SFINKS DİZİ ŞİFRELEME ALGORİTMASININ VHDL İLE YAZILIMI VE FPGA ÜZERİNDE GERÇEKLENMESİ

Proje Sahibi: Ahmed Yasir DOĞAN
Üniversite: İstanbul Teknik Üniversitesi
Bölümü: Elektronik Mühendisliği,
Danışmanı: Yrd. Doç. Dr. S. Berna ÖRS YALÇIN

MAYIS 2006

1. GİRİŞ

1.1 Giriş ve çalışmanın amacı

Kriptografi şifre bilimidir, Çeşitli iletilerin, bilgilerin herkese açık ortamlarda iletilirken istenmeyen kişiler tarafından kullanılmasını veya değiştirilmesini önlemek amaçlı kullanılır. Kriptografi, bilgiyi güvenli bir şekilde sadece istenilen kişiye ulaştırmak amacıyla uzun süredir kullanılmaktadır. Tarihin bilinen ilk şifreleme yöntemi yer değiştirme ve harf değiştirme yöntemidir. Bu yöntemlerden ilki bir yazıdaki harflerin yerlerini değiştirerek, ikincisi ise harfleri başka harflerle değiştirerek gerçekleştirilir.

Kriptografik sistemler önceleri askeri amaçlı kullanıldıysa da gelişen teknoloji ile birlikte ortaya çıkan güvenlik açığını kapatmak ve bilginin güvenilir bir şekilde taşınmasını sağlamak amacıyla sivil yaşamda da yerini almıştır.

İletilmek istenen veri açık ağlar üzerinden iletildiğinde veri istenmeyen kişiler tarafından dinlenme veya değiştirilme tehdidi altındır. Söz konusu veri şifrelenmemiş düz metindir (plaintext). İletinin içeriğini saklamak üzere ileti üzerindeki gizleme işlemine şifreleme (encryption) denir. Bu işlem sonucunda düz metin şifrelenmiş olur ve böylelikle bilginin içeriği başkaları tarafından anlaşılamayacak hale gelir. Şifreleme işlemi sonucu oluşan metine şifreli metin (ciphertext) denir. Şifreleme amaçlı kullanılan algoritmanın güvenliği bu algoritmanın saklı tutulması ile sağlanıyorsa bu bir sınırlandırılmış algoritmadır. Bu tür algoritmalar günümüz şartlarına pek uymamaktadır. Algoritmanın güvenliği sadece gizli tutulmasıyla sağlandığından istenmeyen bir durumda algoritmanın açığa çıkması ile kullanılan tüm sistemin değiştirilmesi gerekir. Aynı zamanda bu tür algoritmaların gizliliği nedeniyle kalite kontrolüne ve standardizasyona olanak tanımadığından bu tür algoritmalar güvenli bir şifreleme sağlamazlar. Günümüzde bu sorun açık algoritma, gizli anahtar tekniği ile giderilmektedir. Günümüzde kullanılan şifreleme algoritmaları artık gizli değildir. Bilginin güvenliği algoritmanın gizlenmesi ile değil, iletiyi şifrelemek amaçlı kullanılan anahtarın saklı tutulması ile sağlanır. Kullanılan anahtar çok çeşitli değerler alabilir ve yalnızca bu anahtara sahip kişiler istenilen bilgiye ulaşabilir. Şifreleme sistemlerinde kullanılan anahtarların özgün olması ve kendini tekrarlamaması istenir.

Gizli anahtar kullanan şifreleme sistemleri iki gruba ayrılır; bunlar blok şifreleme (block cipher) ve dizi şifreleme (stream cipher) sistemleridir. Blok şifreleme sistemleri şifrelenecek olan veriyi bloklara ayırır ve her birim zamanda bir bloğu şifreler. Her bloğu şifreleme işlemi birbirinden bağımsız olarak gerçekleştirilir.

Dizi şifreleme sistemleri ise blok boyları bir olan blok dönüştürücüler olarak düşünülebilir.Dizi şifreleyicilerde her birim zamanda şifrelenecek olan verinin yalnızca bir biti şifrelenir, bu şifreleme işlemi bir exor kapısı ile gerçekleştirilir ve kullanılan anahtar uyarınca diğer bitlerde sırasıyla şifrelenir.

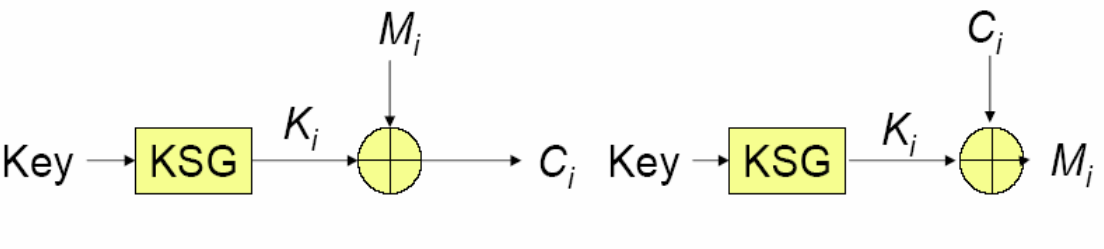
Bu çalışmanın konusu olan SFINKS algoritması da bir dizi şifreleme algoritmasıdır. Her birim zamanda verinin bir biti bu algoritma uyarınca oluşturulan anahtar ile exorlama işlemine tabi tutulur. SFINKS algoritması daha önceden gerçekleştirilmemiş olması ve Avrupa Birliği Standardının seçilmesi amacıyla düzenlenen ECRYPT Dizi Şifreleme Projesi' ne aday olan algoritmalardan bir tanesi olması bakımından önemli bir yere sahiptir. Çalışmanın ileriki bölümlerinde dizi şifreleme yöntemi detaylı olarak incelenecek ve daha sonra bir dizi şifreleme algoritması olan SFINKS algoritması üzerinde hususla durulacak, çalışma ilkesi ayrıntıyla ele alınacaktır. SFINKS algoritması FPGA üzerinde gerçeklenecek ve bu gerçekleme işlemi ayrıntıyla ele alınacaktır. Ayrıca algoritmanın FPGA üzerinde gerçeklenmesi nedeniyle FPGA yapısı hakkında genel bilgiler de SFINKS algoritmasının ele alınmasından önce sunulacaktır.

2. DİZİ ŞİFRELEME SİSTEMLERİ

2.1 Genel Bilgiler

Günümüzde kullanılan kriptografik sistemler genel olarak kullanılan anahtarın açık veya gizli tutulmasına göre ikiye ayrılırlar. Anahtarı açık tutulan kriptografik sistemler iki farklı anahtara sahiptirler; bu anahtarlardan biri açık ve herkes tarafından bilinebilen bir anahtar iken diğer anahtar ise gizli tutulur. Gizli tutulan anahtara sahip kişi veya kurumlar sadece gönderilen iletiyi anlayabilir. Banka uygulamalarında açık anahtar uygulaması kullanımı oldukça kolaylık sağlar. Gizli anahtar kullanılan sistemlerde ise anahtar sadece haberleşmek isteyen kişi yada kurumlarda mevcuttur.

Dizi şifreleme sistemleri gizli anahtarlı şifreleme algoritmalarının önemli bir sınıfını oluşturur. Dizi şifreleyiciler zamanla değişen anahtar yardımıyla düz metinin her bir bitini sırayla şifrelerler. Dizi şifreleyiciler yüksek hızlı iletişim için en iyi alternatiflerden birisidir. Ayrıca donanım ekipmanı genel itibariyle çok basit olup, blok şifreleyicilere oranla donanım üzerinde daha yüksek hızlara sahiptirler. Yüksek hatalı iletişim ortamlarında hata riskini azalttığından tercih sebebidirler.



Şekil 2. 1 Senkron dizi şifreleyicilerin genel modeli

Dizi şifreleme sistemleri genel olarak 2.1 de verilen yapıya sahiptirler. Şifrelenecek olan düz metnin bit dizisi şeklindeki ifadesi olan M dizisinin bir biti ile üretilen anahtar dizisinin bir biti exor işlemine tabii tutularak şifrelenmiş metine ait olan c biti elde edilir. Bu işlem düz metine ait bütün bitler için uygulanarak istenilen şifrelenmiş metin elde edilir. Şifrelenmiş

metinden düz metin elde edilmek istendiği takdirde ise exor işleminin özelliğinden faydalanarak ; aynı şekilde şifrelenmiş metinin bir biti, biti şifrelemek için kullanılan aynı anahtar biti ile exor işlemine tabii tutulur. Bu işlem şifrelenmiş metine ait bütün bitler için tekrarlandığı takdirde düz metin elde edilmiş olur.

Kullanılan dizi şifreleme algoritması yeterince güvenli ise şifrelenmiş metinden düz metine geçebilmek için anahtarın bilinmesi şarttır. Anahtarın bulunabilmesi ancak bütün olasılıkların denenmesi ile mümkün olabilir, fakat bu işlem çok uzun zaman alır ve bu süre zarfında anahtar değiştirilmiş ve yeni bir anahtar kullanılmaya başlanmış olur.

Dizi şifreleme sistemleri genel olarak makul bit uzunluklarına sahip anahtar tohumu dizilerini ve/veya şifrelenmemiş metini kullanarak olabildiğince özgün, rasgele, kendini tekrarlamayan ve tahmin edilmesi zor olan anahtar dizisi üretilmesinde kullanılır. Üretilen anahtarın kalitesi dizi şifreleme sistemlerinin temel problemidir. Üretilen anahtarın çok uzun zaman sonra kendini tekrarlaması kullanılan anahtarın bulunmasını zorlaştırır ve bu da algoritmanın güvenliği açısından oldukça büyük öneme sahiptir. Dizi şifreleme sistemleri rasgele bit üreten sonlu durum makineleri olarak algılanabilir. Burada rasgeleden kasıt önceki bitlere bakarak bir sonraki bitin tahmin edilemez olmasıdır. Rasgele bir dizi elde etmenin mümkün olmadığı göz önünde bulundurulursa, burada amaç eldeki sistemlerle olabildiğince karmaşık bir bit üretme algoritması kullanarak yeterince rasgele diziler elde etmektir.

2.2 Sınıflandırma

2.2.1 Senkron dizi şifreleyiciler

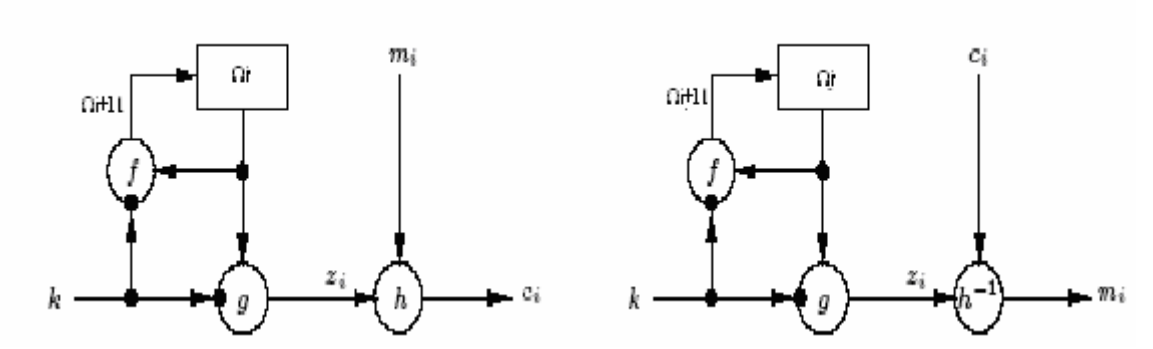
En basit dizi şifreleyiciler anahtar dizisini düz metinden yada diğer adıyla şifrelenmemiş metinden bağımsız olarak üreten senkronize sistemlerdir. Bu tür dizi şifreleyicileri kullanılan bir anahtar oluşturma algoritması uyarınca sisteme girilen anahtar tohumu dizisinden rasgele bir anahtar dizisi üretir. Senkron dizi şifreleyiciler için şifreleme aşamaları genel olarak 2.1, 2.2, 2.3 eşitlikleri ile ifade edilebilir.

$$\Omega_{i+1} = f(\Omega i; k); \quad (2.1)$$

$$z_i = g(\Omega i; k); \quad (2.2)$$

$$c_i = h(zi ; mi); \quad (2.3)$$

Bu ifadeler de Ω_0 ilk durumu ifade ederken f fonksiyonu ise bir sonraki durumu ifade eder. Bir sonraki durum şu an ki durum ile girilen anahtar tohumu dizisinin değerine bağlı olarak belirlenir. Yukarıdaki ifadelerde yer alan g fonksiyonu ise bir sonraki biti elde etmek için kullanılan fonksiyonu ifade eder ve üretilen bit şu an ki duruma ve girilmiş olan anahtar tohumu dizisinin değerine bağlı olarak değişir. h fonksiyonu ise üretilen şifrelenmiş bitin elde edilmesinde kullanılan fonksiyonu ifade eder. İkinci eşitlikten elde edilen anahtar biti ile düz metine ait şifrelenecek bit bu fonksiyon uyarınca işleme alınır ve şifrelenmiş bit elde edilir. Kriptografik sistemlerde h fonksiyonu çoğu kez exor işleminden oluşur. Senkron çalışan dizi şifreleyicilerine ait şifreleme ve şifre çözme süreçlerine ait şemalar 2.2 ve 2.3' te verilmiştir.



Şekil 2. 2 Senkron dizi şifreleyicilerin genel modeli

Senkron dizi şifreleyicilerinin kullanımında doğru olarak şifre çözme işlemi gerçekleştirilmek isteniyorsa alıcı ve gönderen taraf tamamen senkronize olmalıdır, aynı anahtar tohumunu kullanmalı ve aynı durumlarda işlem yapmalıdırlar. İletinin gönderilmesi sırasında karşılaşılabilecek herhangi bir kayıp esnasında şifre çözümünde hata oluşur. Bu sorunu gidermek için sistemin yeniden senkronize edilmesi zorunluluğu ortaya çıkar. Fakat şifrelenmiş metinin bazı bitleri gönderim sırasında değişime uğradığı takdirde diğer bitler bu değişimden etkilenmeyecek ve doğru olarak şifre çözümü gerçekleşecektir. Veri kaybından veya değişimden doğabilecek hataları gidermek amacı ile bu sistemlere ek olarak ayrı mekanizmalar kullanılır ve böylece bu sorunlar giderilir.

2.2.2 Asenkron dizi şifreleyiciler

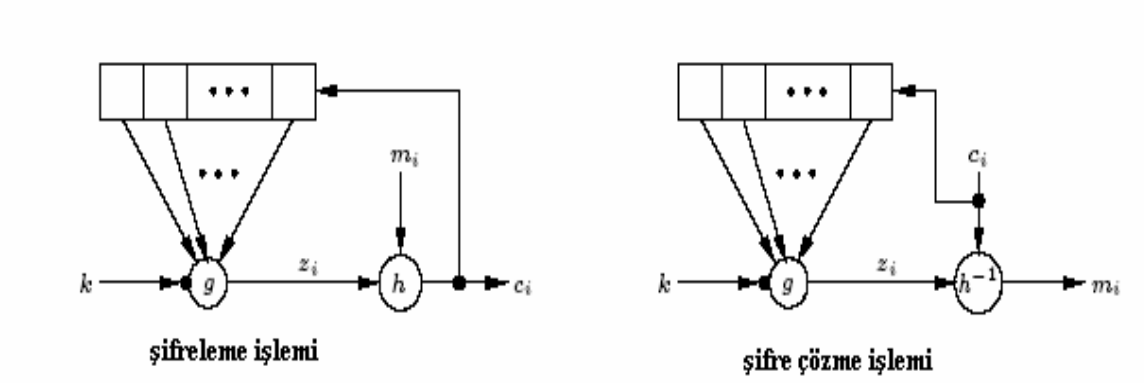
Senkron dizi şifreleyicilerinin yanı sıra bir de asenkron veya kendinden senkronize dizi şifreleyiciler mevcuttur. Bu tür dizi şifreleyiciler anahtar dizisini sisteme girilen anahtar tohumlarıyla birlikte şifrelenmiş metinin sabit sayıdaki şifreli bitlerini kullanarak oluştururlar. Bu türe ait şifreleme aşamaları 2.4, 2.5, 2.6 denklemleri ile ifade edilebilir.

$\Omega i = (ci\text{-}t, ci\text{-}t+1, ci\text{-}t+2, \ldots, ci\text{-}1);$ (2.4)

$zi = g(\Omega i; k);$ (2.5)

$ci = h(zi ; mi);$ (2.6)

İlk eşitlikte yer alan $\Omega_0=(c_{-t}, c_{-t+1}, c_{-t+2},..., c_{i-1})$ ifadesi ilk durumu belirtir. Senkron sistemlere ait eşitliklerde de olduğu gibi g anahtar biti üreten fonksiyonu ve h ise şifrelenmiş biti üreten fonksiyonu ifade etmektedir. Bu tür dizi şifreleyicilerinde bulunulan durumu şifreleniş metinin sabit sayıdaki bazı bitleri belirler. Asenkron dizi şifreleyicilerine ait şifreleme ve şifre çözme aşamaları aşağıdaki şemalar da gösterilmiştir.



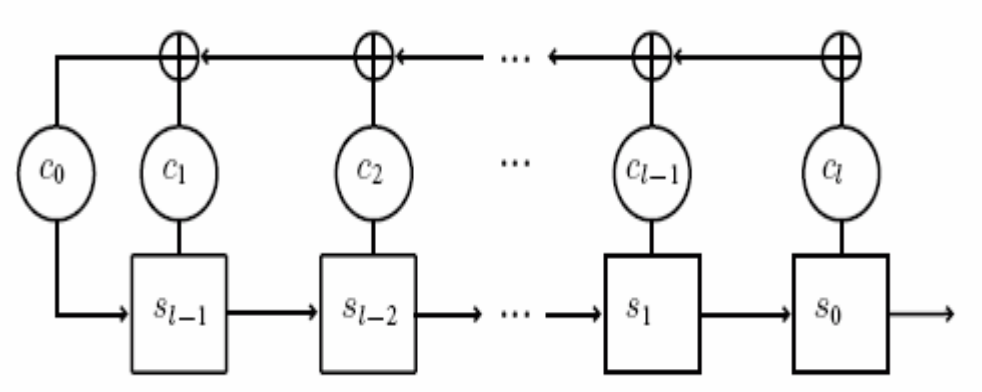
Şekil 2. 3 asenkron dizi şifreleyicilerin genel modeli

Gönderilen şifrelenmiş veride herhangi bir kayıp olması durumunda senkron sistemlerde şifre çözmek mümkün olmadığı halde asenkron veya kendinden senkronize dizi şifreleyicilerinin kullanıldığı kriptografik sistemlerde şifre çözümü mümkündür. Şifre çözme haritası sadece sabit bir şifrelenmiş bit dizisine bağlı olduğundan senkronize halin kaybolması durumunda otomatik olarak sistem kendini yeniler ve sadece iletim sırasında kaybolan veriler dışında herhangi bir kayıp yaşanmaz. Fakat şifrelenmiş bitler üzerinde herhangi bir değişiklik olduğu durumda tamamen doğru bitlerden oluşan durum oluşuncaya kadar elde edilen bitlerde hata meydana gelmiş olur. Hata riskini azaltmak amacıyla bu sistemlere ek olarak farklı mekanizmalar kullanılır.

2.3 Lineer geri beslemeli ötelemeli yazıcılar(LFSR)

Dizi şifreleyicilerin ürettikleri anahtarın düzgün bir dağılıma sahip olması ve tahmin edilemez olması istenir. İhtiyaç duyulan özelliklere sahip anahtar oluşturmada geri beslemeli ötelemeli yazıcılar sıkça kullanılır. Geri beslemeli ötelemeli yazıcılar her birim zamanda lineer bir

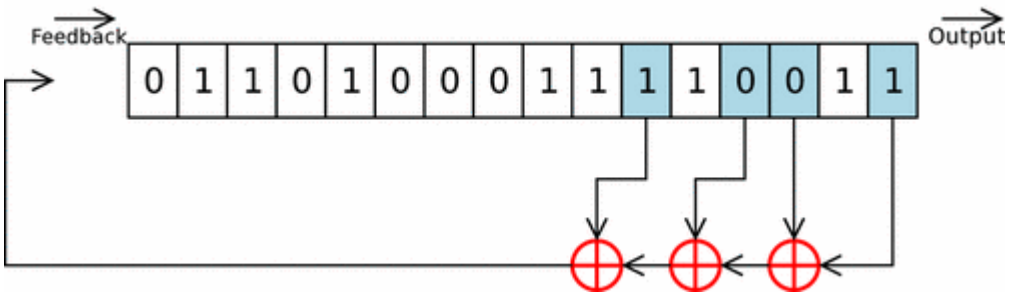
fonksiyonla belirlenen giriş verisini yazılması istenen yazıcıya kaydederken, geri kalan yazıcıların içeriğini bir kaydırır ve böylece her birim zamanda bir sembol saklamış olur. Genel halde geri beslemeli lineer yazıcılara ait blok şema 2.3' te verilmiştir.



Şekil 2. 4 Genel halde LFSR yapısı

Blok şemada yer alan s harfleri lineer geri beslemeli ötelemeli yazıcılara ait hafıza birimlerini ifade ederken c harfleri ise geri besleme ifadesine ait gerekli katsayıları belirtir. Yazıcı l bitlik olup c_0 katsayısı 1 alınır. Her $t \geq 0$ anında LFSR içeriğini geri beslemeden gelen giriş uyarınca içeriğini günceller ve s_0 bitini üretir. LFSR ın başlangıç konumundaki durumu tohum olarak adlandırılır. LFSR yapısının işlevi belirli, tamamen deterministik olduğundan; LFSR tarafından üretilen bit dizisi tamamen bir önceki duruma bağlıdır. Aynı zamanda LFSR' ın sonlu duruma sahip olduğu göz önüne alındığı takdirde kendini tekrarlayan bir dizi üreteceği açıktır. Fakat iyi seçilmiş bir geri besleme fonksiyonuna sahip bir LFSR yapısı rasgele görünümlü ve kendini çok uzun zaman sonra tekrarlayan bit dizisi üretebilir.

LFSR' ın bir sonraki durumunu belirleyen bitler yani geri besleme fonksiyonunda sıfırdan farklı C_i katsayılarına sahip bitlerin sahip oldukları değerler exor işlemine tabi tutulur ve LFSR' ın giriş biti belirlenir. LFSR yapısının sahip olduğu geri besleme fonksiyonu 2 tabanında modüler aritmetiğe sahip bir polinom olarak ifade edilebilir. Katsayıların ancak 0 veya 1 değerlerinden birine sahip olabilecekleri açıktır.



Şekil 2. 5 LFSR örneği $G(x)=x^{11} + x^{13} + x^{14} + x^{16} + 1$

Örneğin; 2.4' te verilen LFSR şemasında geri besleme fonksiyonunu besleyen bitler 16.,14.,13.,11. bitlerdir ve bu LFSR yapısına ait geri besleme fonksiyonu G

G(x)=$x^{11} + x^{13} + x^{14} + x^{16} + 1$ (mod 2) eşitliğiyle ifade edilir. Bu ifadedeki 1 herhangi bir biti ifade etmezken X^n değerleri sol taraftan saymaya başlayarak hangi bitlerden geri besleme alındığını gösterir. Kullanılan LFSR yapısının ürettiği bit dizisinin maksimum periyoda sahip olabilmesi için sağlaması gerekli şartlar mevcuttur. Bunlar; geri besleme polinomunun ilkel olması, yani kendinden daha basit polinomlara bölünememesi gerekir, aynı zamanda geri besleme polinomunu oluşturan bitlerin sayısı çift olmalıdır. LFSR yapısı sonlu durum makinesi olarak düşünülebilir ve n bitlik bir sonlu durum makinesi en fazla 2^n -1 duruma sahip olabilir, yani n bitlik bir LFSR' ın maksimum dizi uzunluğu 2^n -1 olabilir ki bu geri besleme polinomunun beklenen şartları sağlamasıyla mümkün olur. Bir LFSR yapısı farklı geri besleme polinomları kullanarak maksimum dizi şartını sağlayabilir.

2.4 Boole Fonksiyonları

Boole fonksiyonları dizi şifreleyici yapılarında sıkça kullanılan fonksiyonlardır. Genel olarak bir veya birden fazla giriş değişkenini alarak tek bir bitlik çıkış üreten fonksiyonlardır.

X=(X_1, X_2,..., X_n) $X_i \in F_2$ olmak üzere X giriş vektörüne karşılık tek bir çıkış biti üretilir ve f: $F_2^n \rightarrow F_2$ ifadesi ile gösterilir. n değişkenli bir sistemde 2^{2^n} adet boole fonksiyonu tanımlanabilir ve bu boole fonksiyonları kümesi β_n olarak ifade edilir. Boole fonksiyonlarını ifade etmenin bir kaç yolu mevcuttur, giriş sayısı n' in küçük değerleri için doğruluk tablosu oluşturulabilir ve böylece f $\in$ β_n fonksiyonu f=[f(0,0,..0), f(0,0,..1), …,f(1,1,..1)] dizisi olarak ifade edilebilir.

Tablo 2.1 f(X1, X2, X3)= X1*X2+ X2*X3+ X3

x_1	x_2	x_3	$f(\mathbf{x})$
0	0	0	0
0	0	1	1
0	**1**	0	0
0	**1**	1	0
1	0	0	0
1	0	1	1
1	**1**	0	1
1	**1**	1	1

Örneğin tablo 2.1' de $f(X_1, X_2, X_3)= X_1*X_2+ X_2*X_3+ X_3$ boole fonksiyonuna ait doğruluk tablosudur ve bütün girişler için üretilen çıkış tabloda verilmiştir.

2.5 S –kutuları

Dizi şifreleyicilerde kullanılan bir diğer yapı ise S-kutularıdır. S-kutuları boole fonksiyonlarından oluşan bir çıkış vektörüne sahiptir.

$$S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m. \qquad (2.7)$$

2.7 ifadesi ile gösterilir, bu ifade de n giriş vektörünün uzunluğunu gösterirken, m ise çıkış vektörünün uzunluğunu ifade eder. S[X]=Y olarak da ifade edilir, burada Y m bitlik modülo 2 bir çıkış vektörünü ifade ederken, X n bitlik modülo 2 bir giriş vektörünü ifade eder. S-kutuları boole fonksiyonlarından oluştuğu için;

$$S[X]=(f1(X),f2(X),\ldots,fm(X)) \qquad (2.8)$$

$$X=(x_1,x_2,..,x_n) \qquad (2.9)$$

olarak ifade edilebilir.

S-kutusunun karakteristiğini belirleyen çıkış değerlerini oluşturan f boole fonksiyonlarıdır. S kutusunun işlevini belirleyebilmek için f fonksiyonlarının sonuç değerleri incelenmelidir. n giriş vektörünün uzunluğunun yeterince küçük olduğu değerler için S-kutusunun farklı giriş vektörleri için üretmiş olduğu çıkış vektörleri tablo halinde elde edilebilir. Bu durumda yazılımsal veya donanımsal olarak gerçekleştirilen S-kutularının sağlamış oldukları giriş çıkış vektörleri tablo halinde sisteme girilebilir ve sistem verilen giriş vektörü için oluşacak çıkış vektörünü olabildiğince seri üretir. n giriş vektörü uzunluğunun yeterince küçük olmadığı durumlarda yani giriş çıkış tablosunun oluşturulamadığı durumlarda S-kutuları matematiksel olarak gerçekleştirilmelidir. Matematiksel olarak gerçekleştirilen S-kutularına örnek olarak girişine gelen vektörün verilen bir polinoma göre tersini alma işlemini gerçekleştiren S-kutularıdır.

2.6 SFINKS algoritması

Çalışmanın konusu oluşturan SFINKS algoritması bir dizi şifreleyicidir ve dizi şifreleyicilerin sınıflandırılması sırasında ayrıntıyla açıklanan senkron dizi şifreleyici sınıfına ait bir örnektir. SFINKS dizi şifreleyicisine ait algoritmanın oluşum aşamasında yukarıda açıklaması yapılan lineer geri beslemeli ötelemeli yazıcılardan faydalanılmıştır. Ayrıca SFINKS algoritmasında boole fonksiyonlarından yararlanılmış ve sistem içerisinde bir S-kutusunun oluşturmuş olduğu çıkışlar yardımıyla anahtar dizisi üretilmiştir. 4. bölümde SFINKS algoritmasına ait yapılar ayrıntılı olarak incelenecek, bu yapıların gerçekleştirme aşaması verilecektir.

3. SAHADA PROGRAMLANABİLİR KAPI DİZİLERİ

3.1 Genel bilgi ve tarihsel gelişim

Sahada programlanabilir devre elemanları elektriksel olarak programlanabilir eleman ve arabirimlerden oluşan bir tüm devredir. Sahada Programlanabilir kapı dizileri kısaca FPGA (field pogrammable gate array) olarak adlandırılır. Programlanabilir eleman ve arabirimler AND, OR, XOR, NOT işlemlerini veya daha karmaşık olan dekoder, multiplekser gibi matematiksel işlemleri gerçekleştirmek amacıyla programlanabilir. FPGA yapısında bulunan arabirimler tasarımdaki bağlantılar göz önüne alınarak elektriksel olarak programlanabilir ve istenildiği kadar programlanabilme yapısına sahip olduğundan tasarımlarda büyük kolaylık sağlar. Pek çok FPGA yapısı programlanabilir eleman ve ara birimlere ek olarak hafıza birimleri de bulundururlar. Bu hafıza birimleri ayrık flip-flop yapılarından veya hafıza bloklarından oluşabilir.

Tarihsel gelişim incelendiğinde ilk programlanabilir birimin PROM (programmable read only memory) olduğu gözlenir. PROM' lar bir kez programlanabilme özelliğine sahiptirler. PROM' lardan sonra tarihsel gelişimi sırasıyla EPROM (erasable programmable read only memory) ve EEPROM (electrically erasable programmable read only memory) birimleri izler. Bu birimler oldukça kısıtlı yeteneklere sahiptirler. Daha sonraları EEPROM' lara göre daha gelişmiş olan PAL (programmable array logic) ve PLA (programmable logic array) yapıları geliştirilmiştir. PLA yapısında programlanabilen VE ve VEYA matrisleri bulunurken, PAL yapısında ise sabit VEYA matrisi ve programlanabilen VE matrisi mevcuttur. PAL ve PLA yapıları bir fonksiyonu çarpım veya toplamlar şeklinde ifade edebilir ve gerçekleyebilir. Pek çok PLA ve PAL yapılarının bir araya gelmesi ile oluşturulmuş bir diğer yapı ise CPLD' dir. CPLD' ler FPGA yapısına oranla girişlere daha hızlı tepki verebilirken, daha küçük lojik birimlere sahip olması nedeniyle küçük tasarımların gerçekleştirilmesinde kullanılabilir.

Programlanabilir yapılar örneğin FPGA' ler ASIC tasarımlara oranla daha yavaş çalışmalarına rağmen tekrar programlanabilme özelliği ve tasarımların kontrol edilebilir olması nedeniyle tasarımın daha ucuza mal edilebilmesi açısından önemli avantajlara sahiptir.

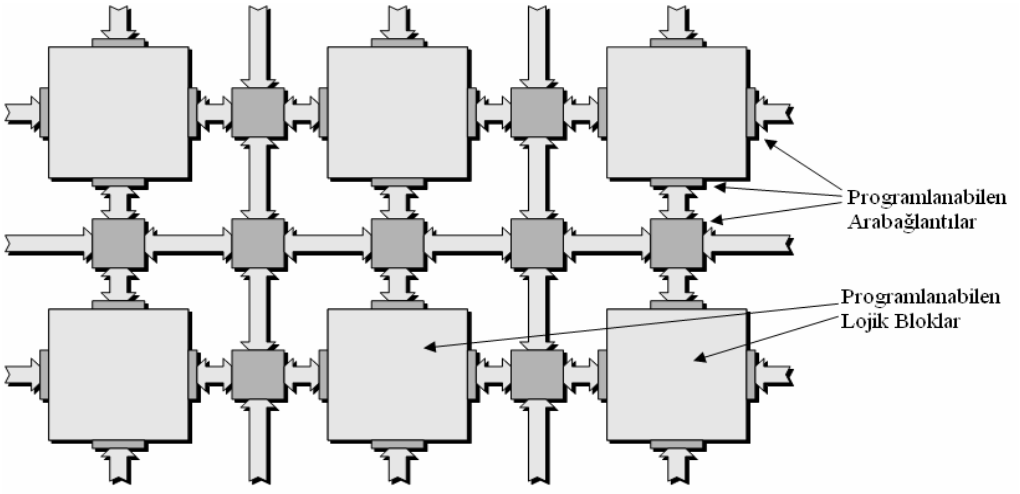
3.2 Kullanım alanları ve üretici firmalar

FPGA yapısı, gelişimi ile birlikte sayısal işaret işleme (DSP) , yazılımsal radyolar, savunma sistemleri , tıbbi görüntüleme, ses tanıma, kriptografi gibi pek çok alanda kullanılmaktadır. Ayrıca ASIC tasarımların üretime gönderilmeden önce istenildiği gibi çalışıp çalışmadığını anlamak amacıyla da kullanılması açısından önemli bir yere sahiptir. Bu çalışma sırasında gerçekleştirilecek olan SFINKS algoritması kriptografi alanında FPGA programlanabilir arabiriminin kullanımına örnek teşkil edebilir. İlk üretiminden itibaren gerçekleştirebilecekleri lojik yapılar ve gerçekleştirebilme hızları açısından gelişmekte olan FPGA programlanabilir birimleri bu gelişim ile birlikte pek çok yeni alanda kullanılmaya başlanmıştır

ASIC tasarımın çok pahalıya mal olması nedeniyle ihtiyaç duyulan özelliklere sahip programlanabilir birim açığını ilk olarak Xilinx firması 1980'li yıllarda FPGA üretimini gerçekleştirerek kapatmıştır. İlk üretimden beri FPGA üretiminde lider kabul edilen firmalar arasında yer alan Xilinx dışında piyasadaki FPGA üretiminde yer alan Altera, Lattice semiconductor, ActelQuick logic gibi firmalarda mevcuttur. En çok kullanılan FPGA programlanabilir birimleri Xilinx ve Altera firmalarına ait FPGA' lerdir.

3.3 FPGA genel yapısı

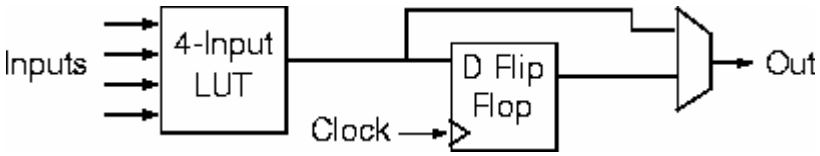
Tipik bir FPGA yapısı konfigüre edilebilir lojik bloklar (CLB) ve yönlendirilebilir kanallardan oluşur.



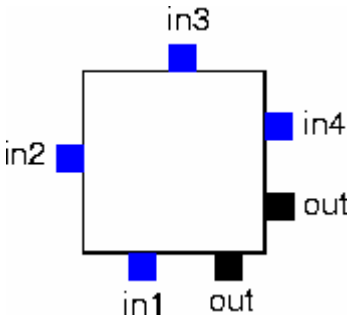
Şekil 3.1 FPGA yapısı

CLB yapıları genel olarak 5 girişten oluşular, bu girişlerden 4 ü LUT (look-up table) girişleri iken diğer giriş ise D flip flop yapısının saat girişidir. Konfigure edilebilir lojik blok yapısına ait genel blok şema ve giriş ve çıkış pinlerinin FPGA programlanabilir birimi üzerindeki yerleşimi 3.2 ve 3.3' te verildiği gibidir.

.



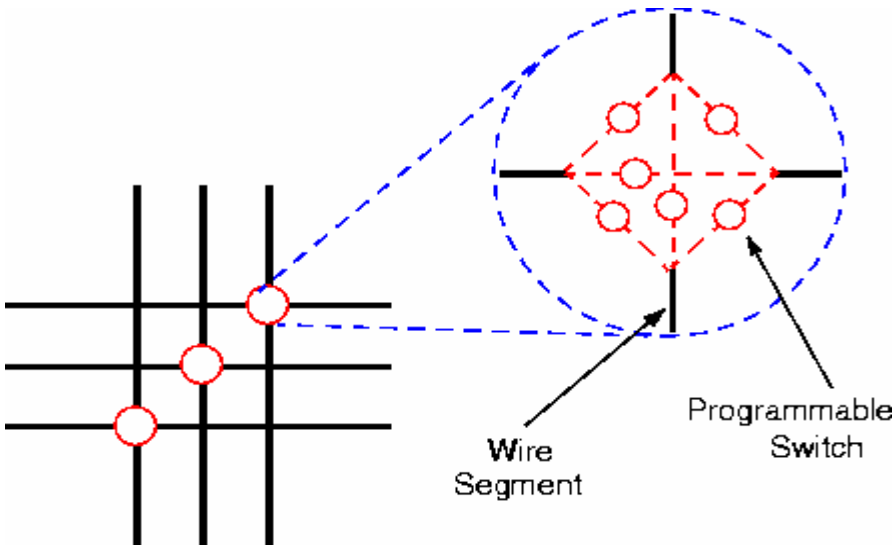
Şekil 3.2 CLB yapısı



Şekil 3.3 CLB giriş çıkışlarının FPGA üzerindeki görünümü

CLB giriş çıkışlarının FPGA üzerindeki konumlarının şemada gösterildiği gibi olmasının nedeni hat üzerinde oluşabilecek gecikmeleri, hat uzunluklarını eşit yaparak olabildiğince eşit olmasını sağlamaktır.

Yatay ve düşey konumlarda bulunabilen CLB' ler istenildiği durumlarda kas kat bağlanabilmekte ve bu bağlantılar programlanabilir ara bağlantılar yardımıyla gerçekleştirilmektedir. FPGA üzerindeki programlanabilir ara bağlantı yapısı 3.4' te verilmiştir.



Şekil 3.4 Programlanabilir ara bağlantılar

Yapının sahip olmuş olduğu programlanabilir anahtarlar sayesinde istenilen tasarım gerekli anahtar bağlantılarının yapılması ile gerçekleştirilebilir.

Ayrıca bu yapılara ek olarak saat dağıtımını uygun bir şekilde gerçekleştirebilmek amacıyla FPGA' ler üzerinde dağıtım hatları bulunmakta ve ardışıl birimlere saat işaretinin tasarımın çalışmasını etkilemeyecek şekilde ulaşması sağlanır.

3.4 FPGA programlama

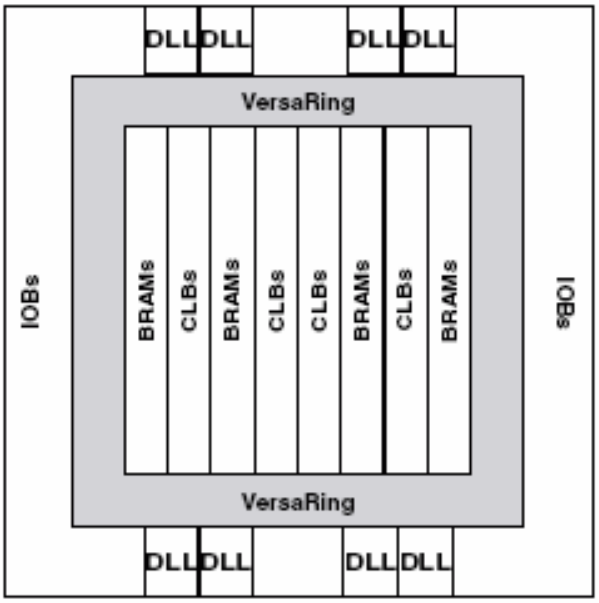
FPGA yapısının nasıl davranacağının belirlenmesi amacıyla tasarımcı donanım tanımlama dillerini (HDL) veyahut şematik tasarım araçlarını kullanır. Donanım tanımlama dillerinin en çok bilinenleri VHDL ve Verilog-HDL dilleridir. Donanım tanımlama dilleri tasarım sırasında kullanılacak olan FPGA programlanabilir biriminin hangi üretici firmaya ait olduğuna göre farklılık gösterebilir. Tasarımcı ilk olarak gerçekleştirmek istediği fonksiyona ait şematik veyahut HDL dillerinden birinde tasarımını gerçekleştirir ve gerekli simülasyonları yapar, bu simülasyonlar sonucunda tasarım istenildiği gibi çalışıyorsa, tasarım için gerekli lojik birimlerin FPGA üzerindeki dağılımını belirlemek amacıyla serim ve hatların yönlendirilmesi kısmı gerçekleştirilir. Bu kısımda tasarımın gerçekleştirilmesi için gerekli lojik birimler belirlenir ve FPGA içerisine gerekli anahtar bağlantıları da yapılarak yerleştirilir. Bu aşamanın da başarıyla gerçekleştirilmesi sonrası tasarımın tekrar simülasyonu yapılır ve devrenin son çalışma hali gözlenir. Gerekli bütün aşamaların başarıyla sonuçlanması durumunda devrenin son hali üretici firmanın sunmuş olduğu paralel kablolar yardımıyla

bilgisayar kullanılarak FPGA üzerine yüklenir ve FPGA programlanabilir birimi istenildiği işi yapmak üzere programlanmış olur.

3.5 Gerçekleme sırasında kullanılan FPGA ve programlanması

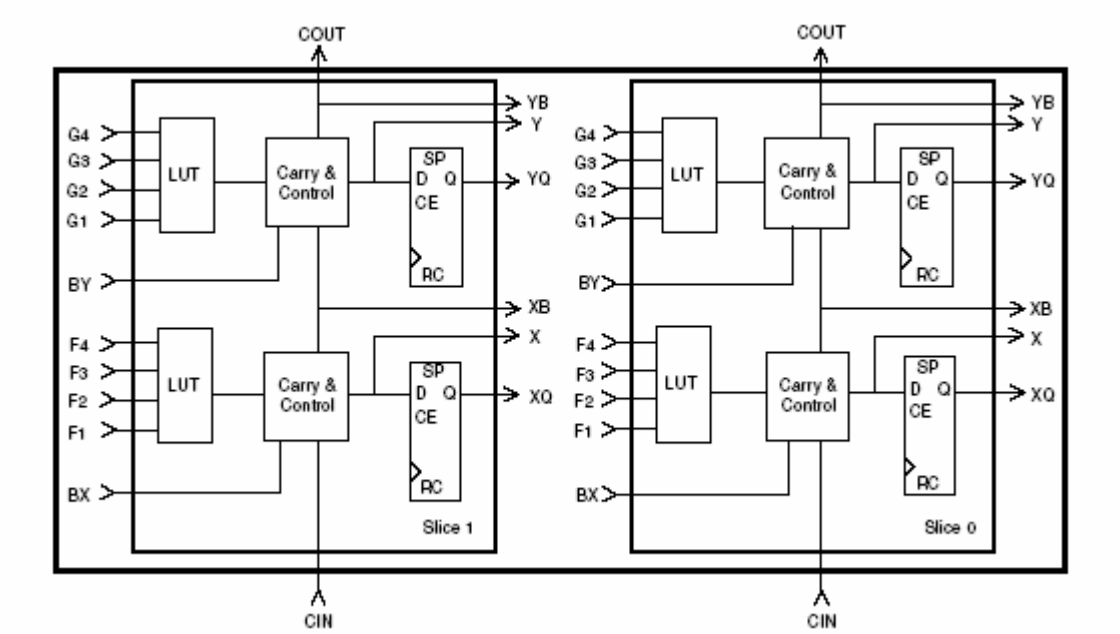
SFINKS algoritmasını gerçekleme sırasında kullanılan FPGA Xilinx firmasına ait olup Virtex-E (XCV1000E)' dir. Virtex-E yapısı da diğer FPGA' lere benzer olarak CLB, programlanabilir arabirimler ve giriş çıkış pinlerinden oluşur. CLB' ler lojik fonksiyonları gerçeklerken, giriş çıkış pinleri FPGA birimiyle CLB' ler arasındaki ara yüzü oluştururlar.

Virtex-E FPGA'ine ait genel yapı 3.5 ' de verilmiştir.



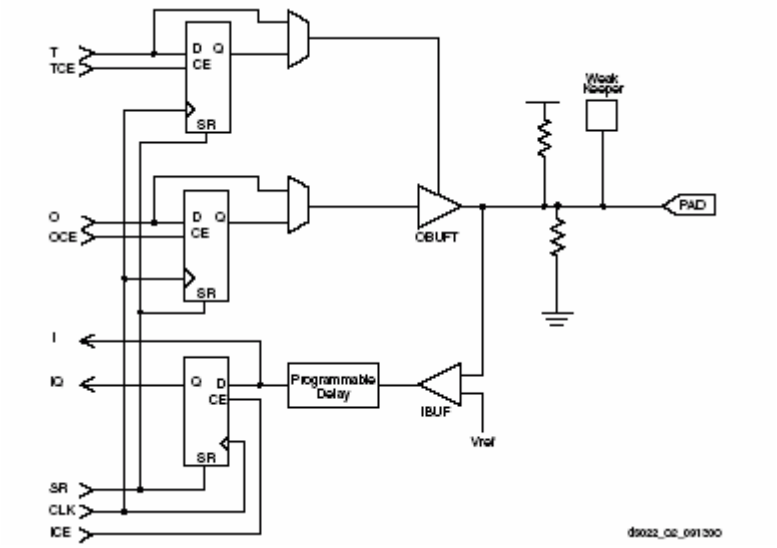
Şekil 3.5 Virtex-E FPGA yapısı

Virtex-E FPGA yapısında bulunan CLB' ler içerisindeki LUT' lar aynı zaman da SRAM görevi görürler. Virtex-E FPGA de CLB yapısına ait genel yapı 3.6' da verildiği gibidir.



Şekil 3.6 Virtex-E FPGA CLB yapısı

Ayrıca Virtex-E yapısı iyi konfigure edilmiş giriş çıkış düzeneğine de sahip olup, pek çok giriş çıkış sinyal standardını desteklemektedir. Giriş çıkış bloklarına ait şema 3.7' de verilmiştir.



Şekil 3.7 Virtex-E FPGA giriş çıkış pad yapısı

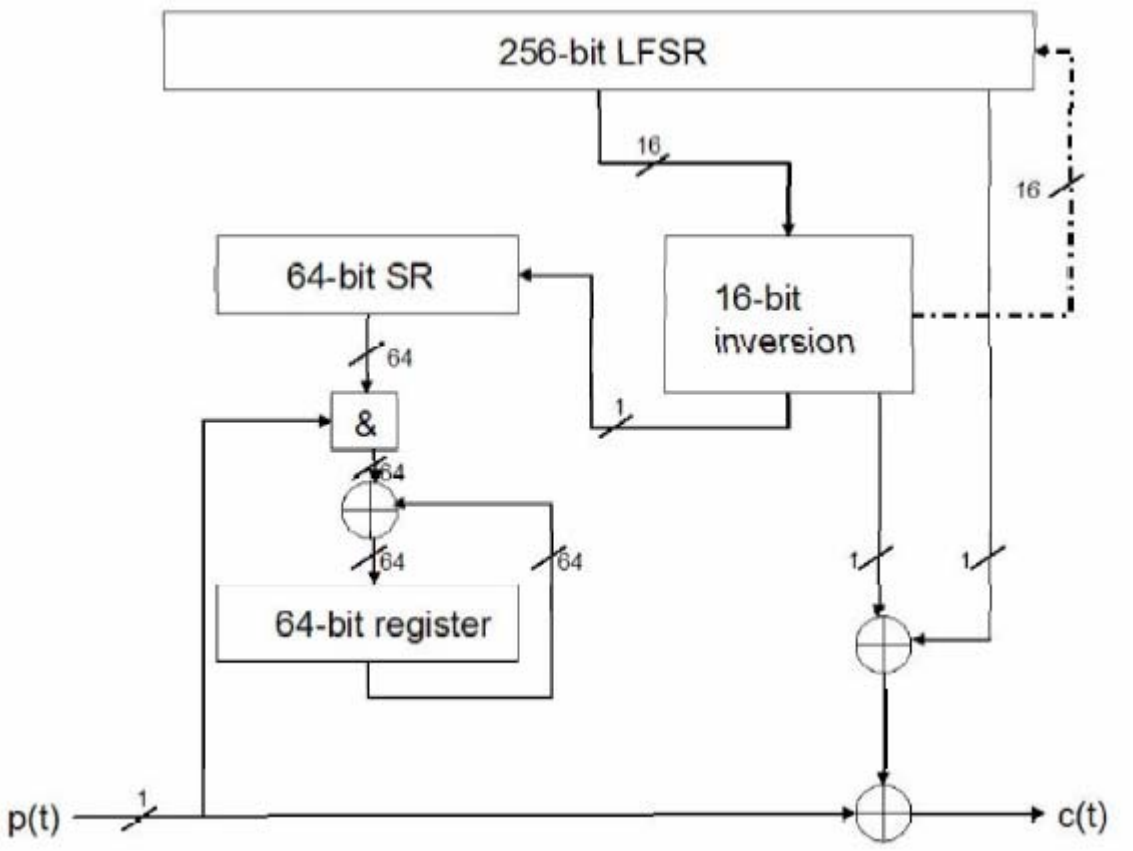
SFINKS tasarımının gerçekleştirilmesi sırasında donanım tanımlama dillerinden olan VHDL dili kullanılmıştır. İlk olarak tasarımı tanımlayan kodlar VHDL dili ile yazılmış ve devrenin davranışsal simülasyonu (behavioral simulation) gerçekleştirilmiştir. Davranışsal simülasyonun istenildiği gibi çalışması sağlanarak lojik sentezleme adımına geçilmiştir ve gerekli araçlar kullanılarak sentezleme işlemi gerçekleştirilmiştir, yani VHDL dili olarak yazılmış tasarıma ait kodlar kapı düzeyine indirilmiştir. Sentez sonrası gerekli testlerin

yapılması ile tasarımın FPGA içerisine yerleştirilmesi (MAP) aşamasına geçilmiştir. Bu aşamada gerekli giriş çıkış pinleri atanmış ve yazılımsal olarak gerçekleştirilen devrenin donanımsal olarak tasarımı gerçekleştirilmiştir. Bu aşamanın sonunda gerçekleştirilen testlerin de başarıyla sonuçlanması sonrası son olarak FPGA üzerine yüklenecek olan program dosyası oluşturulmuş ve tasarımın FPGA üzerinde gerçeklenebilir hale gelmesi sağlanmıştır.

4. SFINKS TASARIM VE GERÇEKLEMESİ

4.1 SFINKS algoritması

SFINKS algoritması, lineer olmayan anahtar dizisi üretecine dayalı senkron bir dizi şifreleyicisidir. Algoritmanın yapısında durum göstergesi olarak 256 bitlik LFSR kullanılmıştır. Boole fonksiyonu olarak 16 bitlik ters alma işlemini gerçekleştiren S-kutusu kullanılmıştır. Algoritmaya ait tasarım 4.1 figüründe verilmiştir.



Şekil 4.1 SFINKS

Şemada görülen p(t), t anında düz metine ait şifrelenecek biti, c(t) ise t anında şifrelenmiş biti ifade etmektedir. Algoritmanın orta derecede güvenlik sağlaması hedeflenmiş ve bu amaçla anahtar tohumu olarak 80 bitlik K=k_{79}……..k_0 kullanılmıştır. Ayrıca başlangıç durumunu belirlemek amacıyla 80 bitlik IV=iv_{79}……..iv_0 vektörü kullanılmıştır. Tasarımda iki adet kaydedici kullanılmıştır, bunlardan ilki MAC için kullanılan 64 bit kaydedici R= r_t^{63}, r_t^{62},

r_t^{61},…., r_t^0 ve diğeri ise 64 bitlik M= m_t^{63}, m_t^{62}, m_t^{61},…., m_t^0 kaydedicisidir. Ayrıca SFINKS tasarımında 256 bit uzunluğuna sahip LFSR kullanılmıştır ve tasarım sırasında bu LFSR ın içeriği S=s_{255} s_{254} s_{253} ,…., s_0 olarak gösterilmiştir. Kullanılan LFSR a ait geri besleme polinomu ise;

$$P(X)=1+X^{44}+X^{62}+X^{64}+X^{69}+X^{93}+X^{105}+X^{131}+X^{141}+X^{149}+X^{171}+X^{190}+X^{192}+X^{204}+X^{208}+X^{242}+X^{256}. \text{ (mod 2)} \quad (4.1)$$

diğer bir yazılımla;

$$S_{t+256}= S_{t+212} +S_{t+194} +S_{t+192} +S_{t+187} +S_{t+163} +S_{t+151} +S_{t+125} +S_{t+115} +S_{t+107} +S_{t+85} +S_{t+66} +S_{t+64} +S_{t+52} +S_{t+48} +S_{t+14} +S_t \text{.(mod 2)} \quad (4.2)$$

olarak ifade edilmiştir, S_{t+j} ifadesindeki j LFSR ın kaçıncı bitine ait olduğunu gösterir.

Tasarımda lineer olmayan filtre fonksiyonu ile anahtar dizisi üretimi sağlanmıştır. Kullanılan filtre fonksiyonu f her t anında LFSR ın 17 bitini giriş olarak alır ve çıkış olarak bir anahtar biti z_t oluşturur. Giriş olarak alınan 17 bitlik vektör tasarım süresince X_t=x_t^{16} ,x_t^{15} ,x_t^{14} ,…., x_t^0 olarak gösterilmektedir. f fonksiyonu aşağıdaki gibi açıklanabilir;

$$z_t=f(X_t)=f(x_t^{16}, x_t^{15}, x_t^{14},\ldots. x_t^1, x_t^0)=(INV(x_t^{16}, x_t^{15}, x_t^{14},\ldots. x_t^1)\&1)+ x_t^0 \text{ (mod 2)} \quad (4.3)$$

$$X_t \in F_2^{17} \quad (4.4)$$

x_t^j ifadesi t anında X_t vektörünün j. bitini ifade eder.

Lineer olmayan filtre fonksiyonunun ana bloğu çarpmaya göre ters alma işlemini gerçekleştiren INV bloğudur. Giriş olarak alınan x_t^{16}, x_t^{15}, x_t^{14},…. x_t^1 vektörünün x^{16}+ x^5+x^3+x^2+1 ilkel polinomu uyarınca çarpmaya göre tersi olan Y_t =y_t^{15}, y_t^{14},…., y_t^0 vektörü bulunur. Elde edilen 16 bitlik Yt vektörünün en düşük anlamlı biti y_t^0 , x_t^0 biti ile exor işlemine tabii tutularak z_t anahtar biti elde edilir. Ayrıca her t anında şifreleme işlemi sırasında hesaplanan Y_t vektörünün 1. biti yani y_t^1 R kaydedicisinin 63. bitine yazılır ve kaydedicinin diğer bitleri birer sağa kaydırılır. Gelen p(t) bitinin değeri sıfır ise M kaydedicisinin içeriği aynen korunur, aksi takdirde M kaydedicisinin içeriği R kaydedicisinin içeriği ile bit bit exorlama işlemine tabii tutularak M kaydedicisinin yeni içeriği belirlenir. R ve M kaydedicilerinin içeriğinin belirlenmesinde gerçekleştirilen işlemler aşağıdaki gibi ifade edilebilir.

$$r_t^j= r_{t-1}^{j+1} \quad j=0,1,\ldots,62 \quad (4.5)$$

$$r_t^{63}=y_t^1 \quad (4.6)$$

$$^{M}_{t} = {}^{M}_{t-1} + p_t * R_{t-1} \quad (4.7)$$

şifreleme işleminin bitiminden 64 t anı sonrasında elde edilen M kaydedicisinin içeriği alıcıya gönderilir ve iletim sırasında oluşabilecek kayıplardan kaynaklı hatalar oluşturulan M kaydedicisi sayesinde giderilir.

Sistemin yeniden senkronize edilebilmesi için ilk olarak tüm iç durumlar sıfırlanır, başlangıç vektörü (IV) ve anahtar tohumu (K) LFSR' a 4.8 ve 4.9 da verildiği gibi yüklenir.

$$S_{-128}^{255} = iv_{79}\ ,\ S_{-128}^{254} = iv_{78},\ \ldots\ldots,\ S_{-128}^{176} = iv_0 \quad (4.8)$$

$$S_{-128}^{175} = k_{79}\ ,\ S_{-128}^{174} = k_{78},\ \ldots\ldots,\ S_{-128}^{96} = k_0 \quad (4.9)$$

ayrıca $S_{-128}^{95}=1$ yapılır ve ardından t=-128 anından itibaren senkronizasyon işlemine başlanır, senkronizasyon işlemi t=0 anına kadar sürdürülür. Bu işlem sırasında Y_{-134},

Y_{-133}, Y_{-132},…, Y_{-129}, Y_{-128}, vektörleri sıfır alınır. 128 adımda gerçekleşen senkronizasyon sırasında LFSR' ın özel olarak kurulumu gerçekleştirilir. LFSR geri besleme polinomunu kullanarak gerçekleştirdiği işleve ek olarak LFSR ın bazı bitleri ters alma işlemi sonrası elde edilen vektörlerin bitleri ile exor işlemine tabii tutulur, bu işlemler aşağıdaki gibi gerçekleştirilir;

$$S_t^j = S_{t-1}^{j+1} + Y_{t-7}^{j\ (\mathrm{mod}\ 16)} \ (\mathrm{mod}\ 2) \quad (4.10)$$

$Y_t^{j(\mathrm{mod}\ 16)}$ ifadesi Y vektörünün t anındaki j(mod 16) nolu bitini ifade etmektedir. j ye ait 16 değer j={11,17,41,52,66,80,111,118,142,154,173,179,204,213,232,247} olarak verilmiştir. Y vektörünün ilk 7 değeri algoritma uyarınca 0 olarak alınmıştır. 128 adımın her birinde geri besleme polinomunun gerçekleştirmiş olduğu işleve ek olarak LFSR ın j. bitleri Y_{t-7} vektörünün j (mod 16) bitleri ile exor işlemine sokulmuş ve LFSR' ın yeni bitleri hesaplanmıştır. Ayrıca senkronizasyon sırasında gerçekleştirilen 128 adım boyunca M ve R kaydedicilerinin içerikleri anahtar biti üretimi sırasında olduğu gibi belirlenir. Gerçekleştirilen 128 adımın sonunda sistem yeniden senkronize edilmiş olur, ve anahtar üretimine başlanabilir.

4.2 Çarpmaya göre ters alma işlemi

Çarpmaya göre ters alma işlemi verilen 4.11 eşitliği ile ifade edilebilir;

$$\beta.\gamma=1 \pmod{p(x)} \qquad \beta, \gamma \in GF(2^m) \qquad (4.11)$$

Ters alma işlemi için uygulanabilecek en basit yöntem tablo kullanmaktır. Fakat tablo kullanma yöntemi m<8 için uygun bir yöntem olabilmektedir. m<8 durumu için tabloyla gerçekleştirilen ters alma işlemi hem alan açısından hem de işlem hızı açısından bir algoritma kullanarak gerçekleştirilen ters alma işlemine nazaran daha uygun bir yöntemdir. Fakat m>8 için bir ters alma algoritması kullanarak bu işlemi gerçekleştirmek tasarım açısından daha uygun olmaktadır.

En iyi bilinen ters alma algoritmaları Fermat ve Euclid algoritmalarıdır. Bunlardan ilki olan Fermat algoritması;

$$\beta^{2\hat{}m}= \beta \qquad (4.12)$$

$$\beta^{-1}= \beta^{(2\hat{}m-2)} \qquad (4.13)$$

denklemlerine dayanır ve

$2^m-2=2+2^2+\ldots+2^{m-1}$ eşitliği göz önüne alındığı takdirde;

$\beta^{-1}= \beta^2 * \beta^{2\hat{}2} * \ldots * \beta^{2\hat{}(m-1)}$ olarak bulunur.

Galois uzayında normal gösterilimde kare alma işlemi sağa kaydırma işlemine denktir ve bu tür algoritmalar gerçekleştirilirken normal gösterilim tercih sebebidir. Kare alma işleminin kaydırma işlemi gibi basit bir yöntemle yapılabilir olmasına karşın bu algoritma çerçevesinde gerçekleştirilen ters alma algoritmalarında paralel çalışan normal tabanlı çarpma devrelerine ihtiyaç duyulması nedeniyle alan çok büyük olur ve dolaysıyla bu tür algoritmalar VLSI tasarımlarda pek tercih edilmezler.

Bu çalışma kapsamında gerçekleştirilen ters alma işlemi en çok bilinen ters alma algoritmalarından Euclid algoritmasına dayanır. Euclid algoritmasını kullanarak ters alma işlemi için yeni bir yöntem öneren Brunner, Curieger, Hofstetter isimli araştırmacıların çalışması referans olarak kullanılmıştır. Bu yöntem uyarınca ilk olarak Euclid algoritmasından

faydalanarak iki sayının en büyük ortak bölenini bulma metodu geliştirilmiştir ve geliştirilen yöntem daha sonra ters alma işlemine uyarlanmıştır.

Euclid algoritması uyarınca iki polinomunun en büyük ortak böleni bulmak için takip edilmesi gerekli adımlar şöyle sıralanabilir. GCD ortak bölen yöntemi olmak üzere;

GCD(A(x),B(x))

A_0: = A(x); B_0:=B(x)

tekrarla i:i+1

Q_i:=A_{i-1}/$B_{i-1;}$

R_i:= A_{i-1}-Q_i* $B_{i-1;}$

A_i=B_{i-1}; B_i =$R_{i;}$

kadar R_i=0; Bu işlem belirtildiği üzere R_i =0 oluncaya kadar devam ettirilir ve R_i =0 olduğunda iki polinomun en büyük ortak böleni A_i olarak bulunur. En büyük ortak böleni bulma yöntemi biraz incelendiği takdirde;

$A(x)=Q_1*B(x)+R_1$ (4.14)

$B(x)=Q_2*R_1+R_2$ (4.15)

$R_1=Q_3*R_2+R_3$, $R_2= Q_4*R_3+R_4$,……., $R_{i-2}= Qi*R_{i-1}+R_i$ olarak yazılabilir ve aynı zamanda $R_i = R_{i-2} - Qi*R_{i-1}$ olarak yazılabilir. Genel olarak kalan polinomu ifade eden $R_i=W_{i*}A(x)+U_{i*}B(x)$ olarak yazıldığı takdirde;

$R_i = R_{i-2} - Qi*R_{i-1}$

$=(W_{i-2}*A(x)+ U_{i-2}*B(x))- Q_i*(W_{i-1}*A(x)+ U_{i-1}*B(x))$

$= (W_{i-2}- Q_i* W_{i-1})*A(x)+(U_{i-2}- Q_i* U_{i-1})*B(x)$

$= W_i*A(x)+U_i *B(x)$

olarak yazılabilir. Bu işlemler göz önüne alınır ve GCD algoritmasında gerekli değişiklikler yapılırsa, ikinci bir GCD algoritması geliştirilmiş olur ki bu algoritma aşağıdaki gibi özetlenebilir.

GCD(A(x),B(x))

R_{-1}:=A(x); W_{-1}:=1; U_{-1}:=0;

R_0:=B(x); W_0:=0; U_0:=1;

i:=0;

tekrarla

i:=i+1;

Q_i:=R_{i-2}/ $R_{i-1;}$

R_i= R_{i-2}-Q_i* $R_{i-1;}$

W_i= W_{i-2}-Q_i* W_{i-1};

U_i= U_{i-2}-Q_i* U_{i-1};

kadar R_i=0;

Bu işlem R_i=0 oluncaya kadar devam ettirilir ve R_i=0 olduğu zaman

$$GCD(A(x),B(x))= W_{i-1}*A(x)+U_{i-1}*B(x) \quad (4.16)$$

olarak bulunur.

Giriş polinomlarından herhangi birinin örneğin A(x) polinomunun F(x) ilkel polinomuna eşit olduğu takdirde F(x) polinomu kendinden daha basit polinomlara ayrılamaması nedeniyle GCD(F(x),B(x))=1 olacaktır. Şayet geliştirilmiş en büyük ortak bölen algoritması A(x) girişinin ilkel F(x) polinomuna eşit olduğu durum için düzenlendiği takdirde gerçekleştirilen işlemin sonunda;

$$R_{i-1}=1=W_{i-1}*F(x)+U_{i-1}*B(x) \quad (4.17)$$

elde edilir ve ters alma işlemi sırasında $GF(2^m)$ üzerinde gerçekleştirilen tüm aritmetik işlemlerin mod F(x) te yapıldığı göz önüne alınırsa W_j*F(x)=0 olacaktır ve R_{i-1} ifadesi tekrar düzenlenirse;

$R_{i-1}=1= U_{i-1}$ *B(x) veya $U_{i-1}= B^{-1}$ (x) elde edilir ve böylece bulunması istenen B(x) polinomunun tersi U_{i-1} elde edilmesiyle bulunabilir. Geliştirilmiş GCD yöntemi kullanılarak istenilen bir polinomun tersini bulmak için izlenmesi gerekli yol şöyle ifade edilebilir;

S:=F(x); V:=0;

R:=B(x); U:=1;

tekrarla

Q=S/R;

tmp:=S-Q*R; S:=R; R:=tmp;

tmp:=V-Q*U; V:=U; U:=tmp;

kadar R=0;

Bu işlem sırasında gerçekleştirilen Q=S/R bölme işlemi bu yöntemin uygun bir tasarım yöntemi olmasını engeller ve bu dezavantajdan kurtulmak gerekir. Dikkat edilirse Q değerinin hesaplanmasının S-Q*R, V-Q*U kalan değerleri için gereklidir. Brent ve Kung adlı araştırmacıların geliştirdiği yöntemle bu kalan değerler bölme işlemi yapılmadan gerçekleştirilebilir. Kalan değerlerinin bulunabilmesi için Brent'in ve Kung'un kullandığı yöntemi şöyle ifade edebiliriz.

S MOD R: (dereceS>=dereceR)

delta:=dereceS-dereceR;

tekrarla

R1:=x^{delta}*R;

U1:= x^{delta}*U;

S:=S-R1; V:=V-U1;

delta:=dereceS-dereceR;

kadar delta<0; (* S MOD R=S, V:=V-(S/R)*U)

Bu işlem basit bir yöntemdir , kalem ve kağıtla yapılan uzun bölme işleminin aynısıdır. Bu işlem delta<0 oluncaya kadar devam ettirildiği takdirde kalanlar sırasıyla S ve R olur .

Bu kalan değerleri bulma yöntemi üzerinde biraz değişik yapılırsa r_m ve s_m ,R ve S polinomlarının en yüksek anlamlı bitlerini göstermek üzere;

S MOD R:

delta:=0;

tekrarla{

şayet r_m=0 **başla{**

R:=x*R; U:=x*U (MOD F)

delta=delta+1;

aksi halde{

şayet s_m=0 **başla**

S:=x*S; U:U/x (MOD F)

aksi halde{

S:=S-R; V:=V-U (MOD F);

S:x*S; U:=U/x (MOD F)

}

bitir

delta=delta-1;**}**

bitir;

kadar delta<0;

Örneğin $S=x^7+ x^5+ x^4+ x^3+ x^2+ x+1$, $R= x^3+ 1$, $U=x+1$, $V= x^4+ x^3$ değerleri için kalan algoritmasının ilk hali ve son olarak geliştirilmiş halinin adım adım gerçekleştirilmiş durumu aşağıdaki tablo 4.1' de verilmiştir.

Tablo 4.1 kalan algoritması örneği

delta	R1	S	U1	V
		$x^7+ x^5+ x^4+ x^3+ x^2+ x+1$		$x^4+ x^3$
4	$x^7+ x^4$	$x^5+ x^3+ x^2+ x+1$	$x^5+ x^4$	$x^5+ x^3$
2	$x^5+ x^2$	$x^3+ x+1$	$x^3+ x^2$	$x^5+ x^2$
0	x^3+1	x	$x+1$	$x^5+ x^2 +x+ 1$
-2				

Tablo 4.2 geliştirilmiş kalan algoritmasının örneği

delta	R	S	U	V
	x^3+1	$x^7+ x^5+ x^4+ x^3+ x^2+ x+1$	$x+1$	$x^4+ x^3$
1	$x^4+ x$	$x^7+ x^5+ x^4+ x^3+ x^2+ x+1$	$x^2+ x$	$x^4+ x^3$
2	$x^5+ x^2$	$x^7+ x^5+ x^4+ x^3+ x^2+ x+1$	$x^3+ x^2$	$x^4+ x^3$
3	$x^6 +x^3$	$x^7+ x^5+ x^4+ x^3+ x^2+ x+1$	$x^4+ x^3$	$x^4+ x^3$
4	$x^7+ x^4$	$x^7+ x^5+ x^4+ x^3+ x^2+ x+1$	$x^5+ x^4$	$x^4+ x^3$

4	x^7+x^4	$x^5+x^3+x^2+x+1$	x^5+x^4	x^5+x^3
3	x^7+x^4	$x^7+x^5+x^4+x^3+x^2$	x^3+x^2	x^5+x^3
2	x^7+x^4	$x^5+x^3+x^2$	x^3+x^2	x^5+x^2
1	x^7+x^4	$x^7+x^5+x^4$	$x+1$	x^5+x^2
0	x^7+x^4	x^5	$x+1$	x^5+x^2+x+1
-1				

Geliştirilen kalan algoritmasının ters alma algoritmasına uyarlanmasıyla elde edilen ters alma işlemi algoritmasının gerçeklenebilmesi için yapılması gereken adımlar sırasıyla;

Ters alma işlemi:

F:=F(x);

S:=F(x); V=0;

R:=B(x); U:=1;

delta:=0;

i:=1 **den** 2*m **'e kadar**

şayet r_m=0 **başla**

R:=x*R; U:=x*U (MOD F)

delta=delta+1;

aksi halde{

şayet s_m=1 **başla**

S:=S-R; V=V-U (MOD F)

bitir

S:=x*S;

şayet delta=0 **başla**

R↔S; U↔V;

U:=x*U (MOD F) delta:=1;

aksi halde

U:=U/x (MOD F);

delta=delta-1;

bitir

bitir

bitir

Bu işlemin 2*m defa gerçeklenmesi durumunda elde edilen B^{-1}=U=V olarak bulunur. Her adımda r_m , s_m bitlerinin ve deltanın değerlerine göre R, S, U, V, delta kaydedicilerinin içeriğinde yapılması gerekli değişikler 4.3 tablosunda verilmiştir.

Tablo 4.3 R,S,U,V kaydedicilerinin ve deltanın fonksiyonel tablosu

r_m	s_m	delta=0	R	S	U	V	delta
0	**-**	**-**	**x*R**	**S**	**x*U**	**V**	**delta+1**
1	**0**	**0**	**x*S**	**R**	**x*V**	**U**	**delta+1**
1	**0**	**1**	**R**	**x*S**	**U/x**	**V**	**delta-1**
1	**1**	**0**	**x*(S-R)**	**R**	**x*(V-U)**	**U**	**delta+1**
1	**1**	**1**	**R**	**x*(S-R)**	**U/x**	**V-U**	**delta-1**

Bu yöntemde R ve S polinomları m+1 bitlik U ve V polinomları ise m bitlik kaydedicilerde saklanır ve tasarım sırasında delta $\log_2(m+1)$' lik ileri ve geri yönde sayıcı olarak kullanılır. Ayrıca tabloda yer alan çıkarma işlemlerinin de GF(2) de işlem yapıldığı göz önüne alınırsa bit bit exor işleminin yapılmasına karşılık geldiği görülür. Yöntemin daha iyi anlaşılabilmesi için bu yöntem uyarınca gerçekleştirilen ters alma işlemine bir örnek vermek gerekirse;

$B(X)=X^2+X+1$ polinomunun $F(X)=X^4+X+1$ polinomuna göre tersi;

m=4 , $R=B(X)= X^2+X+1$,S=F(X), V=0, U=1 bu değerler ikilik düzende ifade edilirse

R="00111" S="10011", V="0000", U="0001" işlemler yapılırsa;

i=1 için;

R="01110", S="10011", V="0000", U="0010" delta=1

i=2 için;

R=”11100”, S=”10011”, V=”0000”, U=”0100” delta=2

i=3 için;

R=”11100”, S=”11110”, V=”0100”, U=”0010” delta=1

i=4 için;

R=”11100”, S=”00100”, V=”0110”, U=”0001” delta=0

i=5 için;

R=”01000”, S=”11100”, V=”0001”, U=”1100” delta=1

i=6 için;

R=”01000”, S=”11100”, V=”0001”, U=”1011” delta=2

i=7 için;

R=”10000”, S=”11000”, V=”1010”, U=”1100” delta=1

i=8 için;

R=”10000”, S=”10000”, V=”0110”, U=”0110” delta=0

olarak bulunur ki B^{-1}=U=”0110” olur.

4.3 SFINKS algoritmasının gerçeklenmesi

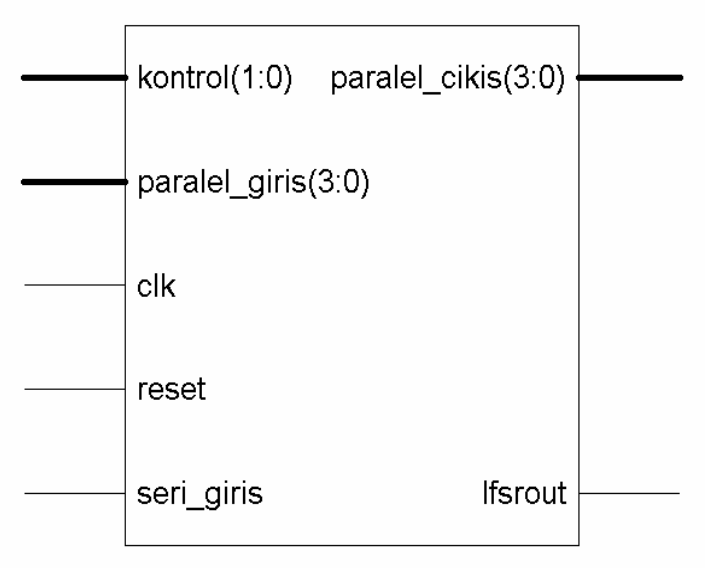
4.3.1 Lineer geri beslemeli ötelemeli yazıcının(LFSR) gerçeklenmesi

SFINKS algoritmasının gerçeklenmesi aşamasında ilk olarak LFSR tasarımı gerçeklenmiştir. En genel halde paralel ve seri girişi olan, öteleme yönü değişebilen paralel çıkış alınabilen bir LFSR tasarımı gerçekleştirilmiştir. Tasarımı gerçekleştirilen LFSR a ait giriş ve çıkışlar aşağıda verildiği gibidir.

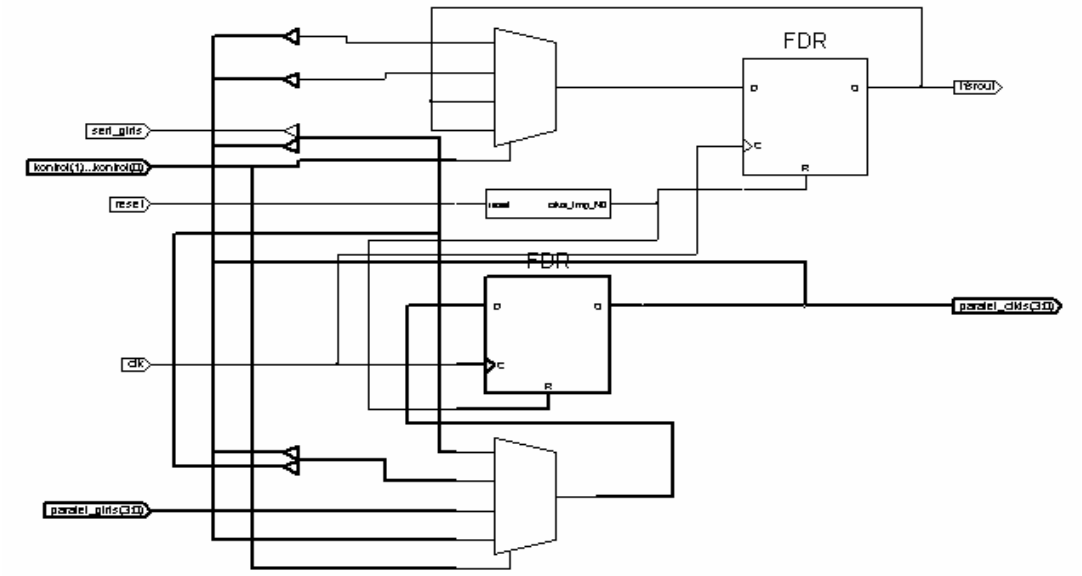
- clk (saat girişi)
- reset: (devreyi sıfırlamak için kullanılan giriş)
- paralel_giris (paralel yükleme girişi N bitlik)
- seri_giris (LFSR a seri yükleme için kullanılan giriş)

- kontrol (öteleme yönünü ve paralel yükleme işlemini belirlemek amaçlı kullanılan giriş 2 bitlik)
- paralel_cikis (paralel çıkış alma amaçlı kullanılan N bitlik çıkış)
- lfsrout (seri çıkış alma amaçlı kullanılan çıkış)

Tasarımı gerçekleştirilen LFSR' ın ilk olarak davranışsal simülasyonu gerçekleştirilmiş ve elde edilen sonuçların istenildiği gibi olduğu görülmüştür. Daha sonra sentezleme aşamasına geçilmiş ve sentezleme işlemi başarıyla gerçekleştirilmiştir. Sentezleme sonrası elde edilen sonuçlarda LFSR' ın çalışabileceği maksimum frekansın 287.356 MHZ olduğu görülmüştür. Aşağıda gerçekleştirilen LFSR a ait genel şematik ve transfer lojiği şematiği verilmiştir.

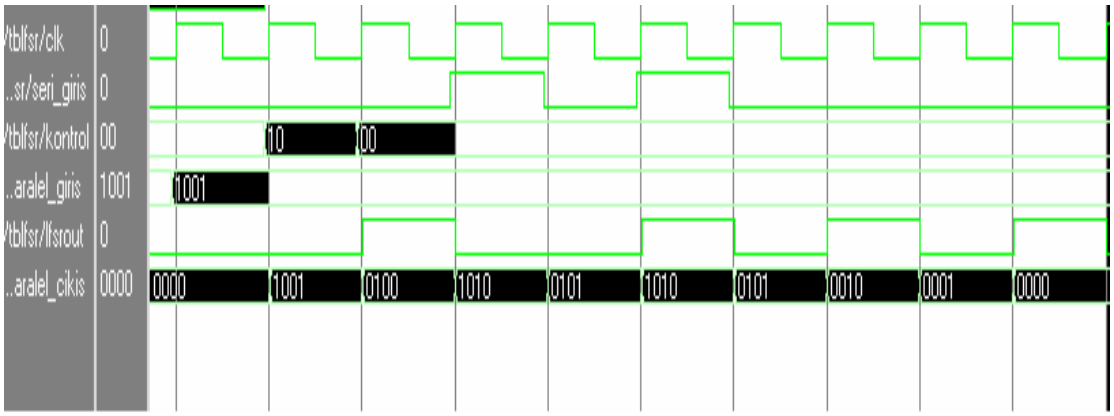


Şekil 4. 1 LFSR' ın genel şematiği



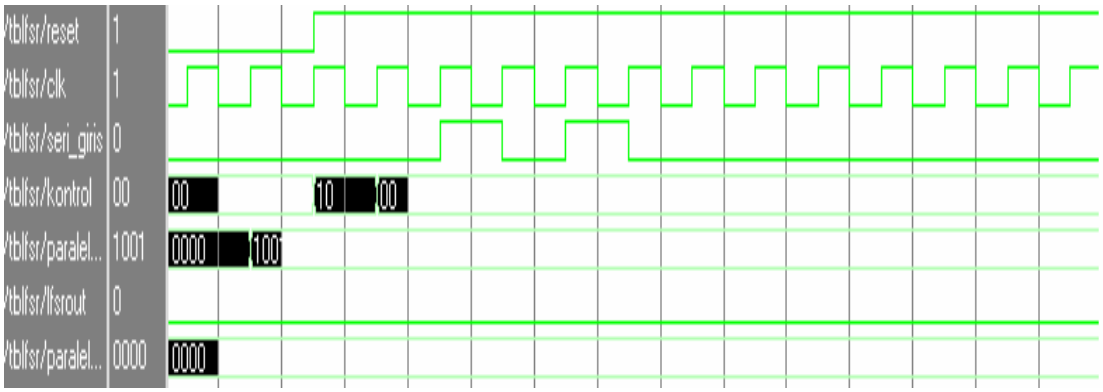
Şekil 4. 2 LFSR' ın yazıcı transfer lojiği şematiği

Sentez sonrası gerçekleştirilen simülasyon için öncelikle test kodları oluşturulmuş ve bu test kodlarının LFSR bloğuna uygulanması sonucu elde edilen sonuçlar şekil 4.3' te verilmiştir. Bu sonuçlarda incelemenin kolay yapılabilmesi amacıyla LFSR ın uzunluğu 4 bit olarak alınmıştır.



Şekil 4. 3 En genel halde LFSR ın sentezleme sonrası simülasyonu

şekil 4.3' ten görüleceği gibi ilk olarak LFSR' a kontrol girişinin "10" yapılması ile "1001" girişi paralel olarak yüklenmiş ve daha sonra kontrol girişinin "00" yapılması ile LFSR ın sağa öteleme yapması sağlanmıştır. Maksimum frekanstan daha büyük frekanslarda yapının çalışmayacağını görmek amacıyla 287.356 MHZ den daha yüksek frekansta saat işareti verilmiştir, elde edilen sonuçlar şekil 4.4' te verilmiştir.

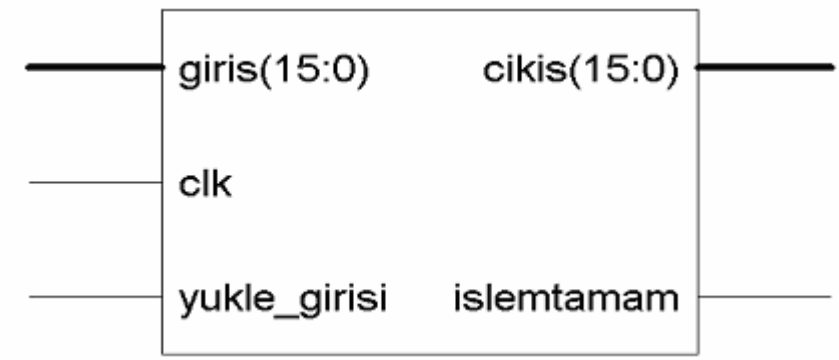


Şekil 4. 4 LFSR a maksimum frekanstan büyük saat darbesi uyguladığı zamanki durum

Şekil 4.4' ten de görüldüğü gibi LFSR maksimum frekanstan daha yüksek frekanslarda çalışmamaktadır.

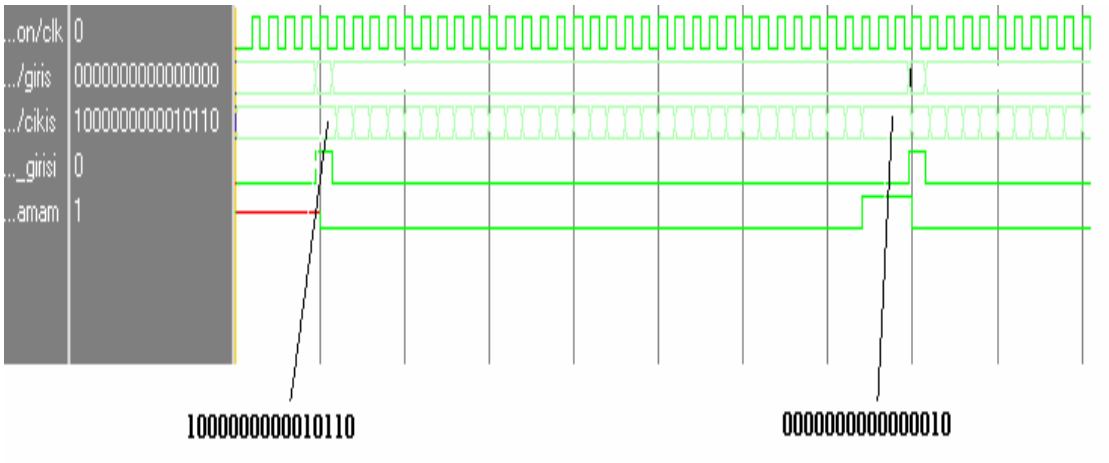
4.3.2 Çarpmaya göre ters alma bloğunun gerçekleştirilmesi

Tasarım sürecinde ikinci olarak çarpmaya göre ters alma işlemini gerçekleştiren blok tasarlanmıştır. Ters alma bloğu gerçekleştirilirken bölüm 4.2 de ayrıntılı bir şekilde ele alınan ters alma algoritması kullanılmıştır. Ters alma işleminin GF(2^{16})' da olması gerektiği düşünülürse kullanılan ters alma algoritması uyarınca 2*m yani 32 defa R, S, U ve V kaydedicilerinin içeriği değiştirilecektir. Bu özellikler göz önüne alınarak ters alma bloğu tasarlanmıştır. Tasarlanan ters alma bloğuna ait şematik blok diyagram şekil 4.5' te verildiği gibidir.



Şekil 4. 5 ters alma bloğu

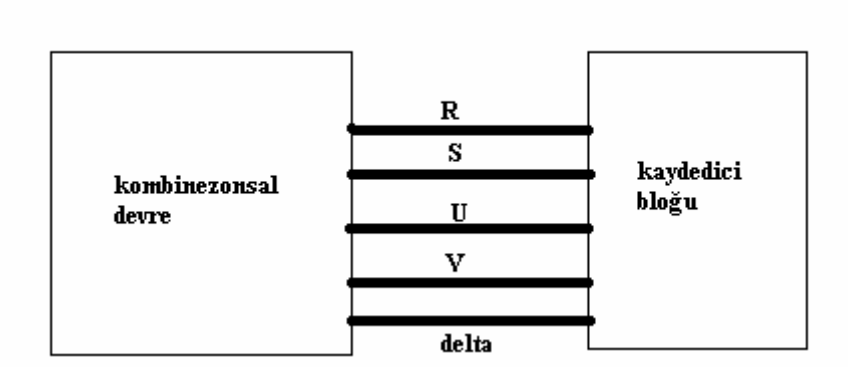
Tasarlanan ters alma bloğunda yükle_girişinin '1' olmasıyla tersi alınacak veri 'giris' ten alınır ve 32 saat işareti sonrasında çıkışa tersi alınmış veri verilir. Yapılan simülasyonlar neticesinde elde edilen sonuçlar şekil 4.6' da verilmiştir.



Şekil 4. 6 ters alma bloğuna ait simülasyon sonucu

Ters alma bloğu sentezlenmiş ve devrenin çalışabilmesi için verilebilecek maksimum frekansın 133.941 MHz olduğu gözlenmiştir. Ters alma bloğu incelendiğinde simülasyon sonuçlarından da görüleceği gibi devre aslında 32 saat işaretinde bir ters alma işlemini gerçekleştirmektedir. Dizi şifreleyicisi devresinin hızını ters alma bloğunun belirleyeceği açıktır. Ters alma işlemi sonucunda hesaplanacak olan sonucun anahtar dizisini oluşturacağı ve bu anahtar dizisinde düz metine ait bitleri şifreleyeceği göz önüne alınırsa aslında sentez sonucu hesaplanan maksimum frekansın dizi şifreleyici devrenin çalışabileceği maksimum frekans olmadığı fark edilebilir. Dizi şifreleyicinin çalışabileceği asıl frekansın 32 saat işaretinde bir ters alma işlemi gerçekleştiğinden hesaplanan maksimum frekansın 32 de biri yani 4 MHz civarında olduğu açıktır. Bu sonuç ta tasarlanan dizi şifreleyicisinin çok yavaş çalışacağını gösterir ki bu da dizi şifreleyicilerin genel özelliği olan yüksek frekanslarda çalışabilmeleri karakteristiğine ters düşer. Bu nedenden dolayı 32 saat işaretinde bir ters alma işlemini gerçekleştiren ters alma bloğunda değişikler yapılmasına karar verilmiştir.

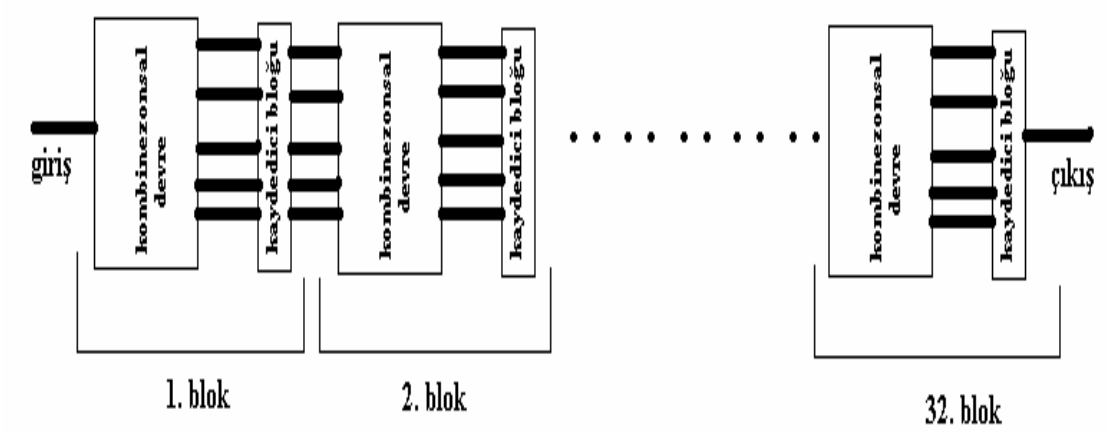
En uzun kombinezonsal yolu değiştirmeden bloğun her saat işaretinde bir ters alma işlemini gerçekleştirmesi sağlanarak yüksek frekanslara ulaşabileceği düşünülmüştür. Bunun için ters alma bloğunda ters alma işlemi sırasında 32 adımın her birinde gerçekleştirilen işlemler kombinezonsal olarak tanımlanmış ve ayrıca her adım sonucunda kombinezonsal devrenin çıkışındaki verilerin kaydedilebilmesi amacıyla kaydedici bloğu tasarlanmıştır. Böylece ters alma işlemi için gerekli olan 32 adımın sadece birini gerçekleştiren devre tasarlanmıştır. Tasarlanan bu devreye ait blok şema aşağıdaki gibidir.



Şekil 4. 7 gerekli 32 adımın sadece birini gerçekleyen devreye ait blok şema

Ters alma işleminin gerçekleşebilmesi amacıyla gerekli 32 adımın her birinde R, S, U, V ve delta kaydedicilerinin içeriklerinin değişeceği göz önüne alınarak kombinezonsal devrenin çıkışındaki R, S, U, V ve delta çıkışları kaydedici bloğuna kaydedilmektedir.

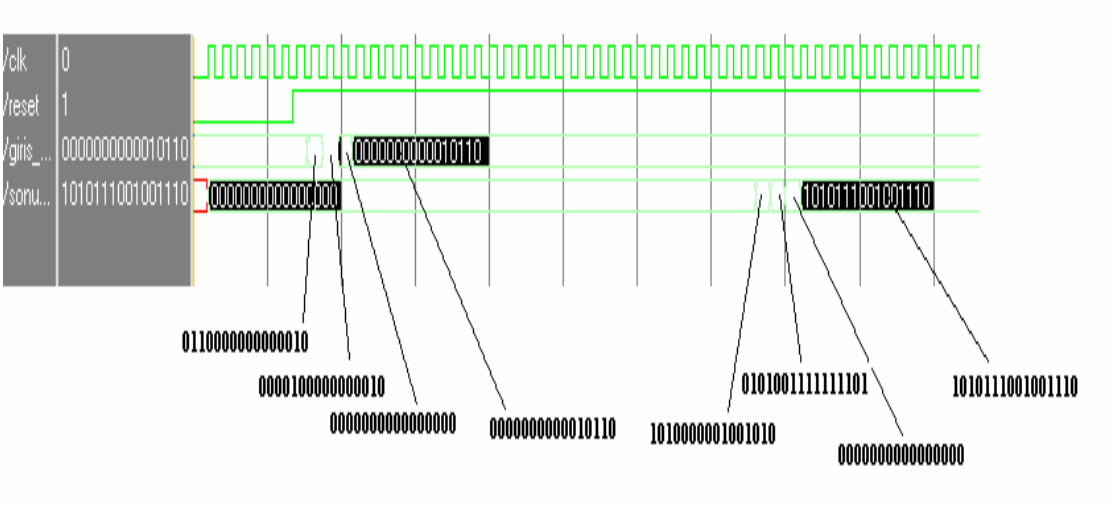
Bir adımı gerçekleştiren bloğun 32 defa art arda bağlanmasıyla her saat işaretinde bir adet ters alma işleminin gerçekleştirilmesi sağlanabilir. Bir adımı gerçekleştiren devrenin art arda bağlanması ile elde edilen devrenin blok şeması aşağıdaki verildiği gibi olacaktır.



Şekil 4. 8 kaskat bağlı 32 adet bloktan oluşan ters alma devresine ait blok şema

32 adet bloğun kaskat bağlanması ile elde edilen ters alma devresi istenildiği gibi her saat işaretinde tersi alınmış bir veriyi çıkışına verir. Fakat bu işlem ilk 32 saat işareti için geçersizdir bu süre boyunca veri girişten alınmasına karşın herhangi bir çıkış oluşturulmaz ve ilk tersi alınmış veri 32. saat darbesiyle çıkışa verilir ve bundan sonra her saat darbesinde 32 saat darbesi önce girişe gelen verinin tersi çıkışa verilir . Böylece ilk 32 saat işaretinde ters alma bloğunun kurulumu gerçekleşir. Tasarımı tamamlanan devrenin sentez işlemi gerçekleştirilmiş ve ters alma bloğunun çalışabileceği maksimum frekansın 119.389MHZ olduğu görülmüştür. Kaskat bağlama sonucu ters alma bloğunun her ne kadar kapladığı alanda artış olsa da çalışabileceği frekans 4 MHZ den 119.389 MHZ e yükseltilmiştir.

Frekansta yapılan yaklaşık 30 kat iyileştirmenin çok büyük bir iyileşme olduğu açıktır.Gerçekleştirilen ters alma bloğuna ait simülasyon sonuçları ise şekil 4.8' de verildiği gibidir.



Şekil 4. 9 Kas kat bağlı 32 adet bloktan oluşan ters alma devresine ait simülasyon sonuçları

Simülasyon sonucundan da görüleceği gibi 32. saat işaretinde ilk çıkış verilmektedir ve 32. saat işaretinden sonra her saat işaretinde 32 saat işareti önce alınan verinin tersi çıkışa verilmektedir. Ayrıca gerçekte tersi olmayan sıfır değerinin SFINKS algoritması uyarınca tersinin yine kendisi yani sıfır olduğu kabul edilmiştir. Kullanılan ters alma algoritmasının sıfır girişi için geçerli olmayacağı açıktır. Devrenin tasarımı gerçekleştirilirken ilk olarak girişin sıfır olup olmadığı kontrol edilir, girişin sıfır olması durumunda çıkışına sıfır verilir. Aksi takdirde ise ters alma algoritması uyarınca hesaplanan değer çıkışa verilir.

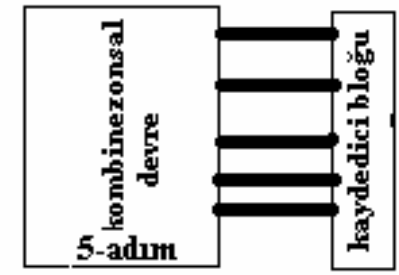
Bölüm 4.1 de SFINKS algoritması uyarınca sistemin yeniden senkronizasyonu sırasında LFSR' ın bazı bitlerinin 7 saat işareti öncesinde hesaplanan tersi alınmış vektörün bazı bitleri ile exorlandığı ifade edilmiştir.

$$S_t^j = S_{t-1}^{j+1} + Y_{t-7}^{j\,(\mathrm{mod}16)} \quad (\mathrm{mod}2) \qquad (4.18)$$

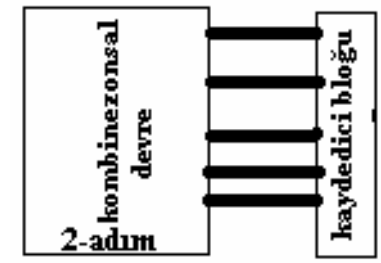
j= {11, 17, 41, 52, 66, 80, 111, 118, 142, 154, 173, 179, 204, 213, 232, 247}

Tasarlanmış olan ters alma bloğunun girişine gelen verinin tersini 32 saat işareti sonra ancak hesaplayabildiğinden, bu blok senkronizasyon sırasında kullanılamaz ve senkronizasyon işleminin gerçekleşebilmesi için en geç 7 saat işareti sonra girişine gelen verinin tersini hesaplayan bir ters alma bloğu kullanılmak zorundadır. Senkronizasyon işlemi sırasında

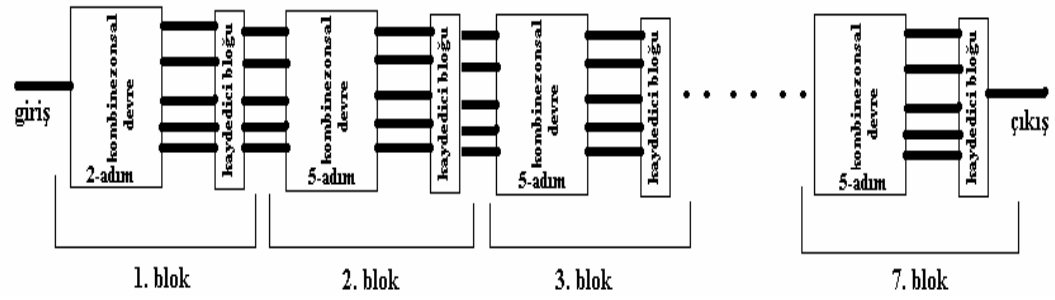
kullanılacak olan ters alma bloğunun tasarımı sırasında ters alma algoritması uyarınca 32 adımın sadece 1 adımını gerçekleştirilen kombinezonsal devrenin 5 adet kopyası birbirine kas kat olarak bağlanmış ve çıkışına da bir adet kaydedici bloğu eklenmiştir. Kaydedici bloğu ve ters alma işleminin 5 adımını gerçekleştiren kombinezonsal devre bloğu çiftinden 6 adet kas kat bağlanmıştır. Bu kas kat bağlama sonucu toplam olarak ters alma işleminin 32 adımından 30 adımı gerçekleştirilmiş olur. Son olarak da 2 adımı gerçekleştiren kombinezonsal devrenin bu kas kat yapıya bağlanması sonucu gerçekleştirilmesi gerekli 32 adım tamamlanmış olur. Senkronizasyon işlemi sırasında kullanılacak olan 7 saat darbesi sonra çıkışa girişin evriğini veren ters alma devresine ait blok şema şekil 4.8, 4.9, 4.10 da verilmiştir.



Şekil 4. 10 ters alma işlemi için gerekli 5 adımı gerçekleştiren devreye ait blok şema



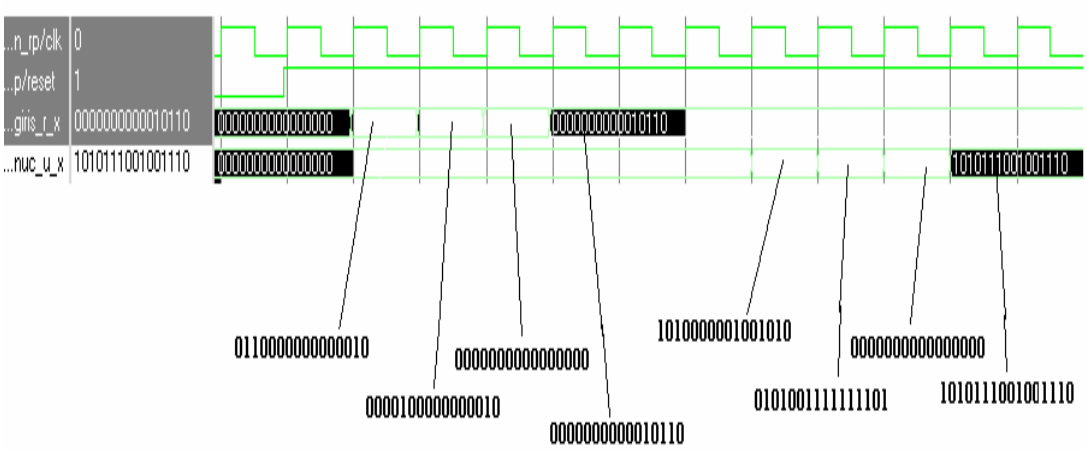
Şekil 4. 11 ters alma işlemi için gerekli 2 adımı gerçekleştiren devreye ait blok şema



Şekil 4. 12 senkronizasyon işlemi sırsında kullanılan ters alma bloğuna ait kaskat blok şema

Bu devrenin sentezlenmesi sonucu elde edilen sonuçlar uyarınca senkronizasyon aşamasında kullanılacak olan ters alma bloğunun çalışabileceği maksimum frekansın 33.710 MHz olduğu görülmüştür. Görüldüğü gibi senkronizasyon aşamasında kullanılacak olan ters alma

devresinin en uzun kombinezonsal yolu anahtar üretimi sırasında kullanılacak olan ters alma bloğunun en uzun kombinezonsal yoluna nazaran daha uzun olduğundan gecikme daha fazla olur ve böylece devrenin çalışabileceği maksimum frekans düşer. Devreye ait simülasyon sonuçları şekil 4.13 de verilmiştir.



Şekil 4. 13 senkronizasyon işlemi sırsında kullanılan ters alma bloğuna ait simülasyon sonucu

Simülasyon sonucundan da görüleceği gibi girişin gelmesinden 7 saat darbesi sonra evrik değer hesaplanmış ve çıkışa verilmiştir.

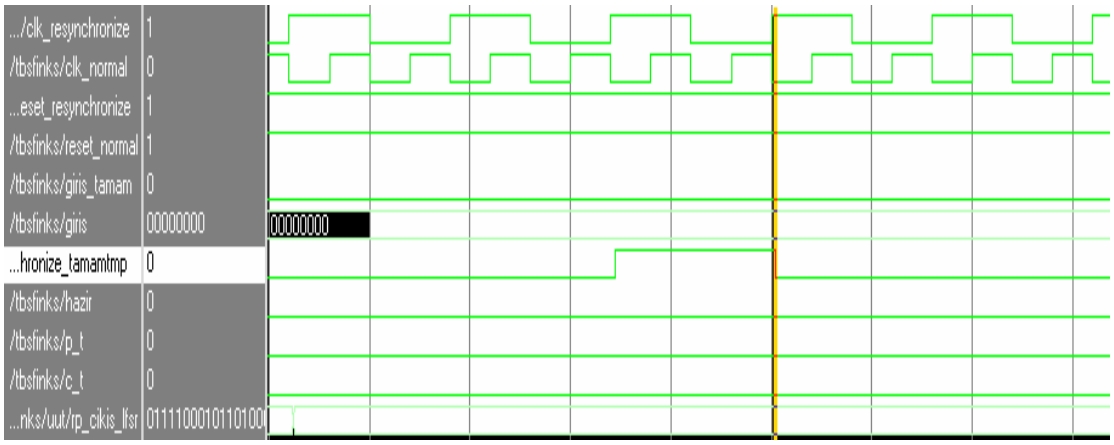
4.3.3 LFSR ve ters alma bloklarının birleştirilmesi

SFINKS algoritmasının gerçekleştirilmesi sürecinde bölüm 4.3.1 ve 4.3.2 de ayrıntıyla ele alındığı üzere bir adet LFSR bloğu tasarlanmıştır, ayrıca senkronizasyon aşamasında ve anahtar üretim aşamasında kullanılmak üzere iki adet birbirinden farklı ters alma bloğu tasarlanmıştır. Senkronizasyon aşamasında SFINKS dizi şifreleyici devresinin, normal üretim aşamasına oranla çok daha yavaş çalışacağı görülmüştür. Senkronizasyon işleminin sadece ihtiyaç duyulduğunda yapılacağı göz önüne alınarak SFINKS dizi şifreleyicisi devresinin senkronizasyon bloğu ve anahtar üretim bloğu olmak üzere birbiriyle haberleşebilen iki bloktan oluşturulmasına karar verilmiştir.

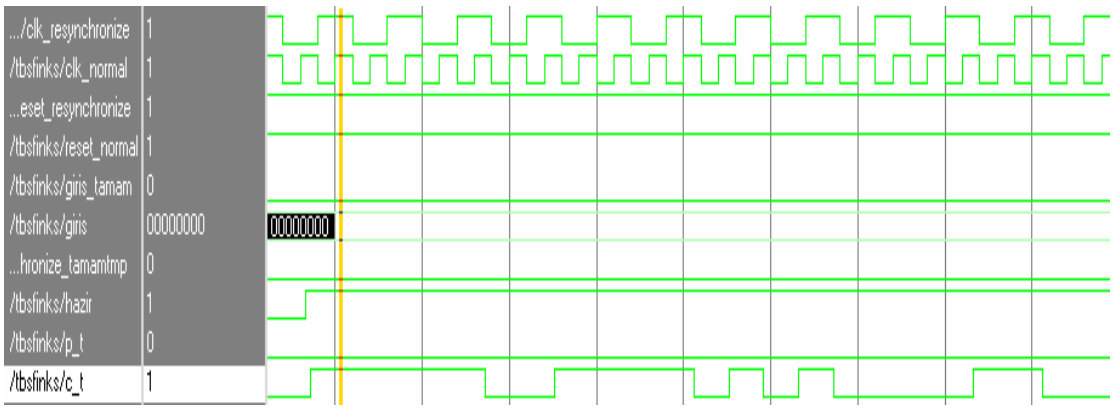
Bir sistemin hızının sisteme ait en yavaş çalışan devre tarafından belirlendiği göz önüne alınarak, SFINKS dizi şifreleyicisinin tek saat işareti kullanılması durumunda senkronizasyon bloğunun hızında çalışacağı açıktır. Bu sorunun giderilmesi amacıyla sistemin iki saat işareti ile çalıştırılmasına karar verilmiştir. Böylece yavaş çalışan senkronizasyon bloğu tasarımın hızını belirlemeyecek ve devrenin yüksek hızlarda çalışması mümkün olacaktır.

Senkronizasyon işleminin tamamlanması ile senkronizasyon bloğu anahtar üretim bloğuna işlemin bittiğini haber verecek ve senkronizasyon bloğunun çıkışındaki LFSR, R ve M kaydedicilerinin içeriğinin anahtar üretim bloğu tarafından alınmasını sağlayacaktır.

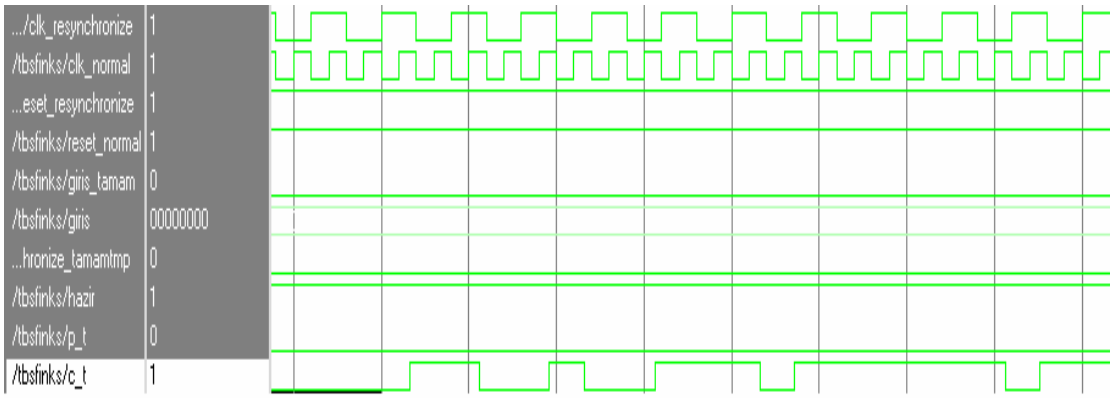
Senkronizasyon ve anahtar üretim blokları LFSR ve ters alma bloklarının uygun bir şekilde birleştirilmesi ile ayrı ayrı tasarlanmıştır. Tasarımı gerçekleştirilen senkronizasyon ve anahtar üretim bloklarının kaskat bağlanması ile SFINKS dizi şifreleyici devresinin tasarımı gerçekleştirilmiştir. SFINKS devresinin sentezleme işlemi yapılmış ve sentezleme işleminin ardından devrenin FPGA üzerine serimi ve devreye ait giriş çıkış pinlerinin atama işlemi yapılmıştır. Tüm bu işlemlerin gerçekleştirilmesi ile FPGA içerisine yerleştirilen devrenin serim sonrası simülasyonu yapılmıştır.



Şekil 4. 14 SFINKS devresinin serim sonrası simülasyon sonucu



Şekil 4. 15 SFINKS devresinin serim sonrası simülasyon sonucu



Şekil 4. 16 SFINKS devresinin serim sonrası simülasyon sonucu

16'lık tabanda K="00000000000000000001" ve IV="00000000000000000000" değerleri için gerçekleştirilen serim sonrası simülasyon sonuçları şekil 4.14, 4.15, 4.16' te verilmiştir. Şekil 4.14' te senkronizasyon işleminin tamamlanması ile senkronizasyon bloğuna ait 'resynchronize_tamam' çıkışı lojik '1' olmuş ve böylece LFSR, R ve M kaydedicilerinin içeriği anahtar üretme bloğuna aktarılmıştır. Şekil 4.15 ve 4.16' te ise anahtar bitlerinin 'hazır' çıkışının lojik 1 olmasıyla üretilmeye başlandığı anlaşılır ve anahtar bitleri okunmaya başlanır.

Tasarımı ve gerçeklenmesi tamamlanan SFINKS dizi şifreleyicisinin FPGA üzerinde serim ve yollandırma aşamalarının tamamlanması ile devrenin son haline ait zaman ve alan raporları incelenmiştir. Elde edilen raporları özetlemek gerekirse; anahtar üretim bloğuna ait maksimum saat frekansı (clk_normal) 120 MHz, senkronizasyon bloğuna ait maksimum saat frekansı(clk_resynchronize) 33 MHz' dir. Bu sonuçlara göre SFINKS devresi senkronizasyon sırasında 33 MHZ ile çalışabilirken, anahtar üretim aşamasında 120 MHZ' e kadar olan hızlarda çalışabilir. Ayrıca SFINKS devresinin FPGA üzerinde gerçeklenmesi sonucu Virtex-E (XCV1000E) yapısında bulunan 24576 flip flop'un 3004 adetini, 24576 adet LUT hücresinin 11205 adetini kullanılmışır.

4.4 SFINKS dizi şifreleyici devresinin test edilmesi

SFINKS dizi şifreleyici devresinin senkronizasyon ve anahtar üretim aşamasında karmaşık işlemler gerçekleştirdiği, ayrıca farklı anahtar tohumu ve başlangıç vektörleri için elle test edilebilmesi mümkün olmadığından doğruluğunun kontrol edilebilmesi amacıyla MATLAB' de algoritmaya ait kod yazılmıştır. Testin yapılabilmesi için bir adet anahtar tohumu dizisi ve başlangıç vektörü örneği alınmıştır. Alınan örnek başlangıç vektörü ve anahtar tohumu dizisi

16’lık tabanda K=”00000000000000000001” ve IV=”00000000000000000000” ‘dır. MATLAB kodunun çıktıları ile SFINKS dizi şifreleyici devresinin yaklaşık 6 saat süren simülasyonu sonrası elde edilen çıktıları karşılaştırılmıştır. Yapılan bu karşılaştırma sonucu 2 çıktının da birbirleri ile uyumlu olduğu gözlenmiştir. Böylece SFINKS devresinin doğru olarak çalıştığı anlaşılmıştır.

5. SONUÇ

Bu çalışmada bir senkron dizi şifreleyicisi olan SFINKS dizi şifreleyicisinin, sayısal devre tasarımı amacıyla kullanılan donanım tanımlama dillerinden VHDL ile yazılımı ve yazılan VHDL kodunun donanım olarak FPGA üzerinde gerçeklenmesi konusunda çalışılmıştır. SFINKS algoritması daha önceden gerçekleştirilmemiş olması ve Avrupa Birliği Standardının seçilmesi amacıyla düzenlenen ECRYPT Dizi Şifreleme Projesi' ne aday olan algoritmalardan bir tanesi olması bakımından önemli bir yere sahiptir. İlk olarak bloklar halinde gerçeklenen devre, daha sonra blokların uygun bir şekilde birleştirilmesi ile SFINKS devresi elde edilmiştir. Tasarlanan blokların birleştirilmesi ile FPGA üzerinde yerleştirme ve yollandırma işlemleri gerçekleştirilmiş ve devreye ait alan ve zaman bilgileri elde edilmiştir.

Çalışma sırasında dizi şifreleyicilerin genel özelliği olan yüksek frekanslarda çalışabilme karakteristiğine uygun olarak SFINKS devresinin daha hızlı çalışabilmesi amacıyla alternatif gerçeklemeler yapılmıştır. Yapılan alternatif gerçeklemeler sonucu devre daha yüksek hızlarda çalışabilir hale getirilmiştir ve böylece devre beklenen hızlarda çalışmaya başlamıştır.

KAYNAKLAR

[1] **Stinson, D.R.,** 2002. Cryptography, Chapman & Hall/CRC Press Company, United States of America.

[2] **Hsu, Y.C., Tsai, K.F., Liu J.T. and Lin, E.S.,** 1995. VHDL Modeling For Digital Synthesis, Kluwer Academic Publishers, Riverside.

[3] **Savaş, E.,** 1994. Dizi Şifreleme Sistemleri ve Doğrusal Karmaşıklık, *Yüksek Lisans Tezi,* İ.T.Ü. Fen Bilimleri Enstitüsü, İstanbul.

[4] **Menezes, A., Van Oorschot, P.,Vanstone, S.,** 1996. Handbook of Applied Cryptography, CRC Press Company.

[5] **Apohan, A.M.,** 1993. Kriptoloji, *Yüksek Lisans Tezi,* İ.T.Ü. Fen Bilimleri Enstitüsü, İstanbul.

[6] **Braeken, A., Lano, J., Mentens, N., Prenel, B., Verbauwhede, I.,** 2005. SFINKS :A synchronous Stream Cipher for Restricted Hardware Environments , Belgium.

[7] **Brunner H., Curiger A., Hofstetter M.** , 1993. On Computing Multiplicative İnverses in GF(2^m), IEEE Transactions on Computers, **42,** 1010-1015.

AHMED YASİR DOĞAN
Denizabdal Mah. Denizabdal Çesme sok., No:17\21 Fatih - İSTANBUL
Phone: (0212) 587 34 40 Mobile: (0536) 361 66 95 e-mail :ahmed.dogan@gmail.com

EDUCATION

2002-Present	**Electronics Engineering, Istanbul Technical University** GPA 3.64 / 4.00	Istanbul
2001-2002	1 year **English Preparation** class**, Istanbul Technical University**	Istanbul
1998-2001	**Ankara Science High School** GPA 4.80 / 5.00	Ankara

WORK EXPERIENCE

2005	**Siemens A.G.** Summer Internship 1 month Industrial Solutions Programming with PLC and Controlling motor by Micromaster inverter	Istanbul
2005	**Eta Tasarım Merkezi** Summer Internship 1 month VLSI design by using CAD programs such as CADENCE	Istanbul
2003	**Nokta Endüstriyel Otomasyon** Summer Internship 1 month Industrial Automation with PLC	Istanbul

LANGUAGES

English (Reading and Writing fluently)

COMPUTER SKILLS

- Operating Systems: Windows98, Windows XP Professional, Microsoft Office Programs
- Programming with C language
- Pspice ,MicroSim and Cadence electronic CAD programs
- Programs about electronics engineering such as VHDL,VERILOG, MWOffice

SEMINARS AND COURSES TAKEN

2004	**Cisco CCNA** course Mayasoft Bilişim 2 months	Istanbul
2005	Programming with **C#** Elektrik Mühendisleri Odası 1 month	Istanbul