YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DEĞİŞKEN UZUNLUKLU RSA ŞİFRELEME SİSTEMLERİNİN TASARIMI ve GERÇEKLENMESİ

Müh. Emin Aytaç DERELİOĞLU

FBE Elektronik ve Haberleşme Anabilim Dalı Elektronik Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Prof. Dr. Atilla ATAMAN

Prof. Dr. Günhan DÜNDAR

Prof. Dr. Vedat TAVŞANOĞLU

İSTANBUL, 2005

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

C	ChipherText, şifreli mesaj
d	Açık anahtar
e	Özel anahtar
n	Modül değeri
P	PlainText, şifrelenmemiş veya şifresi çözülmüş mesaj
p, q	Asal sayılar
r	Radix değeri

KISALTMA LİSTESİ

ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
IEEE	Institute of Electrical and Electronic Engineers
LUT	Look Up Table
MIT	Massachusetts Institute of Technology
PCB	Printed Circuit Boards
PCI	Peripheral Component Interconnect
PKC	Public-Key Cryptosystems
RAM	Random Access Memory
RSA	Rivest, Shamir, Adleman
VHDL	Very High Speed Integrated Circuits Hardware Description Language

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖZET

Bu tez çalışmasında, dünyada en çok kullanılan açık anahtarlı şifreleme sistemi olan RSA algoritmasının gerçeklenmesi yapılmıştır. Klasik simetrik şifreleme algoritmalarına kıyasla, internet gibi kullanıcı sınırı konulamayan sistemlerde, anahtar dağıtımı açısından çok avantajlı olan bu algoritma yapısı, teknolojinin gelişmesi ve daha güvenli sistemlere duyulan ihtiyaç doğrultusunda her geçen gün kapasitesini arttırmaktadır. Matematiksel olarak modüler üs alma işlemini gerçekleştiren algoritmada, günümüzde güvenlik açısından çok büyük sayılar kullanılması gerekmektedir, ancak kullanılan büyük sayılar daha fazla tümdevre alanı demektir. Bu tasarımcının daha az alan kullanarak daha hızlı çalışan sistemler oluşturması gerekliliğini ortaya koyar.

Bu çalışma içerisinde RSA sisteminin alternatif matematiksel yöntemleri ve bu yöntemlerin literatür çalışmaları doğrultusunda en az donanımsal yapıyla nasıl gerçeklenebilecekleri araştırılmıştır. Çalışmalar sonucunda karmaşıklığı en aza indirmek amacıyla birbirinin aynı yapıların ard arda eklenmesi ile oluşan ve yol gecikmelerinin optimum olduğu sistolik yapı tercih edilmiştir.

Tez çalışmasında, değişken uzunluklu RSA sistemlerinin, yazılımdaki tek bir değişkenin değerinin değiştirilmesi ile farklı bit uzunlukları için uyarlanabileceği gösterilmiş, 256 bitlik RSA sistemi için ise yapı donanımsal olarak gerçeklenmiştir. Kişisel bilgisayarlara bağlanabilen RSA şifreleme veya şifre çözme modülü ile gerçek zamanlı işlemlerin yapılabildiği gösterilmiş ve çalışma içerisinde örneklerle sunulmuştur.

Anahtar Kelimeler: RSA, kriptoloji, simetrik ve asimetrik sistemler, açık ve özel anahtar, binary yöntemi, modüler çarpma, modüler üs alma, montgomery yöntemi, sistolik yapılar

ABSTRACT

In this thesis, the most popular public-key crypto-system, RSA, was implemented successfully. In comparison with the classical symmetric crypto-systems, this algorithm has a big advantage over key-distribution problem especially the systems that have limitless users like internet. With the improvement of technology and the necessity of more security, RSA systems improve its capacity continuously. Even today's security conditions make designers use very very big numbers(1024,2048.. bits), but the bigger numbers you use, the larger silicon area you need, so the designers try to develop more secure systems in a smaller chip area.

Throughout this work, the alternative mathematical methods and optimum hardware structures of these mathematical methods were searched for the better implementations of RSA systems. Finally, the Montgomery systolic architecture that is composed of the same architecture units connected in an ordered way with each others was chosen to decrease the hardware complication and routing delays.

This thesis presents the variable length RSA systems can be adapted for different bit lengths by changing only one variable in the software and this structures hardware implementation was completed for 256 bits RSA system. At the end, real-time applications were presented with the encryption and decryption module that can be connected over PCI slot with personal computers and an example was shown in this thesis.

Key Words: RSA, cryptology, symmetric and asymmetric systems, public and private keys, binary method, modular multiplication, modular exponentiation, montgomery method, systolic architecture

ÖNSÖZ

Tüm yüksek öğrenimim boyunca değerli desteklerini esirgemeyen hocam Sn. Prof. Dr. Atilla ATAMAN'a, yüksek lisans çalışmalarım için verdikleri izinler ve yardımlar için TUBİTAK-UEKAE'deki yöneticilerime, tez çalışmama katkılarından dolayı değerli çalışma arkadaşlarım Sn. Levent ORDU ve Sn. Eşref DERİNKAYA'ya, son olarak da manevi desteklerinden ötürü sevgili aileme teşekkür ederim.

Haziran 2005 Emin Aytaç DERELİOĞLU

1. GİRİŞ

İletişim ve bilgisayar teknolojilerindeki büyük gelişmeler farklı noktalardaki insanlar ve bilgisayarlar arasında çok hızlı ve etkili haberleşmeyi mümkün kılmaktadır. Teknolojiyle ortaya çıkan elektronik posta, internet, elektronik para transferleri, elektronik ticaret gibi çeşitli hizmetler, hız ve maliyet açısından geleneksel sistemlere göre büyük avantajlara sahiptir. Bu tip servislerin kullanım alanlarının artması, bilgi alışverişlerinde alıcı ve gönderici tarafların doğrulanması, bilgi gizliliğinin korunması gibi temel gereksinimleri ortaya çıkarmaktadır. Şifreleme teknolojileri bu aşamada devreye girerek ihtiyaç duyulan güvenli ortamı sağlamaya çalışırlar. Örnek olarak açık anahtarlı şifreleme sistemlerinde kullanılan dijital imza gösterilebilir, bu uygulama ile alınan datanın göndericisinin kim olduğu ve datanın transfer esnasında bütünlüğünün bozulup bozulmadığı doğrulanabilir.

İlerleyen bölümlerde daha ayrıntılı olarak değineceğim şifreleme sistemleri temelde iki gruba ayrılır. Bunlar gizli-anahtarlı(simetrik) şifreleme sistemi ve açık-anahtarlı(asimetrik) şifreleme sistemi olarak isimlendirilir. Bu iki sistemden gizli-anahtarlı şifreleme sisteminin güvenliği, algoritmasının ve anahtarlarının gizliliğine dayanmaktadır. Bu sistemler özgün algoritması ile ağırlıklı olarak sınırlı kullanıcılara sahip organizasyonlar(ulusal güvenlik birimleri, askeri birimler vb.) için kullanılmaktadır. Sistem gizlilik koşulları sağlandığı müddetçe güvenlik açışından daha sağlam bir yapıya sahip olmakla birlikte sisteme dahil olacak birimlere anahtar dağıtılması zorluğunu bünyesinde barındırmaktadır. Sistemde haberleşecek kişilerin önceden bir araya gelerek ortak bir anahtar üzerinde anlaşmaları gerekmektedir. Yine de sınırlı sayıda alt birimlere sahip olan organizasyonlara güvenli anahtar dağıtımı sağlanabilir ancak internet gibi milyonlarca kullanıcısı olan ve her gün yeni kullanıcılarla tanışan sınırsız bir dünyada, anahtar dağıtım işlemini bu şekilde gerçekleştirmek imkansızdır. Bu zorluk açık-anahtarlı şifreleme sistemlerinin ortaya çıkmasını gerektirmiştir.

İlk olarak Whitfield Diffie ve Martin Hellman, yayınladıkları bir makale (Diffie ve Hellman,1976) ile açık-anahtarlı şifreleme sistemi fikrini ortaya atmışlar, dijital imzayı ve bu yapıların güvenlik sınırlarını tartışmışlardır. Bu fikir, alıcı ve gönderici tarafların önceden bir araya gelerek ortak bir anahtar üzerinde anlaşmaya ihtiyaç duymadan güvenli bir şekilde haberleşebilmelerini sağlamayı hedeflemektedir. Bu yayını takiben Rivest, Shamir, Adleman üçlüsü ilk açık-anahtarlı şifreleme sistemi olarak RSA'i önermişlerdir (Rivest vd.,1978). RSA sisteminin güvenliği çok büyük sayıların çarpanlarına ayrılmasının zorluğuna dayandırılmaktadır. RSA şifreleme sistemi zamanla en çok bilinen ve kullanılan açık-anahtarlı şifreleme sistemi olmuştur. Sistemin geliştirilmesi ve kırılmasına yönelik çok sayıda makale

yayınlanmıştır.

Bu tez kapsamında öncelikli olarak şifreleme sistemleri ve dijital imzalar üzerine genel bilgiler sunulmuş, ardından tez çalışmasının temelini oluşturan RSA şifreleme sistemi ayrıntılı olarak ele alınmıştır. RSA sisteminin standart matematiksel yapısı, sayısal elektronik tasarımlar için uygun olmadığından gerçeklemeye yönelik matematiksel çalışmalar araştırılmış ve araştırma sonucunda seçilen en uygun yapı ile FPGA gerçeklemesi farklı bit uzunluklarına sahip sistemler için başarı ile tamamlanmıştır.

Simülasyon sonuçlarının doğruluğu görüldükten sonra, VHDL dili ile yazılmış program PCI bağlantısı mevcut bir PCB üzerindeki Xilinx firmasının FPGA entegrelerine (XCV400E ve XC2S200) yüklenmiş ve gerçek zamanlı olarak PCB'nin bilgisayardan gelen verileri şifreleyerek veya şifrelerini çözerek tekrar bilgisayara geri gönderebildiği gösterilmiştir.

Sonuçta; bu donanıma sahip farklı noktalardaki birçok bilgisayarın anahtar dağıtım problemi yaşamadan birbirleri ile RSA şifreleme sisteminin sağladığı güvenlik çemberi altında haberleşebileceği ortaya konulmuştur.

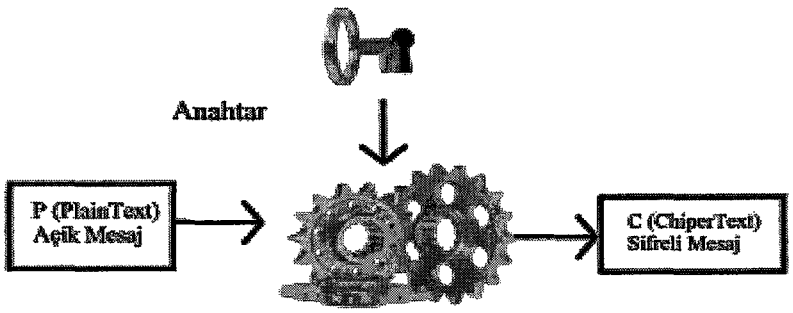
2. ŞİFRELEMENİN TEMELLERİ

Basit olarak güvenli olmayan bir kanal üzerinden iki nokta arasında güvenli haberleşmeyi sağlama bilimi olarak nitelendirilebilen şifreleme (kriptografi) ve böyle gizli bir haberleşme düzeneği üzerinden bilgi edinme bilimi olarak nitelendirilebilen şifre kırma (kripto analiz) kavramlarınının bütününe şifre bilimi (kriptoloji) adı verilir.

Şifreleme ile ilgili kaynaklar incelendiğinde standart bazı gösterimler kullanıldığı görülür Tez kapsamında da kullanılan bu gösterimleri bu aşamada belirtmek faydalı olacaktır. Şifrelenecek mesaj için genellikle *plaintext* (*P*) veya *cleartext* kullanılır. Şifreleme işlemi *encryption* olarak adlandırılır ve fonksiyon gösteriminde E(), şifreleme işlemini tanımlar. Şifrelenen mesaj ise *ciphertext* olarak isimlendirilir ve *C* ile gösterilir.. Aynı şekilde şifre çözme işlemi *decryption* olarak adlandırılır ve D() şeklinde sembolize edilir.

2.1 Anahtarlar ve özellikleri

Evimizin içindekileri korumak için kapımızı anahtarla kilitleriz. Temelde, anahtar asıl koruma görevini yapacak olan kapının açılmaması için üzerindeki mekanizmayı harekete geçiren bir araçtır. Kapıyı bir yönde kapatır, diğer yönde açarız. Tıpkı bu işlem gibi şifreleme algoritması bir kapıdır ve onu gizli numaralardan oluşan bir anahtarla kilitlememiz gerekir. Algoritma, aşama aşama anahtarın verilerini kullanarak şifreleme işlemini gerçekleştirir. Ev anahtarlarında olduğu gibi ancak doğru anahtar kapıyı açar yada doğru anahtar şifre çözme işlemini gerçekleştirebilir.



Şekil 2.1 Anahtar kullanımı

Şekil 2.1'de simetrik algoritma tipi gösterilmiştir. Bu algoritmalarda şifreleme ve şifre çözme işlemleri için kullanılan anahtarlar aynıdır. Bu nedenle, bu tip yapılara iki tarafta da aynı anlamını vermek için simetrik isimlendirilmesi yapılmıştır.

Anahtar konusunda zaten şifreleme yapan bir algoritma var neden ikinci bir bilgiye yani anahtara ihtiyaç duyuluyor sorusu akla gelebilir. Bunun nedeni hangi algoritma olursa olsun

onun asla kırılamaz olduğu iddia edilemez, her algoritma belirli bir zaman içinde kırılabilir. İşte bu noktada anahtar devreye girer, kod-kırıcı şifreleme algoritmasını kırmış olsa bile halen anahtar bilgisine sahip olmadığı için mesaja veya bilgiye erişimi mümkün değildir. Keza siz ne kadar güvenilir olursanız olun, sizin birlikte çalıştıklarınız veya onların birlikte çalıştıkları kişiler sizin kadar güvenli olmayabilir. Bir de genelde anahtar bilgisi algoritma ile karşılaştırıldığında çok daha küçüktür ve bu bilgiyi gizli tutmak algoritma bilgisini gizli tutmaktan daha kolaydır. Bunlara ek olarak farklı bilgilerinizi farklı anahtarlarla şifreleyebilirsiniz, bunun anlamı anahtarlarınızdan biri kırılsa bile bilginizin halen güvenli olabileceğidir.

Anahtarın bu kadar önemli olduğu bilgisinin ardından, kod-kırıcının algoritmayı elde ettikten sonra şifre kırma işlemi için sizin anahtarınızın da ne olduğunu bilmesinin gerekliliğini kavramış oluyoruz. Anahtarı kırmaya ilk yaklaşım "kaba saldırı" (brute-force attack) olarak isimlendirilen, anahtarı doğrudan algoritmada yerine koyup deneme yöntemidir. Bu yöntemle olası anahtarlar tek tek denenir ve sonuç gözlenir. Denenen anahtar ile sonucun anlamlı olup olmadığı genellikle hafızasında anlamlı kelimeler bulunan bilgisayar programları ile yapılır, program yan yana gelen harflerin uygun bir harf serisine sahip olup olmadığını test eder. Deneme sonucu uygunsa kalır, değilse bir sonraki anahtara geçer. Genellikle tek bir anahtarı denemek için gereken süre çok kısadır. Kod-kırıcının yazdığı program bir saniyede bir çok anahtarı deneyebilir ve sonunda tüm anahtarları belirli zaman sonunda denemiş olur. Bir de doğru anahtarın nerede çıkacağı belli değildir, belki ilk denemede belki de son. Genelde doğru anahtarı bulma başarısının tüm anahtar sayısının yarısına yakın noktalarda olduğu gözlenmiştir.

Anahtarın deneme-yanılma yolu ile bulunmasını zorlaştırmak için anahtar uzunluğunun seçimi önemlidir, örneğin biz 0 ile 100 milyar arasında bir değer olan anahtarımızı 6 ay kullanıyorsak ama kod-kırıcı o anahtarı 3 aylık deneme sonucunda elde edebiliyorsa bu sistem güvenli değildir, bizim anahtarımızın boyunu uzatmamız, deneme aralığımızı da örneğin 0 ile 100 milyar milyar milyar arasında yapmamız gerekmektedir. Anahtar işlemleri için kullanılan sayılar 40,56,128 bit vb uzunluklarda olabilir. 40 bit anahtar uzunluğu demek olası anahtarların 0 ile 1 trilyon arasında , 56 bit anahtar uzunluğu demek olası anahtarların 0 ile 72 kuatrilyon arasında olduğu anlamına gelmektedir. 128 bitlik anahtar için ise sadece 128 bitlik anahtar demek daha kolaydır (Burnett ve Paine, 2001). Anahtar boyuna eklenen her bir bit deneme-yanılma yöntemi için kullanılacak zamanın iki katına çıkması demektir. Mesela 64 bitlik bir anahtarı denemek 4 saat sürüyorsa, 65 bitlik bir anahtarı denemek 8 saat

sürecektir çünkü eklenen bit en ağırlıklı bit olarak bize iki olasılık ('0' yada '1') dolayısıyla iki kat anahtar uzayı sunar .

Kısaca, şifre kırma işini zorlaştırmak için daha büyük anahtarlar kullanmamız gerekmektedir. Daha uzun anahtar, daha güvenli sistem demektir. RSA Laboratuarları bazı zamanlarda farklı anahtar uzunluklarının kullanıldığı mesajlarının kırılması için para ödüllü yarışmalar düzenlerler. Bu yarışmalar esansında 1997 yılında 40 bit anahtar uzunluğuna sahip sistemin anahtarının 3 saatte, 48 bitlik sistemin anahtarının 280 saatte bulunduğu gözlenmiştir. 1999'da yapılan yarışmada ise 54 bitlik sistemin anahtarı 24 saate çözülmüştür ve bu sistemlerin hepsinde sonuç daha tüm olasılıkların yarısına ulaşılmadan elde edilmiştir.

Anahtar kırma işlemi için kullanılan yapının ne olduğunu ancak tahmin edebiliriz ve bu yapının mevcut tüm işlemcilerinin tek bir amaç için, sadece anahtar denemek için çalıştığını düşünmeliyiz. Mesela elimizdeki bir sistemin 56 bit uzunluğundaki bir anahtar uzayının %1'lik kısmını 1 saniyede taradığını farz edelim. Bir de yine bu sistemin, 56 bitlik yapının %50'sini 1 dakikada yaptığını farzedelim. Bu hesap genişletildiğinde 90 bit için aynı sistemin 321 asır çalışması gerektiği görülür. Anahtarın bit uzunluğu ve sürenin birbirleri ile olan ilişkileri Şekil 2.2'de gösterilmektedir.

Table 2-1 A Worse Than Worst-Case Scenario: How Long a Brute-Force Attack Will Take for Various Key Sizes

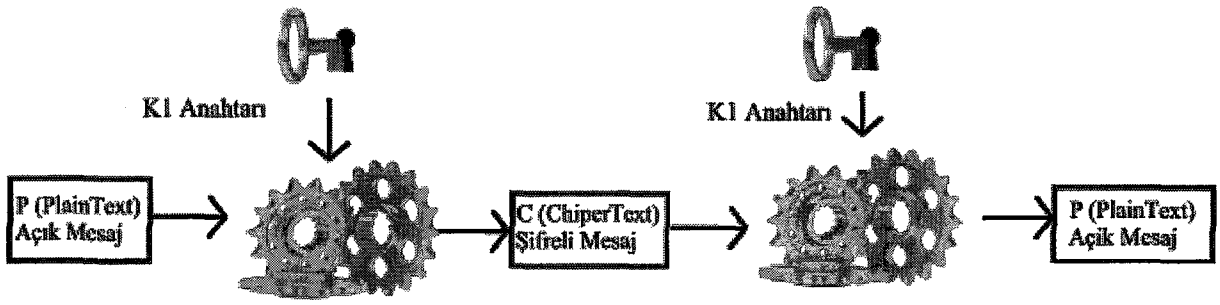
Bits	1 percent of Key Space	50 percent of Key Space
56	1 second	1 minute
57	2 seconds	2 minutes
58	4 seconds	4 minutes
64	4.2 minutes	4.2 hours
72	17.9 hours	44.8 days
80	190.9 days	31.4 years
90	535 years	321 centuries
108	140,000 millennia	8 million millennia
128	146 billion millennia	8 trillion millennia

Şekil 2.2 Anahtar uzunluğu ve kırma süresi ilişkisi (Burnett ve Paine, 2001)

2.2 Şifreleme algoritmaları

2.2.1 Simetrik şifreleme algoritmaları

Anahtar yapısına ve kullanımına bağlı olarak algoritmalar simetrik ve asimetrik olmak üzere ikiye ayrılırlar. Simetrik algoritmalarda şifreleme ve şifre çözme işlemleri aynı anahtar ile gerçekleştirilmektedir.



Şekil 2.3 Simetrik şifreleme yapısı

Simetrik algoritmalar daha farklı isimlerle de adlandırılmaktadırlar; geleneksel algoritmalar, gizli-anahtar algoritmaları, tekil-anahtar algoritmaları bunlardan bazılarıdır. Bu tip algoritmalarda taraflar gizli haberleşmeye başlamadan önce şifreleme işlemlerinde kullanacakları ve önceden üzerinde anlaşma sağladıkları bir anahtara ihtiyaç duyarlar. Bu anahtar güvenli olduğu kabul edilen yol üzerinden taraflara dağıtılır. Haberleşmenin gizliliği öncelikle anahtarın gizliliğine bağlıdır.

Simetrik algoritmalarda şifreleme ve şifre çözme işlemleri sembolik olarak denklem (2.1) ve (2.2) de gösterilmiştir.

$$E_K(M) = C \tag{2.1}$$

$$D_K(C) = M \tag{2.2}$$

Simetrik algoritmalar kendi içerisinde dizi ve blok şifreleme algoritmaları olarak ikiye ayrılırlar. Dizi şifreleme algoritmaları açık metindeki bit veya sekizliler üzerinde, blok şifreleme algoritmaları ise sabit uzunluktaki bloklar üzerinde işlem yaparlar.

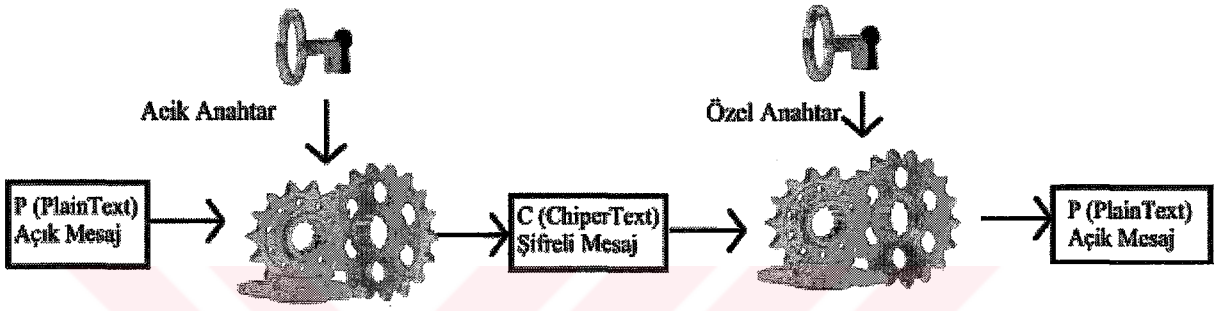
Aslında simetrik şifreleme sistemlerinin ne yaptığını zihnimizde canlandırmak kolaydır. Anahtar kullanılarak adım adım izlenen yol ile karıştırılmış çıkış datası elde edilir. Şifreyi çözmek içinde aynı işlem tersinden yapılmalıdır. Yapılan en son şey veriyi (data) kaydırmak ise çözerken yapılacak ilk şey de veriyi tersi yönde bir o kadar kaydırmak olacaktır. Açık-anahtarlı sistemlerde durum buradakinden çok farklıdır. Çünkü yapılan işlem ancak matematiksel bir temelde açıklanabilir, basit olarak tersi işlem yapmak bizi sonuca ulaştırmaz.

2.2.2 Asimetrik şifreleme algoritmaları

Açık anahtar şifreleme algoritmaları olarak da bilinen asimetrik algoritmaların en önemli özelliği şifreleme ve şifre çözme işlemleri için farklı iki anahtara ihtiyaç duymasıdır. Açık-anahtar şifreleme algoritmaları denmesinin sebebi ise şifreleme anahtarının herkesçe

bilinmesinde bir sakınca olmamasıdır. Şifreleme anahtarından şifre çözme anahtarının elde edilebilmesinin, günümüz işlem gücü göz önüne alındığında oldukça zaman gerektiren bir çalışma olması bu tip şifreleme sistemlerinin güvenlik temelini oluşturmaktadır.

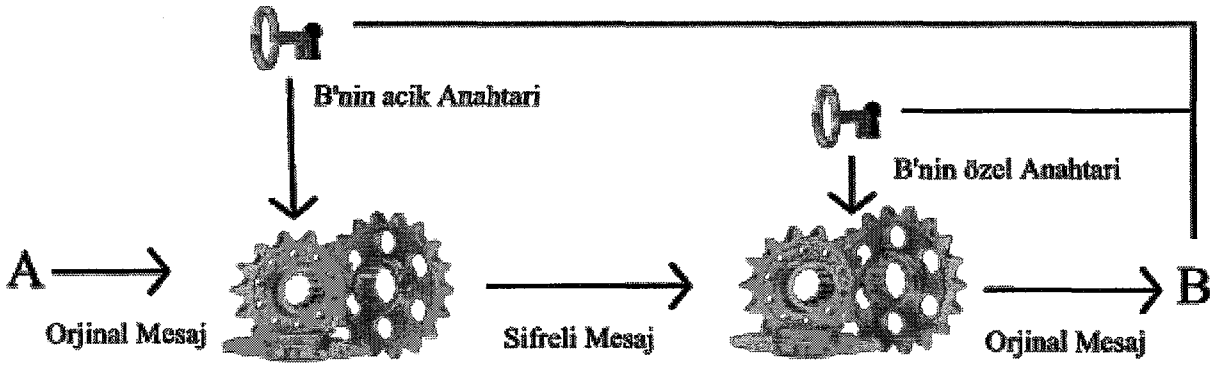
Asimetrik şifreleme sistemlerinde, şifreleme anahtarı genellikle açık anahtar, şifre çözme anahtarı ise özel anahtar olarak adlandırılır. Özel anahtara, gizli anahtar da denilmekle birlikte simetrik algoritmalarda kullanılan gizli anahtarla karıştırılmaması açısından özel anahtar adının kullanılması daha doğrudur. Açık-anahtar ve özel anahtar kullanılarak yapılan işlem Şekil 2.4'de gösterilmiştir.



Şekil 2.4 Asimetrik şifreleme yapısı

Asimetrik şifreleme sistemleri, simetrik sistemler anlatılırken değinildiği gibi matematiksel bir temele sahiptir ve matematiksel içeriği değişmediği müddetçe hep aynı prosedürü aynı yönde işletir. Bu açıdan matematiksel işlem, dolayısıyla da asimetrik sistemler tek yönlüdür denilir. Asimetrik sistemlerin bu özelliği bazı kaynaklarda bir tek kaçış kapısına sahip tek yönlü kapalı bir yol olarak ifade edilmektedir, buradan geriye hareket ederek sonuca ulaşamazsınız, sonuca ulaşmak için ancak tek kaçış kapısını kullanabilirsiniz ki onunda anahtarı özel anahtardır.

Açık-anahtarlı(asimetrik) şifreleme sistemlerinin iki farklı anahtara (şifreleme ve şifre çözme) sahip olmaları onları anahtar dağıtımı konusunda simetrik şifreleme sistemlerinden çok avantajlı bir konuma getirmiştir. Açık-anahtarlı yapılarda sisteme giren her kişi mesajlarını şifreleyebilmek ve çözebilmek için bir anahtar çifti üretir. Her sistemde kişilerin anahtarlarından birisi herkese açık bir veritabanına kaydedilir, diğer anahtar ise anahtarı üreten tarafından gizli tutulur. Eğer sistemdeki A kişisi B kişisine mesaj yollamak isterse mesajı veritabanından temin edebileceği B'nin açık-anahtarı ile şifreler. Şifrelenen mesaj B'ye gönderildikten sonra B, o mesajı kendi özel anahtarı vasıtası ile çözer ve mesajı güvenli bir şekilde elde etmiş olur (Şekil 2.5). Gönderim esnasında mesaj başkası tarafından alınsa bile kişi B'nin özel anahtarına sahip olmadığı için mesajı çözemez.



Şekil 2.5 Asimetrik mesajlaşma modeli

Asimetrik algoritmaların güvenliği konusuna gelindiğinde ise kod-kırıcı algoritmayı zaten bildiği için sadece gizli anahtarı bulmaya çalışacaktır. Asimetrik sistemlerde gizli anahtar ile açık-anahtar birbirleriyle matematiksel olarak ilişkilidirler. Ancak matematiksel işlemler kullanılarak anahtarların birbirlerinden elde edilmeleri çok zaman gerektirir. Açık anahtarlı şifreleme sistemlerine de bu zaman avantajına dayanılarak güvenlidir denilir. Açık-anahtarlı sistemler güvenlik açısından gizli anahtarın taşınmasında oluşabilecek her türlü aksilik ve düşman tarafından doğrudan taşıyıcının ele geçirilmesi gibi bir takım simetrik algoritma problemlerinden de sıyrılmıştır, bu da açık- anahtar sisteminin artılarındadır. Bu tip sistemlerin en önemli dezavantajı, sistemin simetrik yapılara göre çok yavaş çalışmasıdır. Bu nedenle uygulamalar her ikisini de içerecek şekilde yapılır, yani simetrik sistemin gizli anahtarının dağıtımı açık-anahtarlı sistem ile gerçekleştirilirken, asıl şifreleme algoritması hız avantajı göz önüne alınarak simetrik sistem ile şifrelenir.

Bu tez kapsamında yapılan uygulamada kullanılacak asimetrik şifreleme algoritması RSA olarak seçilmiştir. İlerleyen bölümlerde RSA konusu ayrıntılı olarak incelenecektir.Asimetrik algoritmalarda açık anahtarla şifrelenen mesaj özel anahtarla çözülmektedir. Aslında bunun terside gerçeklenebilirdir. Yani özel anahtarla şifrelenen mesaj açık anahtar ile açılabilmektedir. Bu esneklik sayısal imza yapılarının oluşmasını sağlamıştır.

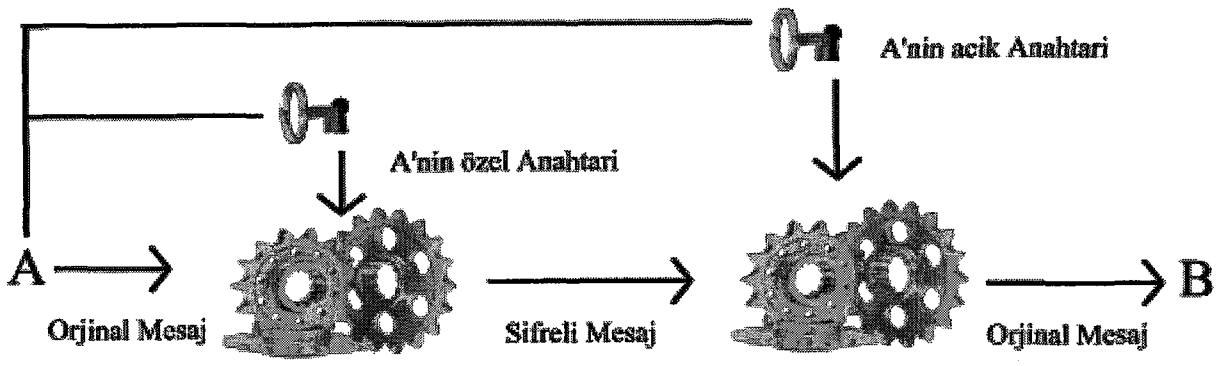
2.3 Sayısal imza

Günlük hayatta kullanılan kişisel imzalar yazılı anlaşmalarda, ,kimlik doğrulamalarda önemli yer tutmaktadır. Dokümanların imzalanması, dokümanı alana, yazanın veya onaylayanın kimliğini tanıtmaktadır. Bununla birlikte kişisel imzaların kullanılmasının başka nedenleri de vardır. Aynı imza taklit edilememekte, doküman imzalandıktan sonra üzerinde değişiklik yapılamamakta ve dokümanı imzalayan daha sonraki bir zamanda imzalamadığını iddia edememektedir.

Bilgisayar ortamında benzer işlemler düşünüldüğünde işlerin daha zor olduğu görülmektedir. Bilgisayar dosyaları kolaylıkla kopyalanabilir, üzerlerinde değişiklikler yapılabilir, kişinin imzası ne kadar karmaşık olursa olsun taşınabilirdir.

Burada saydığımız sorunları çözecek, günlük hayatta kullanılan kişisel imzaların yerini sanal ortamda alacak bir sistem, şifreciler tarafından sayısal imza algoritmaları adı altında geliştirilmiştir. Sayısal imzalar simetrik ve asimetrik sistemlerle gerçeklenenler olmak üzere ikiye ayrılırlar. Simetrik kripto sistemleri kullanıldığında haberleşecek taraflar öncelikle güvenilir bir üst birime ihtiyaç duyarlar. Haberleşmeler bu birim üzerinden gerçekleştirilir. Taraflar birbirlerinin gizli anahtarlarını bilmemekte, güvenilir birim ise ortamda bulunan herkesin gizli anahtarını bilmektedir. Gelen mesajları açar, gönderenin kimliğini ve mesajı alıcının özel anahtarıyla kapatarak alıcı tarafa yollar. Alıcı gelen mesajı kimlik doğrulama bilgisini gizli anahtarıyla açar ve mesajın kim tarafından gönderildiğini öğrenir.

Simetrik kripto sistemlerinde sayısal imzaların gerçeklenmesi, istenen özellikleri sağlamakla birlikte oldukça zaman alıcı ve karmaşık işlem adımları içermektedir. Ortamda güvenilir bir birime gereksinim duyulması, bütün haberleşmelerin bu birim üzerinden yapılması, giden ve gelen mesajların kimlik doğrulamada kullanılmak üzere bir veritabanında tutulması beklenen başarımı düşürmektedir. Teoride güvenilir olduğu düşünülen sistem uygulamada aynı sonucu vermemektedir. Sayısal imzaların gerçeklenmesinde kullanılan diğer bir yöntemde asimetrik kripto sistemleridir. Bu, asimetrik şifreleme algoritmalarının her zaman hem şifrelemede hem de imzalamada kullanılabileceği anlamına gelmez. Asimetrik kripto sistemlerinin kullanılmasıyla belirtilmek istenen imzalama işlemlerinde herkesçe bilinen (açık-anahtar) ve yalnızca imza sahibi tarafından bilinen (özel anahtar) bir anahtar çiftinden yararlanılmasıdır. Bu işlem için sistemdeki A kişisi B'ye mesaj yollamadan önce mesajını kendi özel anahtarı ile şifreler, B mesajı aldıktan sonra mesajı, veritabanından elde edebileceği A'nın açık-anahtarı ile çözer, keza başka bir anahtar mesajı çözemez, buradan B kişisi mesajın A kişisinden geldiğinden emin olur yani A kişisi mesajını sayısal olarak imzalamıştır. Burada, eğer data gönderim esnasında tahrip edilmişse mesaj bu kez A'nın açık-anahtarı ile de çözülemeyecektir. Dolayısıyla, bu işlem hem kullanıcının, hem de gönderilen bilginin doğruluğunu test etmektedir (Şekil 2.6).



Şekil 2.6 Sayısal imza modeli

3. RSA AÇIK ANAHTAR ŞİFRELEME SİSTEMİ

1970'lerin ortalarında Diffie-Hellman ikilisinin yayınlamış oldukları açık-anahtarlı şifreleme sistemleri ile ilgili makalenin (Diffie ve Hellman,1976) ardından MIT'de profesör olan Ron Rivest, bu konu üzerinden yeni ve eşsiz bir şifreleme sistemi oluşturmaya karar vererek Adi Shamir ve Len Adleman ile birlikte çalışmaya başlamıştır. 1977 yılında bu üçlü verileri tamamıyla şifreleyebilen bir sistem oluşturmuşlardır. 1978 yılında yayınladıkları (Rivest vd.,1978) bu sistemin adı mucitlerinin soy isimlerinin baş harflerinden esinlenilerek RSA konulmuştur. O zamandan bu güne farklı zamanlarda farklı açık-anahtar şifreleme sistemleri bulunmakla birlikte halen anahtar dağıtım probleminin üstesinden gelmek için en çok kullanılan yöntem RSA dir.

3.1 RSA'in matematiksel temelleri

RSA asimetrik şifreleme algoritması temelde üç sayıdan oluşmaktadır; bunlar modül, açık anahtar ve özel anahtardır. Buradaki modül iki çok büyük asal sayıdan elde edilir. Literatürde, bu sayılar genellikle n(modül), e (açık-anahtar), d (özel-anahtar), p ve q (modülü oluşturan asal sayılar) harfleriyle ifade edilirler.

RSA şifreleme işlemine başlarken yapılması gerekenler aşağıda sıralanmıştır.

1) Öncelikle çarpımları, istenilen modül uzunluğuna (256,512,1024,2048,..bit) erişebilen 2 adet asal sayı seçilir. (p ve q)

2) Seçilen asal sayılar çarpılarak tüm şifreleme işlemlerinde kullanılacak ve herkes tarafından bilinmesinde sakınca olmayan modül sayısı elde edilir (3.1).

$$n = p * q \tag{3.1}$$

3) Buradan sonra n'in Euler fonksiyonu tanımlanır (3.2).

$$\phi(n) = (p-1) * (q-1) \tag{3.2}$$

4) $\phi(n)$ ile en büyük ortak böleni '1' olan keyfi bir e açık-anahtarı seçilir

$$OBEB(\phi(n), e) = 1 \tag{3.3}$$

5) $\phi(n)$ Euler fonksiyonu ve e açık anahtarı kullanılarak d özel anahtarı hesaplanır.

$$d = e^{-1} \bmod(\phi(n)) \tag{3.4}$$

$$e * d = 1 \bmod(\phi(n)) \tag{3.5}$$

$$e*d = 1 + k*\phi(n) \tag{3.6}$$

Bu beş aşamalı işlemler dizisinden sonra şifreleme için ihtiyaç duyulan modül ve anahtar çifti elde edilmiş olur. Açık mesajı P ile şifreli mesajı C ile gösterdiğimizi hatırlarsak şifreleme için denklem (3.7), şifre çözme için ise denklem (3.8)'deki işlemler yapılır.

$$C = P^e \bmod(n) \tag{3.7}$$

$$P = C^d \bmod(n) \tag{3.8}$$

Denklem (3.7) ile elde edilen sonucun, denklem (3.8) kullanılarak aynı üstel modül alma işleminden geçirilip nasıl orjinal giriş değerini verdiğini denklem (3.9)'daki "Euler Teoremi"ni kullanarak açıklayabiliriz (Ek-13).

$$P^{\varphi(n)} = 1 \bmod(n) \tag{3.9}$$

Denklem (3.8)'deki C değeri yerine denklem (3.7)'de elde edilen sonuç konulursa;

$$P = \left(P^e\right)^d \bmod(n) = P^{e*d} \bmod(n) \tag{3.10}$$

elde edilir. Denklem (3.6) ile elde edilen çarpım kullanılarak denklem (3.10) yeniden yazılırsa denklem (3.11)'e ulaşılır.

$$P = P^{1+k*\varphi(n)} \bmod(n) = P * P^{k*\varphi(n)} \bmod(n) = P * (P^{\varphi(n)})^k \bmod(n) \tag{3.11}$$

Euler teoremi doğrultusunda denklem (3.11)'i basitleştirirsek;

$$P = P*1^k \bmod(n) = P*1 \bmod(n) = P \bmod(n) = P \tag{3.12}$$

sonucuna ulaşılır ve orjinal giriş değeri elde edilmiş olur.

RSA sisteminin çalışmasını basit sayısal bir örnekle açıklamaya çalışalım.

Asal sayılarımız $p = 19$ ve $q = 47$ olsun

--. $n = p * q = 19 * 47 = 893$ (3.13)

--$\phi(n) = (p-1) * (q-1) = (19-1) * (47-1) = 18 * 46 = 828$ (3.14)

-- $e = 7$ olarak seçelim

-- $OBEB\ (7,828) = 1$ (3.15)

$$d = 7^{-1} \bmod(828) = 355 \tag{3.16}$$

$$355*7 = 2485 = 828*3+1 \tag{3.17}$$

Şimdi elimizdeki bilgilerle (e = 7, d = 355, n = 893) ve mesajımızın 25 olduğunu kabul ederek şifreleme yaparsak;

$$C = 25^{7} \bmod(893) = 6103515625 \mod(893) = 6834843 * 893 + 826 \tag{3.18}$$

$$C = 25^{7} \bmod(893) = 826 \tag{3.19}$$

olarak buluruz, şifreli mesajı çözmek için modüler üs alma işlemini özel anahtar ile tekrar yaparsak ;

$$P = 826^{355} \bmod(893) = 3{,}3722269802\ 3096698738\ 8643901671\ 4 \cdot e^{1035} \bmod(893) = 25 \tag{3.20}$$

orjinal mesajımızı elde etmiş oluruz.

Burada şifreyi kırmak isteyen kişinin elinde zaten modül ve açık anahtar değerleri vardır. İşlem basite indirgendiğinde görülür ki şifre kırıcının p ve q değerlerini bulması diğer tüm değerlere vakıf olmasını sağlar. n değeri p ve q değerlerinin çarpımıdır, o zaman problem n'in çarpanlarına ayrılması problemi olarak özetlenebilir ki bu da RSA in esas güvenlik dayanağıdır. Burada RSA'in bir nevi çarpma işlemi olduğu ve bu işlemin tersinin de gayet tabi bölme gibi basit bir çözümü olduğu düşünülebilir ama bu ancak bölen sayı bilindiği müddetçe kolaydır.

Çarpanlara ayırma zorluğunu örneklemeye çalışırsak; n'in 55 olduğunu düşünelim burada soru p ve q'nun ne olacağıdır. Biz hemen bu iki asal sayının 5 ve 11 olduğunu söyleyebiliriz. Şimdi n'in 1457 olduğunu düşünelim, buradan p ve q değerlerini elde etmenin 55'ten elde etmekten çok daha fazla zaman kaybettirdiği aşikardır. Sayı ne kadar büyürse o sayının çarpanlarına ayrılması için gereken süre de artar. Araştırmacılar sayıları çarpanlarına ayırmak için bir takım bilgisayar programları yazmışlardır ve 1457 sayısı bu tip programlar için çok kolaydır ancak tıpkı insanlarda olduğu gibi sayılar çok çok büyüdüğünde hızlı bilgisayarlar da tıpkı bizim basit örneğimizdeki gibi daha fazla zamana ihtiyaç duyarlar (1457 sayısı 31 ve 47 sayılarının çarpımıdır). Aralık 2000 tarihli kayıtlar (Burnett ve Paine, 2001) 512 bit uzunluğundaki bir RSA sisteminin çarpanlara ayırma işlemi ile kırılmasının o zamanki teknolojik şartlarda hiç durmadan paralel olarak çalışan 292 adet bilgisayar ile ancak 5 aydan biraz daha fazla sürede gerçeklenebildiğini göstermiştir. Bu yapının en anlamlı kısmına eklenecek her bir bitin kırma süresini 1035 ile 1036 kat artırdığı bilindiğine göre 1024 bitlik bir sistemin aynı tip teknoloji ile ancak 3 ila 30 milyon yıl arasında kırılabilir olduğunu

söyleyebiliriz.

Tüm şifreleme sistemleri gibi RSA sistemini kırmanın bir diğer yolu kaba atak (brute-force attack)' tır. RSA sisteminde kaba atak, iki farklı değişkeni bulmak için kullanılabilir. Bunlardan birisi tüm sistemlerdeki gibi özel anahtar değişkeninin denenerek bulunması, bir diğeri de *n* modül değerimizin çarpanlarından birinin denenerek bulunmasıdır. Çarpanlarından birinin teker teker denenmesi işleminde her bir işlem sonunda denenen sayının n modül değerimizi kalansız bölüp bölmediğine bakılır.Eğer koşul sağlanamazsa bir sonraki denemeye geçilir. 1024 bitlik bir sistemde genellikle çarpanlardan birisi 512 bit uzunluğundadır. Biz biliyoruz ki bu sayı tektir, dolayısıyla en az ağırlıklı biti kesin 1 olacaktır, bir de 512 bit uzunluğunda olduğu için en ağırlıklı biti de 1'dir, deneme işleminde bunlar göz önüne alındığında deneme uzayı ancak 510 bit seviyesine indirgenebilir ki bu da halen makul bir aralık değildir. İkinci yöntem ise doğrudan anahtar değerinin kaba atak ile denenmesidir. Biliyoruz ki anahtar uzayı büyüdükçe güvenlik seviyesi de bir o kadar artmaktadır. RSA laboratuarlarının Nisan 2000 de yayınlamış oldukları bir yazıda, 10 milyon $ bütçe ile farklı anahtar uzunluklarındaki sistemlerin kırılması için ne kadar süre ve belleğe ihtiyaç duyulacağı Şekil 3.1'de gösterilmiştir.

Table 4-1
Time to Break Keys of Various Sizes with $10 Million to Spend

Symmetric Key (Size in Bits)	ECC Key (Size in Bits)	RSA Key (Size in Bits)	Time to Break	Number of Machines	Amount of Memory
56	112	430	Less than 5 minutes	105	Trivial
[illegible]	[illegible]	760	500 months	1,000	1GB
96	192	1,020	3 million years	116	170GB
128	256	1,620	10^{12} years	0.16	120TB

Şekil 3.1 10 milyon $ bütçe ile anahtar bulma denemesi (Burnett ve Paine, 2001)

RSA Laboratuarları zaman zaman farklı bit uzunluklarındaki sayıların çarpanlarına ayrılması için para ödüllü yarışmalar düzenlemektedir. Şekil 3.2'de 3 Mayıs 2005 tarihinde RSA Laboratuarlarının internet sayfasından alınmış bir görüntü yer almaktadır.

Challenge Number	Prize ($US)	Status	Submission Date	Submitter(s)
RSA-576	$10,000	Factored	December 3, 2003	J. Franke et al.
RSA-640	$20,000	Not Factored		
RSA-704	$30,000	Not Factored		
RSA-768	$50,000	Not Factored		
RSA-896	$75,000	Not Factored		
RSA-1024	$100,000	Not Factored		
RSA-1536	$150,000	Not Factored		
RSA-2048	$200,000	Not Factored		

Şekil 3.2 Çarpanlara ayırma yarışması [1]

Şekil 3.2'den de görüleceği üzere henüz 640 bitlik sayının çarpanlara ayrılması gerçekleştirilememiştir ve 1024 bitlik yapının çarpanlarına ayrılması karşılığında verilecek ödül 100,000$ dır.

3.1.1 Modüler üs alma

RSA sisteminin çok büyük sayılarla işlem yaptığı ve bu işlem kapasitesinin birçok bilgisayar programının dahi sınırlarını aştığı aşikardır. Bu bölümde RSA sistemini oluşturabilmek için gerekli olan yoğun matematiksel sürecin farklı yöntemlerle nasıl üstesinden gelinmeye çalışıldığı incelenmiştir. Bu farklı yöntemler RSA sisteminin sayısal devre tasarımlarında da büyük önem taşımaktadır.

RSA sistemi kurulurken modül, özel ve açık anahtarlar yayınlanır, kullanıcılar da bunlar ile birlikte imzalama, onaylama, şifreleme ve şifre çözme için hep aynı işlemi gerçeklerler (3.7 ve 3.8). Bu işleme modüler üs alma veya "Moduler Exponentiation" denir. Bu işlem genel bir karıştırma operasyonudur ki RSA dışında Diffie-Helman, El-Gamal, Dijital İmza Standartları

gibi sistemlerde de kullanılmaktadır.

Modüler üs alma işlemini yaparken her bir işlemden sonra sonucun n modül oranında indirgenmesi esastır. (3.7)'i esas alarak örneklersek ; $C = P^e \bmod n$ 'i hesaplamak için öncelikle $C_1 = P \bmod n$ değerini ardından $C_2 = C_1.P \bmod n$ değerini hesaplamalı ve bu işlemler dizisini $C = P^e \bmod n$ değerine ulaşıncaya kadar yapmalıyız. Bu ilk akla gelebilecek yöntem ile e-1 adet işlem yapılması gerekmektedir. Mesela e = 15 olan bir sistemde $P^2, P^3, P^4, \ldots, P^{12}, P^{13}, P^{14}, P^{15}$ değerlerinin hepsinin hesaplanması gerekmektedir ancak "Binary Method" veya "Squaring and Multiplication" olarak adlandırılan başka bir yöntem sayesinde e=15 olan bu işlem için sadece $P, P^2, P^3, P^6, P^7, P^{14}, P^{15}$ değerleri hesaplanarak da aynı sonuç elde edilebilmektedir.

3.1.1.1 Binary yöntemi

Binary yöntemi, üs olarak kullanılan sayının(e veya d) her bitini sağdan-sola (R-L, Right to Left) veya soldan-sağa (L-R, Left to Right) tarar ve her bir taramada bir kare alma işlemi gerçekleştirir. Taranan bitin durumuna göre de o bit için çarpma işlemine gerek olup olmadığına karar verir. Bu iki yöntem aşağıda gösterilmiştir. k'nın üs olan değerin bit sayısı olduğunu kabul ederek;

a) L-R Binary Metodu

Sonuç= $C = P^e \bmod n$

1. **if** $e_{k-1} = 1$ **then** $C := P$ else $C := 1$
2. **for** i = k-2 **downto** 0

   2.a. $C := C \,.\, C \bmod n$

   2.b. if $e_i = 1$ **then** $C := C.P \bmod n$

3. **return** C

b) R-L Binary Metodu

Sonuç= $C = P^e \bmod n$

1. $X_0 = P$; $C_0 = 1$
2. **for** i = 0 to k-1

   2.a. $X_{i+1} := X_i \,.\, X_i \bmod n$

   2.b. if $e_i = 1$ **then** $C_{i+1} := C_i.X_i \bmod n$

2.c. else $C_{i-1} = C_i$

3. return C

İki yöntem de temelde aynı işi yapmasına rağmen R-L Binary Methodunun diğerine göre gerçekleştirmede bazı avantajları vardır. Temel avantajı ise 2.a ve 2.b adımlarındaki işlemlerin birbirlerinden bağımsız olmaları ve paralel olarak işlem görebilmeleridir.

3.1.2 Modüler çarpma

Modüler üs alma işleminin gerçekleştirilebilmesi için binary yöntemi kullanılsın veya kullanılmasın, her adımda modüler çarpma işlemlerine ihtiyaç duyulur (3.21). Eğer "binary yöntemi" kullanılırsa, bu daha da ileri giderek modüler çarpma ve modüler kare alma işlemleri olarak genişletilir.

$$C = P \cdot C \bmod(n) \tag{3.21}$$

RSA sisteminde çok büyük sayılarla makul zamanlarda işlem yapabilmemiz amacıyla modüler çarpma işlemi için de alternatif yollar bulmamız gereklidir. Bu yöntemlerin bazıları çarp ve böl, çarp ve indirge, Brickell's Yöntemi, Blakley's Yöntemi, Montgomery Yöntemi olarak isimlendirilir. Bu tez kapsamında sadece Montgomery Yöntemi kullanıldığı için diğer yöntemler ile ilgili bilgi sunulmamıştır. İsmi sayılı yöntemlerle ilgili ayrıntılı bilgiye Koç, Ç. K.(1994)'dan ulaşılabilir.

3.1.2.1 Montgomery yöntemi

1985 yılında Montgomery, k- bit uzunluğundaki P, C ve n sayıları ile $R = P.C \bmod n$ işlemini gerçekleştirebilen etkili bir algoritma ortaya koymuştur. Bu algoritma, özellikle 2'nin kuvvetlerini kullanarak hızlı aritmetik işlemler yapan genel amaçlı bilgisayarlar (mikroişlemciler, sinyal işlemcileri) ve benzer sayısal tasarımlar için çok ideal bir yapıya sahiptir. Montgomery yöntemiyle, k bit uzunluğundaki R sonucu, n modül değeri ile herhangi bir bölme işlemi yapmadan elde edilebilmektedir. Modül değerine bölme işlemi, 2'nin katlarına bölme işlemi ile temsil edilmekte ve sayılar ikilik tabanda olduğu için uygulama kolaylıkla sayısal tasarıma adapte edilebilmektedir. Farz edelim n k-bit uzunluğunda bir sayı olsun (2^{k-1} ve 2^k aralığında) ve r de 2^k olsun. Montgomery indirgeme yöntemi r ve n'nin aralarında asal olmalarını gerektirir ki n'nin iki asal sayının çarpımı olması bu şartı zaten sağlamaktadır.

$P < n$ olacak şekilde verilen bir P sayısı ile onun r'ye bağlı olan n kalanını (3.22) ile tanımlayabiliriz.

$$\bar{P} = P \cdot r \bmod(n) \tag{3.22}$$

Burada işlem sonucunun 0 ile n-1 arasında olacağı açıktır ve tüm sayıların bu aralıkta bir karşılığı bulunmalıdır. Montgomery indirgeme yöntemi, bu bilgi doğrultusunda *n*-kalanları hesaplanan iki sayıyı kullanarak sonucun *n*-kalanını hesaplamanın, sayıların normal değerlerini kullanarak hesaplamaktan çok daha hızlı olduğunu ortaya koymuştur. Bu nedenle, montgomery algoritmasına girmeden sayılarımızın *r*'ye bağlı *n* kalanlarını hesaplamamız gerekmektedir. Kalan değerleri hesaplanarak algoritma çalıştırıldığında (MonPro($\bar{P} \cdot \bar{C}$,*n*)) denklem 3.23'deki sonuç elde edilir. Yani montgomery algoritması sonuca bir r^{-1} çarpanı eklemektedir.

$$\bar{R} = \bar{P} \cdot \bar{C} \cdot r^{-1} \bmod(n) \tag{3.23}$$

Buradaki r^{-1} *r*'nin modül *n*'ye göre tersidir (3.24).

$$r^{-1} \cdot r = 1 \bmod(n) \tag{3.24}$$

(3.23)'ü biraz açarsak;

$$\bar{R} = P \cdot r \cdot C \cdot r \cdot r^{-1} \bmod(n) \tag{3.25}$$

$$\bar{R} = P \cdot C \cdot r \bmod(n) \tag{3.26}$$

olduğu görülür. Yani (3.23) işlemi ile aslında *R*'nin r'ye bağlı *n*-kalanını hesaplamış olduğumuzu görürüz.

Montgomery indirgeme yöntemini anlamak için bir değişkene daha ihtiyaç vardır, n' değeri (3.27)'deki gibi hesaplanır.

$$r^{-1} \cdot r - n \cdot n' = 1 \tag{3.27}$$

Tüm bu bilgiler sonucunda Montgomery indirgeme algoritması aşağıda gösterilmiştir;

MonPro(P,C,n)

1. $t := \bar{P} \cdot \bar{C}$
2. $m := t.n' \bmod r$
3. $u := (t+m.n)/r$
4. **if** $u \geq n$ **then return** $u-n$

else return u

Görüldüğü gibi işlemlere P ve C'nin kalanları ($\bar{P}\cdot\bar{C}$) kullanılarak başlanmıştır. Buradaki en önemli faktör, Montgomery içindeki işlemlerin r ile bölme olarak gerçekleşmesidir ki bu da r'nin 2'nin kuvveti olması sayesinde işlemlerin sayısal tasarım göz önüne alındığında çok hızlı gerçekleştiğini gösterir.

Buraya kadar ki incelemelerimizi Binary yöntemindeki modüler çarpma işlemini efektif hale getirmek için yaptığımızı belirtmiştik. Şimdi Modüler üs alma işlemi için L-R Binary yöntemini, onun içindeki modüler çarpma işlemleri için de montgomery yöntemini kullanarak elde ettiğimiz sonuçları kullanırsak;

ModExp($P.C.n$)

1. $\bar{P}:=P.r \bmod n$

2. $\bar{x}:=1.r \bmod n$

3. for $i:= k\text{-}1$ **downto** 0 **do**

4. $\bar{x}:=\text{MonPro}(\bar{x},\bar{x})$

5. **if** $e_i = 1$ **then** $\bar{x}:= \text{MonPro}(\bar{P}\ \bar{x})$

6. $x =\text{MonPro}(\bar{x},1)$

7. return x

algoritmasını elde etmiş oluruz.

Burada tanıtılan işlemlerin geçerliliğini bir sayısal örnek ile tekrar gözden geçirelim.

$C=7^{10} \bmod 13$ işlemini ModExp algoritması için adım adım yapalım;

- $n = 13$ olduğu için $r =2^4 =16 > n$ olarak seçeriz.
- n` hesaplanır; $16 . 9 - 13 . 11 = 1$'dir, dolayısıyla $r^{-1}=9$ ve $n$`$=11$ olarak elde edilir.
- $\bar{P}$ hesaplanır; $P = 7$ dir ve $\bar{P} = P.r \bmod n =7 . 16 \bmod 13 = 8$ olarak hesaplanır.
- $\bar{x}$ hesaplanır ; $\bar{x} =1.r \bmod n =1.16 \bmod 13 =3$

Buradan sonra döngü işlemi başlar;İkili tabanda $e_i =$"1010" dir.

Çizelge 3.2 MonPro işlem sırası

e_i	4. Adım	5.Adım
1	MonPro(3,3)=3	MonPro(8,3)=8
0	MonPro(8,8)=4	----------
1	MonPro(4,4)=1	MonPro(8,1)=7
0	MonPro(7,7)=12	----------

MonPro(3,3) = 3

t :=3.3=9

m:= 9.11 mod 16 =3

u:=(9+3.13)/16=48/16=3

MonPro(8,3) = 3

t :=8.3=24

m:= 24.11 mod 16 =8

u:=(24+8.13)/16=128/16=8

MonPro(8,8) = 4

t :=8.8=64

m:= 64.11 mod 16 =0

u:=(64+0.13)/16=64/16=8

MonPro(4,4) = 16

t :=4.4=64

m:= 16.11 mod 16 =0

u:=(16+0.13)/16=16/16=1

MonPro(8,1) = 7

t :=8.1=8

m:= 8.11 mod 16 =8

u:=(8+8.13)/16=112/16=7

MonPro(7,7) = 12

t :=7.7=49

m:= 49.11 mod 16 =11

u:=(49+11.13)/16=192/16=12

MonPro(12,1) = 4

t :=12.1=12

m:= 12.11 mod 16 =4

u:=(12+4.13)/16=64/16=4

Sonuç $7^{10} \bmod 13 = 4$ tür.

Son MonPro işlemi (MonPro(12,1)) r'ye bağlı n-kalanı elde edilmiş olan sonucu, kalan değerinden sıyırmak için yapılmaktadır.

$$Mon\Pr o(\overline{R},1,n) = \overline{R} \cdot 1 \cdot r^{-1} \bmod(n) \tag{3.28}$$

$$Mon\Pr o(\overline{R},1,n) = R \cdot r \cdot r^{-1} \bmod(n) \tag{3.29}$$

$$Mon\Pr o(\overline{R},1,n) = R \cdot \bmod(n) \tag{3.30}$$

Tüm işlemler bittikten sonra ulaşılmak istenen R değeri elde edilmiş olur.

Böylece, RSA hesaplamalarına temel oluşturan modüler üs alma ve modüler çarpma işlemlerinin alternatif yöntemlerle de çalıştığını örneğimizle ispatlamış olduk. Bundan sonraki bölümlerde bu yöntemlerin nasıl donanımsal tasarım ve gerçekleme de kullanıldıklarını inceleyeceğiz.

4. RSA SİSTEMİNİN DONANIMSAL GERÇEKLENMESİ

Bu bölümde önceki bölümlerde modüler üs alma ve modüler çarpma işlemleri için sunulan alternatif çözümlerin bit veya bit katarları ile işlem yapabilecek şekilde uyarlanmış hallerini göreceğiz. Bu bilgiler ışığında donanımsal olarak gerçeklemeler yapılırken izlenen yöntemleri inceleyeceğiz.

Koç,Ç. K.(1994,1995), Daly, A.ve Marnane, W.,(2002), Koç, Ç. K. ve Walter, C. D.(2003) kaynakları ayrıntılı incelenirse bir önceki bölümde sunmuş olduğumuz MonPro işleminin donanımsal gerçekleme için bit bazında işlem yapabilecek şekilde uyarlanmış olan aşağıdaki algoritmalar ile aynı görevi üstlendiği görülür. Bit bazında çalışma, algoritmaya giren sayıların belirli bit uzunluklarındaki parçalarının sırayla aynı algoritmadan sayının tüm bitlerine bu işlem yapılıncaya kadar geçirilmesi ile gerçekleşir.

MonPro1 (A,B,n) – k-bit sayılar

Sonuç : $P = A.B.r^{-1} \bmod n$

$P_{-1} = 0;$

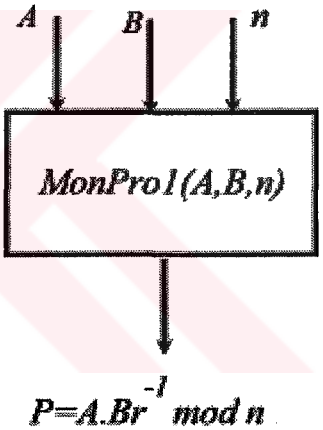
1. **For** $i = 0$ **to** $k-1$ **do**

   1.a $q_i = (P_{i-1} + a_i . B) \bmod 2$;

   1.b $P_{i+1} = (P_{i-1} + q_i . n + a_i . B) / 2$;

2. **End for;**

3. **Return** P_{k-1}



Şekil 4.1 MonPro1 sembolik gösterimi

MonPro1 algoritması, algoritmaya giren A ve B sayılarının her döngüde tek bir bitinin taranması mantığı üzerine kurulmuştur. Her seferinde tek bir bit taramak bu döngüyü bit sayısı kadar tekrarlamak demektir. Burada, işlem sayısını azaltmak için döngülerde taranacak olan bit sayısını arttırabiliriz. Tarama adımı ne kadar büyürse, tarama sayısı o oranda azalacaktır. MonPro1 algoritmasının herhangi bir r tabanına göre genellenmiş hali (r eğer 4 ise her döngüde 2 bit taranır ve buna radix-4 tabanına göre işlem yapılıyor denilir) MonPro2 olarak elde edilebilmektedir (Şekil 4.2).

MonPro2 (A,B,n) – k-bit sayılar

Sonuç : $P = A.B.r^{-m} \bmod n$

$P_0 = 0;$

$m = k / log_2 r$

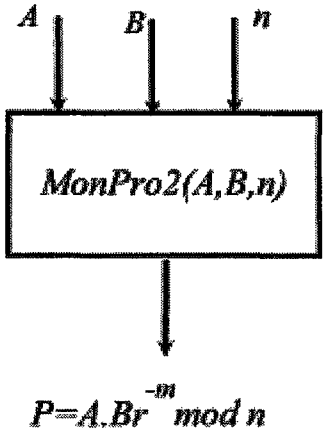
1. **For** $i = 0$ **to** $m-1$ **do**

   1.a $q_i = (P_0 + a_i.b_0) \,.\, (r\text{-}n_0)^{-1} \bmod r$;

   1.b $P_{i+1} = (P_i + q_i \,.\, n + a_i.B) / r$;

2. **End for;**

3. **Return** P_{k-1}



Şekil 4.2 MonPro2 sembolik gösterimi

İkinci algoritmanın 1.a ile işaretlenmiş satırına dikkat edersek burada $a_i.b_0$ işlemini görürüz, bu çarpım B sayısının tarama adımı boyutunda taranacak en az anlamlı kısmı (b_0) ile A sayısının yine aynı boyutta taranacak ilgili döngüdeki değerinin (a_i) çarpımıdır. Bu bilgi ışığında algoritmayı daha basit bir hale getirmek için B sayısını r ile çarparak (r, ikinin katı olduğu için her çarpım bit bazında sola kaydırmak, yani sağdan bir bitin 0 yapılması anlamına gelmektedir) B sayısının tarama adımı boyutundaki taranacak en az anlamlı kısmının hep 0 olmasını garanti etmiş oluruz. Buradan $a_i \,.b_0$ çarpımın sonucu hep 0 olacaktır ve 1.a satırındaki toplama işlemimiz için etkisiz kalacaktır. Yapılan basitleştirme karşılığında veri kaybı olmaması için aynı döngü 2 kez fazladan çalışmalıdır, bu fazladan çalışılacak 2 döngü ve dolayısıyla yitirilecek zaman, gerçek zamanlı işlemlerdeki toplam döngü sayısının yanında çok küçük kalacaktır. Bu son değişiklik ile algoritmanın tezimizde de kullanacağımız son hali ortaya çıkmaktadır (Şekil 4.3). MonPro3 algoritması işletilmeye başlamadan önce B değerinin r ile çarpılması gerekmektedir, çarpım sonucu oluşan bu değişken B`olarak sembolize edilmiştir.

MonPro algoritmalarının hepsinin sonunda tarama aralığına bağlı olarak sonuca eklenen çarpımlar vardır. MonPro3 algoritmasında görüldüğü üzere sonuç değerinin içinde olmaması gereken $(r^{-(m+1)})$ çarpanı vardır. Buradaki r modülün tabanını, m ise kaç adet döngü yapılacağını göstermektedir.

MonPro3 (A,B',n) – k-bit sayılar

Sonuç : $P = A.B.r^{-(m+1)} \bmod n$

$P_0 = 0;$

$B` = B.r;$

$m = k / log_2 r$

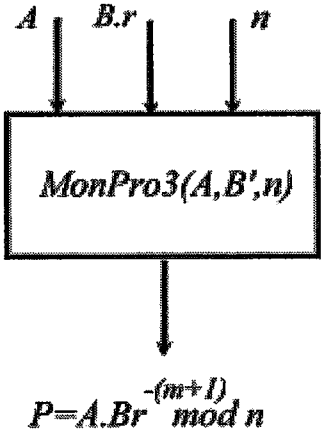
1. **For** $i = 0$ **to** $m+1$ **do**

   1.a $q_i = (P_0) \,.\, (r\text{-}n_0)^{-1} \bmod r$;

   1.b $P_{i+1} = (P_i + q_i \,.\, n + a_i.B`) / r$;

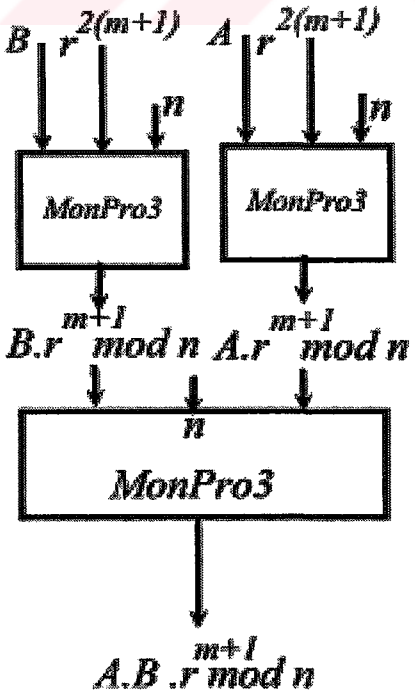
2. **End for;**

3. **Return** P_{k-1}



Şekil 4.3 MonPro3 sembolik gösterimi

Fazladan sonuca eklenen bu çarpım değeri, aslında MonPro algoritmasına bir önceki bölümde anlatıldığı gibi neden sayıların kendilerinin değil de sayıların r'ye bağlı n-kalanlarının sokulması gerektiğinin de bir göstergesidir. Bu fazladan oluşan değerlerin bir takım hesaplamalar ile ortadan kaldırılması ve sonucun n-kalanının elde edilmesi gerekmektedir. Hesaplamaların sıralaması Şekil 4.4'de gösterilmiştir.



Şekil 4.4 MonPro3 sonuç gösterimi

Görüldüğü üzere sonucun *n*-kalanını elde etmek için öncelikle $r^{2(m+1)}$ değerinin hesaplanması ve işlemlere dahil edilmesi gerekmektedir. Bunu matematiksel olarak ifade etmeye çalıştığımızda A değerinin *r*'ye bağlı *n*-kalanının hesaplamasına dönmemiz gerekir. MonPro3 algoritmasının A, $r^{2(m+1)}$ ve *n* değikenleri ile işleme sokulması sonucunda, aslında A sayısının *n*-kalanı hesaplanmış olmaktadır.

$$Mon\Pr o(A, r^{2(m+1)}, n) = A \cdot r^{2(m+1)} \cdot r^{-(m+1)} \bmod(n) \tag{4.1}$$

$$Mon\Pr o(A, r^{2(m+1)}, n) = A \cdot r^{(m+1)} \bmod(n) \tag{4.2}$$

Burada dikkat edilmesi gereken noktalardan birisi $r^{2(m+1)}$ değerinin modül sayısından daha büyük olduğudur. Bu nedenle bu değer algoritmaya doğrudan sokulmaz, onun yerine sayının *n*'e göre modülü alınmış değeri algoritmaya girilir (4.3).

$$C = r^{2(m+1)} \bmod(n) \tag{4.3}$$

Hesaplamada kullanılan değişkenlerden *m* değeri sadece bit uzunluğu ile bağlantılı ($m=k/\log_2 r$), radix-r değeri de tasarlanan sistem için önceden planlanan bir değer olduğu için *C*, bit sayısı veya modül değeri değişmediği müddetçe sabit kalacaktır. Bu nedenle tez çalışmasında bu değişkene "constant" adı verilmiştir. Bu değerin değişmeyecek olması, onun tıpkı anahtarlar ve modül değeri gibi veri tabanına kaydedilebileceği anlamına gelmektedir, daha hızlı bir sistem için her başlangıçta bu değerin hesaplanması yerine hafızadan kullanılması çok daha uygundur.

MonPro3 işleminin donanımsal olarak uyarlanmış halini kullanrak üstel işlemler yapabilmemiz için bu algoritmaları binary yöntemi ile ortak kullanmamız gerekmektedir. Aşağıda, içine MonPro3 algoritması yerleştirilmiş R-L Binary Yöntemi'nin tez kapsamında da kullanacağımız son hali verilmiştir.

MonExp(*P,e,n*)–*P* şifrelenecek veya çözülecek mesaj, *e* üs (açık veya özel anahtar), *n* modül, *k* modülün bit sayısı, *z* ise üs değerinin bit sayısıdır.

$m=k / log_2 r$

MonExp(*P,e,n*)

{ ***C*:=** $r^{2(m+1)}$ **mod** *n* --------**(constant hesaplaması)**

***P*:= MonPro**(*C,P,n*) --------**(*n*- kalanı hesaplaması)**

***R*:= MonPro**(*C*,1,*n*) --------**(*n*- kalanı hesaplaması)**

For i = 0 **to** *z*-1 **do**

if (ei =1) **then**

***R*:= MonPro**(R,P,n) --------**(Multiplication)**

End if;

***P*:= MonPro**(P,P,n) --------**(Squaring)**

End for;

***R*:= MonPro**(1,R,n)

Return R;

}

Buraya kadar tanımlanan fonksiyonlardan MonPro3 ve MonExp'i kullanarak, radix değerimizin de 4 olduğunu kabul ederek (radix-4) sayısal bir örnek yapalım; yapılacak işlem 3^{11} mod 37 olsun. 37 sayısı 2'lik düzende "100101" dir ve k=6 bit uzunluğundadır.

MonExp(3,11,37**);**

$m = k/\log_2 4 = 6 / 2 = 3$

Burada $\log_2 4$ olmasının nedeni işlemlerin radix-4 e göre yapılmasından kaynaklanmaktadır.

{ ***C*:=** $4^{2(m+1)}$ **mod** $n = 4^8$ mod 37 = 65536 mod 37 = 9

***P*:= MonPro**(C,P,n) = **MonPro**(9,3,37) =28

***R*:= MonPro**(C,1,n) = **MonPro**(9,1,37)=34

For Döngüsü ***e* "1011"**

i = 0 => e_0 = '1' ***R*:= MonPro**(R,P,n) = **MonPro**(34,28,37) =28

***P*:= MonPro**(P,P,n) = **MonPro**(28,28,37) =10

i = 1 => e_1 = '1' ***R*:= MonPro**(R,P,n) = **MonPro**(28,10,37) =30

***P*:= MonPro**(P,P,n) = **MonPro**(10,10,37) =16

i = 2 => e_2 = '0' ***R*:= ————**

***P*:= MonPro**(P,P,n) = **MonPro**(16,16,37) =1

i = 3 => e_3 = '1' ***R*:= MonPro**(R,P,n) = **MonPro**(1,30,37) =27

***P*:= MonPro**(P,P,n) = **MonPro**(1,1,37) =12

***R*:= MonPro**($1,R,n$) = **MonPro**(27,1,37) =28

sonucuna doğru olarak ulaşılır.

Algoritma incelenirken önemle dikkat edilmesi gereken nokta "Multiplication" ve "Squaring" olarak adlandırılan kısımların bulundukları döngü içinde birbirlerinin sonuçları ile ilgilenmemeleri, yani bağımsız olmalarıdır. Sonuçlar ancak içinde bulunulan döngü bittikten sonra diğer döngü içerisinde değerlendirilmektedir. Buradan sayısal tasarımda aynı donanımsal yapıdan iki adet birbirinden bağımsız çalışan birim oluşturabileceğimizi anlarız. Bu tip bir tasarım hız açısından büyük fayda sağlayacağı için çok hızlı sistemlerle çalışmaya uygundur ancak aynı sistemden 2 adet kullanılıyor olması istenmeyen alan işgaline neden olacaktır. Bir de donanımsal birimin tek olduğu ancak işlemlerin uygun saat darbeleri ile birbirlerinden ayrılabileceği yapılar tasarlanabilir ki bu tip yapılar literatürde "Systolic Arthitecture" olarak isimlendirilir. Tez kapsamında da bu temele oturtulmuş bir RSA sistemi tasarlanmıştır.

4.1 Sistolik Yapılar

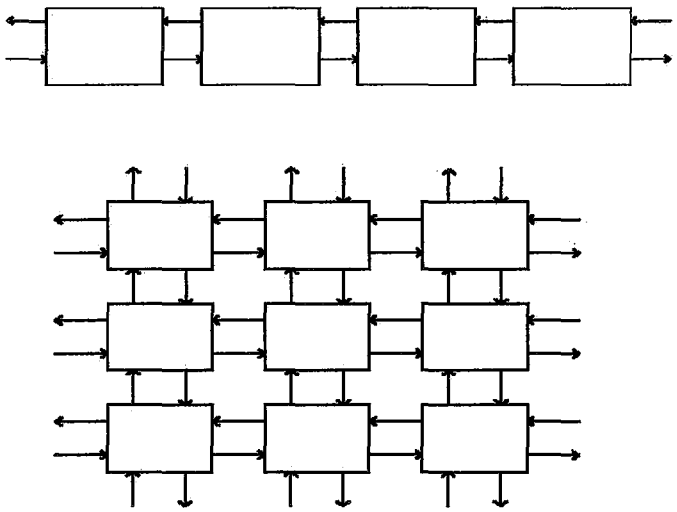
Birbirleriyle tamamen aynı özelliklere sahip daha küçük parçaların belirli bir düzene göre bir araya gelerek oluşturdukları bütüne sistolik yapılar adı verilir. Bu yapıların uyması beklenen bazı temel kurallar vardır. Bunlar;

• Her bir parçacık birbirinin aynı olmalıdır ve eş zamanlı olarak çalışmalıdır, böylece yapı yeni parçacıklar eklenerek kolaylıkla genişletilebilmelidir.

• Parçacıklar komşu diğer parçacıklarla bağlıdırlar dolayısıyla yapı ne kadar büyürse büyüsün bağlantı ve yol gecikmeleri minimum seviyede sorun oluşturmalıdır.

• Parçacıklar olabildiğince basit yapılmalıdır böylece yüksek frekanslarda çalışmaya olanak sağlamalıdır.

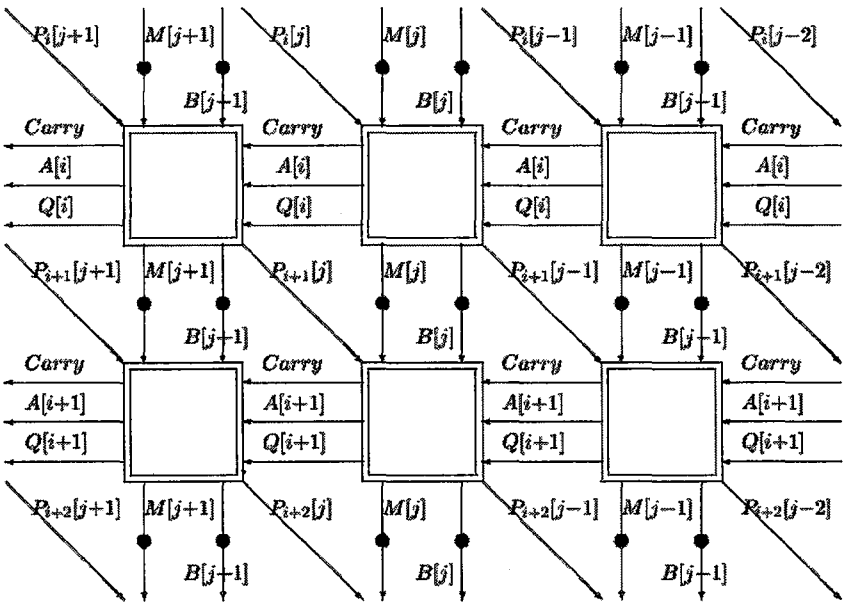
Şekil 4.5'de iki farklı sistolik yapıya örnek verilmiştir. Örnekler, küçük parçaların sadece komşu parçalarla haberleştiği, kolaylıkla eklemeler yapılabilecek basit yapılar dizisinden oluşmaktadır. Sistolik yapıların Montgomery algoritması ile de uygun çalışabildiği gözlenmiştir.



Şekil 4.5 Sistolik yapı gösterim örnekleri (Yesil, 2004)

4.2 Montgomery Yönteminin Sistolik Yapısı

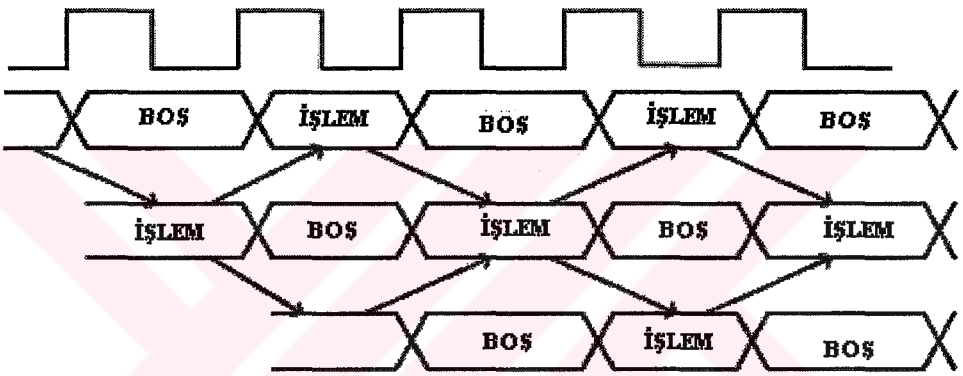
Bundan önceki bölümlerde örneklerle açıkladığımız üzere, Montgomery Yöntemi belirli sayıdaki bitler üzerinde(radix-2, radix-4,...)işlemler yapmakta, daha sonra kaydırma yaparak bir sonraki bit setini almakta ve aynı işlemleri onların üzerinde yapmaktaydı. MonPro3 algoritmasını incelersek işlem en az ağırlıklı kısım olan P_0 dan başlamaktadır ve bu bilgiyle q_i elde edilmektedir. Elde edilen q_i ile de bir toplama işlemi sonucunda bir sonraki döngü için gerekli olan çıkışlar oluşturulmaktadır.. Şekil 4.6'da gösterildiği gibi satırlar her yeni döngüde yapılan işlemleri, sütunlar ise tek bir bitin (veya bit katarının) ardıl saat darbeleri ile yaptığı hesaplamaları göstermektedir.



Şekil 4.6 Montgomery sistolik yapısının sembolik gösterimi (Walter, 1993)

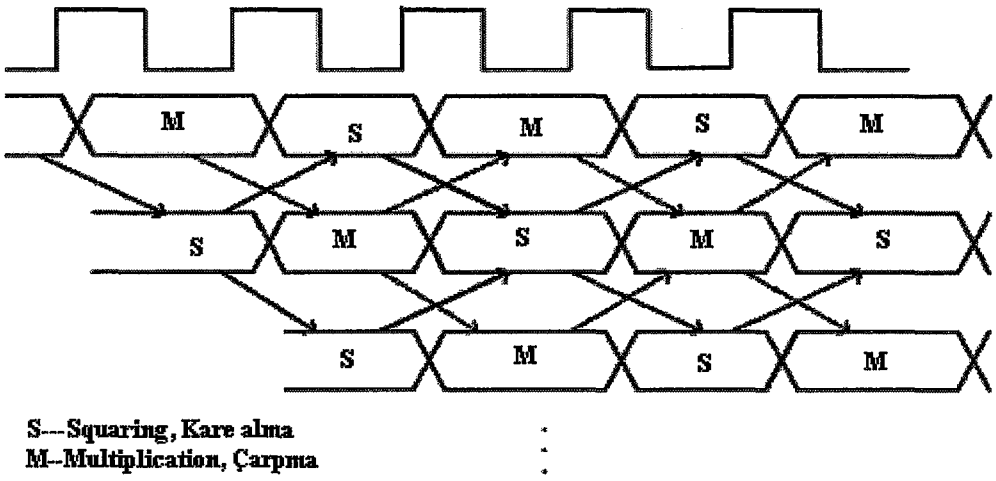
Şekil 4.6'dan da anlaşılacağı üzere Montgomery yöntemi sistolik yapının gereklilikleri ile örtüşmektedir. Bu nedenle tez kapsamında tasarım yapılırken Montgomery yönteminin sistolik gerçeklenebilirliğinden yararlanılmıştır.

Burada tek bir hücre bir tek bit (veya bit katarı) üzerinde işlem yapar ve çıkış değerlerini (elde dahil) bir sonraki saat döngüsünde kullanılmak üzere solundaki hücrenin girişine bırakır. Gerekli bilgileri makul bir zamanda girişinde bulan (i+1). sütundaki hücre işlemini gerçekleştirir ve sonucu sağ tarafındaki hücreye gönderir. Burada dikkat edilmesi gereken i. sütundaki işlemin (i+1). sütundaki işlemin sonucuna bağlı kalmasıdır. Bu nedenle (i+1). sütunda işlem yapılırken i. sütunda işlem yapılmaz, sadece (i+1). sütunun cevabı beklenir. Bu durum Şekil 4.7'de gösterilmiştir.



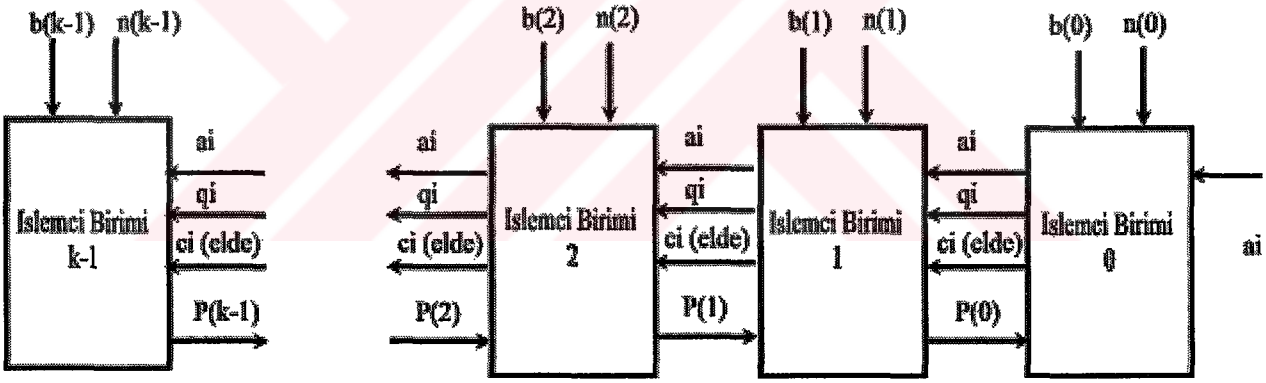
Şekil 4.7 Montgomery yöntemi ile paralel çalışma sembolik gösterimi

Daha önceki bölümde değindiğimiz üzere Montgomery Yöntemi ile işlem yaparken paralel ve bağımsız ama aynı özellikli iki donanım birimi ile işlem yapıldığında her ikisinde de şekil 4.7'de gördüğümüz gibi, bir çalışan bir boş duran zaman diyagramı elde ederiz. Bu yapı saat darbelerini ancak %50 performansla kullanabilmektedir. Burada karşımıza çıkan boş durma süreleri birbirlerinden bağımsız çalışan çarpma (Multiplication) ve kare alma (Squaring) işlemleri için kullanıldığında aynı donanımsal yapıdan 2 adet oluşturma sorunumuz ortadan kalkmaktadır. Bağımsız çalışan kare alma işlemi, çarpma işleminin boşluklarına serpilmektedir.Böylece ardıl saat darbeleri ile herhangi bir boş zaman bırakılmadan işlem yapabilmektedir (Şekil 4.8).



Şekil 4.8 Araya şıkıştırma (interleaving) yönteminin sembolik gösterilimi

Şekil 4.8 ile gösterilen yapıya literatürde "Interleaved Multiplication and Squaring" adı verilmektedir. Tez kapsamında tasarlanan RSA sistemimiz de bu ilke ile çalışmaktadır. Tasarlanan sistemin her bir hücresine "İşlem Birimi" adını verirsek sistemimiz kabaca şekil 4.9'daki gibi olacaktır.



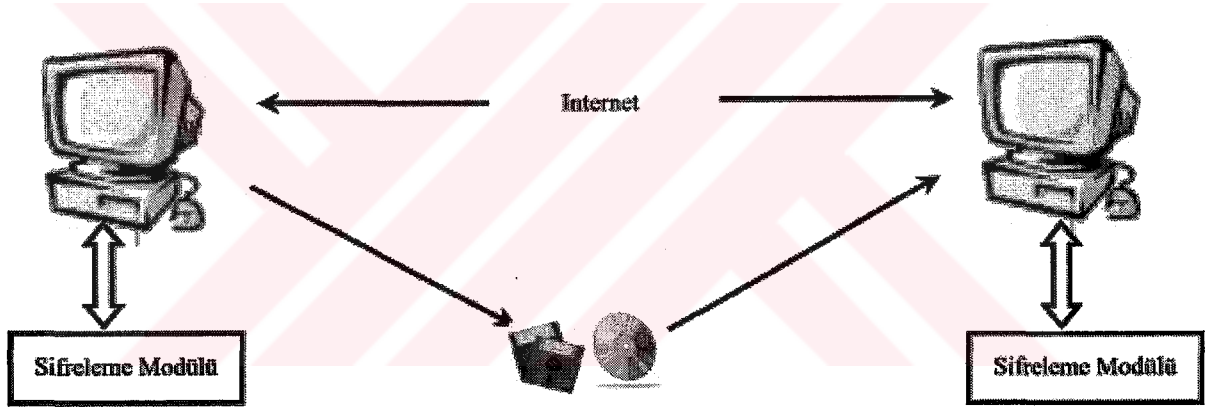
Şekil 4.9 Tasarlanan sistemin sembolik gösterilimi

Burada tasarımın sistolik yapıya benzerliği bir kez daha ortaya konulmuştur. Bildiğimiz, buradaki her bir işlemci biriminin aynı görevi yani algoritmanın $P_{i+1} = (P_i + q_i \cdot n + a_i.B) / r$ kısmının hesaplamasını yaptığıdır ancak qi değerinin hesaplanması için algoritmada bulunan $q_i = (P_0) \cdot (r\text{-}n_0)^{-1} \bmod r$; kısmının da ayrıca hesaplanması gerekmektedir ve şekil 4.9'da en sağda görülen işlemci birimi bu görevi yapar. Tasarımda, q_i hesaplaması için farklı ve işlemci modülüne göre daha sade, içinde sadece basit birkaç işlem ve look-up table (LUT) bulunduran yeni bir hücre kullanılmıştır. İlerleyen bölümlerde her iki işlemci biriminin de donanımsal yapısı ayrıntılı olarak gösterilecektir.

5. SİSTEM MİMARİSİ ve ÇALIŞMASI

Bu bölümde, daha önceki bölümlerde anlatılan konular doğrultusunda tasarlanmış sistem, bir bütün olarak ele alınmıştır. Sistem mimarisi genelden ayrıntıya gidilecek şekilde konulara ayrılmış ve tüm modüller öncelikle donanımsal olarak incelenmiş, ardından da çalışma şekilleri anlatılmıştır.Yine kullanılan modüller ile ilgili yazılımlar ve sonuçlar ekler içinde sunulmuştur.

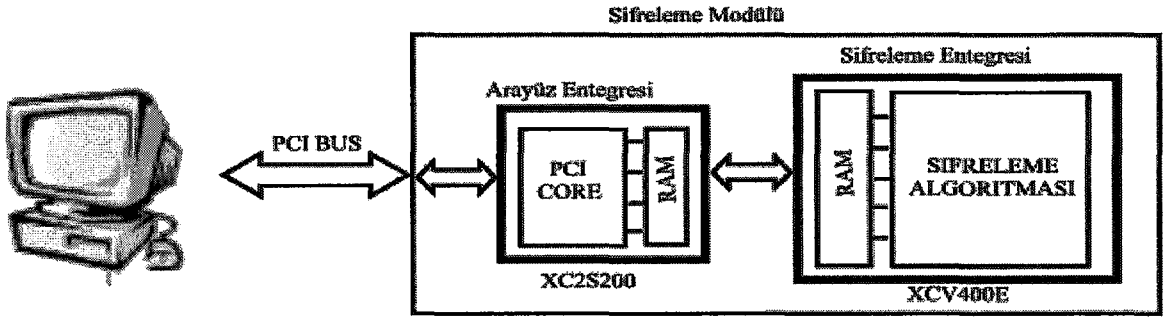
Tez çalışması sonucunda oluşturulan sistem bilgisayarımızın PCI Slot'una bağlı bir şifreleme modül kartı ile bilgisayarımızdan klavye yardımı ile şifrelenmesi için girdiğimiz bilgileri alıp şifreleme işleminden geçirerek, tekrar şifreli mesajı bilgisayarımıza geri göndermektedir, aynı işlem prosedürü şifre çözme içinde benzer veri akışı ile sağlanabilmektedir. Bu çalışma ile şifrelenen dosyalar, internet üzerinden gönderilebilir veya taşınabilir aygıtlar yardımı ile yine aynı modüle sahip başka bir bilgisayara taşınabilirler (Şekil 5.1).



Şekil 5.1 Genel sistem çalışması

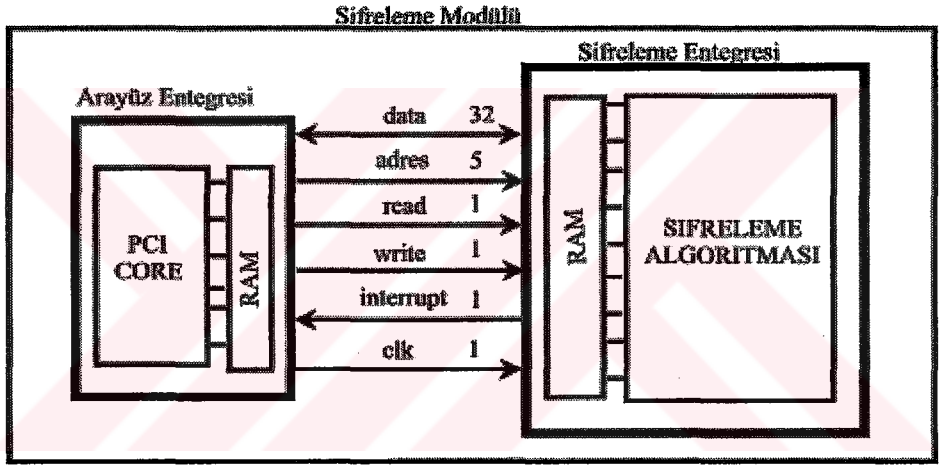
5.1 Sistem mimarisi

Şifreleme işlemi yapacak birimlerden bir tanesinin sahip olduğu donanım daha ayrıntılı olarak Şekil 5.2'de gösterilmiştir. Burada şifreleme kartı kişisel bilgisayarın PCI Slotuna yerleştirilir. Bilgisayarda çalıştıracağınız program ile kart bilgisayar tarafından tanınır, böylece veri gönderme ve alma işlemleri başlatılabilir. Bilgisayar sadece ara-yüz entegresi ile bağlıdır ve veriler, bu entegrenin RAM'ine yazılır veya okunur.



Şekil 5.2 Bilgisayar ile şifreleme kartı bağlantısı

Şekil 5.2'de belirtildiği üzere kullanılan entegreler XILINX firmasının ürünleridir. Ara-yüz entegresi olarak Spartan XC2S200 kullanılırken şifreleme algoritmasının çalışması için VIRTEX XCV400E entegresi kullanılmıştır. Bu entegreler ile ilgili olarak (Ek-12)'ye bakılabilir.

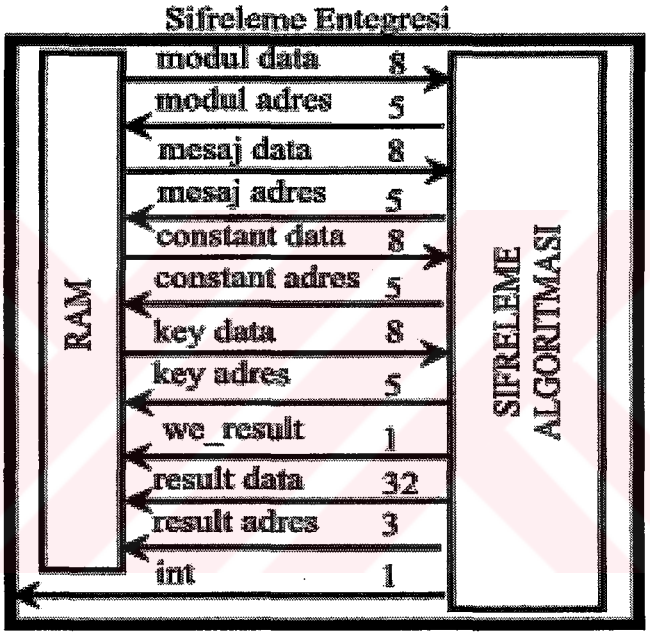


Şekil 5.3 Ara-yüz ve şifreleme entegrelerinin haberleşmesi

Şekil 5.3'de iki entegrenin birbirleri ile olan haberleşmeleri gösterilmiştir. Entegreler arasındaki haberleşme işlemi ara-yüz entegresinin bilgisayardan aldığı verileri kendi RAM bölgesine yerleştirmesinden sonra başlar. Ara-yüz entegresi öncelikle data hattına ilk veriyi ve bu verinin şifreleme entegresinin RAM bölgesinde yazılmasını istediği adresi gönderir, ardından yazma işlemi olduğu için (şifrelenen bilgi geri alınırken okuma yapılır.) "write" ucundaki sinyali 3 saat darbesi genişliğinde "1" seviyesinde tutar (Ek-1). Şifreleme entegresi ara-yüz entegresi tarafından hatlara gönderilen bilgiyi "write" sinyali 1 iken alır ve ilgili datayı RAM deki ilgili adres alanına yazar. Bu işlem aynı kapasiteye sahip iki RAM arasındaki tüm datalar transfer edilinceye kadar 32'lik paketler halinde devam eder. Transfer işlemi bittikten sonra şifreleme entegresi, yazılımın ürettiği dahili bir sinyal ile şifreleme algoritmasını devreye sokar. Anahtar uzunluğuna bağlı olarak değişen şifreleme süresi sonunda elde edilen sonuç RAM de kendisi için ayrılan alana yazılır ve ara-yüz entegresi

sonucun hazır olduğunun bildirilmesi amacıyla "interrupt" sinyali '1' yapılarak uyarılır(Ek-2). Bu sinyal ile uyarılan ara-yüz entegresi, sonuçları kendi ram bölgesine transfer etmek için tekrar adres bilgisini hatta göndererek yine 3 saat darbesi genişliğinde ama bu kez "read" sinyalini '1' seviyesinde tutar (Ek-3). Bu esnada şifreleme entegresi hattaki adres bilgisinin belirtmiş olduğu ram bölgesinden ilgili datayı alır ve hatta gönderir. Böylece ara-yüz entegresinin sonucu içeren RAM bölgesi dolmuş olur ve bu bilgi PCI-Core yardımı ile bilgisayara gönderilir.

Buradan sonra konumuz gereği ağırlıklı olarak şifreleme algoritmasının işletildiği entegrenin iç yapısı incelenecektir.

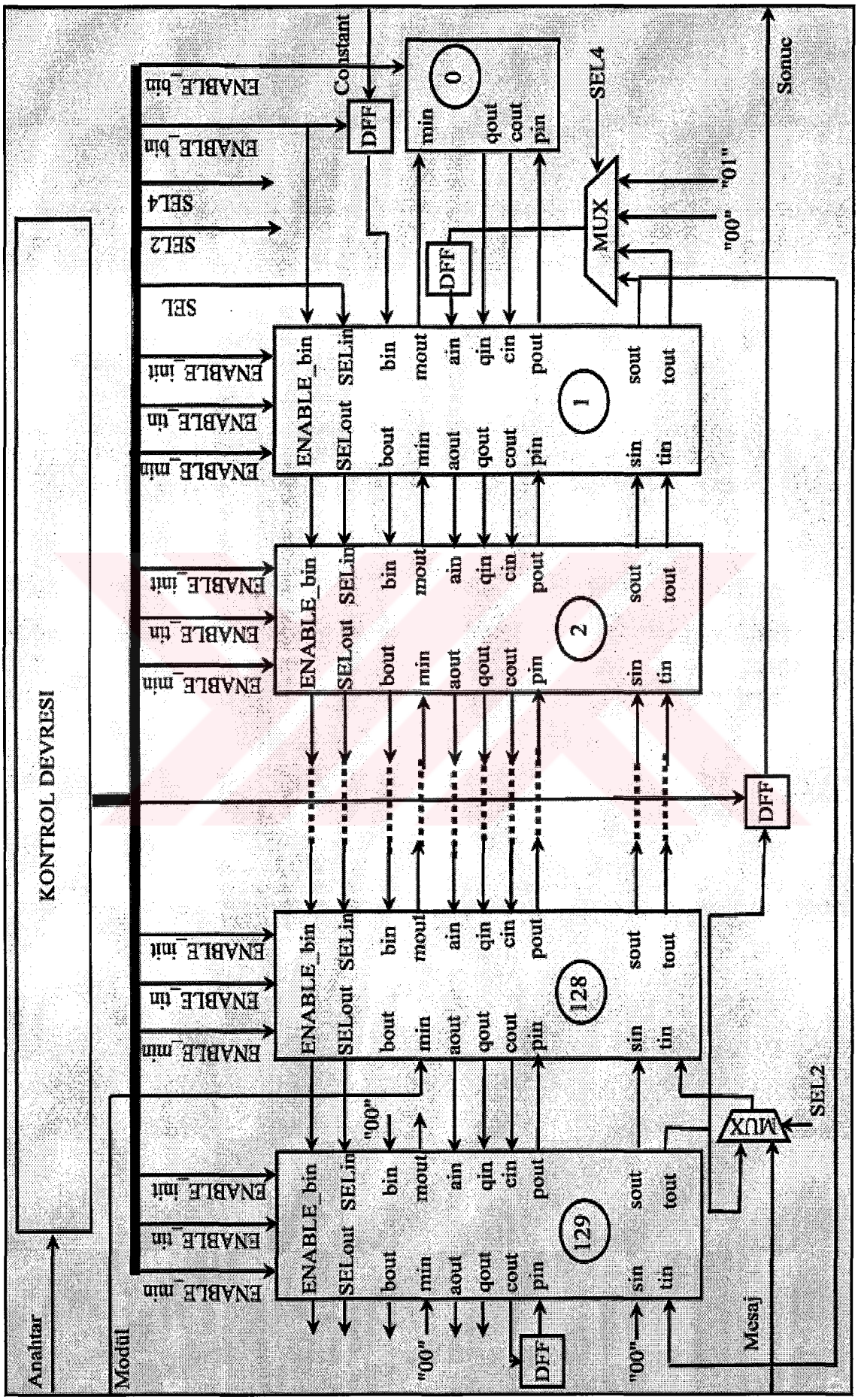


Şekil 5.4 Şifreleme entegresinin dahili sinyal yapısı

Şifreleme entegresinin içinde RAM ile algoritma arasındaki haberleşme Şekil 5.4'de gösterilmiştir. Burada algoritma, şifreleme için ihtiyaç duyduğu bilgilerin adresini paralel olarak çalışan RAM'lere gönderir ve hem şifreleme öncesinde hem de şifreleme esnasında hatta bu adresler doğrultusunda bilgileri hazır bulunur. Sekiz bitlik olan veriler bir süre hatta kalmalıdır çünkü algoritmaya key bilgisi tek, modül, mesaj ve constant bilgileri ise ikilik paketler halinde sokulmaktadır. Her sekizlik data ikişer ikişer tarandıktan sonra adres bilgisi 1 arttırılarak bir sonraki sekiz bitlik pakete ulaşılır ve aynı işlemler tekrarlanır. Bu paket uzunluğu tamamen sistemin tasarımına bağlıdır. Daha önceki bölümlerde de belirttiğimiz gibi aynı anda taranacak bit sayısının genişliği arttıkça, tarama sayısı azalır ancak işlem karmaşıklığı artar, dolayısıyla sistemin çalışabileceği frekans değeri de düşer.

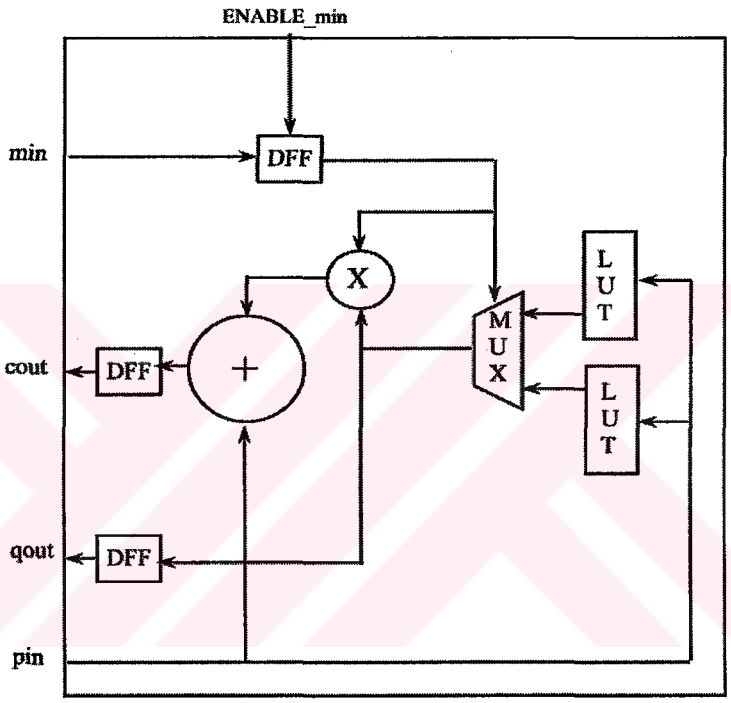
Şifreleme işlemi anahtar boyutuna bağlı olarak, değişken bir zaman sonunda sonucu, en az

ağırlıklı 2 bitinden başlayarak iki bitlik paketler halinde üretmeye başlar. Bu aşamaya gelinince algoritma kısmı "we_result " bağlantısını '1'yaparak RAM'i yazma moduna geçirir. Tıpkı dataların alınması gibi sonuç datası çıkarken, bilgi, ikişer bit halinde kaydırma işlemi ile RAM'in sonuç gözüne yerleştirilir. Sonucun yerleşmesini takiben yine algoritma kısmı bu kez ara-yüz entegresini uyarmak için "int" (interrupt) hattını '1'seviyesine çıkarır (Ek-2) ve işlem akışı bir üst paragrafta anlatıldığı şekilde ilerler. Şekil 5.5 şifreleme algoritmasının iç yapısını 256 bitlik RSA sistemine göre göstermektedir.



Şekil 5.5 Şifreleme algoritmasının iç yapısı

Şekil 5.5'den 256 bitlik RSA sistemi için toplam 130 adet işlem birimi gerektiği görülmektedir. Buradaki 130 sayısı genelleme yapılırsa k bit sayısı olmak üzere radix-4 için k/2+2 dir. İç yapılarına daha sonra değineceğimiz hücrelerden 0. işlemci birimi dışındakilerin yapıları tamamen aynıdır. Şekil 5.5'den de görüldüğü üzere yapımız sistolik yapının gereklilikleri ile örtüşmektedir. Algoritmada anahtar taşıyan veri hattı ve kontrol sinyalleri dışında kalan tüm hatlar 2 bit genişliğindedir. Bunun nedeni daha önce de belirttiğimiz üzere radix-4 tabanlı bir yapı kullanmamızdır. 0.işlemci biriminin iç yapısı Şekil 5.6'da gösterilmiştir.



Şekil 5.6 0. işlemci biriminin iç yapısı

Şekil 5.5'de 0.olarak belirtilen işlemcinin görevi algoritmadaki $q_i = (P_0) \cdot (r\text{-}n_0)^{-1} \bmod r$ satırının işlevini yerine getirmektir çünkü diğer işlemci birimleri algoritmada hemen bu satırı takip eden $P_{i+1} = (P_i + q_i \cdot n + a_i . B) / r$ işlemini gerçekleştirirler. Bu farklı işlemi yapan birimin neden en sağa yerleştirildiği ileride anlatılacaktır. Tekrar $q_i = (P_0) \cdot (r\text{-}n_0)^{-1} \bmod r$ işlem satırına dönersek, burada r ile sembolize edilen değer bit tarama için kullanılan değerdir, bu sistemin radix-4 tabanlı olduğunu bildiğimize göre değer 4'tür. Bu işleme giren diğer değerlerden biri modülün en az ağırlıklı iki biti n_0 , bir diğeri ise başlangıç değeri sıfır olan, ancak her yeni döngüde farklı değerler alabilecek olan P_0 dır. Şekil 5.6 incelenirse yapının tablolara (LUT- Look Up Table) dayalı olduğu görülür, bunun nedeni bu işlemin olası sonuçlarının az olması dolayısıyla giriş değerine göre çıkış değeri atamanın işlem yaparak çıkış değeri üretmekten daha basit olmasıdır. Matematiksel olarak iki farklı tabloya

ihtiyacımız vardır. Bu tablolar aşağıdaki işlemlerle elde edilir. İşlemlere başlamadan modül değerinin tek sayı olduğunu çünkü 2 asal sayının çarpımından oluştuğunu (2 pratikte kullanılmaz) 3. bölüm konularından hatırlamalıyız. Bu durum modülün son iki bitlik paketi için olası sonuçların "01"(1) ve "11"(3) ile sınırlı olduğunu göstermektedir.

$q_i = (P_0) . (r\text{-}n_0)^{-1} \bmod r$

$q_i = (P_0) . (4\text{-}n_0)^{-1} \bmod 4$

n_0= "01"(1) olsun

$q_i = (P_0) . (4\text{-}1)^{-1} \bmod 4$

$q_i = (P_0) . (3)^{-1} \bmod 4$

$q_i = (P_0) . 3 \bmod 4$

Buradan ;

P_0=0 ise $q_i = 0$

P_0=1 ise $q_i = 3$

P_0=2 ise $q_i = 2$

P_0=3 ise $q_i = 1$ değerleri çıkar

$q_i = (P_0) . (r\text{-}n_0)^{-1} \bmod r$

$q_i = (P_0) . (4\text{-}n_0)^{-1} \bmod 4$

n_0="11" (3) olsun

$q_i = (P_0) . (4\text{-}3)^{-1} \bmod 4$

$q_i = (P_0) . (1)^{-1} \bmod 4$

$q_i = (P_0) . 1 \bmod 4$

Buradan ;

P_0=0 ise $q_i = 0$

P_0=1 ise $q_i = 1$

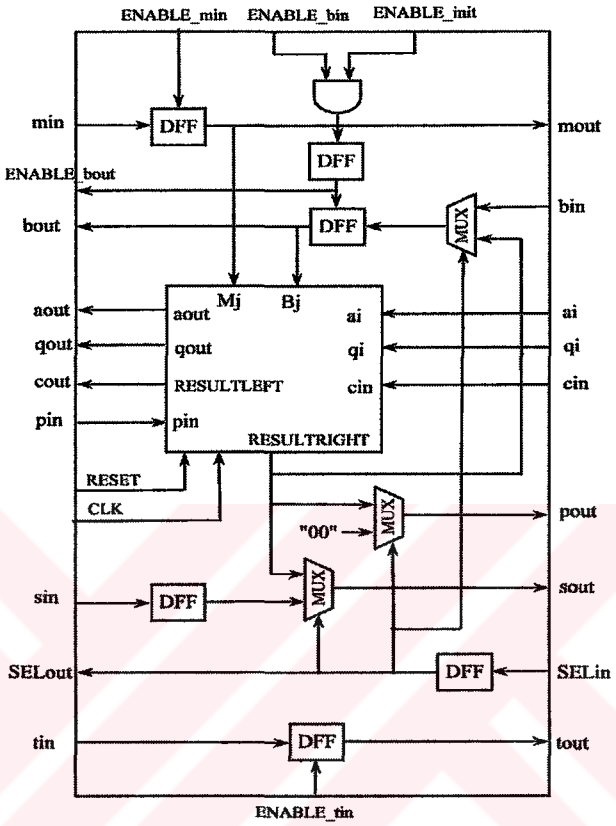
P_0=2 ise $q_i = 2$

P_0=3 ise $q_i = 3$ değerleri çıkar

Modül ve P_0 değerleri göz önüne alınarak Şekil 5.6'daki yapı ile algoritmanın ilk satırının görevini yerine getirebiliyoruz. 0.işlemci modülü için VHDL dilinde yazılan program pe_first.vhd (Ek-4) adıyla kullanılmıştır.

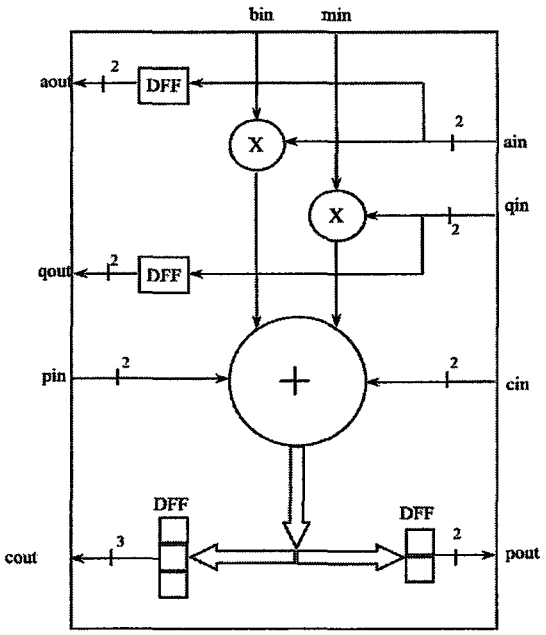
İlk işlemci birimi dışındaki tüm birimlerin ortak iç yapısı ise Şekil 5.7'de gösterilmektedir. Yapının içerisinde veri akışını saat darbeleri ile kontrol edebilmek için çok sayıda bellek elemanı kullanılmıştır. İşlemci birimlerinin ard arda bağlanması ile sistem içerisinde saat kontrolünde uzun mesafeli bir veri akışı sağlanmış olur. Asıl işlemi gerçekleştiren kısım şeklin ortasındaki hücredir. Diğer kısımlar bu hücreye ve çevredeki diğer işlemci birimlere bilgi akışını sağlamakla görevlidir. $P_{i+1} = (P_i + q_i . n + a_i . B) / r$ satırında kullanılan isimlendirmelerden sadece n değeri Şekil 5.7'de Mj ile gösterilmektedir, diğer isimlendirmeler aynıdır. İşlemci birimleri çalışma esnasında birbirini takip eden saat darbelerinde aynı işlemi farklı girişler için gerçekleştirmektedirler. Bir saat darbesinde "çarpma işlemi"(multiplication) işlemi yapılırken, takip eden saat darbesinde "kare

alma"(squaring) işlemi yapılmaktadır. Burada kullanılan MUX'lar şifreleme algoritmasına çalışma esnasında bu işlemler için farklı girişler sağlayabilmek amacıyla kullanılmaktadır. 0. işlemci birimi dışındaki işlemciler için VHDL dilinde yazılan program PE.vhd (Ek-5) adıyla kullanılmıştır.



Şekil 5.7 Genel işlemci birimi

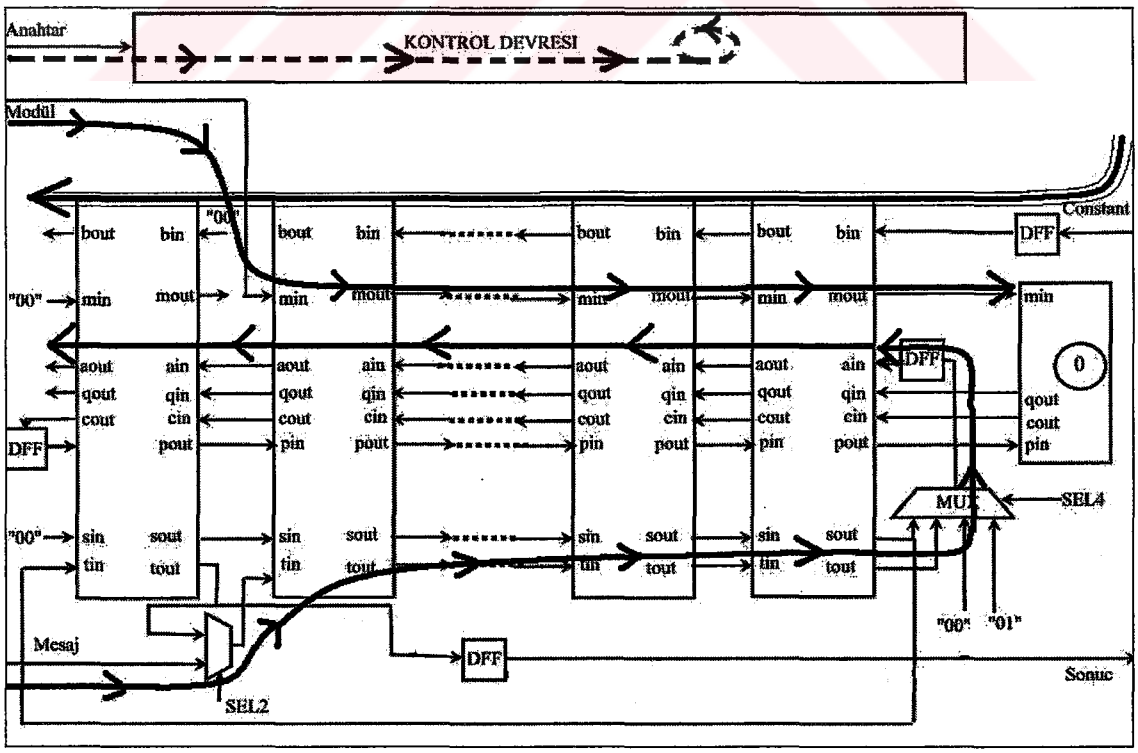
Şekil 5.8 işlemci biriminin çekirdeğini oluşturan ve montgomery olarak isimlendirilen yapıyı ayrıntılı olarak göstermektedir. İki bitlik veri hatları ile işlem yapan bu kısım sadece çarpma ve toplama bloklarından ibarettir. Dikkat edilirse girişlerin bazıları sağdan devreye sokularak soldan dışarı gönderilmiş, bazıları ise soldan giriş yapıp sağdan devreyi terk etmiştir. Bunlar devrenin bütünü düşünüldüğünde işlemci birimin işlem yapmak için hem sağındaki hem de solundaki işlemci biriminden veri alması gerektiğini gösterir ki bu da, zaman sınırlamalarının devre için çok hassas olduğuna işarettir. Her iki komşu işlemci birimden giriş alınarak oluşturulan sonuç, yine her iki komşu işlemci birime paylaştırılmaktadır. Montgomery yapısıyla ilgili VHDL dilinde yazılan kod montgomery.vhd adıyla kullanılmıştır (Ek-6)



Şekil 5.8 Montgomery yapısı

5.2 Sistemin çalışması

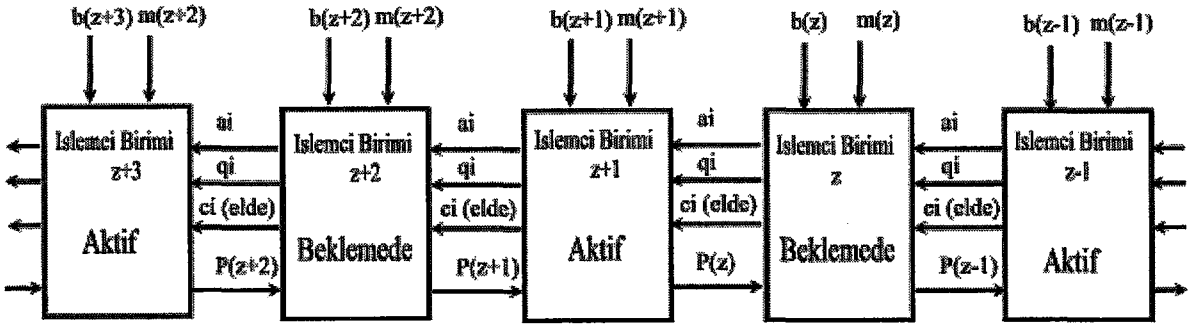
Çalışmanın ilk aşamasında, henüz algoritma işletilmeye başlamadan önce kullanılacak modül ve constant değerleri tüm işlemci birimlerine ulaştırılarak şifrelemenin ihtiyaç duyduğu ilk koşul oluşturulmaya çalışılır. Bunun için modül (mi), anahtar ve mesaj soldan, constant (bi) sağdan sisteme giriş yaparak, sistemi Şekil 5.9'da görüldüğü gibi baştan sona tararlar.



Şekil 5.9 İlk değerlerin yerleşmesi

Gerekli tüm datalar yerlerine yerleştikten sonra 1. işlemci birimi ain girişi şifrelenecek mesajımızın en az ağırlıklı iki biti, diğer girişleri ise "00" olacak şekilde şifrelemenin ilk işlem yordamını başlatır. Buradan elde edilen sonuç iki farklı bölüme sahiptir, bunlardan sol tarafa aktarılan 3 bit genişliğindeki kısım sonucun iki bitten taşan kısmı elde(cout), sağ tarafa aktarılan diğer kısım ise sonucun iki bitlik kalanıdır. Bir sonraki saat darbesi ile 1. işlemci biriminin girişine yeni değerler gelirken 1. işlemci biriminin ain ve qin değerleri artık 2. işlemci biriminin girişlerine kaydırılmıştır. Diğer giriş olan cin her yeni işlem sonunda oluşan eldedir dolayısıyla bu değer için sabit bir kaydırma söz konusu değildir. İkinci saat darbesi ile hem 1. hem de 2. işlemci birimleri birbirlerinden bağımsız işlemler yaparlar. Bu aşamaya kadar 0. işlemci biriminden ilk koşullar nedeniyle qin ve cin değerleri için sadece "00" sonuçları elde edilir, ancak 0. işlemci birimi 1. saat darbesi ile 1. işlemci biriminden sağ tarafa pout ucu ile gönderilen sonucu 2. saat darbesi esnasında işler ve 3. saat darbesine gelindiğinde artık 1. işlemci girişindeki cin ve qin değerleri "00" dan farklıdır ve algoritmanın önceden ürettiği sonuçlara bağlıdır. Kabaca 3. saat darbesi geldiğinde 1. işlemci girişlerinde yeni değerler belirirken bir darbe önce giriş olan değerler bir soldaki işlemci birimlerinin girişleri olacak şekilde kaydırılırlar. Kaydırılan ain, qin bilgileri belirli bir kaydırma işleminden sonra en soldaki işlemci biriminde de işlem görürler ve bir daha kullanılmazlar ancak cin değeri bir sonraki işlemde de kullanılmak için pin girişinden tekrar döngüye sokulur. İşlemci birimlerinin aynı saat darbesi esnasında birbirlerinden farklı değerleri işlediklerini belirtmiştik, bir sonraki paragrafta bunun nedenlerini inceleyeceğiz.

Bölüm-4'de MonPro ve MonExpo algoritmalarına değinmiş ve bunlarla ilgili sayısal bir örnek çözmüştük. MonPro3 algoritmasını ve şekil 4.9'u incelediğimizde bit tabanlı işlem yaparken işlemci birimlerimizin bir diğer işlemci biriminin sonucuna ihtiyaç duyduğunu görüyoruz. İşlemci birimlerimizin bütün çıkışlarının önünde data akışını kontrol etmek amacıyla bir bellek ünitesi bulunduğunu daha önce belirtmiştik, bunu da dikkate alarak örneğin z. işlemcinin h. saat darbesi ile girişlerindeki bilgiler doğrultusunda bir işlem gerçekleştirdiğini kabul edelim. h. darbe süresince yapılan işlemler sonucunda oluşan bilgi z. işlemcinin bellek birimlerinin önüne gelir. (h+1). saat darbesi ile hem sonuç bilgileri hem de kaydırılması gerekli bilgiler bu kez (z+1). işlemci biriminin girişlerine gelmiştir, bu esnada z. işlemci biriminin de girişlerine yeni bilgiler gelmiştir ancak z. işlemcinin ihtiyaç duyduğu ve (z+1). işlemcinin (h+1). saat darbesi süresince üretip (h+2). saat darbesi ile çıkışa vereceği P(z) değeri eksiktir. Bu nedenle z. işlemci 1 saat darbesi boyunca boş duracak ve solundaki işlemcinin çıktısını bekleyecektir. Bu durum tek bir saat darbesi için şekil 5.9'da gösterilmiştir.



Şekil 5.9 Bir aktif, bir beklemede çalışmasının gösterimi

MonExpo algoritmasının aşağıdaki parçasını incelediğimizde;

For i = 0 **to** *z-1* **do**

İf (*ei* =1) **then**

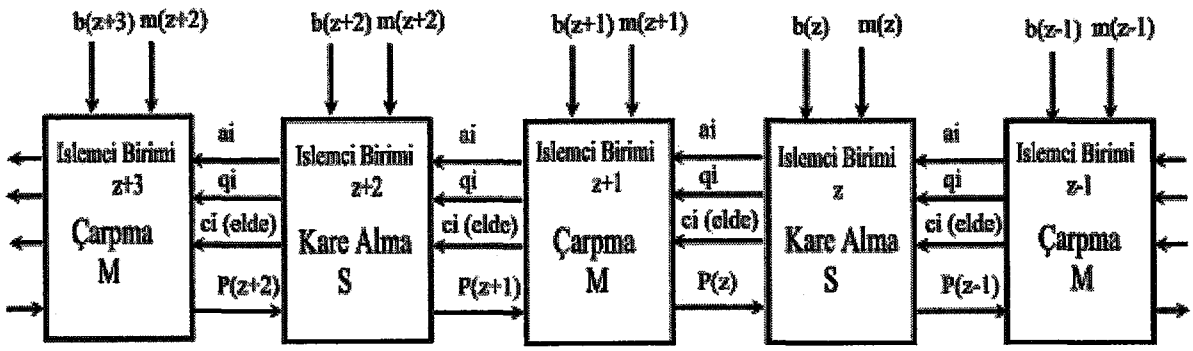
R:= MonPro(R,P,n) --------**(Multiplication)**

End if;

P:= MonPro(P,P,n) --------**(Squaring)**

End for;

P:= MonPro(P,P,n) işleminin ve R:= MonPro(R,P,n) işleminin birbirlerine göre tek bir döngü için bağımsız olduklarını görmekle birlikte aynı algoritma yordamını işlettiklerini (MonPro) ve girişlerinin iki tanesinin de ortak olduğunu (P,n) fark ediyoruz. Bu işlemler farklı olan üçüncü girişlerinin durumuna göre isimlendirilirler, eğer üçüncü giriş P'den farklıysa çarpma (multiplication), P'ye eşitse kare alma(squaring) olarak isimlendirilirler. Burada birbirleri ile paralel çalışan algoritma parçaları için aynı yapılardan iki tane kullanmak yerine Şekil 5.10'da gösterildiği gibi bekleme ile kaybedilen zamanda diğer işlemin yapılması donanımsal olarak büyük avantaj sağlamakta ve hiç boş saat darbesi bırakmamaktadır.



Şekil 5.10 Araya-sıkıştırma(interleaving) mantığı gösterimi

Böylece normalde bekleme yapacak olan işlemci modüller her iki işlemi aynı yapıda ve aynı zaman zarfında bitirebilmektedirler. Bu işlemler dizini hem çarpma hem de kare alma (tek bir MonPro yordamı) için k bit sayısı olmak üzere k+4 saat darbesi sonucunda elde edilmiş olur.

Burada dikkat edilmesi gereken diğer bir nokta da MonExpo algoritmasının, içinde paralel çalışan MonPro'lar barındıran birden fazla döngüden oluşmasıdır. Bir döngüden çarpma ve kare alma için sonuç elde edildikten sonra bir sonraki döngüye geçildiğinde R:= MonPro(R,P,n) işlemi diğer işlemin sonucuna ihtiyaç duyar. Dolayısıyla k+4 saat darbesi sonucunda ilk iki biti ortaya çıkan sonucu yine kademeli olarak girişe yönlendirmeli ve yeniden bir MonPro işlemine başlanmalıdır. Anahtarın işlenen bitinin '1' veya '0' olmasına bağlı olarak R:= MonPro(R,P,n) işlemi bazen yapılmakta bazen yapılmamaktadır. Yapılmadığı zamanlar çarpma kısmı boş olan bir MonPro döngüsü gerçekleşmektedir. Böyle durumlarda daha önceki MonPro işleminin sonucunun bir sonraki döngüde kullanılmak üzere hafızada tutulması gerekmektedir. Şekil5.5'de soldan sağa data akışı sağlayan sin ve tin olarak isimlendirilmiş bilgi hatları ve MUX yapıları bu görevleri yerine getirirler.

En soldaki işlemci modülünün sağında bulunan MUX ilk mesaj datasının sisteme sokulmasında ve her bir MonPro işlemi sonucunda elde edilen ara değerlerin tekrar sistemin girişine tıpkı orijinal mesajda olduğu gibi yönlendirilmesini sağlar Sonuç bilgisi en alt ağırlıklı 2 bitinden itibaren elde edilmeye başlanır. Buradaki MUX ile sonuç en az ağırlıklı 2 bitinden itibaren tin akışına eklenir. Sonucun en ağırlıklı 2 biti elde edilip diğer döngüdeki işlemlere geçileceği zaman bir önceki işlemin sonucunun en az ağırlıklı 2 biti soldan başlayarak tin ve ona bağlı bellek birimleri sayesinde en sağa ulaşmış olur. Buradan anahtarın ilgili bitinin durumuna göre seçme ucu ile sisteme sokulmaya başlanır veya diğer işlem bitene kadar durumunu muhafaza eder. Anahtar ile ilgili işlemleri kontrol devresi yapmaktadır. Anahtar değerleri sistemimize tek bitlik bir hat ile dahil olur. MonExpo'nun her bir MonPro'lu döngüsü için anahtarın tek bir bitine bakılarak karar verilir, ancak anahtarın şimdiki değerinin yanında önceki ve sonraki değerlerinin de dikkate alınması gerekmektedir keza sonuçların bazı zamanlarda anahtar bitinin durumuna göre birkaç döngü boyunca saklanması gerekebilmektedir. Anahtarın tüm bitleri için döngü yapıldıktan sonra MonPro bir kez de algoritmanın sonundaki R:= MonPro(1,R,n) işlemi için yapılır(*n*-kalanından kurtulma işlemi) ve sonuç elde edilir. Bu işlemdeki sabit '1' değerini sisteme dahil etmek amacıyla yapının sağ tarafında bir ucu sabit "01", bir ucu sabit "00" olan 4x1 lik MUX kullanılmıştır. Tüm algoritma işletildikten ve istenen sonuç oluştuktan sonra sonuç, yine en az ağırlıklı 2 bitinden başlayarak en soldaki işlemci biriminin tin çıkışından her saat darbesinde 2 bit olacak

şekilde dışarı gönderilir.

5.3 Bilgisayar ara-yüzünün çalışması

Bir önceki bölümde şifreleme kartının önce bilgisayarla olan haberleşme ağı, ardından şifreleme entegresinin kendi içinde RAM'i ile nasıl haberleştiği ve nihayetinde şifreleme entegresi içinde şifreleme işleminin nasıl gerçekleştiği ayrıntılı olarak anlatılmıştır. Bu bölümde anlatacaklarımız algoritma tasarımından ziyade şifreleme kartımızın bilgisayar ile haberleşmesi esnasındaki ara-yüz programının çalışması ile ilgilidir. Bu bölümde sistem, kullanıcı açısından değerlendirilmektedir. Bölüm içerisinde şifreleme ve şifre çözme için kullanılabilecek 2 farklı ara-yüz tanıtılacaktır ve her biri için mesaj, açık-anahtar ile şifrelenecek ve şifreli mesaj özel anahtar yardımı ile çözülecektir.

Tez kapsamında, kullandığımız PCB kartımız üzerindeki Xilinx XCV400E FPGA'i ile 256 bitlik RSA sistemi, entegrenin %72'lik kısmı kullanılarak gerçekleştirilmiştir. Sistemin daha yüksek bitlik yapılara ulaşması için yazılımda N ile belirtilen kısmın istenilen bit uzunluğuna getirilmesi yeterli olmaktadır. Farklı bit uzunlukları için yazılım dosyası kullanılarak sonuçlar simülasyonda görülebilir, keza 1024,2048,...bitlik sistem ile 256 bitlik RSA sistemi arasında mantıksal hiçbir fark yoktur, sadece gerçeklemelerde daha büyük kapasiteli entegreler kullanılmalıdır. Zaten yapıya istenildiği gibi ekleme yapılabilir olması sistolik niteliğinin getirdiği bir zorunluluktur.

Öncelikle uygulamamızda kullanacağımız RSA sistemimizin gerek duyduğu modül, constant, açık ve özel anahtar girdilerinin hesaplanması gerekmektedir.

Bunun için ilk olarak iki adet büyük asal sayı seçmemiz gerekmektedir. Bunlar ;

Birinci asal sayı ; *p*

Decimal = 2.425.967.623.052.370.772.757.633.156.976.982.469.681--40 digit

Hexadecimal = 721185e72870099062df79115634dac31 ----(132 Bit)

İkinci asal sayı ; *q*

Decimal = 671.998.030.559.713.968.361.666.935.769--30 digit

Hexadecimal = 87b57be17f0ecdbf18a227bd9 --- (100 Bit)

olsunlar. Modül değerimizin n = p x q olduğunu hatırlarsak ; bu değer

Modül Değeri ; $n = p \times q$	
Decimal	= 1.630.245.
	464.892.823.
	711.538.270.
	142.482.769.
	515.544.228.
	277.190.451.
	217.281.557.
	216.919.689

Modül Değeri ; $n = p \times q$	
Hexadecimal =	000003c
	781b3378
	1cb6ea74
	453b5298
	109c822d
	de0c51ef
	5187561b
	c7148089

şeklinde elde edilmiş olur. Sırada n'in Euler fonksiyonunun hesaplanması vardır ki o da aşağıdaki gibi hesaplanmalıdır.

Euler Değeri; $\varphi(n) = (p-1) \times (q-1)$
Decimal=1.630.245.464.892.823.711.538.270.142.480.343.547.920.503.908.387.133. 870.156.218.567.514.240

İkinci aşamada, sistemde kullanılacak olan constant değerinin hesaplanması gerekmektedir. Bu değer, hesaplanmasından da anlaşılacağı üzere sadece modül ve bit uzunluğuna bağlıdır. Yani bit uzunluğumuzu ve modül değerimizi hesapladıktan sonra bu değer ne şifreleme de ne de şifre çözme işleminde değişmez hep aynı olarak kalır. Burada akla bu değerin sabit olmasından dolayı FPGA içine yerleştirilmesinin daha uygun olabileceği düşünülebilir. Ancak böyle yapılırsa aynı bit uzunluğuna sahip fakat farklı bir modül ile şifreleme yapılmak istenildiğinde bu sabit değerin tekrar yazılım yükleme işlemi ile FPGA içerisine yerleştirilmesi gerekmektedir. Bir de RSA sistemi k bitlik ise bu sistem k bit ile k/2 bit arasındaki tüm bit uzunluklarındaki şifreleme işlemini kapsamaktadır Buradaki örnekte RSA sistemimiz 256 bitliktir ancak modül değerimiz 232 bit uzunluğunda kullanılmaktadır. Bu nedenle, constant değerini buradaki gibi sisteme arayüz ile girmek farklı işlemler yapabilme yeteneğimizi arttırır. Constant değerini hesaplarsak;

Constant Değeri ; $C:= r^{2(m+1)} \bmod n$
Decimal= 1.319.011.
445.150.237.
778.761.004.
013.726.023.
827.068.347.
920.295.487.
268.268.612.
651.989.274

Constant Değeri
Hexadecimal = 00000030
ecc3357b
ff3bbfa2
32d2ba16
8bc33d66
eb67cff0
146d2528
cb1e391a

olarak elde ederiz. Buradan sonra anahtar değerlerini hesaplamamız gerekmektedir. Biliyoruz ki anahtarlardan birini $OBEB(\phi(n),e)=1$ koşulunu sağlaması şartıyla keyfi belirleyebiliyoruz. Bu bilgiden hareketle e açık anahtarı ve onun n modul tabanına göre tersi olan d özel anahtarını hesaplayabiliriz.

Açık anahtar ; $e=31$ secilirse
Özel(Gizli) anahtar ; *d*
Decimal = 262.942
816.918.197.
372.828.753.
248.787.152.
185.148.468.
372.320.505.
462.928.422.
349.599.071

Özel(Gizli)anahtar
Hexadecimal = 00000009
c0ca94b0
46b225d0
b053e406
dc4f1e88
33f3bd19
7bc7b1fb
1ad8715f

Böylece açık ve özel anahtar çifti de elde edilmiş olur. Tüm bu bilgiler ışığında herhangi bir mesaj değeri seçerek onu açık-anahtar ile şifreleyebilir, şifrelediğimiz mesajdan da özel

anahtar yardımı ile orjinal mesajımızı yeniden elde edebiliriz. Mesajımız olarak ondalık tabanda 763 sayısını belirlemiş olalım bu sayının hexadecimal olarak değeri "2FB" dir. Mesaj bu forma herhangi bir teknikle getirilmiş olabilir, harfin ASCII değeri, alfabedeki sırası veya daha kullanıcıya özgün bir kodlama kullanılabilir, yeter ki mesaj bir sayı olarak ifade edilsin. 763 değeri ve bulduğumuz parametreler yardımı ile şifreleme matematiksel olarak;

Şifreleme ; $C = P^e \bmod n$

C=763^31mod1.630.245.464.892.823.711.538.270.142.482.769.515.544.228.277.190.451.217.28 1.557.216.919.689

Hexadecimal=	0000000c	Decimal=	332.673.
	56ebd87b		238.353.750.
	28e6d6db		196.998.752.
	8539d8f4		246.827.940.
	96364ada		179.436.866.
	ca0dc74e		915.865.921.
	d4455a89		641.815.777.
	33425d1c		126.538.524.

şeklinde yapılır ve sonucunda C değeri elde edilmiş olur.

Şifrelenen mesajı çözerek orijinal mesaja ulaşmak istersek bu kez özel anahtar ile aynı işlemi tekrar yapmamız gerekmektedir. Bu da;

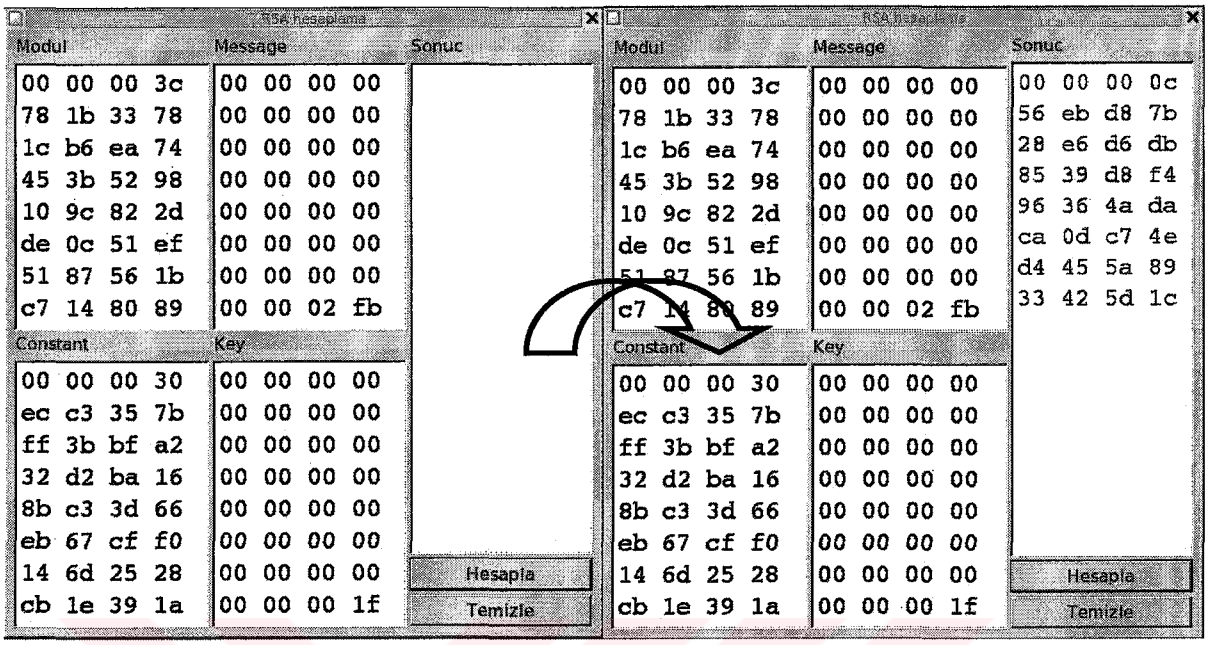
De-şifreleme ; $P = C^d \bmod n$

P=32.673.238.353.750.196.998.752.246.827.940.179.436.866.915.865.921.641.815.777.126.538.524^262.942.816.918.197.372.828.753.248.787.152.185.148.468.372.320.505.462.928.422.349.599.071**mod**1.630.245.464.892.823.711.538.270.142.482.769.515.544.228.277.190.451.217.281.557.216.919.689

P= 763

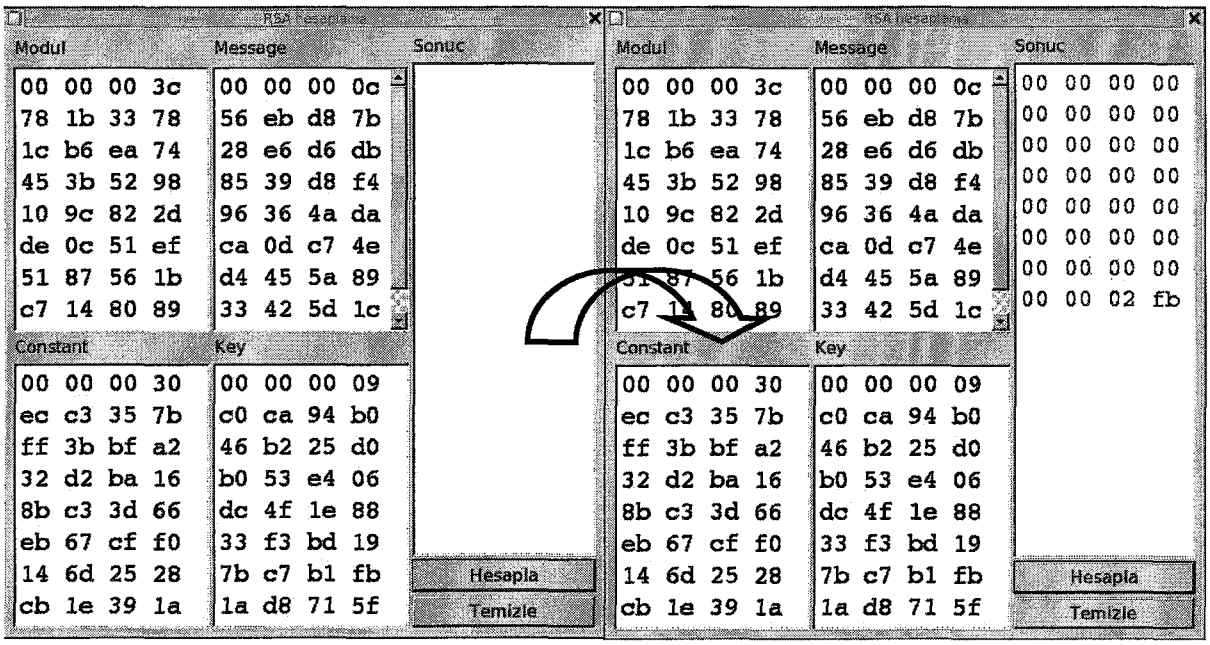
şeklinde elde edilir. Şimdi burada kullanılan bilgiler ile bilgisayar ara-yüz kısmının nasıl şifreleme ve şifre çözme yaptığı anlatılmaya çalışılacaktır.

Bilgisayarda şifreleme işlemi için iki farklı ara-yüz oluşturulmuştur. Bunlardan birinci Şekil 5.11'de gösterilmektedir.



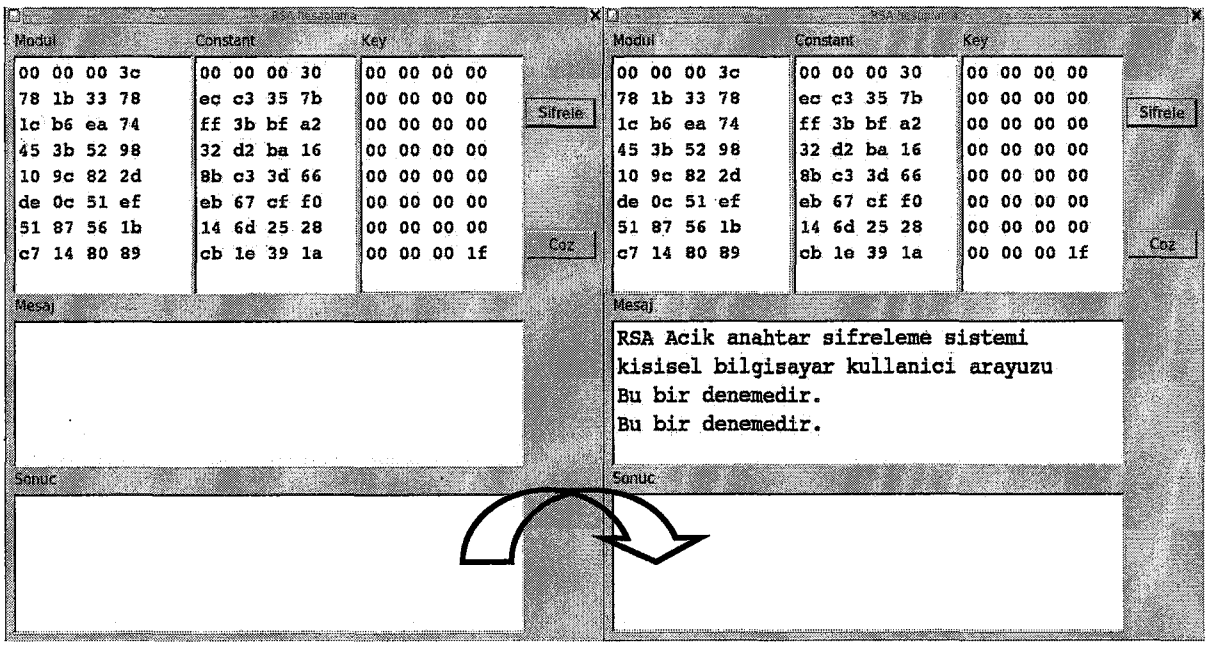
Şekil 5.11 Birinci ara-yüz programı ile şifreleme işlemi

Program çalıştırıldığında karşımıza modul, message, constant ve key değerlerinin hexadecimal formatta girilebilmesi için boş pencerelerden oluşan bir ekran gelir. Buralara RSA sistemi için önceden hesaplamış olduğumuz değerler girilerek hesapla düğmesine bastığımızda daha önce ayrıntılı anlattığımız üzere bilgiler bilgisayarımızın PCI portu ile şifreleme kartındaki arayüz entegresine oradan da şifreleme entegresine gider. Burada şifreleme belirli bir zaman zarfında yapılır ve sonuç tekrar aynı yoldan bilgisayarın PCI portuna oradan da ekranda "sonuç" olarak adlandırılan pencereye yazdırılır (Şekil 5.11). Bu program ile şifrelenen mesajımızı yine hexadecimal formatta elde etmiş oluyoruz. Şifreli mesajımızdan bu program yardımı ile orjinal mesajımızı elde etmek istersek, modül ve mesaj penceremizdeki değerlerimizi değiştirmeden, sonuç penceresinde elde ettiğimiz değeri mesaj penceresine olduğu gibi kopyalıyoruz. Bir de anahtar penceresine bu kez çözme işlemi yapacağımız için özel anahtarımızı gireriz (Şekil 5.12). Şifreleme ve şifre çözme işlemleri aynı yordam ile çalıştığı için tekrar hesapla düğmesine basarak bekliyoruz ve orjinal mesajımızı sonuç penceresinden elde etmiş oluyoruz.



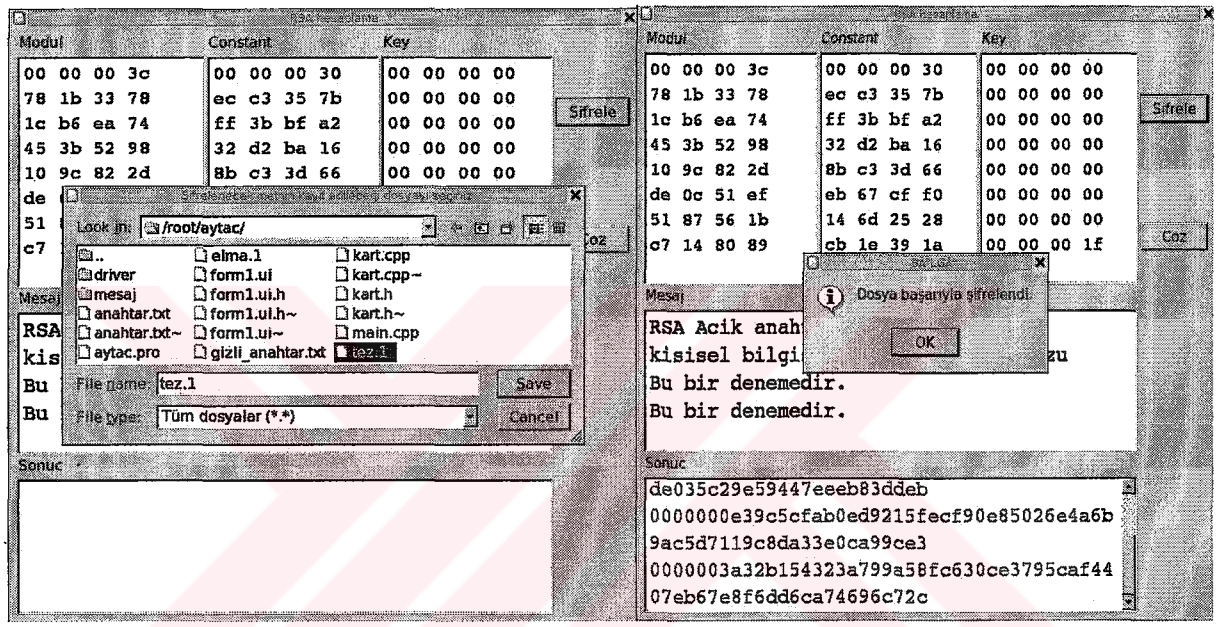
Şekil 5.12 Birinci ara-yüz programı ile şifre çözme işlemi

Birinci ara-yüz programı daha çok programın doğru çalışıp çalışmadığının kararının verilmesinde kullanılabilecek yapıdadır, gerçek zamanlı şifreleme işlemlerinde kullanılmak için uygun bir düzende hazırlanmamıştır. Çünkü şifrelenecek mesajın öncelikle hexadecimal formata çevrilmesi gerekmektedir, uzun bir mesajı şifrelemek istediğimizde o mesajı 256 bitlik dilimler halinde programımıza giriş yapmamız gereklidir vs. bu nedenler birinci programın çok kullanılabilir olamadığını göstermektedir. Tez kapsamında bilgisayar ara-yüzü için ikinci bir program daha oluşturulmuştur.



Şekil 5.13 İkinci ara-yüz programı ile şifreleme işlemi 1

İkinci ara-yüz programı da çalıştırıldığında birinci ara-yüz programındaki gibi hesaplanan değerlerin girilmesi gereken boş pencerelerle oluşturulmuş bir ekranla karşılaşırız. Buradaki pencerelere yine sayısal örneğimizde hesapladığımız modül, constant, key değerlerini girerek sistemimizi ayarlarız.(Şekil 5.13) Mesaj için ayrılan kısma ise şifrelemek istediğimiz mesajı klavyeden normal mesaj formatında gireriz. Mesajımızı bitirdikten sonra şifrele düğmesine basarız. Program, şifreleme işlemi bittikten sonra şifreli bilgilerimizi bir dosyada saklayacağı için bizden o dosyayı oluşturmamızı veya mevcut dosyalar arasından seçmemizi ister. Örneğimizde tez..1 adlı bir dosya oluşturulmuştur.



Şekil 5.14 İkinci ara-yüz programı ile şifreleme işlemi 2

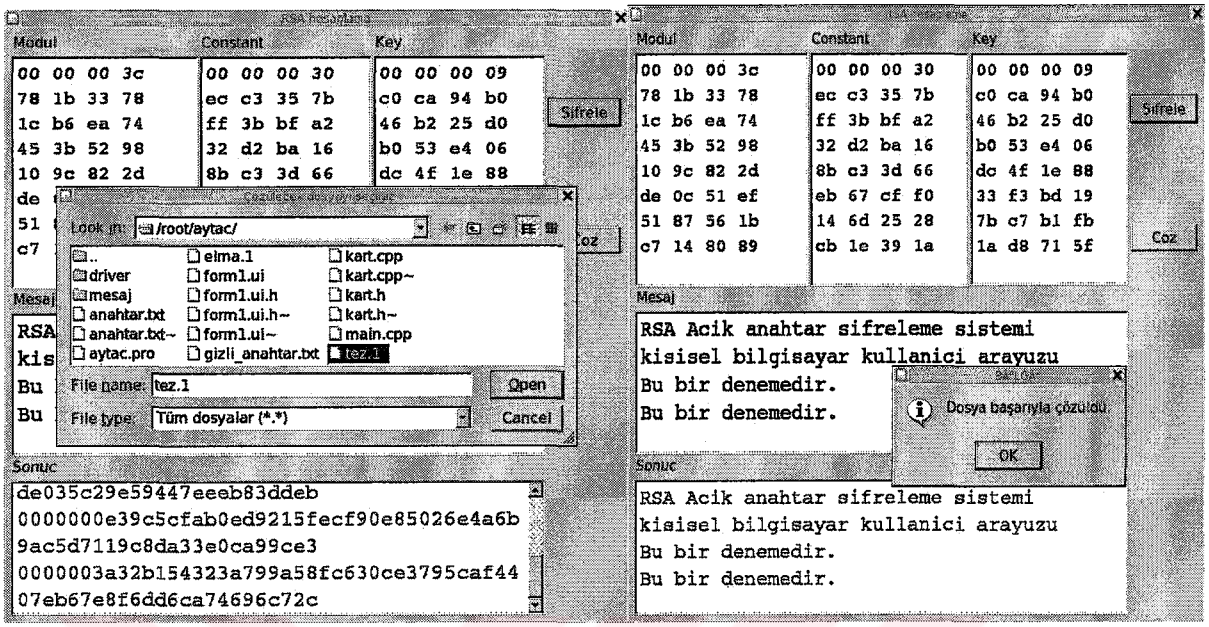
Dosyayı seçtikten sonra şifreleme işlemi aktif hale geçer ve bir süre sonra şifreleme işleminin tamamlandığı bilgisini ekrana yazdırır(Şekil 5.14).

Şifrelenmiş bilgi artık tez.1 isimli dosyadadır. Bu aşamadan sonra dosya ya internet vasıtasıyla elektronik postanıza eklenerek yada taşınabilir bellek üniteleri yardımı ile aynı donamına sahip diğer bir bilgisayarda çözülmek için güvenli bir şekilde taşınabilir.

Herhangi bir yolla diğer bilgisayara ulaşan dosyayı çözmek için yine aynı program çalıştırılmalı ve modül ile constant değerleri aynı olacak şekilde pencereler doldurulmalıdır. Key penceresine ise açık anahtarın çözücüsü olan özel anahtar girilmelidir.

Tüm bilgiler yerleştirildikten sonra çöz düğmesine basılarak çalışmaya başlanır. Program, çözülecek dosyanın belirtilmesi için dosya dizinine girer. Burada çözülecek dosya seçildikten sonra (tez.1) şifre çözme işlemi aktif olur ve çözme süresince bekler ardından şifre çözme işleminin bittiğini belirten bir uyarı oluşturur. Uyarı ile birlikte sonuç penceresinin içinde

güvenli olarak taşınan mesajın orjinal hali de belirmiş olur yani güvenli haberleşme sağlanmıştır(Şekil 5.15)



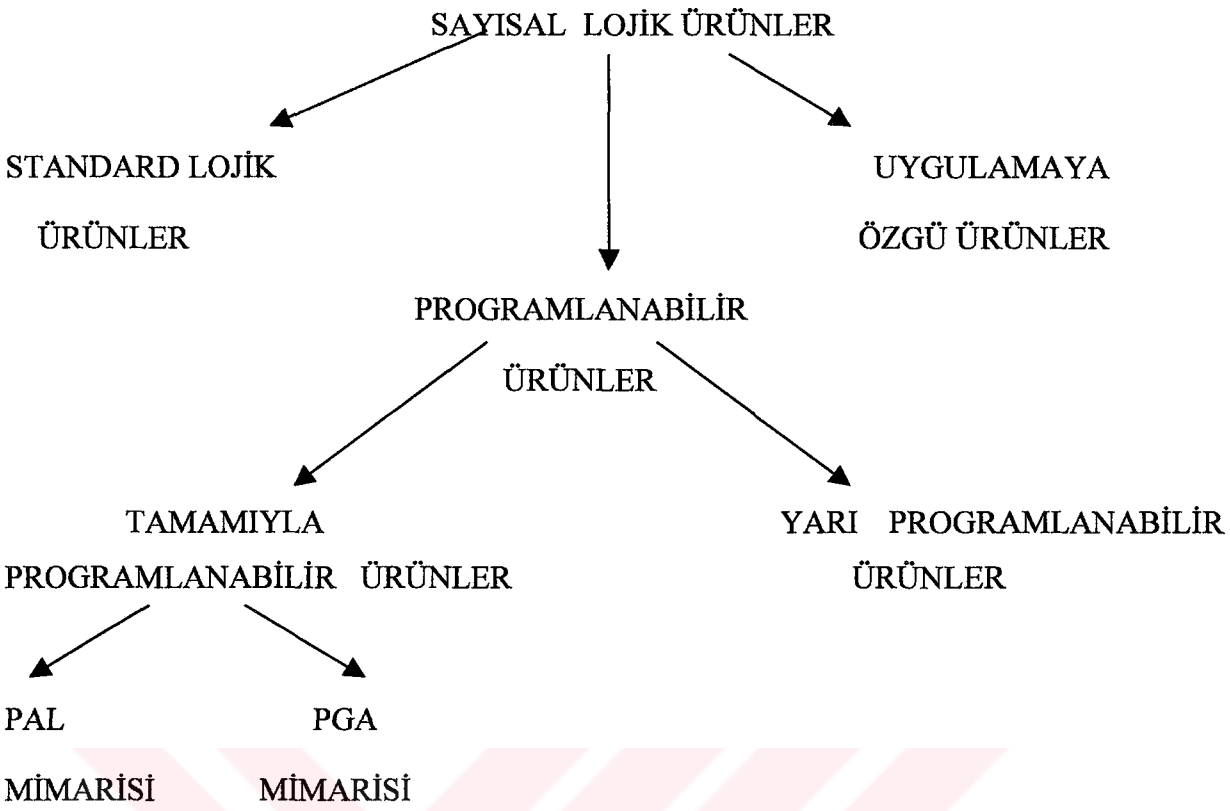
Şekil 5.15 İkinci ara-yüz programı ile şifre çözme işlemi

Buradaki örnekte şifreleme ve şifre çözme işlemleri aynı bilgisayarda ve temizleme işlemi yapılmadan yapılmıştır amaç şifrelenen mesaj ve şifresi çözülen mesajın değişikliğe uğramadan elde edildiğini tek ekranda gösterebilmektir.

5.4 Sahada programlanabilir kapı dizileri (FPGA)

Bu bölümde üzerinde şifreleme algoritmasını koşturduğumuz FPGA'ler ile ilgili genel bir bilgi sunulacaktır. Daha sonra kullandığımız Xilinx firmasının entegrelerinin iç yapısı anlatılacak ve en sonunda da tez çalışmasının yazılım geliştirme ortamından elde edilen sentez sonuçları sunlacaktır.

FPGA'ler sayısal ürünler yelpazesi içinde tamamıyle programlanabilir ürünler arasında yer alır (Şekil 5.16).



Şekil 5.16 Sayısal ürünlerin sınıflandırılması

FPGA'lerin elektronik pazarındaki payı, özellikle yüksek kapı miktarlarına ulaşım ve karmaşık fonksiyonların bu yapı içerisinde kolaylıkla gerçekleştirilebilmesi sayesinde, hızlı bir biçimde artmaktadır. Ayrıca FPGA'ler kullanılarak tasarım ve süre bakımından da önemli avantajlar elde edilir.

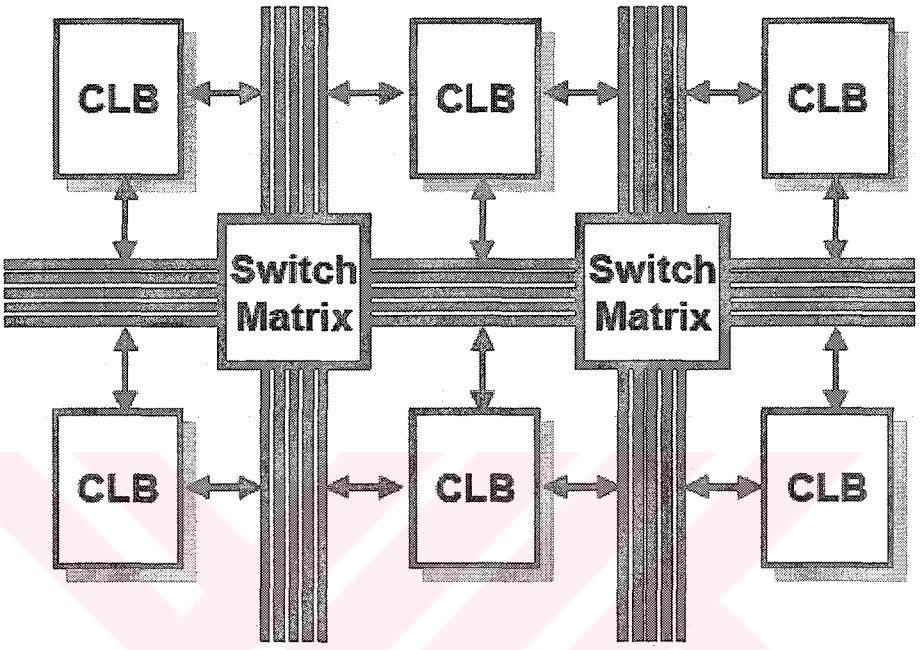
FPGA'lerde iki çeşit programlanabilen anahtar kullanılmaktadır, bunlar SRAM tabanlı anahtarlar ve "anti-fuse" tipi anahtarlardır. Anti-fuse tipi anahtarlar, yarıiletken üzerinde aslında açık devre olan ancak üzerinden büyük bir akım geçirildiğinde iletken hale geçen bölgeler kullanan yapılardır. Bu durum iyi üretilmemiş ayrık yarı iletken elemanlarda da vardır. Bu dezavantaj, FPGA teknolojisinde bir avantaj olarak kullanılmaktadır. SRAM'li anahtarlar yeniden programlanabilir ancak anti-fuse tipi olanlar sadece bir kez programlanabilirler (OTP: One-time programmable).

FPGA yapıları için henüz bir standart yoktur, farklı firmalar farklı teknolojiler ve farklı yapılar kullanılmaktadır. Farklı firmaların ürettiği FPGA'ler karşılaştırılırken, eşdeğer NAND kapısı sayısına bakılır.

Piyasada Xilinx, Actel ve yeni gelişen QuickLogic FPGA'leri bulunmaktadır. Xilinx'in FPGA'leri SRAM tabanlı anahtarlar kullanmakta dolayısıyla bir çok defalar

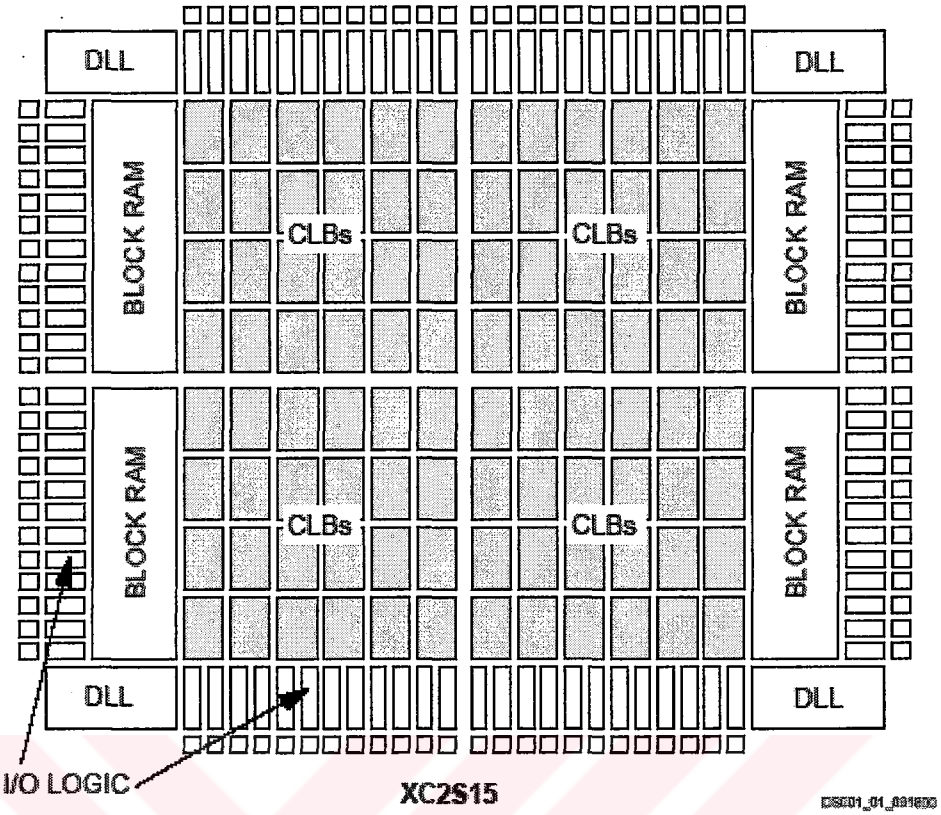
programlanabilmektedir. Ancak Actel ve QuickLogic firmalarının FPGA'leri anti-fuse özelliklidirler. Anti-fuse kullanmanın avantajı tümdevre alanından tasarruf etmektir.

Xilinx FPGA'leri iki boyutta yerleştirilmiş mantık devresi blokları içermekte, bloklar arasında yatay ve dikey bağlantılar yapabilmektedir (Şekil 5.17).



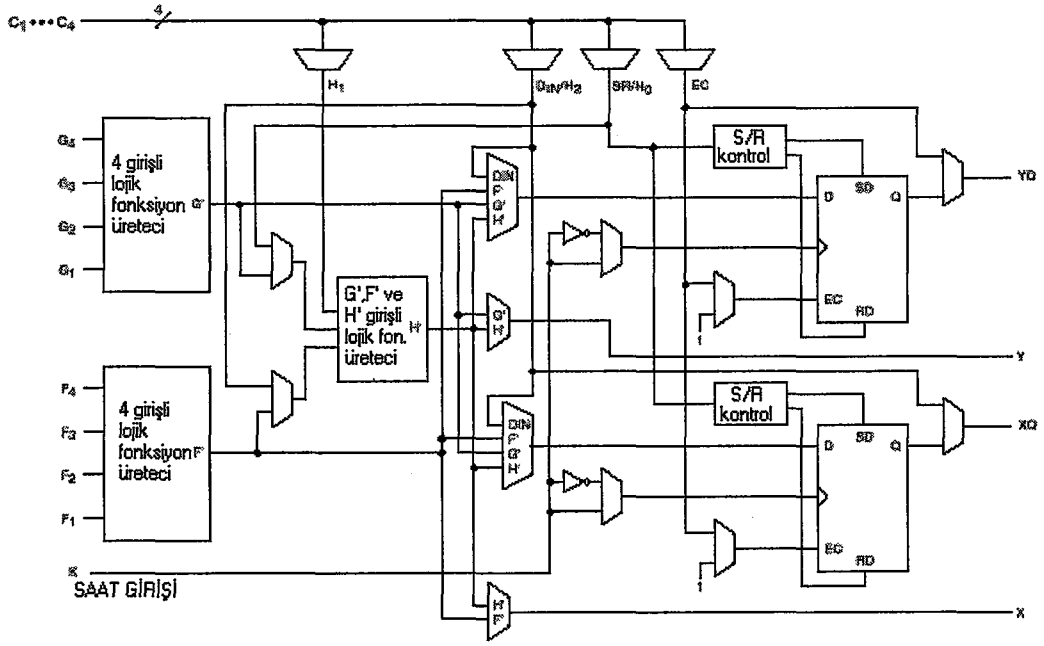
Şekil 5.17 FPGA iç bağlantı yapısı

İlk FPGA Xilinx firması tarafından 1985 yılında XC2000 serisi olarak piyasaya sürülmüştür. Genel olarak FPGA'ler, merkezinde "Mantık Blokları" ki buna CLB (Configurable Logic Blok) adı verilmektedir ve dış dünyaya açıldıkları noktalarda da "Giriş Çıkış Blokları" GÇB içeren bir yapıya sahiptirler (Şekil 5.18).



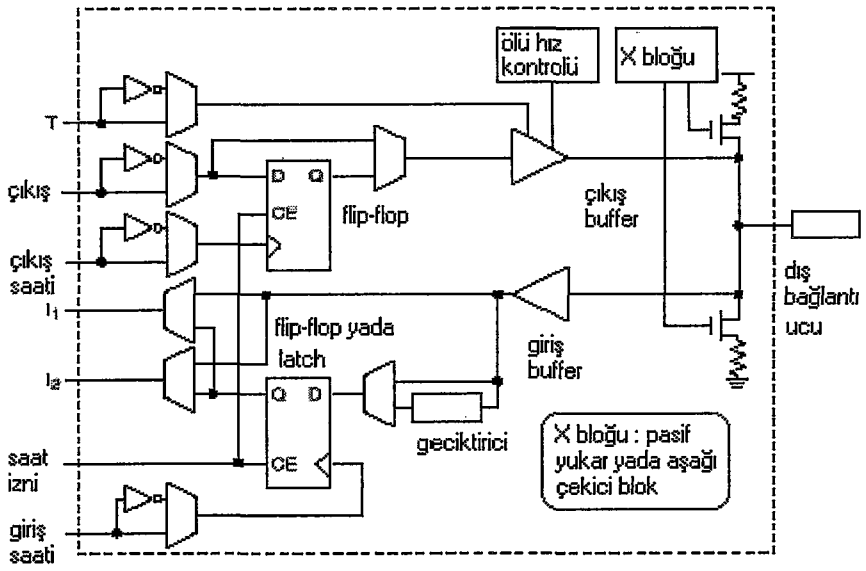
Şekil 5.18 FPGA iç yapısı

CLB'ler, FPGA'in "Boole" fonksiyonlarını gerçekleyen parçalarıdır. Her firmanın FPGA mimarisinin farklı olduğunu belirtmiştik. Şekil 5.19'da Xilinx firmasının 4000E serisinin CLB yapısı gösterilmektedir.



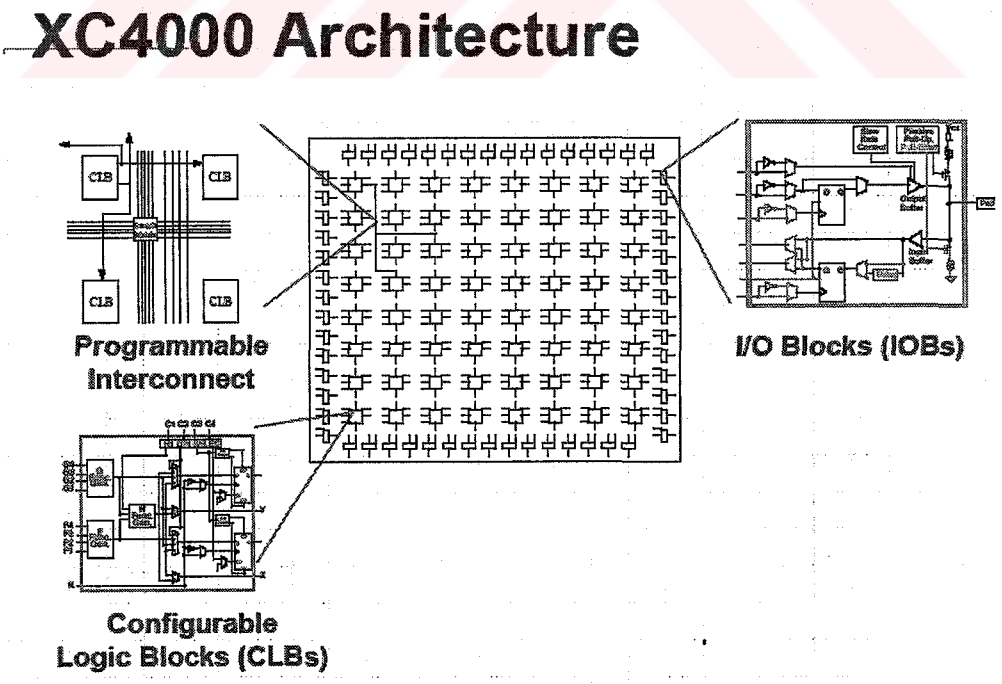
Şekil 5.19 CLB iç yapısı

GÇB'ler FPGA'nın dış dünya ile bağlantısını kuran yapılardır. Şekil 5.20 yine Xilinx firması FPGA'lerine ait GÇB(I/O Logic) yapısını göstermektedir.



Şekil 5.20 GÇB iç yapısı

Şekil 5.20'den de görüleceği üzere X bloğu bağlantı ucunun kullanılmaması durumlarında güç tüketimini azaltmak amacına yönelik olarak oluşturulmuş yukarı yada aşağı çekici programlanabilir dirençtir.



Şekil 5.21 Xilinx XC4000 entegresinin iç yapısı

Tez kapsamında kullanılan XCV400E kodlu ürünün %72 lik kısmı ile 256 bitlik RSA sistemi tasarlanmıştır. Yazılım geliştirme ortamının sentezleme kabiliyeti ile çalışmada tüm FPGA yapısı içerisinde ne kadar hangi elemandan kullanıldığı bilgisine ulaşılabilmektedir. Tasarımımızın sentez sonuçları aşağıda sunulmuştur.

```
=========================================================================
*                          HDL Synthesis                                *
=========================================================================

HDL Synthesis Report

Macro Statistics
# RAMs                  : 17
  8x8-bit dual-port distributed RAM: 16
  8x32-bit dual-port distributed RAM: 1
# Registers             : 1634
  8-bit register        : 16
  5-bit register        : 2
  1-bit register        : 568
  2-bit register        : 915
  32-bit register       : 1
  4-bit register        : 3
  3-bit register        : 129
# Counters              : 6
  9-bit up counter      : 1
  32-bit up counter     : 1
  11-bit up counter     : 1
  5-bit up counter      : 2
  3-bit up counter      : 1
# Shift Registers       : 1
```

3-bit shift register : 1

Multiplexers : 280

2-to-1 multiplexer : 274

8-bit 4-to-1 multiplexer : 4

1-bit 9-to-1 multiplexer : 1

2-bit 9-to-1 multiplexer : 1

Tristates : 1

32-bit tristate buffer : 1

Adders/Subtractors : 402

5-bit adder : 388

9-bit adder : 2

18-bit adder carry out : 3

9-bit adder carry out : 1

4-bit adder : 4

3-bit adder : 1

19-bit adder : 1

11-bit adder : 1

10-bit adder : 1

Multipliers : 260

2x2-bit multiplier : 259

10x9-bit multiplier : 1

Comparators : 19

32-bit comparator greater : 7

32-bit comparator not equal : 1

32-bit comparator lessequal : 5

32-bit comparator less : 3

```
 32-bit comparator greatequal    : 1
 32-bit comparator equal         : 2
=========================================================================
*                      Final Report                      *
=========================================================================
Device utilization summary:
Selected Device : v400epq240-6
 Number of Slices:              3502  out of   4800    72%
 Number of Slice Flip Flops:    3048  out of   9600    31%
 Number of 4 input LUTs:        4053  out of   9600    42%
 Number of bonded IOBs:           41  out of    162    25%
 Number of GCLKs:                  1  out of      4    25%
=========================================================================
TIMING REPORT
Clock Information:
------------------------------+----------------------+-------+
Clock Signal                  | Clock buffer(FF name) | Load  |
------------------------------+----------------------+-------+
CLK                           | BUFGP                | 3209  |
------------------------------+----------------------+-------+
Timing Summary:
Speed Grade: -6
   Minimum period: 28.602ns (Maximum Frequency: 34.963MHz)
   Minimum input arrival time before clock: 12.894ns
   Maximum output required time after clock: 6.514ns
   Maximum combinational path delay: 10.540ns
Completed process "Synthesize".
```

6. SONUÇLAR

Açık-anahtarlı şifreleme sistemleri ve sayısal imza, modern teknolojinin ve modern şifreleme biliminin en önemli gelişmelerinden biridir. Günümüz internet dünyasının sağladığı elektronik ticaret, internet bankacılığı gibi sektörlerin her geçen gün daha fazla değer kazanacağının kesinliğinin yanısıra, RFID gibi henüz yeni duyulan bir çok teknolojide bu tip açık anahtarlı şifreleme sistemlerinin kullanım alanlarına dahil olacaktır.

Bu çalışmada, en çok kullanılan sayısal imza ve açık anahtarlı şifreleme sistemi olan RSA (Rivest, Shamir, Adleman) kullanılmıştır. Matematiksel olarak RSA şifreleme veya şifre çözme işlemini gerçekleştirebilecek daha kolay altenatif yöntemler ve bu yöntemlerin sayısal tasarım açısından gerçeklenebilirlikleri araştırılmıştır.

Çalışma esnasında R-L ve L-R Binary yöntemleri ve bu yapıların içerisinde bulunan Montgomery yöntemi kullanılarak sonuca gidilmeye çalışılmıştır. Sistolik temelli bir yapı seçilerek gereksiz donanımdan ve bağlantı karışıklığından kurtulunmuştur. Yapının daha az işlemci birimi içermesi için radix-4 tabanlı bir sistem kullanılmıştır. Daha üst radix seviyelerine çıkılmamış, böylece sistemin yaklaşık 35 Mhz hızlarında çalışabilmesi sağlanmıştır. En son olarak da yine donanımsal gerekliliği yarı yarıya indiren araya-sıkıştırma (interleaving) yöntemi ve uygun yardımcı donanımlar kullanılarak ard arda gelen saat darbelerinde sistemin sorunsuz çalıştığı gösterilmiştir.

Nihayetinde tez çalışmamızda değişken uzunluklu (1024,2048,4096,...) RSA sistemlerinin tasarlanabileceği donanımsal bir alt yapı ve VHDL dilinde yazılmış bir RSA kodu ortaya konulmuştur. Bunun yanı sıra XCV400E FPGA entegresinin kapasitesi doğrultusunda da %72 doluluk oranıyla 256 bitlik RSA sistemi gerçek zamanlı olarak çalıştırılmış ve şifreleme yordamları başarıyla tamamlanmıştır.

Sonuç olarak, bilgi güvenliğinin öneminin her geçen gün artmasının yanı sıra gelişen teknolojinin getireceği hız karşısında, daha güvenli RSA sistemlerine daha küçük tümdevre alanlarında ihtiyaç olacaktır. Bu tez çalışmasının, geliştirilecek yeni RSA temelli sistemler için iyi bir başvuru kaynağı olabileceği ve bu konularda gerek duyulan türkçe kaynak ihtiyacını karşılayabileceği düşüncesindeyim

KAYNAKLAR

Acar, T., Kaliski, B. S. ve Koç, Ç. K., (1996), "Analyzing and Comparing Montgomery Modular Multiplication Algorithms", IEEE Micro Symp., 16(3):26-33

Blum, T. ve Paar, C., (1999), "Montgomery Modular Exponantiation on Reconfigurable Hardware", IEEE Symp. On Computer Arithmetic, 70-77

Brickell, E. F., (1989), "A Survey of Hardware Implementations of RSA", Advances in Cryptology, Crypto 89, Proceedings, Lecture Notes in Computer Science, 435:368-370

Burnett, S. ve Paine, S. (2001), RSA Security's Official Guide to Cryptography, California.

Chen, P. S., Hwang, A. ve Wu, C. W., (1996), "A Systolic RSA Public Key Cryptosystems ", IEEE International Symposium on Circuits and Systems, 408-411

Daly, A. ve Marnane, W., (2002), "Efficient Architectures for Implementing Montgomery Modular Multiplication and RSA Modular Exponantiation on Reconfigurable Logic", FPGA'02, 24-26 February 2002, California, USA.

Diffie, W. ve Hellman, M. E.,(1976), "New Directions in Cryptography", IEEE Trans. on Information Theory, 22:644-654

Eldridge, S. E. ve Walter, C. D., (1993), "Hardware Implementation of Montgomery's Modular Multiplication Algorithm", IEEE Transactions on Computers, 42:693-699.

Koç, Ç. K., (1994), "High-Speed RSA Implementation", RSA Laboratories technical reports, Redwood City.

Koç, Ç. K., (1995), "RSA Hardware Implementation", RSA Laboratories technical reports, Redwood City.

Nedjah, N. ve Mourelle, L. M.., (2002), "Two Hardware Implementations for the Montgomery Modular Multiplicition: Sequential versusu Parallel", Proceedings of the 15 th Symposium on Integrated Circuits and Systems-Design (SBCC1'02), 2002, Rio de Janerio.

Orup, H. ve Korneup, P.,(1991), „A High- Radix Algorithm for Calculating the Exponenetial M^E Module N", 10. IEEE Symposium on Computer Arithmetic, 51-57

Rivest, R, Shamir, A. ve Adleman, L., (1978), "A Method for Obtaining Digital Signatures and Public-key Cryptosystems", Communications of the ACM, 21:120-126

Tenca, A. F. ve Todorov, G. ve Koç, Ç. K., (2001), "High-Radix Design of a Scalable Modular Multiplier "

Todorov, G. (2000), Asic Design Implementation and Analysis of a scalable High-Radix Montgomery Multiplier, Oregon State University, Oregon.

Yeşil, S. ve İslamoğlu, N. A. ve Tekmen, Y. C. ve Aşkar, M., (2004), "Two Fast RSA Implementation Using High Radix Montgomery Algoritm", 0-7803-8251-X IEEE, 2004

Walter, C., (1993), "Systolic Modular Multiplication" IEEE Transactions on Computers, 42(3):376-378

Walter, C., (1999), "Montgomery's Multiplication Technique: How to Make It Smaller and Faster", CHES'99, 1717:80-93

INTERNET KAYNAKLARI

[1]www. rsa.org

[2]www.engin.umd.umich.edu/ECE/Fac_Staf/elkateeb.html

[3]www.xilinx.com

[4]http://world.std.com/~reinhold/BigNumCalc.html

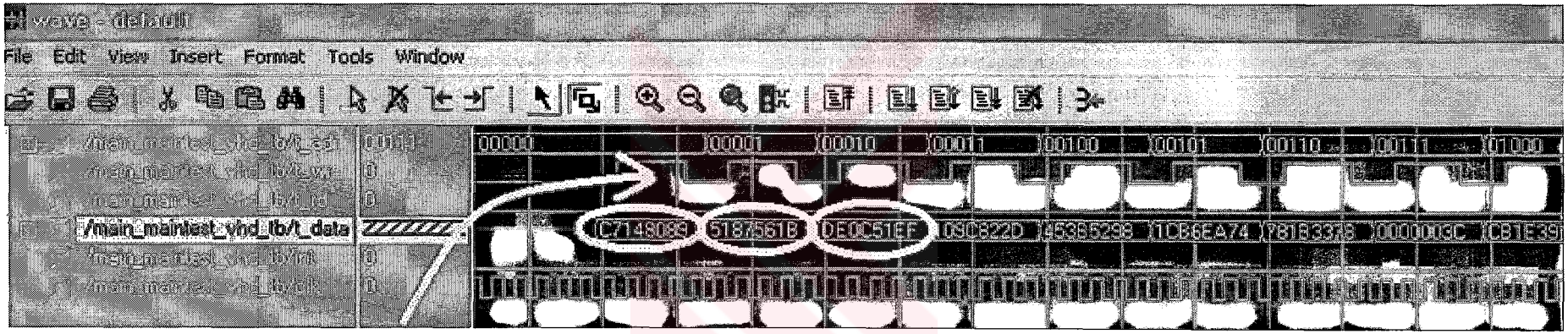
[5]www.rsasecurity.com

[6]http://primes.utm.edu/lists/small/small.html

[7]www.antilles.k12.vi-us/math/cryptotut/rsa2.htm

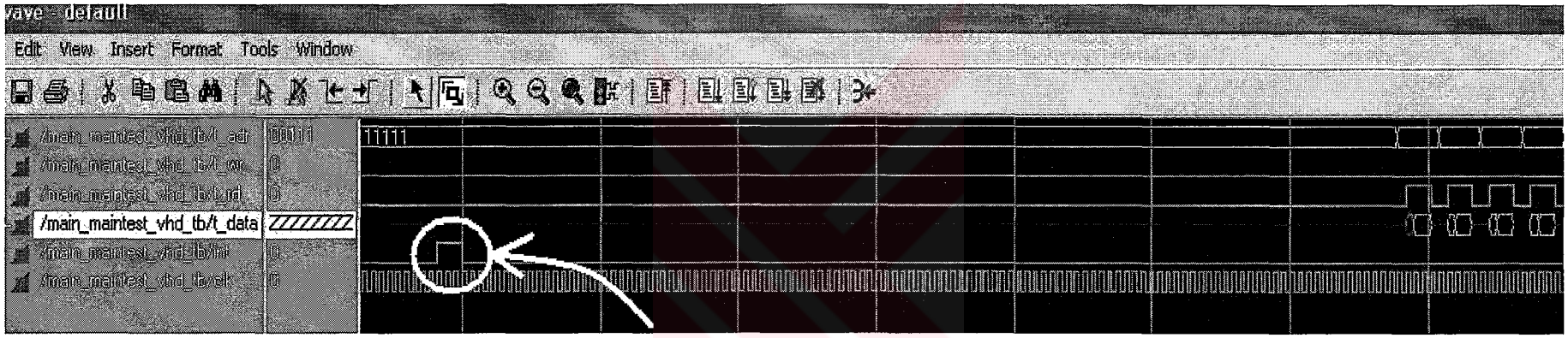
EKLER

Ek 1 Ara-yüz entegresi ile şifreleme entegresi arası haberleşme sinyal yapısı



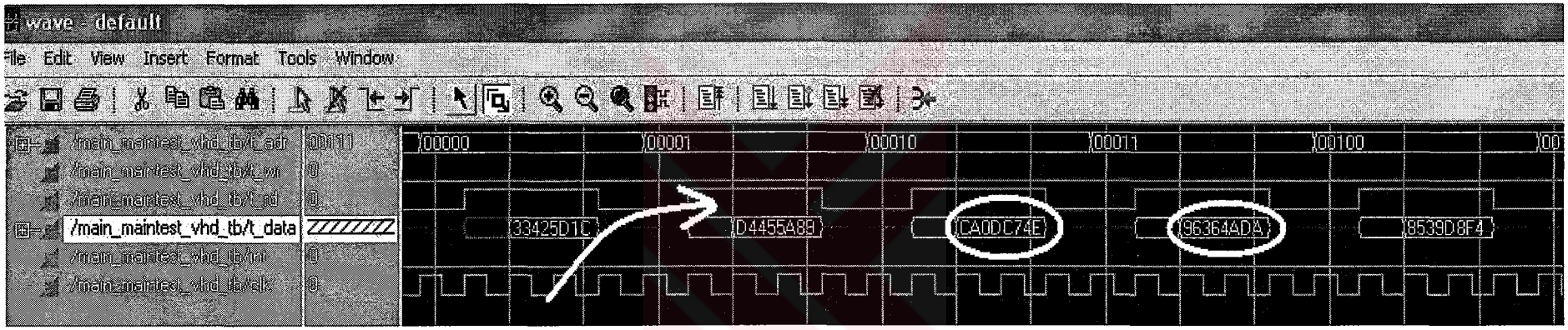
Burada mesajlaşma için kullanılan değerler Bölüm 5.3.1 içinde anlatılmış olan sayısal örneğin bilgileridir. t_wr yani "write" sinyali okla işaretlenmiştir. Buradaki haberleşme iki farklı entegre arasında geliştiği için bilgi aktarımınında sorun olmaması amacıyla "write" sinyali 3 saat darbesi genişliğince '1' değerinde tutulmaktadır ki şifreleme algoritması saatin bu üç yükselen kenarından birisinde kesinlikle bilgiyi aktarabilmiş olsun.

Ek 2 İnterrupt sinyalinin gösterilimi



Burada ara-yüz entegresini şifrelemenin bittiği konusunda bilgilendiren interrupt (int) sinyali okla gösterilmiştir. Sinyal ile uyarılan entegre sonuc bilgisini bir süre sonra bölüm 5.1'de anlatıldığı gibi aktarır.

Ek 3 Sonuç bilgisinin aktarımı



Burada ok ile işaretli sinyal "read" (t_rd) dir. Bu sinyalde, aktarılacak bilgi iki farklı entegre arasında olduğu için 3 saat darbesi genişliğinde tutulmuştur. Böylece alıcı olan (sonuç bilgisini alacak olan) ara-yüz entegresi bilgiyi hiç kaçırmadan aktarabilmektedir.

Ek 4 0.İşlemci modülü yazılımı (pe_first.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pe_first is
Port ( min  : in std_logic_vector(1 downto 0);
       pin  : in std_logic_vector(1 downto 0);
       ENABLE:in std_logic;
       RESET:in std_logic;
       qout : out std_logic_vector(1 downto 0);
       cout  : out std_logic_vector(3 downto 0);
       ooo : out std_logic_vector(1 downto 0);
       CLK : in std_logic);
end pe_first;

architecture Behavioral of pe_first is

signal temp_min  : std_logic_vector(1 downto 0);
signal temp_pin  : std_logic_vector(1 downto 0);
signal temp_pout  : std_logic_vector(1 downto 0);
signal MUX_OUT   : std_logic_vector(1 downto 0);


COMPONENT ff_enable
       PORT(
              DIN : IN std_logic_vector(1 downto 0);
              ENABLE : IN std_logic;
              CLK : IN std_logic;
              RESET : IN std_logic;
              DOUT : OUT std_logic_vector(1 downto 0)
              );
       END COMPONENT;


begin

Inst_ff_enable_mout: ff_enable PORT MAP(
                                    DIN =>min ,ENABLE =>ENABLE ,
                                    CLK =>CLK ,RESET =>RESET ,
                                    DOUT =>temp_min );

process (temp_pin,temp_min)
begin
```

```
        if      (temp_min ="01")then
                case temp_pin is
                        when "00" => MUX_OUT <= "00";
                        when "01" => MUX_OUT <= "11";
                        when "10" => MUX_OUT <= "10";
                        when "11" => MUX_OUT <= "01";
                        when others=> Null;
                end case;
        else
                case temp_pin is
                        when "00" => MUX_OUT <= "00";
                        when "01" => MUX_OUT <= "01";
                        when "10" => MUX_OUT <= "10";
                        when "11" => MUX_OUT <= "11";
                        when others=> Null;
                end case;
        end if;
end process;
        temp_pout<=MUX_OUT ;
        temp_pin <= pin;

process (CLK)
begin
        if(rising_edge(clk)) then
                if RESET='1' then
                        cout<="0000";
                        qout<="00";
                else
                        cout<=(temp_pout*temp_min)+temp_pin;
                        qout<=temp_pout;
                end if;
        else null;
        end if;
end process;

end Behavioral;
```

Ek 5 Genel İşlemci modülü yazılımı (PE.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PE is
Port (  min : in std_logic_vector(1 downto 0);
        bin : in std_logic_vector(1 downto 0);
        pin : in std_logic_vector(1 downto 0);
        sin : in std_logic_vector(1 downto 0);
        tin : in std_logic_vector(1 downto 0);
        ain : in std_logic_vector(1 downto 0);
        qin : in std_logic_vector(1 downto 0);
        cin : in std_logic_vector(2 downto 0);
        SEL : in std_logic;
        ENABLE_min:in std_logic;
        ENABLE_bin:in std_logic;
        ENABLE_init:in std_logic;
        ENABLE_tin:in std_logic;
        mout : out std_logic_vector(1 downto 0);
        bout : out std_logic_vector(1 downto 0);
        pout : out std_logic_vector(1 downto 0);
        sout : out std_logic_vector(1 downto 0);
        tout : out std_logic_vector(1 downto 0);
        aout : out std_logic_vector(1 downto 0);
        qout : out std_logic_vector(1 downto 0);
        cout : out std_logic_vector(2 downto 0);
        SEL_out : out std_logic;
        ENABLE_b_out : out std_logic;
        CLK:in std_logic;
        RESET:in std_logic
        );
end PE;

architecture Behavioral of PE is

signal RESULTRIGHT  : std_logic_vector(1 downto 0);
signal MUX_OUT_pout  : std_logic_vector(1 downto 0);
signal MUX_OUT_sout  : std_logic_vector(1 downto 0);
signal MUX_OUT_bin  : std_logic_vector(1 downto 0);
signal bin_middle  : std_logic_vector(1 downto 0);
signal m_temp  : std_logic_vector(1 downto 0);
signal b_temp  : std_logic_vector(1 downto 0);
signal sin_temp  : std_logic_vector(1 downto 0);
```

```
signal sout_temp  : std_logic_vector(1 downto 0);
signal tin_temp  : std_logic_vector(1 downto 0);
signal SEL_temp  : std_logic;
signal ENABLE_b_temp  : std_logic;
signal ENABLE_bin_middle  : std_logic;

        COMPONENT montgomery
        PORT(
                Mj : IN std_logic_vector(1 downto 0);
                Bj : IN std_logic_vector(1 downto 0);
                Pj : IN std_logic_vector(1 downto 0);
                ai : IN std_logic_vector(1 downto 0);
                qi : IN std_logic_vector(1 downto 0);
                cin : IN std_logic_vector(2 downto 0);
                CLK : IN std_logic;
                RESET : IN std_logic;
                aiout : OUT std_logic_vector(1 downto 0);
                qiout : OUT std_logic_vector(1 downto 0);
                RESULTLEFT : OUT std_logic_vector(2 downto 0);
                RESULTRIGHT : OUT std_logic_vector(1 downto 0)
                );
        END COMPONENT;

        COMPONENT dff_radix_2
        PORT(
                DIN : IN std_logic;
                CLK : IN std_logic;
                RESET : IN std_logic;
                DOUT : OUT std_logic
                );
        END COMPONENT;

        COMPONENT dff_radix_4
        PORT(
                DIN : IN std_logic_vector(1 downto 0);
                CLK : IN std_logic;
                RESET: in STD_LOGIC;
                DOUT : OUT std_logic_vector(1 downto 0)
                );
        END COMPONENT;

        COMPONENT ff_enable
        PORT(
                DIN : IN std_logic_vector(1 downto 0);
                ENABLE : IN std_logic;
                CLK : IN std_logic;
                RESET : IN std_logic;
                DOUT : OUT std_logic_vector(1 downto 0)
                );
        END COMPONENT;
```

```
begin

Inst_montgomery: montgomery PORT MAP(
                Mj =>m_temp ,
                Bj =>b_temp ,
                Pj =>pin ,
                ai =>ain ,
                aiout =>aout ,
                qi =>qin ,
                qiout =>qout ,
                cin =>cin ,
                CLK =>CLK ,
                RESET =>RESET ,
                RESULTLEFT =>cout ,
                RESULTRIGHT =>RESULTRIGHT
        );
-----Mux Yapisi----------------

process (SEL_temp,RESULTRIGHT,sout_temp,bin )
begin
  case SEL_temp is
    when '1' => MUX_OUT_pout <= "00";--sonraki S&M de 0 versin diye muxlaniyor
                MUX_OUT_sout <= RESULTRIGHT;--sout degerinin atanmasi
                MUX_OUT_bin  <= RESULTRIGHT;-- min degerinin atamasi
    when '0' => MUX_OUT_pout <= RESULTRIGHT;
                MUX_OUT_sout <= sout_temp;
                MUX_OUT_bin <= bin;
    when others=> null;
  end case;
end process;

        pout <= MUX_OUT_pout;
        sout <= MUX_OUT_sout;
        bin_middle <= MUX_OUT_bin;
        mout<=m_temp;       --Baslangıcta birer bit saga kaydirmak
        bout<=b_temp;       -- uclarında enable li DFF ler var

        Inst_ff_enable_mout: ff_enable PORT MAP(
                                DIN =>min ,ENABLE =>ENABLE_min ,
                                CLK =>CLK ,RESET =>RESET ,
                                DOUT =>m_temp );

        Inst_ff_enable_bout: ff_enable PORT MAP(
                                DIN =>bin_middle , ENABLE =>ENABLE_b_temp ,
                                CLK =>CLK , RESET =>RESET ,
                                DOUT =>b_temp );

        sin_temp<=sin;      -- enable siz dff ler icin
                            -- bu yapi bir sonraki S&M de kullanilmak
                            --icin olusturulldu
```

```
        DDF1_mout: dff_radix_4 PORT MAP(
                                DIN =>sin_temp ,CLK =>CLK ,
                                RESET=>RESET,DOUT =>sout_temp);

        tin_temp<=tin;          --A (multiplicationu)  kaydirmak icin kullanilir
                                -- xin her saatte yenilenir
                                -- baslangicta girilir sonra kendi doner

    Inst_ff_enable_tin: ff_enable PORT MAP(
                                DIN =>tin_temp  ,ENABLE =>ENABLE_tin ,
                                CLK =>CLK ,RESET =>RESET ,
                                 DOUT =>tout );

        DDF4: dff_radix_2 PORT MAP(
                                DIN =>SEL ,CLK =>CLK ,
                                RESET =>RESET,DOUT =>SEL_temp);

                SEL_out<=SEL_temp;
                ENABLE_bin_middle<=ENABLE_bin and ENABLE_init;

        DDF5: dff_radix_2 PORT MAP(
                                DIN =>ENABLE_bin_middle ,CLK =>CLK ,
                                RESET =>RESET,DOUT =>ENABLE_b_temp);
                ENABLE_b_out<=ENABLE_b_temp;

end Behavioral;
```

Ek 6 Montgomery modülü yazılımı(Montgomery.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Montgomery is
   Port ( Mj : in std_logic_vector(1 downto 0);
          Bj : in std_logic_vector(1 downto 0);
          Pj : in std_logic_vector(1 downto 0);
          ai : in std_logic_vector(1 downto 0);
          aiout : out std_logic_vector(1 downto 0);
          qi : in std_logic_vector(1 downto 0);
          qiout : out std_logic_vector(1 downto 0);
          cin: in std_logic_vector(2 downto 0);
          CLK:in std_logic;
          RESET:in std_logic;
          RESULTLEFT : out std_logic_vector(2 downto 0);
          RESULTRIGHT : out std_logic_vector(1 downto 0)
         );
end Montgomery;

architecture Behavioral of Montgomery is

COMPONENT dff_radix_4
        PORT(
                DIN : IN std_logic_vector(1 downto 0);
                CLK : IN std_logic;
                RESET: in STD_LOGIC;
                DOUT : OUT std_logic_vector(1 downto 0)
                );
END COMPONENT;
COMPONENT dff_radix_8
        PORT(
                DIN : IN std_logic_vector(2 downto 0);
                CLK : IN std_logic;
                RESET : IN std_logic;
                DOUT : OUT std_logic_vector(2 downto 0)
                );
END COMPONENT;

signal pio1  : std_logic_vector(2 downto 0);
```

```
signal pio2  : std_logic_vector(1 downto 0);

signal temp0: std_logic_vector(4 downto 0);
signal temp1: std_logic_vector(3 downto 0);
signal temp2: std_logic_vector(3 downto 0);
signal temp_1: std_logic_vector(4 downto 0);
signal temp_2: std_logic_vector(4 downto 0);

signal aio  : std_logic_vector(1 downto 0);
signal qio  : std_logic_vector(1 downto 0);

begin
                temp1<= ai * Bj ;
                temp2<= qi * Mj ;

                temp_1<= '0' & temp1;
                temp_2<= '0' & temp2;

                temp0<= temp_1+temp_2+ cin + Pj ;

                pio1<=temp0(4 downto 2);
                pio2<=temp0(1 downto 0);

                aio<=ai;
                qio<=qi;


DDF1_RADIX_8: dff_radix_8 PORT MAP(DIN =>pio1 ,CLK =>CLK ,
RESET=>RESET,DOUT =>RESULTLEFT          );
DDF2_RADIX_4: dff_radix_4 PORT MAP(DIN =>pio2 ,CLK =>CLK ,
RESET=>RESET,DOUT =>RESULTRIGHT         );
DDF3_RADIX_4: dff_radix_4 PORT MAP(DIN =>aio ,CLK =>CLK, RESET=>RESET
,DOUT =>aiout );
DDF4_RADIX_4: dff_radix_4 PORT MAP(DIN =>qio ,CLK =>CLK, RESET=>RESET
,DOUT =>qiout );

end Behavioral;
```

Ek 7 Kontrol modülü yazılımı(Enable1.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity enable_1 is
        PORT(ENABLE_bin                 : OUT std_logic;
                ENABLE_init         : OUT std_logic;
                ENABLE_min          : OUT std_logic;
                COUNT_tin           : OUT std_logic;
                en_ex               : OUT std_logic;
                result_coming       : OUT std_logic;
                result_done         : OUT std_logic;
                exp_length          : OUT integer;
                result_ready        : OUT std_logic;
                counter                 : OUT std_logic_vector(10 downto 0);
                say                 : OUT integer;
                SEL                 : OUT std_logic;
                SEL_2               : OUT std_logic;
                SEL_4               : OUT std_logic_vector(1 downto 0);
                expo_out            : OUT std_logic;
                EXPO                : IN    std_logic;
                CLK                 : IN  std_logic;
                RESET               : IN  std_logic
                );
end enable_1;

architecture Behavioral of enable_1 is
signal COUNT_SEL:std_logic_vector(10 downto 0);
signal COUNT_tin_temp:std_logic;
signal COUNT_tin_middle:std_logic;
signal SEL_init:std_logic;
signal SEL_pre:std_logic;
signal SEL_pre_1:std_logic;
signal ENABLE_min_temp:std_logic;
signal ENABLE_b_pre:std_logic;
signal ENABLE_exp_init:std_logic;
signal ENABLE_exp :std_logic;
signal exp_length_1 :integer;
signal expo_ext :std_logic;
signal exponent:std_logic;
signal expo_state:std_logic_vector(3 downto 0);
```

```
signal length:integer range 0 to 256 ;
signal final_sign:std_logic;
signal last_sign_middle1:std_logic;
signal last_sign_middle2:std_logic;
signal result_enable:std_logic;
--signal result_done:std_logic;
signal sayici:integer;
signal COUNT :integer:=0;
signal uzunluk :integer;
signal final:integer;
signal initialise        :std_logic:='0';
signal ready_i : std_logic := '0';
constant N: INTEGER := 256;

        COMPONENT dff_radix_2
        PORT(DIN : IN std_logic;
                CLK : IN std_logic;
                RESET : IN std_logic;
                DOUT : OUT std_logic);
        END COMPONENT;

        COMPONENT ff_enable_radix2
        PORT(DIN : IN std_logic;
                ENABLE : IN std_logic;
                CLK : IN std_logic;
                RESET : IN std_logic;
                DOUT : OUT std_logic);
        END COMPONENT;

        COMPONENT ff_enable_radix2_resetli
        PORT(DIN : IN std_logic;
                ENABLE : IN std_logic;
                CLK : IN std_logic;
                RESET : IN std_logic;
                DOUT : OUT std_logic);
        END COMPONENT;

TYPE exp_description is array (0 to 257) of std_logic;
SIGNAL exp : exp_description;

begin
        DFF_final_sign1: dff_radix_2 PORT MAP(DIN =>final_sign ,CLK =>CLK ,
                                        RESET =>RESET,DOUT =>last_sign_middle1);

        DFF_final_sign2: dff_radix_2 PORT MAP(DIN =>last_sign_middle1 ,
        CLK=>CLK , RESET =>RESET,DOUT =>last_sign_middle2);

--Bit uzunlugu kadar exponent degerinin yerlesmesi icin bit uzunlugu kadar dff erekir

        Inst_ff_expo_17: ff_enable_radix2_resetli PORT MAP(
                                DIN =>EXPO , ENABLE =>ENABLE_exp ,
```

```
                                        CLK =>CLK ,RESET =>RESET ,
                                    DOUT =>exp(N+1)
                                );
-----GENERATE KISMI ------------

GEN2:for j in 2 to N+1 generate
Inst_ff_expo: ff_enable_radix2_resetli PORT MAP(          DIN =>exp(j) ,
                                        ENABLE =>ENABLE_exp ,
                                        CLK =>CLK ,
                                        RESET =>RESET ,
                                        DOUT =>exp(j-1)  );
end generate GEN2;

Inst_ff_expo_extra: ff_enable_radix2 PORT MAP(DIN =>exp(1) ,
                                        ENABLE =>ENABLE_exp ,
                                       CLK =>CLK ,RESET =>RESET ,
                                       DOUT =>expo_ext );
-- Burada exponentin iceriye alinmasi ve boyutunun hesaplanmasi islemi yapilir.
-- ENABLE_exp_init 4 bitlik icin 4 darbe boyunca '1' olarak kalir
-- ENABLE_exp_init 6 bitlik icin 6 darbe boyunca '1' olarak kalir
process (CLK, RESET,length)
begin
        if(rising_edge(CLK)) then
                if RESET='1' then
                        length<=0;
                elsif(ENABLE_exp_init='1') then -- 4 bit için 4 darbe genislikte 1 olur
                        if (EXPO='0') then          -- cunku exponentin uzunlugu 4 tur.
                            length<=length + 1 ;
                        else
                            length<=0;
                        end if;
                else null;
                end if;
        else null;
        end if;
end process;

exp_length_1 <= N - length;---1;-- Burada cikarma yapilacak sayi bit uzunlugudur.
exp_length<=exp_length_1;
uzunluk <= exp_length_1;  -- 4 bit icin [2+8*(lentgh+1)]
final<=(N/2)+((uzunluk+1)*(N+4)); --6 bit icin [3+10*(lentgh+1)]

process (CLK, RESET)
begin
        if rising_edge(CLK) then
                  if RESET='1' then
                        COUNT <= 0;
                        ENABLE_init<='1';
                        SEL_init <='0';          --Baslangicta SEL_init 0 olmali cunku
                        final_sign<='0';  --ilk degerler 0 gelmeli
                        ENABLE_min_temp<='0';--sadece ilk deger 0
```

```
                    ENABLE_exp_init<='1';
                    result_enable <= '0';--result ciktiginda 1 oluyor.yoksa hep 0 dir
                     result_done<='0';
                --  result_ready<='0';
                    ready_i <='0';
                    initialise<='1';
                    elsif initialise = '1' then
                     COUNT <= COUNT + 1;--*******[(Bit sayisi)/2]-1
                     if (COUNT <= (N/2)-1)then                    --6 bit icin 2
                            SEL_init <='0';
                            ENABLE_init<='1';
                            ENABLE_min_temp<='1'
                    ENABLE_exp_init<='1';
                     elsif (COUNT = N/2)then --**********[(Bit sayisi)/2]
                            SEL_init <='0';
                            ENABLE_init<='0';
ENABLE_min_temp<='1';
                            ENABLE_exp_init<='1';
                     elsif (COUNT>N/2 and COUNT <= N-2)then
                            SEL_init <='0';
                    ENABLE_init<='0';
                            ENABLE_min_temp<='0';
                            ENABLE_exp_init<='1';
                elsif (COUNT = final) then.
                            final_sign<='1';
                     elsif (COUNT > final+N+5 and COUNT< final+2*N+13) then
                    result_enable<='1';
                 if (COUNT > final+N+5+3 and COUNT< final+2*N+9) then

        result_done<='1';
                            elsif    (COUNT > final+2*N+9 and COUNT < final+2*N+13)
        then
                                     ready_i <='1';
--                                   result_ready<='1';
                                     result_done<='0';
                            else
                            result_done<='0';
--                                  result_ready<='0';
                                    ready_i <='0';
                            end if;

                    elsif (COUNT > final+ 2*N+13)      then
                            initialise<='0';
                    else
                            ENABLE_min_temp<='0';
                            ENABLE_exp_init<='0';
                            ENABLE_init<='1';
                            SEL_init <='1';
                    final_sign<='0';
                result_enable<='0';
                            result_done<='0';
```

```
                    --      result_ready<='0';
                            ready_i <='0';
                        end if;
                    else null;
                    end if;
                    result_ready<=ready_i;
        else null;
        end if;
end process;

ENABLE_min<=ENABLE_min_temp;

process (CLK, RESET,COUNT,COUNT_SEL)
--variable res:std_logic;
begin
        if(rising_edge(clk))    then
                if RESET='1' then
                   COUNT_SEL <= (others=>'0');
                 else
                   if COUNT > N/2 then
                        COUNT_SEL <= COUNT_SEL + 1;
        --              res:='0';
                        if COUNT_SEL=N+3 then
                   COUNT_SEL<=(others=>'0');
                --          res:='1';
                else null;
                        end if;
                  else null;
                   end if;
                end if;
         else null;
        end if;

        if(rising_edge(clk))    then
                if RESET='1' then
                        sayici<= 0 ;    --sayici '0' lanir..
                else
                --      if res='1' then
                --              sayici<=12;--6* 6 bit icin 6--sayici 4 bit icin 4
                        if final_sign='1' then
                                sayici<=1;
                --      elsif sayici > 0 then
                --              sayici<=sayici-1;
                        else null;
                        end if;
                end if;
         else null;
         end if;
end process;
```

```
process(CLK,RESET,final_sign, last_sign_middle1, last_sign_middle2, COUNT_SEL,
expo_ext, expo_state)
begin
 if(rising_edge(clk))  then
  if RESET='1' then            -- logic
                SEL_4 <= "01";        --**********[(Bit sayisi)/2]+2
        elsif (COUNT < (N/2)+2 ) then  --6* 6 bit icin 5 --  4 bit icin 4 olmali
                SEL_4 <= "01";
        elsif (COUNT = (N/2)+2 ) then        --6* 6 bit icin 5-- 4 bit icin 4 olmali
                SEL_4 <= "11";
 --**********[(Bit sayisi)/2]+3       ile [(Bit sayisi)/2]+3
        elsif (COUNT>=(N/2)+3 and COUNT<=((N/2)+2+(((N/2)-1)*2)-1)) then
                if (COUNT_SEL(0)='0')
                        SEL_4<="01";
                else
                        SEL_4<="00" ;
                end if;
        elsif final_sign='1' then
                SEL_4<= "00";
        elsif last_sign_middle1='1' then
                SEL_4<= "11";
        elsif last_sign_middle2='1' then
                SEL_4<= "00";
        elsif (COUNT_SEL(0)='0' and sayici = 0 ) then
                SEL_4<= "10";
        elsif (COUNT_SEL(0)='1' and sayici = 0 ) then
                if (expo_ext='0') then
                        SEL_4<= "01";
                else
                        SEL_4<= "10";
          end if;
        else
                SEL_4<= "00";
        end if;
 else null;
 end if;

--Buranin mantigi sudur;
--exponent 1 oldugu zaman Count_tin sayma islemi yapar ;
--dolayisiyla sadece o anlarda kayit söz konusudur.
--Expo_state datasinin en agirlikli biti result_enableyi gösterir
--Dolayisiyla result_enable '1' oldugunda exponent otomatik olarak '1' dir.
--cunku cikis degeri t_start uzerinden alinmaktadir ve t_start Count_tin ile calisir.
        case expo_state is
     when "0000" => exponent <= '0';--Bunun disinda exponentin 1 oldugu noktalar
                when "0001" => exponent <= '1';--kosullara bagli olarak ki
                when "0010" => exponent <= '1';--kosullari exponent datasinin kendisi,
                when "0011" => exponent <= '0';--öncesi ve sonrası belirler
                when "0100" => exponent <= '0'; --kulanilarak belirlenir
                when "0101" => exponent <= '1';
                when "0110" => exponent <= '1';
```

```
                when "0111" => exponent <= '1';
                when others =>exponent  <= '1';
  end case;
end process;
        expo_state<=(3=>result_enable,2=>exp(2),1=>exp(1),0=>expo_ext);

        process(COUNT_SEL)
        begin
                if COUNT_SEL=0 then
                        SEL_pre<='1';
                        ENABLE_b_pre<='1';
                else
                        SEL_pre<='0';
                        ENABLE_b_pre<='0';
                end if;
        end process;

        DFF_SEL_pre: dff_radix_2 PORT MAP(DIN =>SEL_pre ,CLK =>CLK ,
                                          RESET =>RESET,DOUT =>SEL_pre_1);
--SEL asagidaki kosullrda 1 olur veya uzatilir.
        SEL<=(SEL_init and (SEL_pre or SEL_pre_1) )or last_sign_middle1 ;
--ENABLE _bin COUNT_SEL '0'iken ve last_sign geldiginde '1' olur
--BURAYI Yeni kaldirdim    ENABLE_b_pre <= not(COUNT_SEL(4) or COUNT_SEL(3)
or COUNT_SEL(2) or COUNT_SEL(1) or COUNT_SEL(0));

        ENABLE_bin<=ENABLE_b_pre or last_sign_middle1 or ENABLE_min_temp;
        ENABLE_exp<= ENABLE_exp_init or ENABLE_b_pre ;

        Count_tin_temp <= not(COUNT_SEL(0));
        Count_tin_middle<=Count_tin_temp and exponent ;
        COUNT_tin<= Count_tin_middle or ENABLE_min_temp ;
--disari atamalar
        en_ex<=ENABLE_exp_init;
        expo_out<=exp(1);
--      sign<=final_sign;
        result_coming<=result_enable ;
        SEL_2<=SEL_init;
        counter<=COUNT_SEL;
        say<= sayici;

end Behavioral;
```

Ek 8 Ana program yazılımı (MAIN.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE ieee.numeric_std.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MAIN is
        PORT(
                t_adr           : IN std_logic_vector(4 downto 0);
                t_wr            : IN std_logic;
                t_rd            : IN std_logic;
                t_data          : INOUT std_logic_vector(31 downto 0);
                int             :OUT std_logic;
                CLK             : IN std_logic ;
                Ready           : OUT std_logic         );
end MAIN;

architecture Behavioral of MAIN is

signal reset :std_logic;
signal result_ready :std_logic;
signal COUNT_SEL :std_logic_vector(10 downto 0);
signal enable_ext :std_logic;
signal expo :std_logic;
signal rd :std_logic;
signal c_start  : std_logic_vector(3 downto 0);
signal p_extra  : std_logic_vector(1 downto 0);
signal m_free  : std_logic_vector(1 downto 0);
signal b_free  : std_logic_vector(1 downto 0);
--signal B_start  : std_logic_vector(1 downto 0);
signal ENABLE_init  : std_logic;
signal ENABLE_min  : std_logic;
signal COUNT_tin  : std_logic;
--signal sign  : std_logic;
signal SEL_2  : std_logic;
signal SEL_4  : std_logic_vector(1 downto 0);
signal MUX_4_OUT : std_logic_vector(1 downto 0);
signal MUX_2_OUT : std_logic_vector(1 downto 0);
signal a_pre : std_logic_vector(1 downto 0);
signal t_start : std_logic_vector(1 downto 0);
signal t_reg : std_logic_vector(1 downto 0);
```

```
signal exp_length:integer;
signal saybak:integer;
signal result_coming : std_logic;
signal ENABLE_result: std_logic;
signal ara:std_logic;
signal init:std_logic:='0';
signal key_init:std_logic:='0';

COMPONENT vr2sp_intf
        PORT(
                t_adr                   : IN std_logic_vector(4 downto 0);
                t_wr                    : IN std_logic;
                t_rd                    : IN std_logic;
                clk                     : IN std_logic;
                modul_address           : IN std_logic_vector(4 downto 0);
                rd                      : IN std_logic;
                message_address         : IN std_logic_vector(4 downto 0);
                key_address             : IN std_logic_vector(4 downto 0);
                pre_address             : IN std_logic_vector(4 downto 0);
                result_data_in          : IN std_logic_vector(31 downto 0);
                result_address          : IN std_logic_vector(2 downto 0);
                we_result               : IN std_logic;
                t_data                  : INOUT std_logic_vector(31 downto 0);
                ready                   : OUT std_logic;
                modul_data_out          : OUT std_logic_vector(9 downto 2);
                message_data_out        : OUT std_logic_vector(9 downto 2);
                key_data_out            : OUT std_logic_vector(7 downto 0);
                pre_data_out            : OUT std_logic_vector(7 downto 0)
                );
        END COMPONENT;

COMPONENT pe
        PORT(
                min : IN std_logic_vector(1 downto 0);
                bin : IN std_logic_vector(1 downto 0);
                pin : IN std_logic_vector(1 downto 0);
                sin : IN std_logic_vector(1 downto 0);
                tin : IN std_logic_vector(1 downto 0);
                ain : IN std_logic_vector(1 downto 0);
                qin : IN std_logic_vector(1 downto 0);
                cin : IN std_logic_vector(2 downto 0);
                SEL : IN std_logic;
                ENABLE_min : IN std_logic;
                ENABLE_bin : IN std_logic;
                ENABLE_init:in std_logic;
                ENABLE_tin : IN std_logic;
                CLK : IN std_logic;
                RESET : IN std_logic;
                SEL_out : OUT std_logic;
                ENABLE_b_out : OUT std_logic;
                mout : OUT std_logic_vector(1 downto 0);
```

```
        bout : OUT std_logic_vector(1 downto 0);
        pout : OUT std_logic_vector(1 downto 0);
        sout : OUT std_logic_vector(1 downto 0);
        tout : OUT std_logic_vector(1 downto 0);
        aout : OUT std_logic_vector(1 downto 0);
        qout : OUT std_logic_vector(1 downto 0);
        cout : OUT std_logic_vector(2 downto 0)
        );
END COMPONENT;

COMPONENT pe_first
PORT(
        min : IN std_logic_vector(1 downto 0);
        pin : IN std_logic_vector(1 downto 0);
        ENABLE : IN std_logic;
        RESET : IN std_logic;
        CLK : IN std_logic;
        qout : OUT std_logic_vector(1 downto 0);
        cout : OUT std_logic_vector(3 downto 0)
        );
END COMPONENT;

COMPONENT DFF_RADIX_4
PORT(
        DIN : IN std_logic_vector(1 downto 0);
        CLK : IN std_logic;
        RESET: in STD_LOGIC;
        DOUT : OUT std_logic_vector(1 downto 0)
        );
END COMPONENT;

COMPONENT dff_radix_2
PORT(
        DIN : IN std_logic;
        CLK : IN std_logic;
        RESET : IN std_logic;
        DOUT : OUT std_logic
        );
END COMPONENT;

COMPONENT dff_radix_8
PORT(
        DIN : IN std_logic_vector(2 downto 0);
        CLK : IN std_logic;
        RESET : IN std_logic;
        DOUT : OUT std_logic_vector(2 downto 0)
        );
END COMPONENT;

COMPONENT ff_enable
PORT(
```

```
            DIN : IN std_logic_vector(1 downto 0);
            ENABLE : IN std_logic;
            CLK : IN std_logic;
            RESET : IN std_logic;
            DOUT : OUT std_logic_vector(1 downto 0)
            );
    END COMPONENT;

    COMPONENT enable_1
    PORT(
            CLK : IN std_logic;
            RESET : IN std_logic;
            EXPO:IN std_logic;
            ENABLE_bin : OUT std_logic;
            ENABLE_init:OUT std_logic;
            ENABLE_min : OUT std_logic;
            COUNT_tin : OUT std_logic;
            en_ex:OUT std_logic;
            result_coming:OUT std_logic;
            result_done:OUT std_logic;
            exp_length: OUT integer;
            result_ready:OUT std_logic;
            counter:out std_logic_vector(10 downto 0);
            say:out integer;
            expo_out:out std_logic;
            SEL : OUT std_logic;
            SEL_2 : OUT std_logic;
  SEL_4 : OUT std_logic_vector(1 downto 0)
            );
    END COMPONENT;

----- RAM DECLARATIONS----------

signal dene          : std_logic;
--signal dublex      : std_logic;
signal result_done1  : std_logic;
--signal we_message  : std_logic;
--signal we_modul    : std_logic;
--signal we_key      : std_logic;
--signal we_pre      : std_logic;
signal we_result     : std_logic;

---RAM MODUL----------

signal modul_z              : integer range 1 to 9 :=1;
signal Modul_address        : std_logic_vector(4 downto 0);
--signal Modul_data_in      : std_logic_vector(7 downto 0);
--signal Modul_data         : std_logic_vector(7 downto 0);
signal Modul_data_out       : std_logic_vector(9 downto 0);
----------
```

```
---RAM MESSAGE---------------------------------------

signal message_z              : integer range 1 to 9;
signal Message_address        : std_logic_vector(4 downto 0);
--signal Message_data_in      : std_logic_vector(7 downto 0);
--signal Message_data                  : std_logic_vector(7 downto 0);
signal Message_data_out       : std_logic_vector(9 downto 0);
signal Message_data_out1      : std_logic_vector(1 downto 0);
-------------------------------------------------------

---RAM CONSTANT---------------------------------------

signal pre_z                  : integer range 0 to 9;
signal pre_address            : std_logic_vector(4 downto 0);
signal pre_address1           : integer range 0 to 31;
--signal pre_data_in           : std_logic_vector(7 downto 0);
--signal pre_data              : std_logic_vector(7 downto 0);
signal pre_data_out           : std_logic_vector(9 downto 0);
signal pre_data_out1                  : std_logic_vector(1 downto 0);
-------------------------------------------------------

---RAM KEY---------------------------------------

signal key_z                  : integer range 0 to 8:=0;
signal key_address            : std_logic_vector(4 downto 0);
--signal Key_data_in          : std_logic_vector(7 downto 0);
--signal Key_data             : std_logic_vector(7 downto 0);
signal Key_data_out           : std_logic_vector(8 downto 0);
signal Key_data_out1                  : std_logic;
-------------------------------------------------------

---RAM RESULT---------------------------------------

signal result_z               : integer range 1 to 31:=1;
signal result_address         : std_logic_vector(2 downto 0);
signal result_address_a       : std_logic_vector(2 downto 0);
signal result_data_in         : std_logic_vector(31 downto 0);
signal result_data_in1        : std_logic_vector(1 downto 0);
--signal result_data           : std_logic_vector(31 downto 0);
--signal result_data_out       : std_logic_vector(31 downto 0);

-------------------------------------------------------


TYPE io_description_1 is array (0 to 131) of std_logic;
TYPE io_description_2 is array (0 to 131) of std_logic_vector(1 downto 0);
TYPE io_description_3 is array (0 to 131) of std_logic_vector(2 downto 0);

SIGNAL m : io_description_2;
SIGNAL b : io_description_2;
SIGNAL p : io_description_2;
```

```
SIGNAL s : io_description_2;
SIGNAL t : io_description_2;
SIGNAL a : io_description_2;
SIGNAL q : io_description_2;
SIGNAL c : io_description_3;

SIGNAL SEL : io_description_1;
SIGNAL ENABLE_bin : io_description_1;

constant N: INTEGER := 256;  ---KAÇ Bitlik RSA ise  sayı girilecek

begin

-------PE with LUT----------
        Inst_pe_first: pe_first PORT MAP(
                min =>m(0) ,
                pin =>p(0) ,
                ENABLE =>ENABLE_min ,
                RESET =>RESET ,
                qout =>q(1) ,
                cout =>c_start ,
                CLK =>CLK
        );

------1. Processing Element---------

          c(1)<=(2=>'0',1=>c_start(3),0=>c_start(2));
Inst_pe_1: pe PORT MAP(
                min =>m(1) ,
                bin =>b(N/2) ,
                pin =>p(1) ,
                sin =>s(1) ,
                tin =>t(1) ,
                ain =>a(1) ,
                qin =>q(1) ,
                cin =>c(1) ,
                SEL =>SEL(1) ,
                ENABLE_min =>ENABLE_min ,
                ENABLE_bin =>ENABLE_bin(1) ,
                ENABLE_init=>ENABLE_init,
                ENABLE_tin =>COUNT_tin,
                mout =>m(0) ,
                bout =>b((N/2)-1) ,
                pout =>p(0) ,
                sout =>s(0) ,
                tout =>t(0) ,
                aout =>a(2) ,
                qout =>q(2) ,
                cout =>c(2) ,
                SEL_out =>SEL(2) ,
                ENABLE_b_out =>ENABLE_bin(2) ,
```

```
                CLK =>CLK ,
                RESET => RESET );

----------GENERATE BASLANGIC KISMI----------

GEN1:for i in 2 to N/2 generate
Inst_generate: pe PORT MAP(
                min =>m(i) ,
                bin =>b((N/2)+1-i) ,
                pin =>p(i) ,
                sin =>s(i) ,
                tin =>t(i) ,
                ain =>a(i) ,
                qin =>q(i) ,
                cin =>c(i) ,
                SEL =>SEL(i) ,
                ENABLE_min =>ENABLE_min ,
                ENABLE_bin =>ENABLE_bin(i) ,
                ENABLE_init=>ENABLE_init,
                ENABLE_tin =>COUNT_tin,
                mout =>m(i-1) ,
                bout =>b((N/2)-i) ,
                pout =>p(i-1) ,
                sout =>s(i-1) ,
                tout =>t(i-1) ,
                aout =>a(i+1) ,
                qout =>q(i+1) ,
                cout =>c(i+1) ,
                SEL_out =>SEL(i+1) ,
                ENABLE_b_out =>ENABLE_bin(i+1) ,
                CLK =>CLK ,
                RESET => RESET );
end generate GEN1;

--------GENERATE KISMI SONU----------

---------Last Processing Element----------

                Inst_pe_N: pe PORT MAP(
                min =>"00" ,
                bin =>"00" ,
                pin =>p_extra(1 downto 0) ,
                sin =>"00",
                tin =>s(0) ,
                ain =>a((N/2)+1) ,
                qin =>q((N/2)+1) ,
                cin =>c((N/2)+1),
                SEL =>SEL((N/2)+1) ,
                ENABLE_min =>ENABLE_min ,
                ENABLE_bin =>ENABLE_bin((N/2)+1) ,
                ENABLE_init=>ENABLE_init,
```

```
            ENABLE_tin =>COUNT_tin,
            mout =>m_free ,
            bout =>b_free ,
            pout =>p(N/2) ,
            sout =>s(N/2) ,
            tout =>t_start ,
            aout =>a((N/2)+2) ,
            qout =>q((N/2)+2) ,
            cout =>c((N/2)+2) ,
            SEL_out =>SEL((N/2)+2) ,
            ENABLE_b_out =>ENABLE_bin((N/2)+2) ,
            CLK =>CLK ,
            RESET => RESET );

    Inst_enable_1: enable_1 PORT MAP(
            ENABLE_min =>ENABLE_min ,
            ENABLE_bin =>ENABLE_bin(1) ,
            ENABLE_init=>ENABLE_init,
            EXPO => Key_data_out1,
            exp_length=>exp_length,
            result_ready=>result_ready,
            counter=>COUNT_SEL,
            say=>saybak,
            COUNT_tin => COUNT_tin,
            en_ex=>enable_ext,
            result_coming=>result_coming,
            result_done=>result_done1,
            expo_out=>expo,
            SEL =>SEL(1) ,
            SEL_2 =>SEL_2 ,
            SEL_4 =>SEL_4 ,
            CLK => CLK,
            RESET =>RESET     );

--- RAM INITIALIZATIONS          ----------

---MODUL PROCESS-------------------------------------------------------
process(clk,reset)
begin
    if reset='1' then
            modul_z<=1;          --Burada "00" olan alt 2 bit kulaniliyor
            modul_address<="00000";
            init<='1';
    elsif rising_edge(clk) and init='1'     then
            if modul_z=9 and modul_address="00000" then
                    modul_address<="00000";
                    modul_z<=1;
                    init<='0';
            elsif modul_z = 9 then
                    modul_z <= 3 ;          -- z=3 en alt biti icin
            elsif modul_z= 7 then--2. adres kismina gecilmeye karar veriyor.
```

```
            modul_address<=modul_address+1; -- 7 de address degisiyor
                    modul_z<=modul_z+2;--ancak diger clk ya kadar bekliyor.
                else
                    modul_z<=modul_z+2;
                end if;
        end if;
end process;
                Modul_data_out(1 downto 0)<="00";
                m(N/2)<=Modul_data_out(modul_z downto modul_z-1);
-------------------------------------------------------------------

---MESSAGE PROCESS-----------------------------------------------------

--      message_z                   <=      modul_z;
        message_address     <=      modul_address;
        Message_data_out(1 downto 0)<="00";
        Message_data_out1 <=Message_data_out(modul_z downto modul_z-1);

-------------------------------------------------------------------

---CONSTANT PROCESS-----------------------------------------------

        pre_data_out(9 downto 8)<="00";
        pre_data_out1<=pre_data_out(pre_z+1 downto pre_z);
        pre_z <= 9-modul_z;  --****************
        pre_address1 <= 31 - conv_integer(message_address) ; --32 Burada yukleme yaparken
en alt adres full 0 olacak
        pre_address <= conv_std_logic_vector(pre_address1,5); --en ust bitten baslanacak

-------------------------------------------------------------------

---KEY PROCESS---------------------------------------------------

process(clk,reset)
begin
        if reset='1' then
                key_z<=0;
                key_address<="00000";
                key_init<='1';
                elsif rising_edge(clk) and key_init='1' then
                        if key_z=7  and key_address="00000" then
                                key_address<="00000";
                                key_z<=8;
                                key_init<='0';
                        elsif key_z = 7 then    --
                                key_z <= 0      ;-- key_z=1 en alt biti icin
                        elsif key_z= 6 then--2. adres kismina gecilmeye karar veriyor.
                        key_address<=key_address+1; --  8 de address degisiyor
                                key_z<=key_z+1;--ancak diger clk ya kadar bekliyor.
                        else
                                key_z<=key_z+1;
```

```
                end if;
        else null;
        end if;
end process;
                Key_data_out(8)<='0';
                Key_data_out1<=Key_data_out(key_z);
-------------------------------------------------------

---RESULT PROCESS-------------------------------------------------------

process(clk,reset)
--variable dene:std_logic;
begin
        if reset='1' then
                result_z<=1;
                result_address<="000"          ;
                dene<='0';
        elsif rising_edge(clk) then-- and result_done1='1' then
                        if result_done1='1' then
                         dene<=not dene;
                                if result_z = 31 then--and dene='1' then
                                        if dene='1' then
                                        result_z<=1;
                                        result_address<=result_address+1;
                                        Result_data_in(result_z downto result_z-1)<=
result_data_in1;
                                        else
                                        Result_data_in(result_z downto result_z-1)<=
result_data_in1;
                                        end if;
                                else
                                        if dene='1' then
                                        result_z<=result_z+2;
                                        Result_data_in(result_z downto result_z-1)<=
result_data_in1;
                                        else
                                        Result_data_in(result_z downto result_z-1)<=
result_data_in1;
                                        end if;
                                end if;
                        else null;
                        end if;
        else null;
        end if;
end process;

--      Result_data_in(result_z downto result_z-1)<= result_data_in1;
        we_result<=result_done1;
-------------------------------------------------------------------------------------------------------
---------
                Inst_vr2sp_intf: vr2sp_intf PORT MAP(
```

```
                t_data                  => t_data,
                t_adr                   => t_adr,
                t_wr                            => t_wr,
                t_rd                            => t_rd,
                clk                             => clk,
                ready                   => reset,
                modul_data_out => modul_data_out(9 downto 2),
                modul_address => modul_address,
                rd => rd,
                message_data_out => message_data_out(9 downto 2) ,
                message_address => message_address,
                key_data_out => key_data_out(7 downto 0),
                key_address => key_address,
                pre_data_out => pre_data_out(7 downto 0),
                pre_address => pre_address,
                result_data_in => result_data_in ,
                result_address => result_address,
                we_result => we_result
        );

-------------------------------------------------------------------------
----4 LU MUX YAPISI----------
--------------------------------
process (SEL_4,s(0),t_reg)
begin
  case SEL_4 is
        when "01"       => MUX_4_OUT <= t_reg;
                when "10"       => MUX_4_OUT <= s(0);
                when "11"       => MUX_4_OUT <= "01";
                when others  => MUX_4_OUT <= "00";
        end case;
end process;
 a_pre <= MUX_4_OUT;

---------------------------
----2 Li MUX YAPISI----------
-------------------------------
process (SEL_2,t_start,Message_data_out1)
begin
  case SEL_2 is
        when '1'            => MUX_2_OUT <= t_start;
                when others  => MUX_2_OUT <= Message_data_out1;
        end case;
end process;
t(N/2) <= MUX_2_OUT;    --degisiklik alani_8   [(Bit_sayisi)/2]
-----------------------------
-----------------------------

process(clk)
begin
if(rising_edge(clk)) then
```

```
        if(reset = '1') then
                p_extra <= (others => '0');     --eldenin transferini burada yaptim
        else
                p_extra <= c((N/2)+2)(1 downto 0); -- en son deger
        end if;
else null;
end if;
end process;


DDF2: DFF_RADIX_4 PORT MAP(DIN =>a_pre ,CLK =>CLK ,
                                        RESET=>RESET,DOUT =>a(1) );

        Inst_ff_enable_b_start: ff_enable PORT MAP(
         DIN =>pre_data_out1 ,ENABLE =>ENABLE_bin(1) ,
                CLK =>CLK ,RESET =>RESET ,
                DOUT =>b(N/2) );

        Inst_ff_enable_t_reg: ff_enable PORT MAP(
                DIN =>t(0) ,ENABLE =>COUNT_tin  ,
                CLK =>CLK ,RESET =>RESET ,
                DOUT =>t_reg );

        ENABLE_result<= COUNT_tin and result_coming;

        Inst_ff_enable_result: ff_enable PORT MAP(
                DIN =>t_start ,ENABLE =>ENABLE_result ,
                CLK =>CLK ,RESET =>RESET ,
                DOUT =>result_data_in1);

        int<=result_ready;
        Ready<=reset;
end Behavioral;
```

Ek 9 Ara-yüz entegresi ile haberleşme modülü yazılımı (vr2sp_intf.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vr2sp_intf is
   Port ( t_data : inout std_logic_vector(31 downto 0);
          t_adr    : in std_logic_vector(4 downto 0);
          t_wr     : in std_logic;
          t_rd     : in std_logic;
          clk      : in std_logic;
          ready    : out std_logic;
          modul_data_out  : out std_logic_vector(9 downto 2);
          modul_address  : in std_logic_vector(4 downto 0);
          rd                  : in std_logic;
          message_data_out  : out std_logic_vector(9 downto 2);
          message_address  : in std_logic_vector(4 downto 0);
          key_data_out  : out std_logic_vector(7 downto 0);
          key_address  : in std_logic_vector(4 downto 0);
          pre_data_out  : out std_logic_vector(7 downto 0);
          pre_address  : in std_logic_vector(4 downto 0);
          result_data_in : in std_logic_vector(31 downto 0);
          result_address  : in std_logic_vector(2 downto 0);
          we_result                : in std_logic);
end vr2sp_intf;

architecture Behavioral of vr2sp_intf is

component lv_ram32x8
   Port ( din             : in std_logic_vector(31 downto 0);
          wr_adr : in std_logic_vector(2 downto 0);
          wr              : in std_logic;
          dout     : out std_logic_vector(7 downto 0);
          rd_adr  : in std_logic_vector(4 downto 0);
          rd              : in std_logic;
          clk             : in std_logic);
end component;

component lv_ram8x32
   Port ( wr_adr        : in std_logic_vector(2 downto 0);
               rd_adr     : in std_logic_vector(2 downto 0);
          data_in         : in std_logic_vector(31 downto 0);
          data_out : out std_logic_vector(31 downto 0);
          wr              : in std_logic;
          rd              : in std_logic;
          clk             : in std_logic);
```

```
end component;

signal sel                  : std_logic_vector(1 downto 0);
signal wr_vector      : std_logic_vector(3 downto 0);
signal result_data    : std_logic_vector(31 downto 0);
signal son_adr        : std_logic_vector(4 downto 0);
signal son_wr         : std_logic;
signal ready_i        : std_logic := '0';
signal ready_1d       : std_logic;
signal ready_2d       : std_logic;
signal ready_3d       : std_logic;
signal ready_rst      : std_logic;

begin

sel <= t_adr(4 downto 3);

process(sel, t_wr)
begin
if(t_wr = '1') then
        case sel is
                when "00"     => wr_vector <= "0001";
                when "01"     => wr_vector <= "0010";
                when "10"     => wr_vector <= "0100";
                when "11"     => wr_vector <= "1000";
                when others   => wr_vector <= "0000";
        end case;
else
        wr_vector <= "0000";
end if;
end process;

MODRAM: lv_ram32x8      port map(
                                din     => t_data,
                                wr_adr=> t_adr(2 downto 0),
                                wr      => wr_vector(0),
                                dout    => modul_data_out(9 downto 2),
                                rd_adr => modul_address,
                                rd      => rd,
                                clk     => clk );

CNSTRAM: lv_ram32x8     port map(
                                din     => t_data,
                                wr_adr=> t_adr(2 downto 0),
                                wr      => wr_vector(1),
                                dout    => pre_data_out(7 downto 0),
                                rd_adr => pre_address,
                                rd      => rd,
                                clk     => clk );

MSGRAM: lv_ram32x8      port map(
```

```
                                    din      => t_data,
                                    wr_adr=> t_adr(2 downto 0),
                                    wr       => wr_vector(2),
                                    dout     => Message_data_out(9 downto 2),
                                    rd_adr => message_address,
                                    rd       => rd,
                                    clk      => clk );

KEYRAM: lv_ram32x8      port map(
                                    din      => t_data,
                                    wr_adr=> t_adr(2 downto 0),
                                    wr       => wr_vector(3),
                                    dout     => Key_data_out(7 downto 0),
                                    rd_adr => key_address,
                                    rd       => rd,
                                    clk      => clk );

RSLTRAM: lv_ram8x32 port map(
                                    data_in  => result_data_in,
                                    wr_adr  => result_address,
                                    wr       => we_result,
                                    data_out  => result_data,
                                    rd_adr   => t_adr(2 downto 0),
                                    rd       => t_rd,
                                    clk      => clk);
t_data <= result_data  when t_rd = '1'                  else (others => 'Z');

process(clk)
begin
if(rising_edge(clk)) then
        son_adr <= t_adr;
        son_wr <= t_wr;
        if(son_adr = X"1F" and son_wr = '1') then
                ready_i <= '1';
        elsif(ready_rst = '1') then
                ready_i <= '0';
        else
                ready_i <= ready_i;
        end if;
        ready_1d <= ready_i;
        ready_2d <= ready_1d;
        ready_rst <= ready_2d;
        ready_3d <= ready_rst;
else null;
end if;
end process;
ready <= ready_3d;
end Behavioral;
```

Ek 10 Kullanılan RAM yapılarının yazılımları(ram_32x8.vhd, ram_8x8.vhd, ram_8x32.vhd)

ram_32x8.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ram32x8 is
   Port ( din   : in std_logic_vector(31 downto 0);
          wr_adr: in std_logic_vector(2 downto 0);
          wr     : in std_logic;
          dout   : out std_logic_vector(7 downto 0);
          rd_adr: in std_logic_vector(4 downto 0);
          rd     : in std_logic;
          clk    : in std_logic);
end ram32x8;

architecture Behavioral of ram32x8 is
component ram8x8
   Port ( din : in std_logic_vector(7 downto 0);
          wr_adr: in std_logic_vector(2 downto 0);
          wr : in std_logic;
          dout: out std_logic_vector(7 downto 0);
          rd_adr: in std_logic_vector(2 downto 0);
          rd : in std_logic;
          clk: in std_logic);
end component;
signal sel : std_logic_vector(1 downto 0);
signal rd_vector : std_logic_vector(3 downto 0);
signal dout_0 : std_logic_vector(7 downto 0);
signal dout_1 : std_logic_vector(7 downto 0);
signal dout_2 : std_logic_vector(7 downto 0);
signal dout_3 : std_logic_vector(7 downto 0);
signal sel_1d : std_logic_vector(1 downto 0);
begin
sel <= rd_adr(1 downto 0);
process(sel, rd)
begin
if(rd = '1') then
case sel is
when "00" => rd_vector <= "0001";
when "01" => rd_vector <= "0010";
when "10" => rd_vector <= "0100";
when "11" => rd_vector <= "1000";
when others =>rd_vector <= "0000";
end case;
```

```
else
rd_vector <= "0000";
end if;
end process;

BYTE0:ram_8x8 port map(din=> din(7 downto 0),
                       wr_adr=> wr_adr,
                       wr=> wr,
                        dout=> dout_0,
                        rd_adr=> rd_adr(4 downto 2),
                         rd=> rd_vector(0),
                         clk=> clk);
BYTE1:ram_8x8port map(din=> din(15 downto 8),
                      wr_adr=> wr_adr,
                       wr=> wr,
                      dout=> dout_1,
                       rd_adr=> rd_adr(4 downto 2),
                       rd=> rd_vector(1),
                       clk=> clk);
BYTE2:ram_8x8port map(din=> din(23 downto 16),
                      wr_adr=> wr_adr,
                        wr=> wr,
                        dout=> dout_2,
                        rd_adr=> rd_adr(4 downto 2),
                        rd=> rd_vector(2),
                        clk=> clk );
BYTE3:ram_8x8port map(din=> din(31 downto 24),
                      wr_adr=> wr_adr,
                      wr=> wr,
                      dout=> dout_3,
                      rd_adr=> rd_adr(4 downto 2),
                      rd=> rd_vector(3),
                      clk=> clk);
process(clk)
begin
if(rising_edge(clk)) then
   sel_1d <= sel;
else null;
end if;
end process;
process(sel_1d, dout_0, dout_1, dout_2, dout_3)
begin
case sel_1d is
        when "00" => dout <= dout_0;
        when "01" => dout <= dout_1;
        when "10" => dout <= dout_2;
        when "11" => dout <= dout_3;
        when others => dout <= (others => '0');
end case;
end process;
end Behavioral;
```

ram_8x8.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ram8x8 is    Port ( din: in std_logic_vector(7 downto 0);
                           wr_adr: in std_logic_vector(2 downto 0);
                           wr : in std_logic;
                           dout: out std_logic_vector(7 downto 0);
                           rd_adr: in std_logic_vector(2 downto 0);
                           rd: in std_logic;
                          clk: in std_logic);
end ram8x8;
architecture Behavioral of ram8x8 is
type ram_type is array(0 to 7) of std_logic_vector(7 downto 0);
signal ram: ram_type;
signal wr_index: integer range 0 to 7;
signal rd_index : integer range 0 to 7;
begin
wr_index <= CONV_INTEGER(wr_adr);
rd_index <= CONV_INTEGER(rd_adr);
process(clk)
begin
if(rising_edge(clk)) then
  if(wr = '1') then
       ram(wr_index) <= din;
--elsif(rd = '1') then
--dout <= ram(rd_index);
   end if;
dout <= ram(rd_index);
else null;
end if;
end process;
end Behavioral;
ram_8x32.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity lv_ram8x32 is
   Port ( wr_adr  : in std_logic_vector(2 downto 0);
          rd_adr  : in std_logic_vector(2 downto 0);
          data_in  : in std_logic_vector(31 downto 0);
          data_out : out std_logic_vector(31 downto 0);
          wr  : in std_logic;
          rd  : in std_logic;
          clk  : in std_logic);
end lv_ram8x32;
architecture Behavioral of lv_ram8x32 is
```

```
type ram_type is array (0 to 7) of std_logic_vector(31 downto 0);
signal ram        : ram_type;
signal wr_index   : integer range 0 to 7;
signal rd_index   : integer range 0 to 7;
begin
wr_index <= conv_integer(wr_adr);
rd_index <= conv_integer(rd_adr);
process(clk)
begin
if(rising_edge(clk)) then
    if(wr = '1') then
    ram(wr_index) <= data_in;
    elsif(rd = '1') then
    data_out <= ram(rd_index);
end if;
else null;
end if;
end process;
end Behavioral;
```

Ek 11 Kullanılan Bellek elemanlarının yazılımları

dff_radix_4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DFF_RADIX_4 is
   Port ( DIN : in std_logic_vector(1 downto 0);
          CLK : in std_logic;
          RESET: in STD_LOGIC;
          DOUT : out std_logic_vector(1 downto 0));
end DFF_RADIX_4;
architecture Behavioral of DFF_RADIX_4 is
begin
process (CLK, RESET)
begin
 if rising_edge(CLK) then
  if RESET='1' then   --asynchronous RESET active High
     DOUT <= "00";
  else
     DOUT <= DIN;
end if;
else null;
end if;
end process;
end Behavioral;
```

dff_radix_2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dff_radix_2 is
Port ( DIN : in std_logic;
        CLK: in std_logic;
        RESET: in STD_LOGIC;
        DOUT : out std_logic);
end dff_radix_2;
architecture Behavioral of dff_radix_2 is
begin
process (CLK, RESET)
begin
  if rising_edge(CLK) then
      if RESET='1' then   --asynchronous RESET active High
       DOUT <= '0';
  else
        DOUT <= DIN;
end if;
else null;
```

```
end if;
end process;
end Behavioral;
```

dff_radix_2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity dff_radix_2 is
Port ( DIN : in std_logic;
                        CLK  : in std_logic;
                        RESET: in STD_LOGIC;
       DOUT : out std_logic);
end dff_radix_2;

architecture Behavioral of dff_radix_2 is

begin
process (CLK, RESET)
begin
  if rising_edge(CLK) then
                if RESET='1' then   --asynchronous RESET active High
                        DOUT <= '0';
        else
        DOUT <= DIN;
                end if;
        else null;
  end if;
end process;


end Behavioral;
```

ff_enable_radix2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
```

```
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ff_enable_radix2 is
Port ( DIN : in std_logic;
                        ENABLE:in std_logic;
                        CLK  : in std_logic;
                        RESET: in STD_LOGIC;
       DOUT : out std_logic);
end ff_enable_radix2;

architecture Behavioral of ff_enable_radix2 is

begin
process (CLK,RESET)
  begin
                if rising_edge (CLK) then
                        if RESET='1' THEN
                                DOUT <= '0';
                                elsif (ENABLE='1') then
                            DOUT <= DIN;
                                end if;
                else null;
     end if;
end process;


end Behavioral;
```

ff_enable_radix_2_resetli.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ff_enable_radix2_resetli is
Port ( DIN : in std_logic;
                        ENABLE:in std_logic;
                        CLK  : in std_logic;
                        RESET: in STD_LOGIC;
       DOUT : out std_logic);
end ff_enable_radix2_resetli;
```

```
architecture Behavioral of ff_enable_radix2_resetli is

begin

process (CLK,RESET) --Sonradan eklendiiiiii
   begin
        if rising_edge(CLK) THEN
                if  RESET='1' then
                                DOUT <='1';
      elsif (ENABLE='1') then
                DOUT <= DIN;
                 else   null;
      end if;
        end if;
end process;


end Behavioral;
```

ff_enable.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FF_enable is
  Port ( DIN : in std_logic_vector(1 downto 0);
                        ENABLE:in std_logic;
                         CLK  : in std_logic;
                         RESET: in STD_LOGIC;
        DOUT : out std_logic_vector(1 downto 0));
end FF_enable;

architecture Behavioral of FF_enable is

begin

process (CLK,RESET)
   begin
                if rising_edge(CLK) then
                        if(RESET='1') then
                                        DOUT <="00";
               elsif (ENABLE='1') then
               DOUT <= DIN;
                                else null;
```

```
            end if;
              else null;
          end if;
end process;



end Behavioral;
```

dff_radix_8.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity dff_radix_8 is
  Port ( DIN : in std_logic_vector(2 downto 0);
                        CLK    : in std_logic;
                        RESET: in STD_LOGIC;
       DOUT : out std_logic_vector(2 downto 0));

end dff_radix_8;

architecture Behavioral of dff_radix_8 is

begin
process (CLK,RESET)
begin
        if rising_edge(CLK) then
          if RESET='1' then   --asynchronous RESET active High
          DOUT <= "000";
        else
        DOUT <= DIN;
                end if;
        else     null;
  end if;
end process;


end Behavioral;
```

Ek 12 XILINX FPGA özellikleri

XCS200 SPARTAN Entegresi;

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O[1]	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	132	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	196	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

XCV400E VIRTEX Entegresi;

Table 1: **Virtex-E Field-Programmable Gate Array Family Members**

Device	System Gates	Logic Gates	CLB Array	Logic Cells	Differential I/O Pairs	User I/O	BlockRAM Bits	Distributed RAM Bits
XCV50E	71,693	20,736	16 x 24	1,728	83	176	65,536	24,576
XCV100E	128,236	32,400	20 x 30	2,700	83	196	81,920	38,400
XCV200E	306,393	63,504	28 x 42	5,292	119	284	114,688	75,264
XCV300E	411,955	82,944	32 x 48	6,912	137	316	131,072	98,304
XCV400E	569,952	129,600	40 x 60	10,800	183	404	163,840	153,600
XCV600E	985,882	186,624	48 x 72	15,552	247	512	294,912	221,184
XCV1000E	1,569,178	331,776	64 x 96	27,648	281	660	393,216	393,216
XCV1600E	2,188,742	419,904	72 x 108	34,992	344	724	589,824	497,664
XCV2000E	2,541,952	518,400	80 x 120	43,200	344	804	655,360	614,400
XCV2600E	3,263,755	685,584	92 x 138	57,132	344	804	753,664	812,544
XCV3200E	4,074,387	876,096	104 x 156	73,008	344	804	851,968	1,038,336

Ek 13 Euler teoreminin ispatı

Euler Teoremi $P^{\varphi(n)} = 1 \bmod(n)$ şeklinde ifade edilir. Asal olan bir sayının euler fonksiyonu $\varphi(n) = n-1$ şeklinde bulunur. Eğer fonksiyonu bulunacak değer RSA algoritmasındaki gibi iki asal sayının çarpımı ise o sayının euler fonksiyonu $\phi(n) = (p-1)*(q-1)$ şeklinde hesaplanır.

İspat için Q ={1,2,3….,*n*-1}şeklinde tanımlanmış bir kümemiz olduğunu düşünelim.

Bir de *n* sayısı ile aralarında asal olan bir *P* sayısı kullanılıyor olsun

Buradan Q kümesinin her bir elemanını ayrı ayrı *P* sayısı ile çarparak yeni bir U kümesi oluşturalım.

U={1**P*,2**P*,3**P*,……,(n-1}**P*}

$U_1 = Q_1 * P$

$U_2 = Q_2 * P$

……

$U_{n-1} = Q_{n-1}$

Eğer U kümesinin her bir elemanı ayrı ayrı (mod *n*) işlemine sokulursa görülecektir ki U kümesindeki her farklı giriş değerine Q kümesinde bir değer karşılık gelecektir. Dolayısıyla, bir yanda U kümesinin tüm elemanları birlikte (mod n) işlemine sokulursa diğer yanda Q kümesinin tüm elemanlarının işlem sonucunda elde edilebilmesi gerekmektedir.

$$U_1 * U_2 * U_3 * ..... * U_{n-1} = Q_1 * Q_2 * Q_3 * ..... * Q_{n-1} \bmod(n)$$

$$P * Q_1 * P * Q_2 * P * Q_3 * ..... * P * Q_{n-1} = Q_1 * Q_2 * Q_3 * ..... * Q_{n-1} \bmod(n)$$

Denklemin her iki yanındaki Q değerlerini sadeleştirdiğimizde;

$P^{n-1} = 1 \bmod(n)$ sonucu elde edilir.

n nin euler fonksiyonunun $\varphi(n) = n-1$ olduğunu bilerek;

$P^{\varphi(n)} = 1 \bmod(n)$ eşitliğini ispatlamış oluruz.

ÖZGEÇMİŞ

Doğum Tarihi	13.11.1979	
Doğum Yeri	Giresun/Bulancak	
Lise	1993-1997	Giresun Süper Lisesi
Lisans	1997-2002	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fakültesi Elektronik ve Haberleşme Mühendisliği Bölümü

Çalıştığı Kurumlar

2002-2003	İris Telekomünikasyon ve Mühendislik Hizmetleri
2003-Devam ediyor	TÜBİTAK-UEKAE

ÖZGEÇMİŞ

Doğum Tarihi	13.11.1979	
Doğum Yeri	Giresun/Bulancak	
Lise	1993-1997	Giresun Süper Lisesi
Lisans	1997-2002	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fakültesi Elektronik ve Haberleşme Mühendisliği Bölümü

Çalıştığı Kurumlar

2002-2003	İris Telekomünikasyon ve Mühendislik Hizmetleri GSM Şebekesi Planlama ve Optimizasyon Mühendisi
2003-Devam ediyor	TÜBİTAK-UEKAE