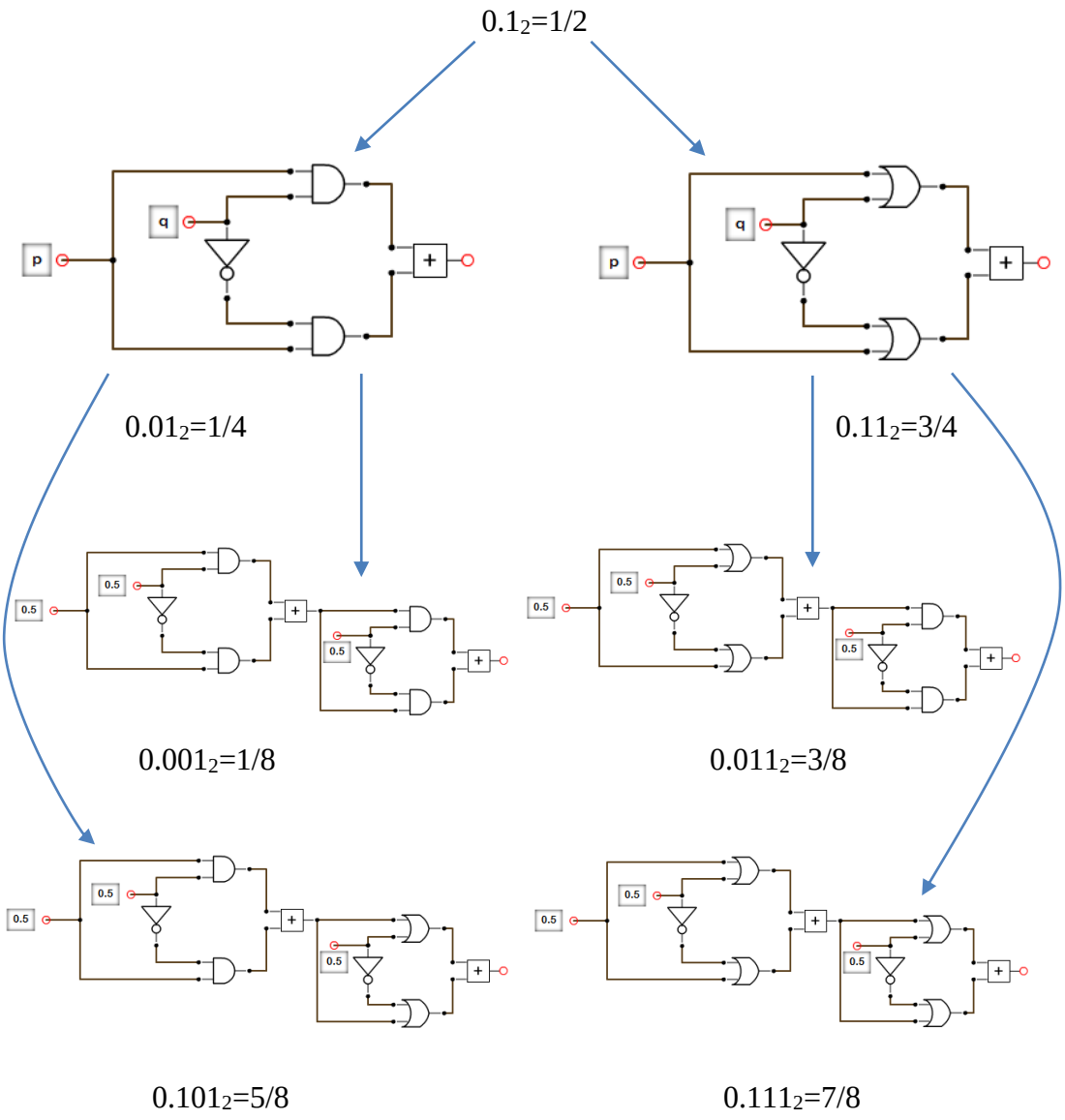
Olasılık Değeri	Lojik Devre
0.5p	
0.5+0.5p	

Girişler (p,q) 0.5 (0.1_2=1/2) alındı. İkili değeri bir basamak sağa kaydırıp, "0" eklemek için VE kapısı ile gerçeklenen blok kullanıldı (0.01_2=1/4). İkili değeri bir basamak sağa kaydırıp, "1" eklemek için VEYA kapısı ile gerçeklenen blok kullanıldı (0.11_2=3/4). Bloklar kaskat bağlanarak ikili tabanda istenen değer hatasız olarak Şekil 3.7'de gösterilen ağaçtaki gibi elde edilebilmektedir.

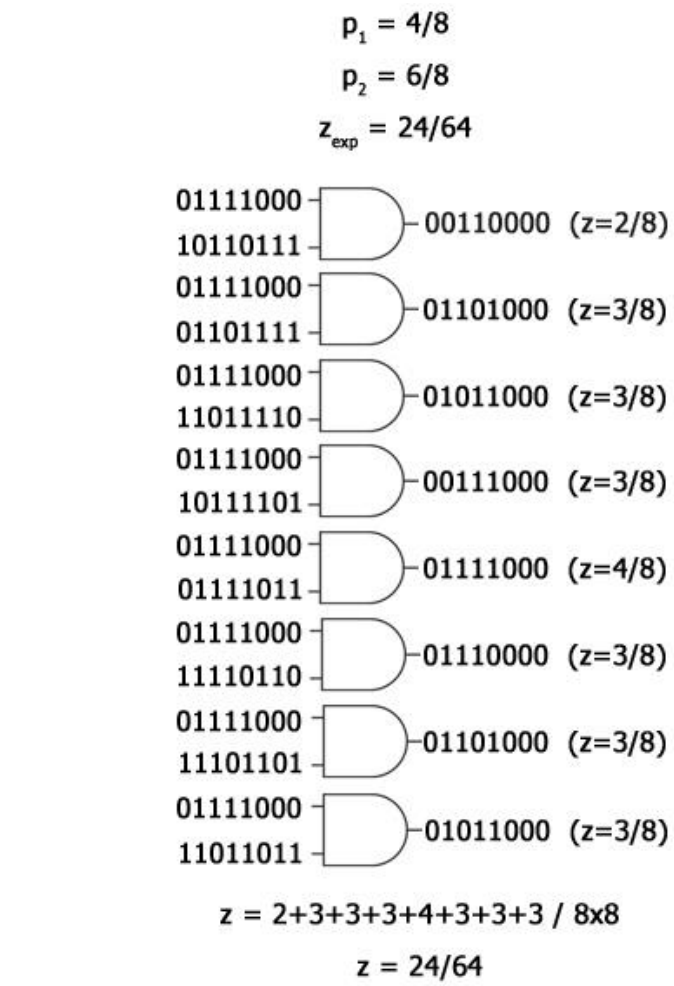
3.3 Bit Bit Kaydırma Yöntemi

0.5 ile hatasız çarpma yaptıktan sonraki hedefimiz, herhangi iki giriş değeri için hatasız çarpma yapmaktı. Herhangi iki giriş değeri ile hatasız çarpma yapmak için bit bit kaydırma yöntemini önerdik. Bu yöntemde girişlerden biri sabit tutulurken, diğer giriş bit bit sola kaydırılır. Giriş dizisinin uzunluğu n ise kaydırma işlemi n-1 kere uygulanır (Örneğin giriş 8 bit ise, kaydırma işlemi 7 kere uygulanır). Kaydırma işlemi uygulanan dizinin ilk konumundaki çarpma da dahil olmak üzere çıkıştan elde edilen bit dizileri birleştirilir.

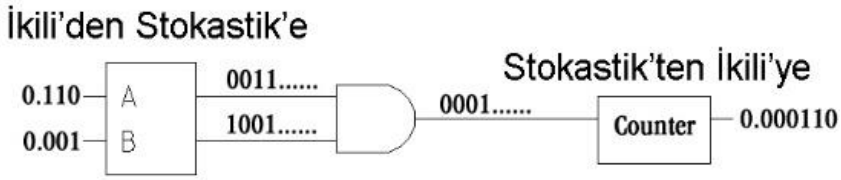
Bu yöntem "İkili'den Stokastiğe Dönüştürücü" yapısı değiştirilerek gerçeklenecektir. Bu sayede Şekil 3.9'daki gibi tek bir VE kapısı kullanılacaktır. Şekil 3.8'de yöntemi açıklamak amacıyla birden fazla VE kapısı kullanılmıştır.



Şekil 3.7 : Olasılık gerçekleme ağacı.

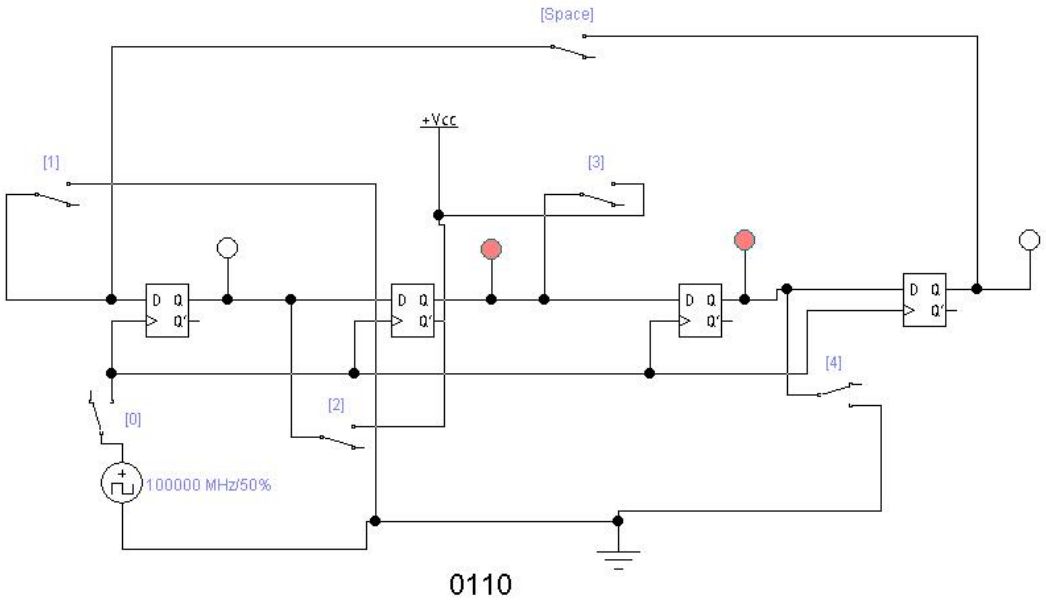


Şekil 3.8 : Bit bit kaydırma yönteminin algoritması.



Şekil 3.9 : Bit bit kaydırma yönteminin teorik uygulaması.

Rastgele bit üreteci için Lineer Geri Beslemeli Ötelemeli Kaydedici yapısı baz alınmıştır. Bu sayede bit bit kaydırarak dizi üretimi gerçekleştirilmiştir. Gerçekleme de ilk olarak anahtarlar yardımıyla D flip-flop'larına ilk değer gömülmüştür. 4 adet çıkış, sırası değişmeden her çevrim için giriş dizisi olarak tek bir hatta verilecektir. Şekil 3.10'da gösterilen devrenin simülasyonu, Electronic Workbench 5.12 programı ile yapılmıştır.

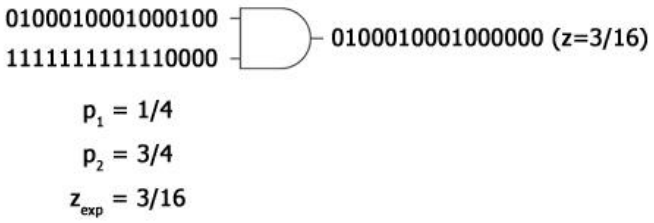


Şekil 3.10 : Bit bit kaydırma yönteminin gerçeklenmesi.

3.4 Dizi Klonlama Yöntemi

Herhangi iki giriş değeri ile hatasız çarpma yapmak için önerdiğimiz bir diğer yöntem dizi klonlamadır. Bu yöntemde ana fikir, girişlerden birisini klonlamaktır.

Girişler p_1=1/4 (0100), p_2=3/4 (1011) ve girişlerin bit uzunluğu n=4 olsun. Girişlerden biri bit uzunluğu kadar kopyalanır (p_1=1/4 (0100010001000100)). Diğer giriş p_2 baz alınarak $p_2.n^2$ adet 1 ve $(1-p_2).n^2$ adet 0 içerecek şekilde Şekil 3.11'deki gibi tekrar hazırlanır.



Şekil 3.11 : Dizi klonlama yönteminin algoritması.

Gerçekleme bit bit kaydırma yöntemiyle benzer şekilde yapılacaktır. Ana fikir bit üretecinin yapısını değiştirmektir.

Dizi klonlama ve bit bit karıştırma yöntemleri için bit üretecinin yapısını değiştirmek rastgeleliği bozmaktadır. Bu yöntemleri geliştirmekteki amacımız iki olasılık değerini daha az tranzistör kullanarak çarpmaktır.

4. SONUÇ VE ÖNERİLER

Bu tez için motivasyonumuz; düşük maliyetli tasarım yapılabilen SH'da [20] hata oranlarını azaltmak ve mümkünse hata oranını sıfır yapmaktı. İlk olarak giriş bitlerinin üretimini değiştirmeyi düşündük. Giriş bitlerinin üretimini değiştirmek amacıyla adını RBKY koyduğumuz yeni bir yöntem önerdik. Literatürde SH için kullanılan RBAY yöntemi ile önerdiğimiz yeni yöntemi karşılaştırdık. Analiz sonuçları önerdiğimiz yöntemin hata oranları bakımından daha iyi sonuçlar verdiğini gösterdi.

SH'da giriş bit dizileri bağımsızdır. Yaptığımız çalışmalar sonucunda bağımsız giriş dizileri ile farklı devre sistemlerinde sıfır hata oranı elde etmenin mümkün olmadığı sonucuna vardık. Bu nedenle devre sistemini iyileştirmek için bağımlılığı kullanmaya karar verdik.

Bağımlılığı kullanarak VE kapısını baz alan 0.5 ile hatasız çarpma yapan bir blok gerçekledik. Blokta kullanılan VE kapıları VEYA kapılarıyla değiştirildiğinde, blok 0.5 ile VEYA'lama işlemini de hatasız gerçekleştirebilmektedir. Bu bloğu, herhangi bir olasılık değerini elde etmek için stokastik anahtarlamalı devrelere uyarladık ve hatasız olarak olasılık değerleri elde ettik. Bu blok ile işlem yaparken hatasız sonuçlar elde etmek için RBKY kullanılmak zorundadır.

Herhangi iki giriş değerini hatasız çarpmak için bit bit kaydırma ve dizi klonlama yöntemini önerdik ve sağlamalarını yaptık. Bu yöntemleri ikili iki girişi çarpmak için kullanmak avantajlıdır. Ancak bu yöntemleri kaskat bağlı devre sisteminde kullanmak pratik değildir. Çünkü girişin uzunluğu ile çıkışta elde edilen bit dizisinin uzunluğu farklıdır.

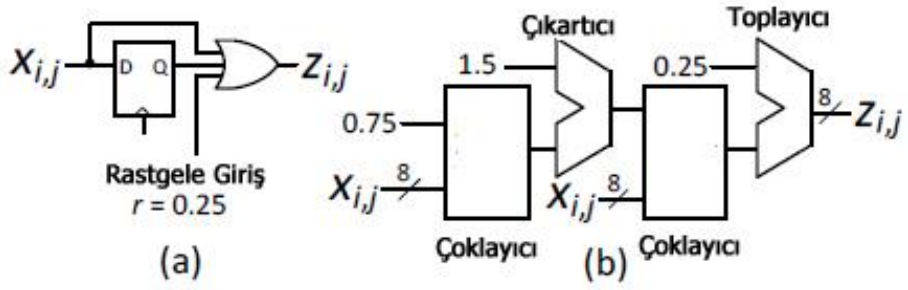
4.1 Çalışmanın Uygulama Alanları

Önerdiğimiz yöntemler/devreler "Gama Düzeltme Fonksiyonları (Gamma Correction Function) ve Düşük Yoğunluklu Parite Kontrolü (Low Density Parity Check) alanlarındaki fonksiyonlara/devrelere de uygulanabilirdir. Bu araştırma alanları bazı kısımlarında SH'yı kullanır.

Gama Düzeltme Fonksiyonu görüntü işleme alanında daha iyi sonuçlar elde etmek için kullanılan lineer olmayan bir işlemdir. Gama Düzeltme Fonksiyonu bir görüntünün her pikseline (**4.1**)'i [10] uygular.

$$z = x^{0.45} \tag{4.1}$$

Bu fonksiyon 6. dereceden Bernstein polinomuna yakınsar [5]. Çalışmamıza uygulanabilecek kısmı Şekil 4.1'de gösterilmiştir.

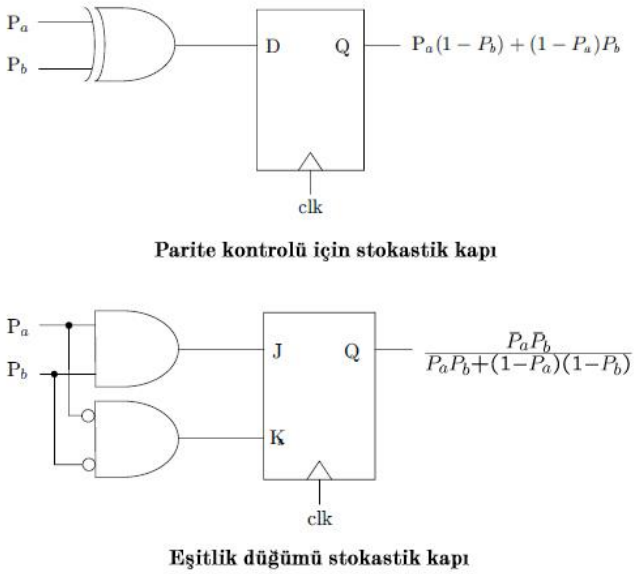


Şekil 4.1 : Gama düzelticiler: (a) Stokastik; (b) Geleneksel [6].

İki gerçeklemenin farkları Şekil 4.2'de gösterilmiştir.

Şekil 4.2 : İki yöntem arasında hata tolerans karşılaştırması [10].

Düşük yoğunluklu parite kontrol kodları Shannon kapasitesine yakaşabilen hata-düzeltme kodları sınıfındandır [7]. İlk olarak Robert Gallager tarafından literatüre sokulmuştur [21]. Günümüzde LDPC kodları WiFi [22] ve sayısal video yayıncılığı [23] gibi haberleşme standartlarında kullanılmaktadır. Çalışmamıza uygulanabilecek kısmı Şekil 4.3'te gösterilmiştir.



Şekil 4.3 : LDPC kodlarının çalışmamıza uygulanabilecek kısmı [24].

KAYNAKLAR

[1] **Gaines, B.R.** (1967). Stochastic computing. *AFIPS Spring Joint Computer Conf.*, 149-156.

[2] **von Neumann, J.** (1963). Probabilistic logics and the synthesis of reliable organisms from unreliable components, *The Collected Works of John von Neumann*, Macmillan.

[3] **Toral, S.L., Quero, J.M., ve Franquelo, L.G.** (2000). Stochastic pulse coded arithmetic, *ISCAS*, **1**, 599-602..

[4] **Mars, P., ve McLean, H.R.** (1976). High-speed matrix inversion by stochastic computer, *Electronics Letters*, **12**, 457-459.

[5] **Qian, W., Li, X., Riedel, M.D., Bazargan, K., ve Lilja, D.J.** (2011). An architecture for fault-tolerant computation with stochastic logic, *IEEE Trans. Computers*, **2**, 93-105.

[6] **Alaghi, A., ve Hayes, J.P.** (2011). Stochastic circuits for real-time image-processing applications, *Design Automation Conference*, paper 136.

[7] **Gross,W.J., Gaudet,V.C., ve Milner, A.** (2005). Stochastic Implementation of LDPC Decoders, *39th Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, California, USA, 8-11 Kasım.

[8] **Brown, B.D. ve Card, H.C.** (2001). Stochastic neural computation I: computational elements, *IEEE Trans. Comp.*, **50**, 891-905.

[9] **Qian, W., ve Riedel, M.D.** (2008). The synthesis of robust polynomial arithmetic with stochastic logic, *Design Automation Conference*, 648-653.

[10] **Alaghi, A., ve Hayes, J. P.** (2013). Survey of Stochastic Computing, *ACM Transactions on Embedded Computing Systems*, **12 (2s)**, 1-16.

[11] **Gupta, P.K., ve Kumaresan, R.** (1988). Binary multiplication with PN sequences, *IEEE Trans. Acoustics, Speech and Signal Processing*, **36**, 603-606.

[12] **Yavuz, S., ve Altun, M.,** (2014). Stokastik Hesaplamada Hata Oranlarını Azaltmak için Rastgele Bit Karıştırma Yöntemi, *Elektrik - Elektronik, Bilgisayar ve Biyomedikal Mühendisliği Sempozyumu (ELECO)*, Bursa, Türkiye, 27-29 Kasım.

[13] **Fisher, R.A., ve Yates, F.** (1948). Statistical tables for biological, agricultural and medical research (3. baskı). London, Oliver & Boyd.

[14] **Durstenfeld, R.** (1964). Algorithm 235: Random permutation, *Communications of the ACM.*, **7**, 420.

[15] **Metropolis, N., Ulam, S.,** (1949). The Monte Carlo method. *Journal of the American Statistical Association*, **44**, 335–341 doi:10.2307/2280232.

[16] **Mano, M.** (2003). Sayısal Tasarım. İstanbul, Literatür Yayıncılık.

[17] **Qian, W., Riedel, M.D., Bazargan, K., ve Lilja, D.J.** (2009). The synthesis of combinational logic to generate probabilities, *International Conference on Computer-Aided Design*, 367–374.

[18] **Wilhelm, D., ve Bruck, J.,** (2008). Stochastic Switching Circuit Synthesis, *ISIT 2008 IEEE International Symposium on Information Theory*, Toronto, Ontario, Canada, 6-11 Temmuz.

[19] **Shannon, C.E.** (1937). *A symbolic analysis of relay and switching circuits*, (yüksek lisans tezi), MIT, Cambridge, MA.

[20] **Alaghi, A., ve Hayes, J.P.** (2014). Fast and accurate computation using stochastic circuits, *Proceedings of the conference on Design, Automation & Test in Europe*, 76.

[21] **Gallager, R.G.** (1962). Low-density parity-check codes, *IRE Trans. Information Theory*, **8**, 21-28.

[22] **IEEE.** (2009). IEEE Standards Association standard, 802.11n for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks.

[23] **ETSI.** (2005). European Telecommunications Standards Institute Standard TR 102 376 V1.1.1. *Digital Video Broadcasting (DVB). User Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications.*

[24] **Tehrani, S.S., Mannor S., ve Gross, W.J.** (2007). Survey of Stochastic Computation on Factor Graphs, *37th International Symposium on Multiple-Valued Logic (ISMVL 2007)*, 54.

EKLER

EK A: RBKY VE kapısı için Matlab kodu

EK B: RBAY VE kapısı için Matlab kodu

EK A

```
% RASTGELE BIT KARISTIRMA VE KAPISI HATA SONUCLARI

n=input('Bit dizisinin uzunlugunu giriniz: ');

   a = zeros(1, n);
   c = zeros(1, n);
   shuf1 = zeros(1, n);
   shuf2 = zeros(1, n);

m=1;

   rng('shuffle');

for k=1:1:n
% a dizisini olustur
   say1=0;
   a(k)=1;

   for s1=1:1:n
      say1=say1+a(s1);
   end

   prob1=say1/n;

    b = zeros(1, n);
% b dizisini olustur

      for d=1:1:n
      say2=0;
      b(d)=1;
               for s2=1:1:n
               say2=say2+b(s2);
               end

               prob2=say2/n;
               err=0;
               avgerr=0;

         for tkr=1:1:1000
%   Aktar & Karistir
               aaa=randperm(n);
               shuf1=a(aaa);

               bbb=randperm(n);
               shuf2=b(bbb);

%      LOJIK VE OP.
      say3=0;
```

```
        c = and(shuf1, shuf2);

        for i=1:1:n
        say3=say3+c(i);
        end
              err = abs((say3/n)-(prob1*prob2))/(prob1*prob2);
              avgerr=avgerr+err;
           end

        avgerr=avgerr*0.001;
        fprintf('%.5f - %.5f  icin ort hata= %.5f\n',prob1,prob2,avgerr);

        G(m,1)=prob1;
        G(m,2)=prob2;
        G(m,3)=avgerr;
        m = m+1;

        end
end

x=G(:,1);
y=G(:,2);
z=G(:,3);

q=1;
for aktarma1=1:n+1:(n*n)
    x_edge(1,q)=x(aktarma1,1);
q=q+1;
end
   x_edge(1,q)=(n+1)/n;

q=1;
for aktarma2=1:1:n
    y_edge(1,q)=y(aktarma2,1);
q=q+1;
end
    y_edge(1,q)=(n+1)/n;

[X,Y]=meshgrid(x_edge,y_edge);
Z= reshape(z,n,n);
Z(n+1,:)=0;
Z(:,n+1)=0;

axes1 = axes('TickDir','out','FontSize',11,'CameraViewAngle',6.6086);
grid(axes1,'on');
hold(axes1,'all');
hidden off
% Create xlabel
xlabel({'p1'});
```

```
% Create ylabel
ylabel({'p2'});

% Create title
title('RASTAGELE BIT KARISTIRMA VE KAPISI HATA SONUCLARI');

% Create colorbar
colorbar('peer',axes1,'FontSize',11);

surf(X,Y,Z)
axis([0,1.1,0,1.1])
disp('Programi kullandiginiz icin TESEKKURLER!')
```

EK B

```
%  RASTGELE BIT ATAMA VE KAPISI HATA SONUCLARI

n=input('Bit Uzunlugunu Giriniz: ');

    a = zeros(1, n);
    b = zeros(1, n);
    c = zeros(1, n);
    m=1;

    rng('shuffle');


for o1=1:1:n
prob1=o1/n;

        for o2=1:1:n
        prob2=o2/n;

    err=0;
    avgerr=0;

            for tkr=1:1:1000

%   Bit base random
                    for i=1:1:n
                       cursor=rand();

        if cursor<=prob1
         a(i)=1;
        else
         a(i)=0;
        end
                    end

                    for i=1:1:n
```

```
                cursor=rand();

      if cursor<=prob2
         b(i)=1;
      else
         b(i)=0;
      end
              end


%      LOJIK VE OP.
       say3=0;
       c = bitand(a, b);

       for i=1:1:n
       say3=say3+c(i);
       end

            err = abs((say3/n)-(prob1*prob2))/(prob1*prob2);
            avgerr=avgerr+err;
         end

       avgerr=avgerr*0.001;
       fprintf('%.5f - %.5f icin ortalama hata = %.5f\n',prob1,prob2,avgerr);

       G(m,1)=prob1;
       G(m,2)=prob2;
       G(m,3)=avgerr;
       m = m+1;

       end
end

x=G(:,1);
y=G(:,2);
z=G(:,3);

q=1;
for aktarma1=1:n+1:(n*n)
   x_edge(1,q)=x(aktarma1,1);
q=q+1;
end
  x_edge(1,q)=(n+1)/n;

q=1;
for aktarma2=1:1:n
   y_edge(1,q)=y(aktarma2,1);
q=q+1;
end
   y_edge(1,q)=(n+1)/n;
```

```
[X,Y]=meshgrid(x_edge,y_edge);
Z= reshape(z,n,n);
Z(n+1,:)=0;
Z(:,n+1)=0;

axes1 = axes('TickDir','out','FontSize',11,'CameraViewAngle',6.6086);
grid(axes1,'on');
hold(axes1,'all');
hidden off
% Create xlabel
xlabel({'p1'});

% Create ylabel
ylabel({'p2'});

% Create title
title('RASTGELE BIT ATAMA VE KAPISI HATA SONUÇLARI');

% Create colorbar
colorbar('peer',axes1,'FontSize',11);

surf(X,Y,Z)
axis([0,1.1,0,1.1])
disp('Programi Kullandiginiz icin TESEKKURLER!')
```

ÖZGEÇMİŞ

Ad Soyad **:** Serter YAVUZ

Doğum Yeri ve Tarihi **:** Fatih 26/04/1986

E-Posta **:** yavuzsert@itu.edu.tr

ÖĞRENİM DURUMU:

- **Lisans** **:** 2010, İstanbul Teknik Üniversitesi, Elektrik-Elektronik Fakültesi, Elektronik Mühendisliği Programı
- **Yüksek Lisans** **:** 2015, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Elektronik ve Haberleşme Anabilim Dalı, Elektronik Mühendisliği Program

MESLEKİ DENEYİM VE ÖDÜLLER:

Nisan 2012'den beri Bilko Bilgisayar, Kontrol ve Otomasyon A.Ş'de çalışmaktadır.

TEZDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- Yavuz S., Altun M., 2014. Stokastik Hesaplamada Hata Oranlarını Azaltmak için Rastgele Bit Karıştırma Yöntem. *Ulusal Elektrik - Elektronik, Bilgisayar ve Biyomedikal Mühendisliği Sempozyumu (ELECO)*, Kasım 27-29, 2014 Bursa, Türkiye.