T.C.
DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

SAYISAL GÖRÜNTÜYE VERİ GÖMMEK VE AYRIŞTIRMAK İÇİN FPGA TABANLI DONANIM MODÜLÜ TASARIMI

Serdar KIRIŞOĞLU

Temmuz 2009

T.C.
DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

SAYISAL GÖRÜNTÜYE VERİ GÖMMEK VE AYRIŞTIRMAK İÇİN FPGA TABANLI DONANIM MODÜLÜ TASARIMI

Serdar KIRIŞOĞLU

DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK EĞİTİMİ BÖLÜMÜ
YÜKSEK LİSANS
DERECESİ İÇİN GEREKLİ ÇALIŞMALAR YERİNE GETİRİLEREK
ONAYA SUNULAN TEZ

Temmuz 2009

Fen Bilimleri Enstitüsü Müdürlüğüne

Bu çalışma jürimiz tarafından Elektrik Eğitimi Anabilim Dalı Yüksek Lisans Tezi olarak kabul edilmiştir

Tez Danışmanı	Prof. Dr. İsmail ERCAN..……	
	(Düzce Üniversitesi)	(imza)
Üye	Prof. Dr. İsmail ERCAN……..	
	(Düzce Üniversitesi)	(imza)
Üye	Yrd. Doç. Dr. Pakize ERDOĞMUŞ…….…...	
	(Düzce Üniversitesi)	(imza)
Üye	Yrd. Doç. Dr. M. Atilla BÜYÜKGÜÇLÜ...…...	
	(Düzce Üniversitesi)	(imza)

ONAYLI

Bu tez, Fen Bilimleri Enstitüsü Yönetim Kurulu'nca belirlenen yukarıdaki jüri üyeleri tarafından uygun görülmüş ve Yönetim Kurulu'nun kararıyla kabul edilmiştir.

İmza

Prof. Dr. Refik KARAGÜL

ABSTRACT

FPGA-BASED MODULE DESING FOR EMBEDİNG AND RECOVERING DATA INTO/FROM DIGITAL IMAGE

KIRIŞOĞLU, SERDAR

Master of Science, Department of Electrical Education

Advisor: Prof. Dr. İsmail ERCAN

JUNE 2008

In this study, moduls have been designed for the purpose of realizing Least Significant Bit (LSB) method of Steganography technique which is a traditional data hiding art by using special aimed Field Programmable Gate Array (FPGA). LSB method uses the least significant bits of binary value of pictures red, green, blue (RGB) components. These bits are replaced with data which will be hidden. These calculations take a long time by programming which are written by programming language. For this reason in order to decrease the load of the computer and to isolate from the programs which are open to threaths, FPGA modules have been designed for this data hiding application. Besides for a better understanding of the operations and the results, the program has been simulated in Matlab's Gui.

Keywords: FPGA, Hardware Module, Steganography, Data Hiding.

ÖZET

SAYISAL GÖRÜNTÜYE VERİ GÖMMEK VE AYRIŞTIRMAK İÇİN FPGA TABANLI DONANIM MODÜLÜ TASARIMI

KIRIŞOĞLU, SERDAR

Yüksek Lisans, Elektrik Eğitimi Anabilim Dalı

Tez Danışmanı: Prof. Dr. İsmail ERCAN

ARALIK 2008, Sayfa

Bu çalışmada özel amaçlı Alan Programlanabilir Kapı Dizileri (Field Programmable Gate Array FPGA) kullanılarak eski bir veri gizleme sanatı olan Steganografi tekniğinin en anlamsız bit (Least Significant Bit LSB) metodunun gerçekleştirilmesine yönelik modüller tasarlanmıştır. LSB metodu resimlerin kırmızı, yeşil, mavi (Red, Green, Blue RGB) bileşenlerinin sayısal olarak ikilik değerlerinin en anlamsız bitlerini kullanır. Bu en anlamsız bitler gizlenecek olan veri ile yer değiştirir. Bu gibi işlemler programlama dilleri kullanılarak yazılmış programlarla yapılırken uzun zaman almaktadır. Bu yüzden bilgisayarın işlemcisinin yükünü azaltmak ve saldırıya açık programlardan izole etmek amacıyla bu veri gizleme işlemi için FPGA modülleri tasarlanmıştır. Ayrıca işlemlerin daha kolay anlaşılabilmesi ve veri gizlemeden sonraki sonuçları rahat görebilmek amacıyla program Matlab'ın Gui'si kullanılarak simüle edilmiştir

Anahtar Kelimeler: FPGA, Donanım Modülü, Steganografi, Veri Gizleme.

Ablam Saliha ERENTURK, Anne ve Babama ithaf ediyorum.

TEŞEKKÜR

Tez danışmanlığımı üstlenerek araştırma konusunun seçimi, yürütülmesi ve sunuma hazırlanması sırasında, değerli bilimsel görüş ve önerilerinden yararlandığım Prof. Dr. İsmail ERCAN'a teşekkür eder ve en içten minnet duygularımı sunarım.

Tezin hazırlanmasında engin bilgi ve tecrübelerinden yararlandığım ve tezime çok emeği geçmiş olan Yrd. Doç. Dr. İbrahim ŞAHİN'e en içten teşekkürleri bir borç bilir, saygılarımı sunarım.

İÇİNDEKİLER

ŞEKİLLER

TABLOLAR DİZİNİ

1. GİRİŞ

Günümüzde bilgi ne nedenli değerli olduğu şüphesiz çoğu insan tarafından kabul edilmektedir. Bilginin bir yerden başka bir yere iletilmesi ise haberleşme kavramını ortaya çıkarmıştır. Teknolojinin gelişmesiyle birlikte eskiden kullanılan haberleşme yöntemleri yerini elektronik ortamlara bırakmıştır. Artık insanlar birbirlerine ağ denilen ortamlarla ve bu ağlara bağlı bazı elektronik cihazlar yardımıyla iletişim kurmakta, önemli veya önemsiz bilgilerini birbirleriyle paylaşmaktadırlar. İnternet ise bilgisayarlar için geliştirilmiş bir ağ ortamıdır ve bilgisayarların işleyebileceği sayısal verileri taşımaktadır. İnternet ortamında iletilen veriler birbirine bağlı birçok noktadan geçerek ulaşacağı yere varmaktadır. Bu nedenle iletişim için kullanılan bu ortamda iletilecek bilgiye alıcıdan önce ulaşmak çok zor değildir. Nitekim bazı önlemler alınsa da, iletişim ortamındaki bu verilere yapılan saldırıların her geçen gün arttığı rapor edilmiştir [1,2]. Bu nedenle bilgileri gizleyebilen sistemler geliştirilmiştir.

Steganografi ise bilgi güvenliğini sağlamak için ortaya çıkmış çok eski bir veri gizleme sanatıdır [3]. Bu veri gizleme sanatı bilgisayar dünyasında haberleşme esnasında yapısından dolayı daha güvenli bir iletişim sunmakta ve çok yaygın olarak kullanılmaktadır. İletilen verinin çeşidi her ne olursa olsun bu sanata adapte etmek mümkün olmaktadır. Ses, resim, yazı bunların başında gelen veri türleridir ve steganografi sanatı en çok bu veriler üzerinde kullanılmaktadır. Bu çeşitli veriler için bu sanatın farklı metotları vardır. Bu sanatı esas alarak günümüz iletim ortamına göre uyarlanmış olan En Anlamsız Bit (Least Significant Bit LSB) metodu geliştirilmiştir. Bu metot ileriki bölümlerde daha detaylı anlatılacaktır.

Çalışmanın Amacı ve Gerçekleşme Aşamaları

Bu çalışmanın asıl amacı Steganografi sanatının LSB metodunu daha hızlı ve daha güvenli bir şekilde uygulamak amacıyla, Alan Programlanabilir Kapı Dizileri (Field Programmable Gate Array FPGA) çipleri üzerinde çalışabilecek, birinin görevi iletilecek olan mesajı taşıyıcı resme gömmek, bir diğerinin görevi de gömülmüş olan mesajı taşıyıcı resimden ayrıştırmak olan iki ayrı donanım modülü tasarlamaktır. Böylelikle görüntü içine veri gömmede, daha ucuz, daha hızlı ve

FPGA çiplerinin tekrar programlanabilme özellikleri sayesinde daha esnek bir yapı ortaya koymaktır. Bu amaçla LSBIN ve LSBOUT olarak iki ayrı modül tasarlanmıştır. Tasarlanan modüller, gerçek veri üzerinde işlemler yapılarak test edilmiş ve modüllerin ürettiği sonuçların doğrulanması yapılmıştır. Aynı veriler değişik özellikteki bilgisayarlarla da işlenerek modüllerin veri işleme hızı genel amaçlı bilgisayarlarla karşılaştırılmıştır.

Tezin İkinci Bölümünde FPGA çipleri, Reconfigurable Computing (RC) sistemler ve veri gizlemede kullanılan Steganografi sanatı ve bu sanatın bir metodu olan LSB yöntemi kısaca özetlenmiştir. Üçüncü Bölümde, tasarlanan modül detaylarıyla anlatılmıştır. Dördüncü Bölümde, yapılan test çalışmaları ve bu çalışmalardan elde edilen sonuçlar sunulmuştur. Beşinci Bölümde ise sonuçlar değerlendirilmiştir. Ayrıca bu bölümde, gelecekte yapılabilecek çalışmalar hakkında önerilerde bulunulmuştur.

2. ÖN BİLGİ VE DİĞER ÇALIŞMALAR

2.1. FPGA Çipleri

FPGA ingilizcesi Field Programmable Gate Array olan ve Türkçeye alan programlanabilir kapı dizileri olarak çevirebileceğimiz çiplerdir. FPGA genel bir tanım olarak "Bir lojik blok dizisi, bu dizinin çevresinde bir halka oluşturan giriş çıkış birimleri ve bütün bu birimleri bağlayan programlanabilir ara bağlantılardan oluşan aygıttır." [4] verilebilir. Bu çipler kullanıcının ihtiyacına göre kullanıldığı yerde programlanabilir esnek bir yapıya sahiptir. Kullanıcı fonksiyonunu gerçekleştirmesi için tasarladığı devreyi kapasitesi dahilinde FPGA çipi içine uygulayabilmektedir. Kullanıcının tasarladığı devre, FPGA tarafından mantıksal bloklar, ara bağlantılar ve giriş çıkış blokları olarak uygulamaya geçirilir. FPGA'ler lojik bloklar ve bunları kaplayan giriş çıkış bloklarından oluşur. FPGA'ler onbinlerce lojik blok ve flip-floplar'dan oluşur [5].

Tabi FPGA çiplerinin teknolojik yeniliklerinin arkasında bir geçmiş bulunmaktadır. Bu programlanabilir yapıların gelişimini aşağıda gösterildiği gibi sıralayabiliriz [4];

Programlanabilir Salt Okunur Bellekler (Programmable Read Only Memory, PROM).

Silinebilir ve Programlanabilir Salt Okunur Bellekler (Erasable Programmable Read Only Memory, EPROM)

Elektriksel Silinebilir ve Programlanabilir Salt Okunur Bellekler (Electronically Erasable Programmable Read Only Memory, EEPROM)

Programlanabilir Lojik Devreler (PLD). PLD'ler ise Programlanabilir Dizi Lojiği (Programmable Array Logic, PAL) ve Programlanabilir Lojik Dizisi (Programmable Logic Array, PLA) olmak üzere iki ayrı isimle karşımıza çıkabilir.

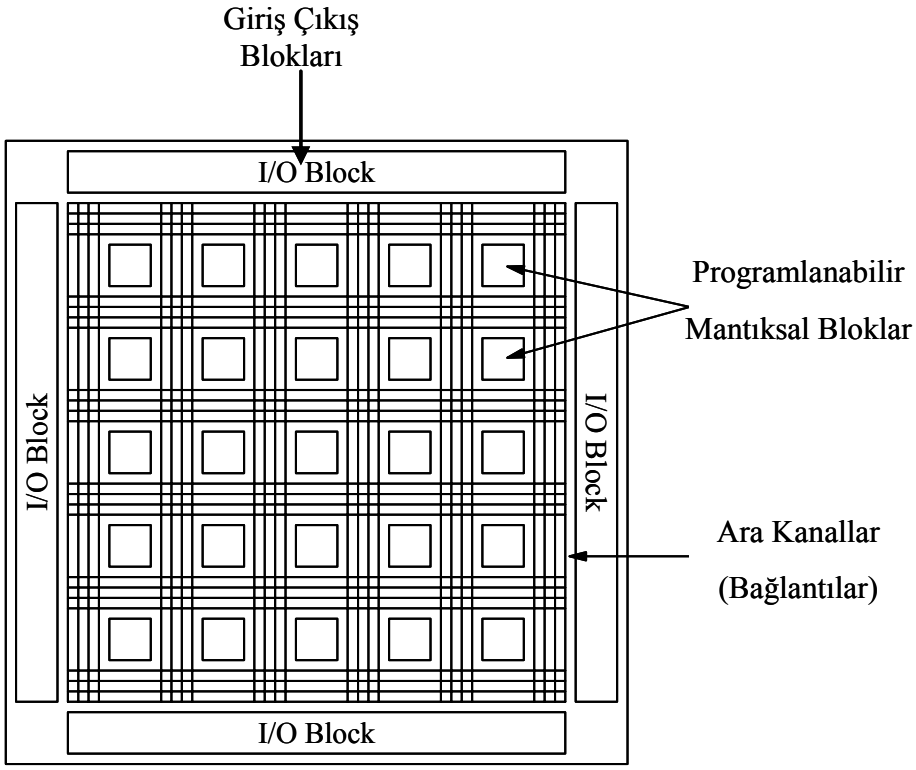
Maske Programlanabilir Kapı Dizileri (Mask Programmable Gate Array, MPGA).

FPGA çiplerini Xilinx firması MPGA ve PLD'lerin üstünlüklerini kullanmışlar ve zayıflıklarını ortadan kaldırarak 1985 yılında piyasaya sürülmüştür.

Günümüzde kullanılan FPGA çiplerini ise iki kategoride inceleyebiliriz [6];

1. Statik RAM (SRAM): SRAM FPGA'lar kullanıcı tarafından defalarca programlanabilirler. Bu tür FPGA'lar RAM tabanlı olduğu için, enerjileri kesilip tekrar verildiğinde yeniden programlanmaları gerekmektedir. Bunu önlemek için seri bir PROM kullanılarak program bilgisinin burada depolanması sağlanabilir.
2. OTP (One Time Programmable) OTP FPGA'lar sadece bir kez programlanabilirler. Bu yüzden seri bir PROM kullanmaya gerek yoktur. Ancak tasarımda yapılacak herhangi bir değişiklikte daha önce programlanmış entegrenin yerine yeni bir entegre kullanılması gerekmektedir.

FPGA çipleri Şekil 2.1'de genel yapısı gösterildiği gibi üç ana bileşenden oluşur;



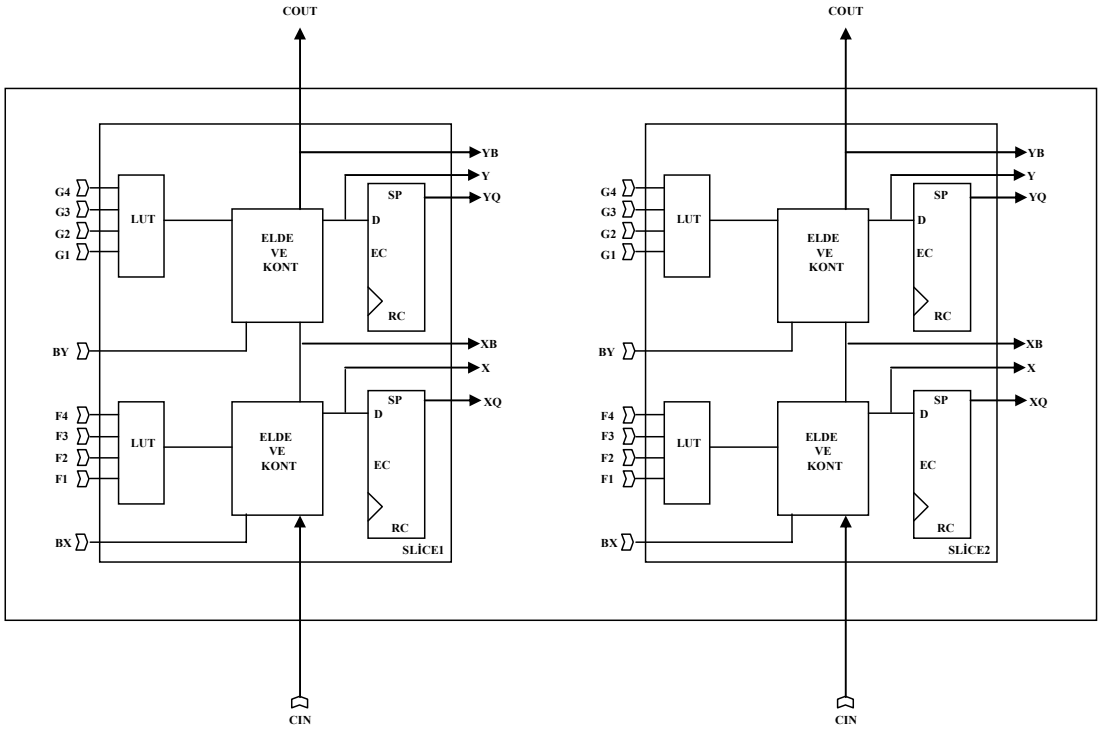
Şekil 2.1. FPGA Genel Yapısı [7].

a. Ayarlanabilir Mantıksal Bloklar (Configurable Logic Blocks (CLB)): Bu lojik blokların gerçekleştirebildiği lojik fonksiyonlar ve yönlendirmenin performansı, yoğunluk ve performansla doğru orantılıdır. Bu lojik bloklar kullanıcının talep ettiği fonksiyonların gerçekleştirilmesini sağlarlar. Şekil 2.2 de gösterildiği gibi CLB blokları içerisinde LUT (Look Up Table)'ler mevcuttur. Bu LUT'ler küçük lojik fonksiyonların geçekleştirebilmeleri için küçük depolama birimlerine sahiptirler [8].

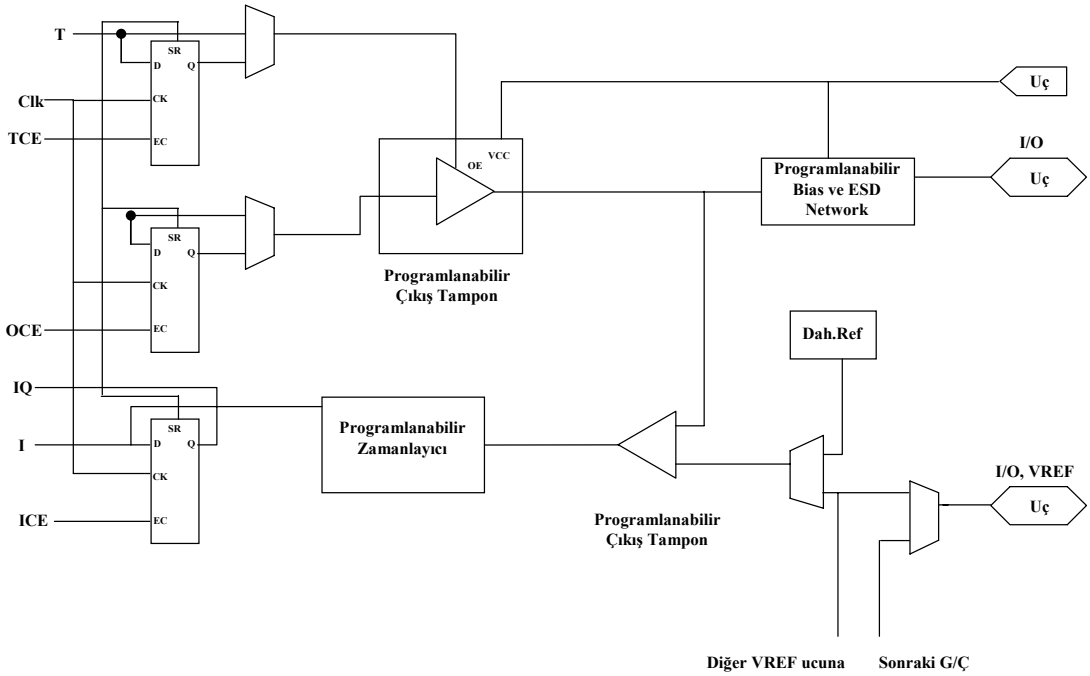
b. Giriş Çıkış Blokları (Input/Output Blocks (IOB)): Şekil 2.3'de yapısı gösterilen IOB'ler FPGA çipinin iç sinyalleri ile yonganın pinleri arasındaki programlanabilir bağlantıyı sağlarlar. Bu bloklar sayesinde çip giriş, çıkış yada çift yönlü programlama esnekliği kazanırlar [4].

c. Ara Bağlantılar (Interconnections): Şekil 2.4'de gösterilen bu bloklar lojik bloklar ve giriş çıkış blokları arasındaki bağlantıları sağlamaktadırlar. Bu bloklar yollandırma kanalları ve durumları değiştirilebilen anahtarlardan oluşmaktadırlar. Yollandırma kanalları lojik blokların en uygun şekilde birbirlerine bağlanmaları açısından önemlidirler. Bu kanalların tasarım için gerekli düzeyde olmaması durumunda çipinizin içindeki diğer blokların fazla olması pek bir anlam ifade etmez. Bu anahtarlar statik RAM hücresi ile kontrol edilen geçiş transistörü, antifuse, EPROM ve EEPROM teknolojileri kullanılarak oluşturulurlar [8].

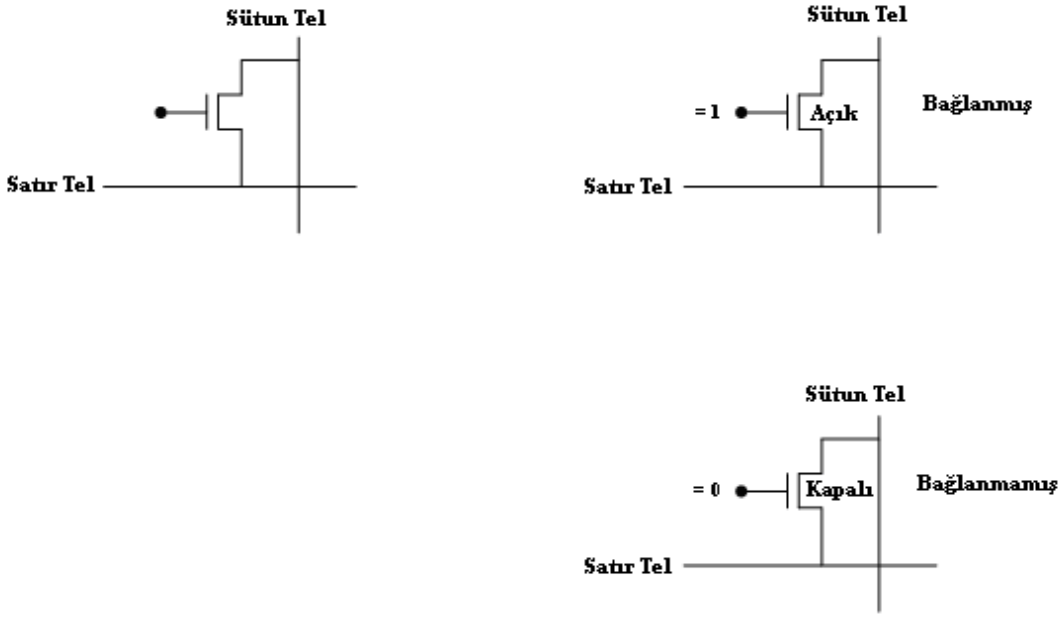
Programlanabilir olduklarından çok esnek bir yapıya sahiptirler [9].



Şekil 2.2 CLB'nin İç Yapısı [10].



Şekil 2.3 IOB yapısı Xilinx Spartan [11].



Şekil 2.4. FPGA Ara Bağlantı Yolları [8].

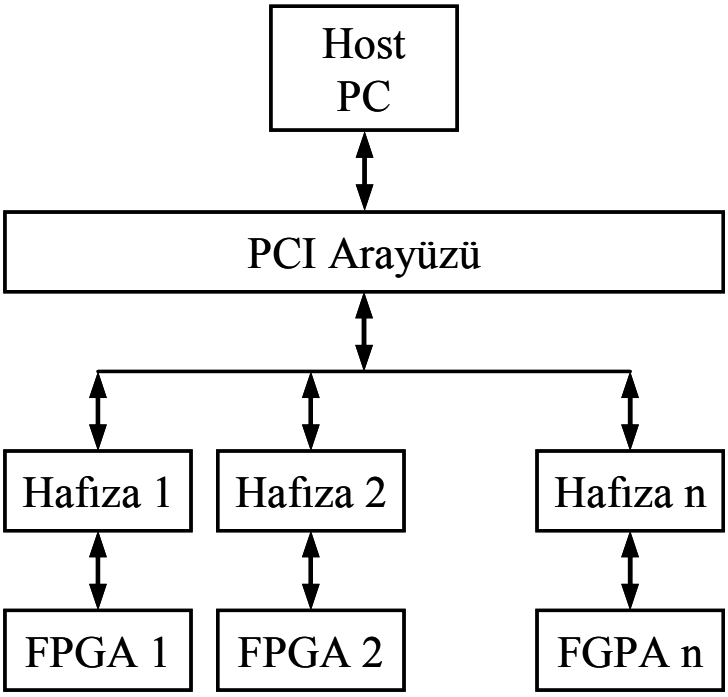
2.2. FPGA Tabanlı Hiper ve Özel Amaçlı Bilgisayarlar

Süper bilgisayarlar gelişmiş özellikleri hızlı işlem yapabilmesi ile dikkat çekmektedirler. Ancak bu bilgisayarlar inanılmaz yeteneklere sahip olmalarına rağmen küçük bir fabrikayı besleyecek kadar çok enerji sarf etmeleri, çok fazla yer kaplamaları, sabit ısı kontrolü gereksinimi ve çok sayıda kablo ve elektrik teli kullanmak gibi dezavantajlara sahiptirler. Fakat bu dezavantajlar FPGA çipleri sayesinde aşılmış, masaüstüne sığacak kadar küçültülmüş ve bir saç kurutma makinesinden fazla enerji harcamayan süper bilgisayarlar yapılmıştır. FPGA teknolojisi ile üretilen HAL-15 adlı bu bilgisayara hiper bilgisayarda denilmektedir [12].

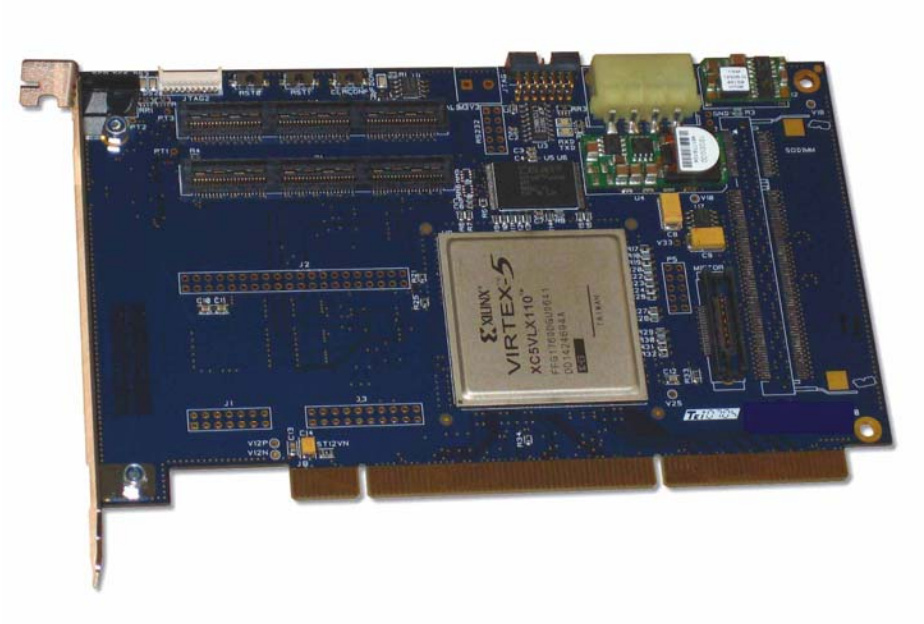
Merkezi ABD'nin Utah eyaletinde bulunan Star Bridge Systems'in geliştirdiği FPGA (Field Programmable Gate Array) teknolojisi ile üretilen süper bilgisayar, geleneksel bilgisayarlardan bin kat daha hızlı. FPGA çipleri kendi kendilerini saniyede yüzler hatta binlerce defa yeniden düzenleyebildiği (reconfigure) için aynı anda çok sayıda işlem yapabiliyor [12].

Hiper bilgisayarı kullanan NASA'daki mühendisler aynı anda milyarlarca işlem yapabilen bu makinenin, piyasadaki bütün süper bilgisayarlardan çok daha hızlı ve çok yönlü olduğunu söylemişlerdir. Yüksek performanslı bu bilgisayarların gücü, klasik CPU'ların (central processing unit) yerine FPGA çipleri kullanmasından kaynaklanmaktadır. FPGA, işlemci üzerindeki milyonlarca transistör ya da geçitin kullanımını en üst seviyeye çıkarıyor (maximize edebiliyor). Bir başka deyişle, FPGA herhangi bir görev için ne kadar transistör gerektiğine karar verebiliyor. Geleneksel işlemcilerde ise birçok uygulama için sadece bir bölüm kullanılıyor [12].

FPGA tabanlı Özel Amaçlı Bilgisayarlar (FPGA-based Custom Computing Machines *F-CCMs*) ise donanım ve yazılımı bir araya getiren özel veri işleme platformlarıdır [13]. Bu platformlar genellikle *Reconfigurable Computing* (RC) sistemler olarak ta adlandırılabilirler. Bu platformlar bir PCI ara yüzüyle PCI yuvasına sahip genel amaçlı bir bilgisayara bağlanabilirler. Bu platformların üzerinde bir yada birden fazla FPGA ve hafıza çipleri bulunabilen elektronik kartlardır [14]. Bu sistemler genel amaçlı bilgisayarın işlemcisinin programlanabilme özelliği ile FPGA çiplerinin hızlı veri işleme özelliğini bir araya getirirler [9]. Bilgisayarın işlemcisini aşırı yoran ve hız gerektiren bazı programlar RC sistemler vasıtasıyla kullanılacak program için özel olarak tasarlanmış modülü barındıran FPGA çipine yaptırılarak 10 ile 100 kat arasında hız kazancı sağlayabilmektedir [7]. Şekil 2.5'te genel bir RC sistemin yapısı görülmektedir. Şekil 2.5'de *n* adet FPGA ve hafıza çipine sahip PCI ara yüzü sayesinde genel amaçlı bilgisayarla haberleşebilen bir yapı görülmektedir. Şekil 2.6'da ise Hitech Global firması tarafından genel amaçlı olarak tasarlanmış bir FGPA kartı görülmektedir.



Şekil 2.5. Tipik Bir RC Sistem Genel Yapısı [9].



Şekil 2.6. Hitech Global Firmasının PCIXSYS-V5 FPGA kartı [15].

FPGA kartlarının kullanılmasında izlenecek yol;

İlk aşamada önceden tasarlanmış donanım modülü hakkındaki konfigürasyon bilgileri PCI veri yolu üzerine takılı olan FPGA çipine yüklenir.

İkinci aşamada FPGA çipinin işlemesi gereken veri PCI veri yolu aracılığı ile kart üzerindeki hafıza birimlerine aktarılır.

Üçüncü aşamada FPGA çipine yollanacak olan başlama sinyali ile verinin işlemeye başlatılması sağlanır. Bu esnada paralel çalışmasının getirdiği avantaj sayesinde ana bilgisayar diğer işlemlere devam edebilmektedir.

Dördüncü aşamada veri işleme bittikten sonra FPGA çipi işlem bitti sinyali ile ana bilgisayar veri işleme işleminin bittiğini kesme sinyali olarak yollar.

Beşinci aşamada ise ana bilgisayar işlenmiş veriyi kart üzerindeki yerel hafızadan okur [14].

FPGA ile bilgisayar arasında hızlı veri iletişimi için DMA (Direct Memory Access-Doğrudan Hafıza Erişimi) kullanır.

2.3.FPGA Çiplerin Avantajları

ASIC (Application Specific Integrated Circuit- Uygulamaya Özel Tümdevre) çiplerinin çeşitli tasarım ve üretim aşamaları uzun zamanlar almaktadır ve bu aşamalardan sonra yapılan yanlışlıklar için geri dönüş mümkün olmamaktadır. Hatalı üretilen çipler ise geri dönüşü olmadığından kullanılamaz olmaktadırlar.

FPGA çipleri ise hızlı ve tekrar programlanabilir yapıları sayesinde çok esnektirler ve bu çipler için üretilen tasarım çok hızlı bir şekilde FPGA çipine yüklenebilirler.

FPGA çipleri tekrar programlanabilme özelliği sayesinde tasarım aşamasında yapılabilecek hataların telafisi mümkün kılınabilmektedir [16].

FPGA çipleri belirli bir problemi çözme için tasarlandıklarından yazılımlara göre çok daha hızlı işlem yapabilmektedirler. Örneğin FPGA çipleri 550 Mhz saat frekansına çıkabilmektedirler [17,18]. Çipler paralel çalışabilir ve bir çip içine kapasitesi dahilinde fazladan modüller yüklenebilir ve bu da bize hız kazancı sağlamaktadır [9].

Altera, Xilinx, Flex, Lucent, Actel firmaları FPGA çipleri üreten belli başlı firmalardır. Firmalar ürettikleri FPGA çiplerine farklı özel isimler vermişlerdir.

Xilinx firması Spartan, Virtex [17] gibi isimler verirken Altera firması ürettiği çiplere Cyclone ve Stratix [13] gibi isimler vermişlerdir.

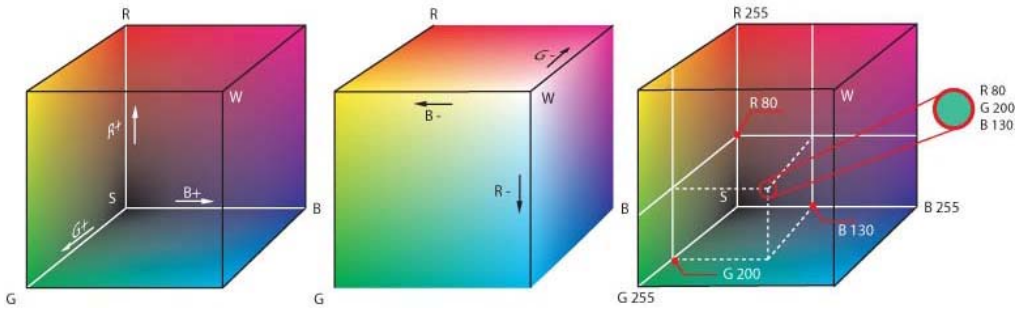
2.4.Sayısal Görüntü

Renkler, Şekil 2.7'de gösterildiği gibi ışığın değişik dalga boylarının gözün retinasına ulaşması ile ortaya çıkan bir algılamadır. Bu algılama, ışığın maddeler üzerine çarpması ve kısmen soğurulup kısmen yansıması nedeniyle çeşitlilik gösterir ki bunlar renk tonu veya renk olarak adlandırılır. Tüm dalga boyları birden aynı anda gözümüze ulaşırsa bunu beyaz, hiç ışık ulaşmazsa siyah olarak algılarız. İnsan gözü 380nm ile 780nm arasındaki dalga boylarını algılayabilir, bu sebepten elektromanyetik spektrumun bu bölümüne görünen ışık denir. Renkler için genelde kulağımızla duyduğumuz ince ve kalın ses analojisi yapılsa da, ses algısının aksine aynı anda gelen ışık frekansları değişik kanallardan algılanamaz (başka bir deyişle göz frekans analizi yapamaz), dolayısıyla aynı anda ince ve kalın sesleri birbirine karıştırmadan duymamıza karşın gözümüz için bu 'çok seslilik' söz konusu olmadığından değişik ışık frekanslarının sadece kombinasyonlarını algılayabiliriz. Bu prensibi açıklamak veya pratik uygulamalarda kullanmak için çeşitli renk modelleri geliştirilmiştir [19].

Renk	Dalga Boyu	Frekans
Kırmızı	~ 625-740 nm	~ 480-405 THz
turuncu	~ 590-625 nm	~ 510-480 THz
Sarı	~ 565-590 nm	~ 530-510 THz
Yeşil	~ 500-565 nm	~ 600-530 THz
Camgöbeği	~ 485-500 nm	~ 620-600 THz
Mavi	~ 440-485 nm	~ 680-620 THz
Mor	~ 380-440 nm	~ 790-680 THz

Şekil 2.7. Renklerin Dalga Boyları ve Frekansları [19].

İşte bu renk modellerinden en yaygın olarak kullanılan model kırmızı, yeşil ve mavi (RGB) renk modelidir. Şekil 2.8'de görüldüğü gibi bütün renkler bu üç ana rengin çeşitli oranlarda karışımından meydana gelmektedir. Bu üç temel rengin farklı kombinasyonlarla bir araya gelmesiyle oluşan diğer renklere ara renkler yada ikincil renkler denir.



Şekil 2.8. Renk Küpleri [20].

Sayısal görüntü ise resimlerin bilgisayarda depolanabilmesi, işlenebilmesi ve görüntülenebilmesi için resimlerin sayılarla ifade edilmesi demektir. Sayısal resimler bilgisayar ekranında oluşturulurken RGB renk modeli kullanılır. Bilgisayar ekranı yatayda ve dikeyde birbirleriyle kesişen gözle görülemeyecek kadar küçük karelerin birleşmesinden meydana gelmiştir. Ekranı oluşturan bu küçük karelerin her birine piksel denir. Renkli bir ekranda Şekil 2.9'da gösterildiği gibi her piksel üç ana rengi oluşturabilecek olan RGB bileşenlerinden oluşur.

Piksellerin bir araya gelmesinden ise resim bilgisayarın ekranında görüntülenebilmektedir. Resmin boyutunu ise piksellerin adedi belirler. Bilgisayar ekranında yatay ve dikey olarak sıralanmıştırlar. Resmin boyutu resmin yatayda ve dikeyde kapladığı piksellerin çarpımıyla hesaplanmaktadır. Örneğin yatayda 100 piksel ve dikeyde 100 piksellik bir yer kaplayan resim 100x100 boyutundadır denilebilir, buna da özel olarak çözünürlük denir.

Ekranların yapılması esnasında farklı teknolojiler kullanılsa da bu piksel yapısına sadık kalınmıştır.

24 bitlik RGB renk modelinde her bir renk bileşeni için 8 bitlik bir aralık ayrılmıştır. Her bir bit farklı 2 olasılık demektir. Bu nedenle her bir renk bileşeni 2^8 farklı tona sahip olmaktadır. Ara renklerde dahil olmak üzere toplamda 2^{32} farklı renk oluşturulabilmektedir.

32 bitlik renk modelinde ise RGB bileşeninin yanı sıra 8 bitlik bir saydamlık (α) bilgisi de bulunmaktadır.

Resimler bilgisayarda depolanırken farklı formatlar halinde depolanabilirler. Bu formatlardan en çok kullanılanları bitmap ve jpeg formatıdır. Kayıpsız sıkıştırma denilen metodu kullanan Bitmap resimler 24 bit RGB renk modelini kullanarak sıkıştırılmadan bilgisayarın hafızasında tutulurlar. Jpeg resimler ise bilgisayarın hafızasında fazla yer kapsamamaları için belirli bir algoritmaya göre sıkıştırılarak depolanırlar, bu depolama esnasında renk bileşenlerinin bazıları tam olarak kodlanamadığından dolayı bu yönteme kayıplı sıkıştırma denilmektedir.

R: 68 G: 43 B: 70	R: 70 G: 43 B: 72	R: 71 G: 44 B: 70	R: 73 G: 43 B: 66	R: 76 G: 43 B: 65	R: 74 G: 41 B: 69	R: 71 G: 42 B: 70	R: 72 G: 44 B: 67	R: 76 G: 54 B: 61
R: 71 G: 44 B: 70	R: 71 G: 44 B: 67	R: 69 G: 42 B: 65	R: 70 G: 43 B: 67	R: 69 G: 41 B: 67	R: 72 G: 46 B: 62	R: 87 G: 64 B: 55	R:110 G: 90 B: 51	R:128 G:116 B: 41
R: 72 G: 44 B: 73	R: 70 G: 44 B: 70	R: 69 G: 43 B: 68	R: 74 G: 49 B: 64	R: 81 G: 64 B: 54	R:102 G: 90 B: 40	R:132 G:121 B: 30	R:148 G:138 B: 25	R:151 G:144 B: 19
R: 72 G: 44 B: 66	R: 75 G: 47 B: 62	R: 92 G: 70 B: 53	R:115 G: 96 B: 44	R:130 G:118 B: 23	R:143 G:133 B: 11	R:152 G:140 B: 11	R:151 G:143 B: 13	R:153 G:148 B: 18
R: 75 G: 56 B: 53	R:103 G: 89 B: 47	R:129 G:120 B: 42	R:135 G:126 B: 27	R:145 G:135 B: 15	R:151 G:141 B: 7	R:153 G:143 B: 8	R:157 G:145 B: 15	R:164 G:150 B: 15
R:115 G:102 B: 45	R:136 G:128 B: 37	R:131 G:126 B: 12	R:142 G:133 B: 11	R:151 G:142 B: 18	R:153 G:143 B: 13	R:157 G:146 B: 10	R:160 G:150 B: 18	R:164 G:155 B: 30
R:135 G:127 B: 29	R:128 G:124 B: 14	R:141 G:133 B: 7	R:153 G:143 B: 7	R:151 G:143 B: 6	R:153 G:145 B: 9	R:160 G:151 B: 19	R:163 G:154 B: 24	R:165 G:154 B: 21

Şekil 2.9. Piksel Değerleri [21].

2.5. Steganografi ve LSB Metodu

Steganografi eski bir veri gizleme sanatıdır [3]. Kelime olarak Yunan alfabesinden türetilmiştir. Kökleri στεγαυο-ς (kaplanmış) ve γραΦ-ειν (yazı) kelimelerinden gelir [22]. Kelime anlamı ise gizli yazı veya örtülü yazı anlamına gelmektedir [23].

Steganografi günümüz sayısal ortamları için ideal bir güvenlik metodudur. Çünkü ağ iletişiminde yollayacağımız herhangi bir mesaj birçok noktadan geçmekte ve saldırıya ve kontrole açık olmaktadır. Ancak mesajımız onu fark ettirmeyecek bir taşıyıcı içine gizlenirse bu mesajımızı daha güvenli bir halde iletmemize olanak sağlar. Taşıyıcılarımız herhangi bir veri dosyası olabilir. Sayısal verilerin farklı formatlarda olması ise seçeneklerimizin çoğalmasını sağlamaktadır. Bu birbirinden farklı formatlara steganografik yöntemlerle verilerimiz saklanabilmektedir [24,25,26,27,28,29,30].

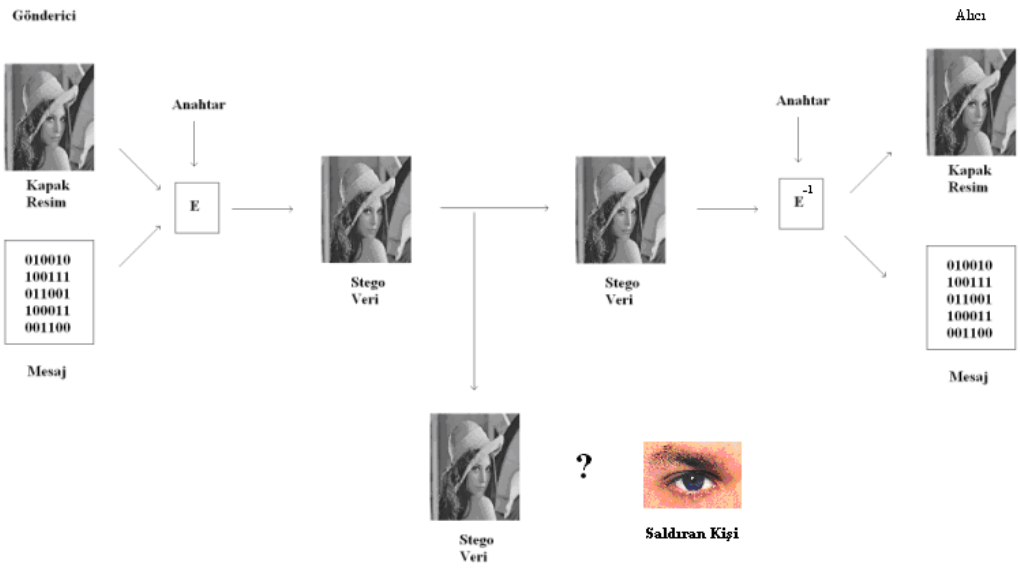
Steganografi'nin amacı gizli bir mesaj yada bilginin varlığını saklamaktır. Steganografi aslında Şekil 2.10'da gösterildiği gibi bazı temel unsurlardan oluşur. Bu unsurlar;

Kapak (cover) : Resim, ses, text vs. dosyasının orjinal halini belirtir.

Anahtar : Gömülecek veriyi şifrelemek için kullanılan anahtar.

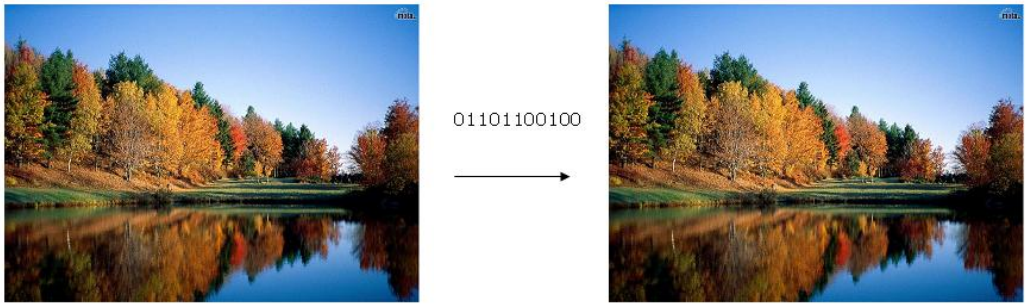
Gömülmüş (embedded) : *Kapak* dosyanın içerisinde gizli olan veriyi ifade eder.

Stego : *Kapak* dosyanın, mesaj gömüldükten sonraki halidir.



Şekil 2.10. Stegosistem Modeli.

Pratikte bu içeriği potansiyel saldıran kişinin dikkatini çekmeyecek olan diğer dijital veride ufak değişiklikler yaparak sağlanır. İnsan görme sistemi ise bu küçük değişimleri farkedememektedir [31]. Şekil 2.11.'de buna bir örnek olan bir manzara resmi gösterilmektedir.



Şekil 2.11. Örnek Manzara Resmi [32].

Steganografi sadece taşıyıcı içindeki verileri değiştirerek mesaj gizleme mantığı değildir. Bazen de taşıyıcı içindeki boşluklar ve gereksiz alanlar kullanılarak

ta yapılabilmektedir [33,29]. Buna en güzel örnek html kodlardır. Html kodlarda özel "<" ve ">" karakterleri arasında etiketler (tag) mevcuttur. Bu etiketler metinlere özellikler vermekte ve birden fazla bir şekilde uygulanabilmektedir. Fazladan özellik verilen bir metnin hangi etiketinin önce hangisinin sonra kapandığı görüntüyü değiştirmediğinden etiketlerin kapama kısımlarının sıralanışının kodlanmasına ve böylece veri saklamaya imkan tanımaktadır. Örneğin dört farklı özellik vermek istediğimiz bir metnin etiketleri ile 4! kadar faklı birer rakam saklayabiliriz. Etiketlerin normal kapanış sırası 1'i ifade ederse, diğer farklı kapanışlar farklı sayıları ifade edecektir.

Bir metnin boşlukları bile veri gizlemek için uygundur. Örneğin bir metinde kelimeler arasındaki bir boşluk veya iki boşluk bulunması durumu kodlanabilir ve buda verimizi gizlememize imkan sağlar [34].

Steganografinin resim uygulamaları aslında ülkemizde oldukça yaygın bir araştırma konusudur. Hatta gri seviyeli resimlere veri gizleyen bir yazılım [27] ve renkli Bitmap formatındaki resimlere veri saklama hizmeti sağlayan bir web sitesi bile mevcuttur [35].

1996 yıllında Bender ve arkadaşlarının kaleme aldığı "Techniques for data hiding" adlı makalede metin, ses, resim gibi birçok dosya türüne veri saklama teknikleri detaylı bir şekilde açıklanmıştır. 2000'li yıllardan sonra ise resim içine veri gizleme teknikleri LSB [27,28], Bit Plane Complexity Segmentation (BPCS) [26], dönüştürme [36,30] ve permütasyon teknikleriyle [37] daha da geliştirilmiştir.

2.5.1. Steganografi tarihteki örnekleri

Mesajı taşıyacak olanın başına dövme ile yazılması.

Bal mumu ile kaplanmış tabletlere mesajın gizlenmesi.

Görünmeyen Mürekkepler.

2. Dünya Savaşında Alman bir casus tarafından gönderilen mesaj [3]; "Apparently neutral's protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by-product, ejecting suets and

vegetable oils.”. Her kelimedeki 2. Harfi birleştirsek: “**Pershing sails from NY June 1”, “Füzenin Newyork'tan Haziran 1'de denize açılacağını.”** [38].

2.5.2. Steganografinin farklı kullanımları

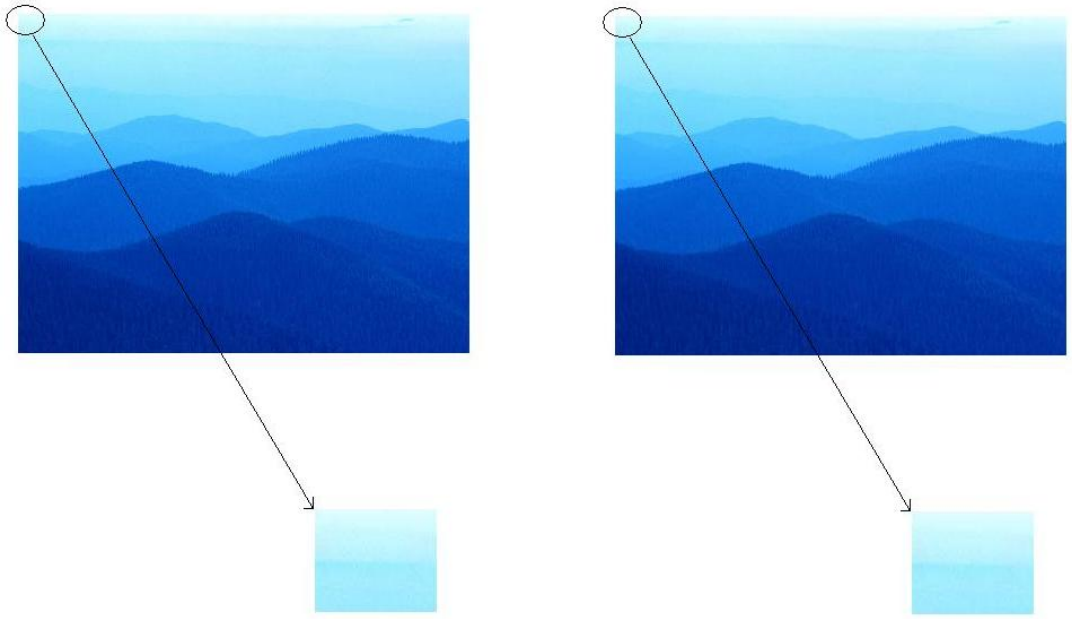
Digital Steganografi: Mesajı gizlemek için kullanılır.

Digital watermarking: Copyright, sahiplik ve lisans bilgilerini gömmek için kullanılır.

Digital fingerprinting: Verinin yasadışı dağıtımını izlemek için kullanılır.

2.5.3. Kullanılan LSB yönteminin özellikleri

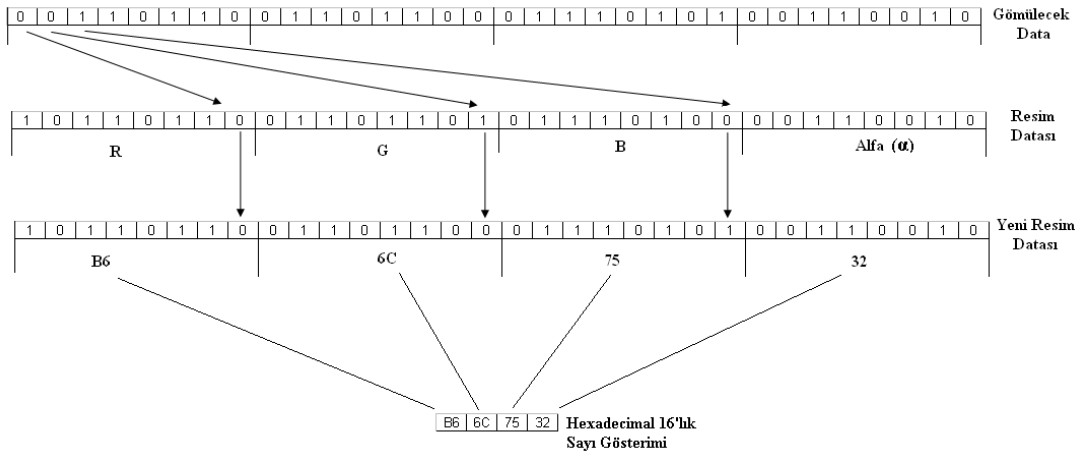
LSB metodu ingilizce least significant bit kelimelerinin baş harflerinden meydana gelen ve 8’er bitlik renk bileşenlerinin en düşük bitlerinde oynamaya yaparak gizlenecek olan veriyi bu bitlerle yer değiştiren metottur. Bu metotta şayet RGB bileşenlerinin sadece en düşük biti kullanılırsa resimdeki değişim miktarı çok az olacağı ve potansiyel saldıran kişinin bu değişimi fark edemeyeceği Şekil 2.12.’de açıkça görülmektedir.



Şekil 2.12. Piksellerin Değişimi.

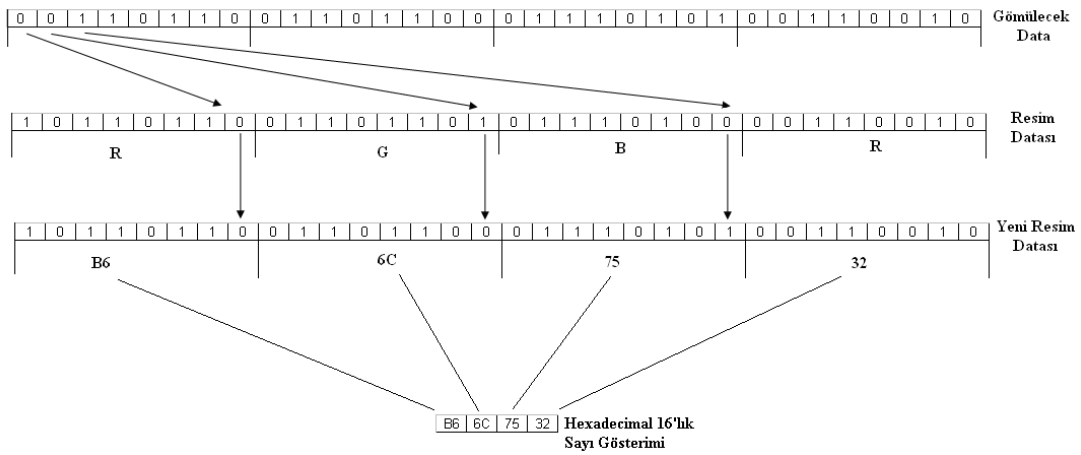
Kullanılan LSB metodunun daha güvenli olabilmesi için üzerinde bazı değişiklikler yapılmıştır. Resimlerimize gömülecek olan veriler 32 bitlik ham veridir ve bu veriler 32 bitlik adres yoluna sahip hafızada tutulmuştur. Resim verisinin bulunduğu her 11 adrese gizlenmek istenen verinin 32 biti gömülmüştür. Resim verisinin bulunduğu her adres bloğunda ilk 8 bitlik veri bloğu hariç geri kalan kısmına 3'er bit veri gömülerek 32 bitlik veri saklanır. Burada önemli olan husus 11 adrese toplam 32 bit veri her adrese 3'er bit düşecek şekilde gömülmek planlanırsa 1 bitlik bir boş alan kalmaktadır. İşte bu bir bitlik son adres satırındaki boşluğa gömülmek istenen veri için üretilen değerlik biti gömülmüştür. Örneğin ikilik tabanda bir sayı şayet tek sayıda lojik olarak '1''e sahipse o sayı için üretilecek değerlik biti lojik olarak tek bitlik '1''dir, bu sayı şayet çift sayıda '1''e sahipse bu sayı için üretilecek değerlik biti lojik olarak tek bitlik '0''dır. Gömülen bu değerlik biti verinin doğru bir şekilde gömülüp gömülmediği konusunda da bize bilgi vermektedir. Ayrıca her resmi içinde veri olup olmadığına dair analiz etmektense veri gömülecek resimlerin tutulduğu ilk 11 adres aynı şekilde son bitleri sıfırlanarak işaretlenmiştir. Böylelikle alınan resimlerin ilk 11 adresine bakılarak içinde veri olup olmadığı anlaşılarak işlem yapılabilmektedir.

Şayet resimler 32 bitlik renk modeline göre bilgisayar hafızasında tutulmaktaysa verilerin değişimi Şekil 2.13 de gösterildiği gibi olmaktadır.



Şekil 2.13. RGB ve Alfa Kanalları.

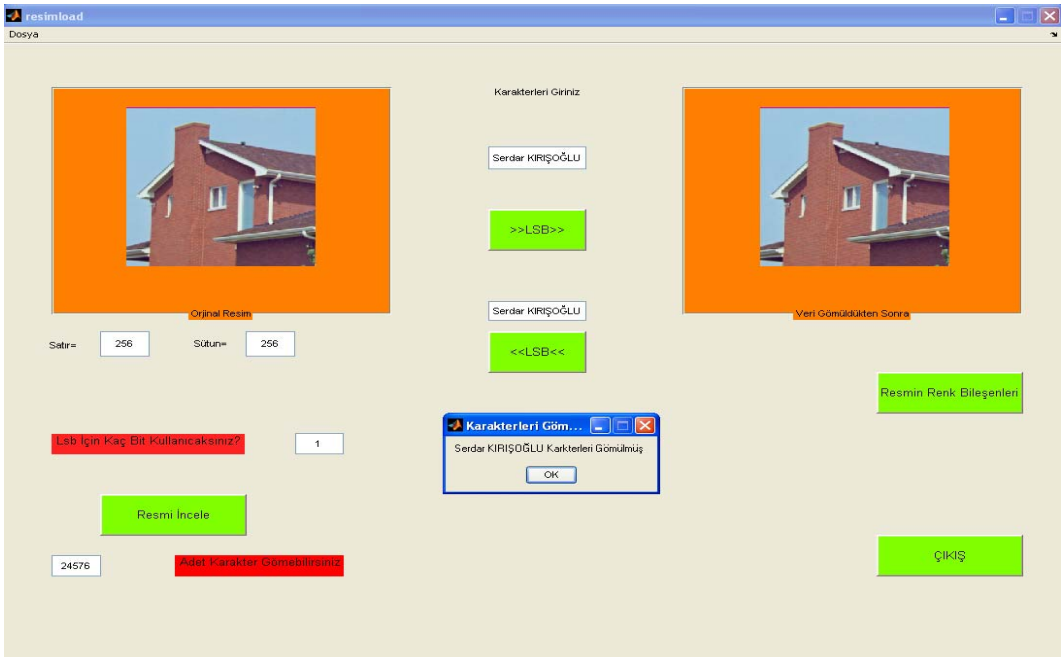
Eğer kullanıcı α (Resmin saydamlık derecesinin tutulduğu son sekiz bit.) kanalını kullanmıyorsa bu seferde Şekil 2.14'deki gibi R-G-B-R modlu 32 bitlik veri içinde resim verisinin o pikseline ait ve o adresteki verinin sadece ilk 8 bitlik bloğunda değişime gidilmemiş olacaktır ki buda zaten asıl LSB'nin gereğidir. Yani LSB metodu kullanılırken her pikselin her bileşenine veri gömmek şart değildir ve aslında güvenlik için istenen durumlardan biride budur.



Şekil 2.14. RGBR Olarak Resim Verisinin Veri Yolundaki Hali.

Şekil 2.12'yi elde edebilmek bu algoritmayı kullanan ve Matlabın grafiksel kullanıcı ara yüzü olan GUI'de Şekil 2.15'de gösterilen bir ara yüz tasarlanmıştır.

2.6. Matlab GUI Simülasyonu



Şekil 2.15. Matlab GUI Ara Yüzü.

Ara yüz öncelikle iki yönlü çalışabilmektedir. Kullanıcı tarafından seçilen bir resmi birinci bölmeye getirdikten sonra <<LSB<< tuşuyla resimde veri olup olmadığına bakmakta ve veri varsa bunu mesaj penceresinde göstermektedir. Bu işlem veriyi geri almakta kullanılmaktadır. Aynı şekilde veriyi gömmek için kullanıcı bir resim seçmekte ve >>LSB>> butonunun üzerindeki text kutucuğuna gizlemek istediği mesajı girdikten sonra >>LSB>> butonuna basarsa veriyi gömmüş olacak ve Stego'yu sağ taraftaki resim bölmesinde görüntüleyecektir. Bu iki bölmede birincisi resmin orijinali ikincisi ise resmin gömüldükten sonraki hali olacağı için kullanıcıya resmin ne kadar değişikliğe uğradığını görmesi amaçlı bir Resim İncele butonuyla resimleri inceleme imkanı sunulmuştur. Ayrıca resimlerin renk bileşenlerini de

inceleyebilmeniz için bir buton tahsis edilmiştir. Programın algoritması ise şu şekildedir.

Veriyi gömerken;

1. Taşıyıcı resmin ilk 11 pikselinin RGB değerlerinin son bitlerini sıfırla (Bu resimde veri olduğunun belirtilmesi için.)

2. Taşıyıcı resmin sonraki 11 pikseline, gömülecek olan mesajın boyutu gömülecek(karakter sayısı olarak, örneğin 3 karakter ise ikilik tabanda '011'). RGB bileşenlerinin son bitlerine gömüleceği ve 11 adet piksel bu işlem için kullanılacağından, 2^32 karakter kadar boyut gömülebilinir.

3. Her bir karakteri ASCII kodlar halinde tek tek her piksele 3'er adet düşecek şekilde bitene kadar yerleştir.

Veriyi ayrıştırırken;

1. Resmin ilk 11 pikselinin RGB değerlerinin son bitlerine bak eğer sıfır değilse dur, sıfırsa devam et.

2. Resmin sonraki 11 pikselinin RGB bileşenlerinin son bitlerinden gömülmüş mesajın boyutunu öğren.

3. Öğrendiğin mesaj boyutu kadar pikselleri okumaya ve mesajı geri almaya devam et.

Bu ara yüzün kodları ekte verilmiştir.

2.7. Diğer Çalışmalar

Resim içine veri gömme metotlarının geliştirilmesine yönelik yapılan çalışmalardan birisi Sedat AKLEYLEK ve Urfat NURİYEV'in "Steganografi ve Steganografinin Yeni Bir Uygulaması." başlıklı çalışmalarıdır [39]. Bu çalışmada yine aynı şekilde veriler LSB metodu ile gizlenerek yollanmakta fakat veriler gizlenmeden önce AS knapsack denilen bir şifreleme yöntemi ile şifrelenmektedir. AS knapsack verilerin bir dizi işlemden geçirilerek belirli bir algoritmaya göre şifrelenmesine dayanmaktadır, bu yöntemde şifreyi çözmek için birde şifrelenirken kullanılan anahtarı karşı stego ile yollamak gerekmektedir.

Lee, Y.K. ve Chen, L.H.'nin yapmış olduğu "High capacity image steganographic model." isimli çalışmalarında, gri seviyeli resimlerde LSB yöntemiyle ve anahtar kullanarak, piksel değerlerinin ilk dört bitlerin modifikasyonu ile veri saklamışlar ve %50 oranında kapasite yakalamışlardır [40-39].

2002 yılında Niimi, M.ve arkadaşlarının yaptığı "High capacity and secure digital steganography to palette-based images." isimli çalışmada resim Steganografi tekniklerinden biri olan BPCS yöntemini temel almışlar ve palet tabanlı resimler içine paletteki renk vektörlerinin sırasına bağlı olmayan bir metotla veri gömmüşlerdir [26].

Diğer bir çalışma ise Noda ve arkadaşları tarafından yapılan "Application of bitplane decomposition steganography to JPEG2000 encoded images." isimli çalışmada, kayıplı sıkıştırma ile oluşturulan resimlerde BPCS yöntemiyle yapılan bir veri saklama çalışmasıdır. Bu çalışmada sıkıştırma yapılırken wavelet katsayılarının niceleme işlemiyle bit düzlemine döndürülmüş hali üzerine BPCS yöntemiyle veri saklamışlar ve %9 ila % 15 arasında değişen başarı kaydetmişlerdir [36].

Sağıroğlu ve Tunçkanat'ın "A Secure Internet Communication Tool." isimli çalışmasında ise gri seviyeli Bitmap resimler üzerinde sıkıştırma çalışmaları yapmışlar ve en önemsiz 4 biti kullanarak resimde değişikliği fark ettirmeksizin veri saklamayı başaran LSB modifikasyonlu bir veri gizleme programı gerçekleştirmişlerdir [27].

Tseng ve Chang jpeg sıkıştırma işlemi esnasında daha fazla saklama kapasitesine sahip bir yöntem geliştirmişlerdir [30].

Brisbane ve arkadaşları ise "*High-capacity steganography using a shared colour palette*." isimli çalışmalarında palet tabanlı renkli resimlere veri gömme işlemini gerçekleştirmişler ve bu işlemi gerçekleştirirken şeffaflığa zarar vermeksizin yüksek veri saklama kapasitesine ulaşmışlardır [37].

Öcal ise taşıma ortamından bağımsız bilgi güvenliğini ve gizliliğini sağlamak amacıyla şifreleme çalışması yapmıştır. Bu çalışmada şifreleme ortamı olarak FPGA platformunu seçmişlerdir. Şifreleme standartlarına sadık kalınması için her türlü saldırıya karşı güvenliği test edilmiş Ulusal Standart ve Teknolojiler Enstitüsü'nün (NISTAmerika) kabul ettiği ileri şifreleme standardını (AES) tercih etmişlerdir. Şifreleme metodu olarak şifre geribesleme metodu (Cipher Feedback, CFB) ve

donanım elemanı olarak XILINX'in ürünü SPARTAN kullanılmıştır. Bu çalışmada FPGA platformunun seçilmesinin sebebi, donanımın yazılıma oranla daha yüksek güvenliğe sahip ve daha hızlı olması olarak gösterilmiştir.[4]

Mahmoud M.I ve arkadaşlarının yaptığı bizimkine benzer bir çalışmada ise; gizlenecek olan veri boyutunu arttırmak ve şifreleme işlemini bir adım daha öteye taşımak amaçlı Wavelet denilen bir dönüşüm içinde Achterbahn-128 adı verilen bir şifreleme algoritması kullanılmıştır. Gizlenecek olan verinin önce Achterbahn-128 algoritması kullanılarak şifrelenmiş hali elde edilmekte daha sonra bu şifreli verinin Wavelet katsayıları şeklinde taşıyıcı resme gömülmesi esas alınmıştır. Bu çalışmanın FPGA modülüne gömülmesiyle işlem daha hızlı ve uygulanabilir hale getirilmiştir [41].

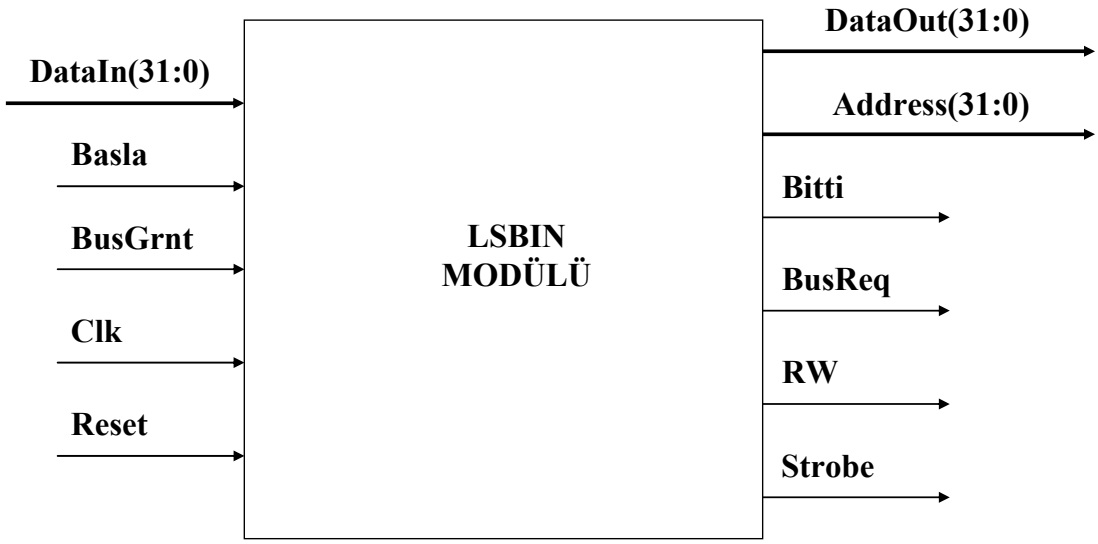
Maity S.P. ve arkadaşlarının yaptığı bir diğer çalışmada ise; gri skala resimler içine resim gizleme üzerine FPGA modülü tasarlamışlardır. Bu modül üzerinde bir novel algoritması çalıştıracak şekilde dizayn edilmiş ve uygun FPGA çipine uygulanmıştır. Bu çalışmada taşıyıcı olarak kullanılan resimdeki değişiklikleri azaltmak için kanal kodlama ve uzaysal iki faz modülasyon tekniğini kullanmışlardır. Bu çalışmada FPGA kullanılmasının nedenini ise, gerçek zamanlı multimedya veri transferi kullanan uygulamalarda kullanılabilecek kadar hızlı olması olarak açıklamışlardır. Ayrıca Maity ve arkadaşları FPGA çipine yerleştirdikleri VLSI dizaynının devresinin kameralar üzerine de yerleştirilebileceğini vurgulamışlardır.[42]

3. VERİ GÖMMEK VE AYRIŞTIRMAK İÇİN FPGA MODÜL TASARIMI

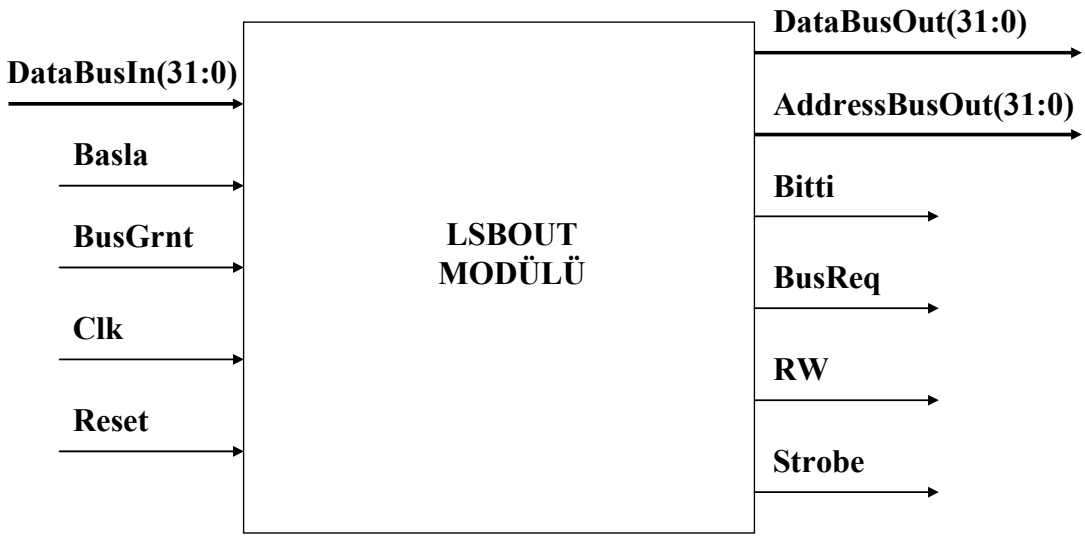
Bu çalışmada Steganografi sanatının Least Significant Bit(LSB) metodunu kullanarak sayısal görüntüye veri gömmek ve ayrıştırmak için Field Programmable Gate Array (FPGA) tabanlı iki ayrı modül tasarlanmıştır. Şekil 3.1'de veri gömen modülün en üst seviye blok şeması görülmektedir. Veri gömmek için tasarlanan *LSBIN* modülümüz *Controller* ve *DataPath* olmak üzere iki ayrı alt modülden oluşmaktadır. *LSBIn* modülümüz 32 bitlik bir *DataIn* girişine ve birer bitlik *Basla*, *BusGrnt*, *Clk* ve *Reset* girişlerine sahiptir. Yine aynı modülümüz çıkış olarak 32 bitlik *Adress*, *DataOut* ve birer bitlik *Bitti*, *BusReq*, *RW*, *Strobe* olmak üzere altı çıkışa sahiptir.

Şekil 3.2'de veri ayrıştıran modülün en üst seviye blok şeması görülmektedir. Veri ayrıştırmak için tasarlanan modülümüz yine aynı şekilde *Controller* ve *DataPath* olmak üzere iki alt modüle sahiptir. Bu modülümüz 32 bitlik bir *DataBusIn* ve birer bitlik *Basla*, *BusGrnt*, *Clk* ve *Reset* girişlerine sahiptir. Veri ayrıştıran modülümüz çıkış olarak 32 bitlik *AdressBusOut* ve *DataBusOut* ve birer bitlik *Bitti*, *BusReq*, *RW*, *Strobe* olmak üzere altı çıkışa sahiptir. *Reset*, *Basla ve Bitti* sinyalleri modüllerin zamanlaması ve modül ile modülün bağlı olduğu bilgisayar arasında senkronizasyonu (hand-shaking) sağlamak için kullanılır.

Bu iki modül, bir donanım tanımlama dili olan VHDL'de tasarlanmış ve Xilinks'in ISE 9.2 yazılımı kullanılarak değişik FPGA çiplerine göre sentezlenmiştir.

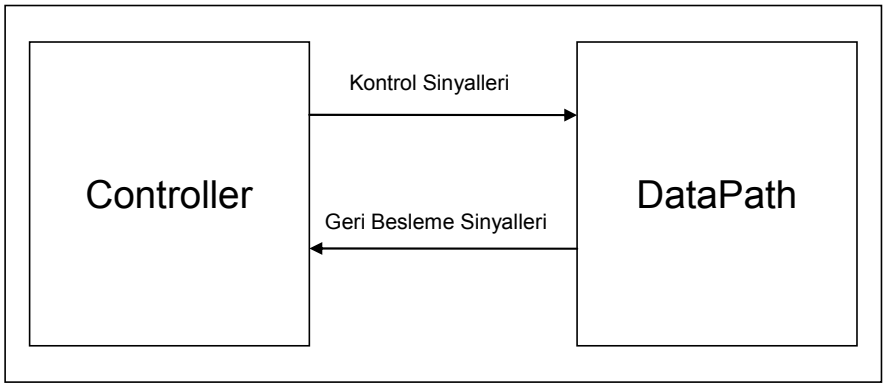


Şekil 3.1.*LSBIN* Modülü.



Şekil 3.2.*LSBOUT* Modülü.

Şekil 3.3'de Her iki modül içinde genel bir ikinci seviye blok diyagramı görülen modüllerimiz kontrol ünitesi ve veri yolundan gelen verileri kontrol ünitesinden gelen sinyallere göre işlemek ve sonuçları yine bu kontrol sinyallerine göre istenilen adreslere yazmayı amaçlayan birer modülden oluşmaktadır.

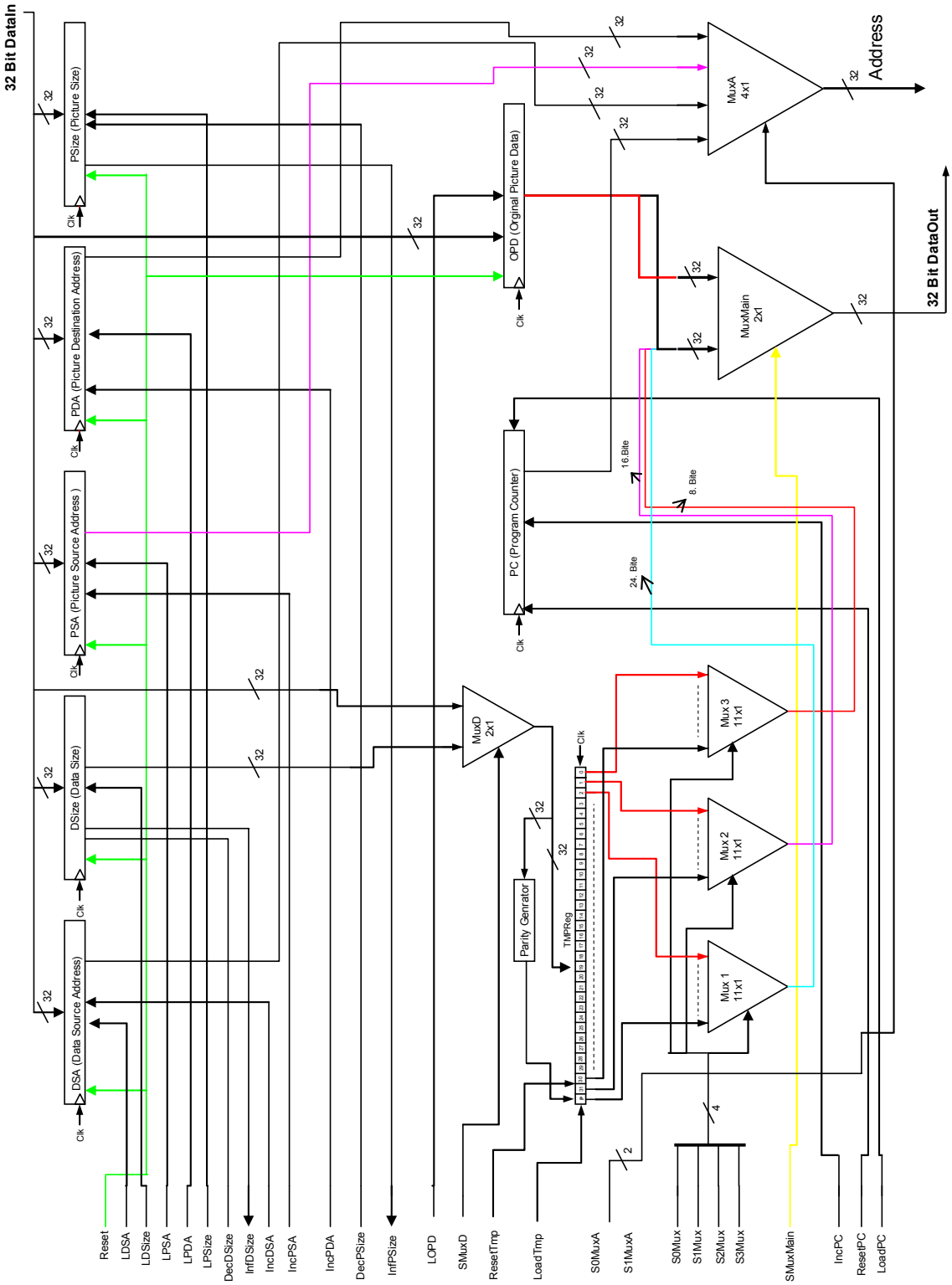


Şekil 3.3. Modüllerin Ortak Genel İkinci Seviye Blok Diyagramı.

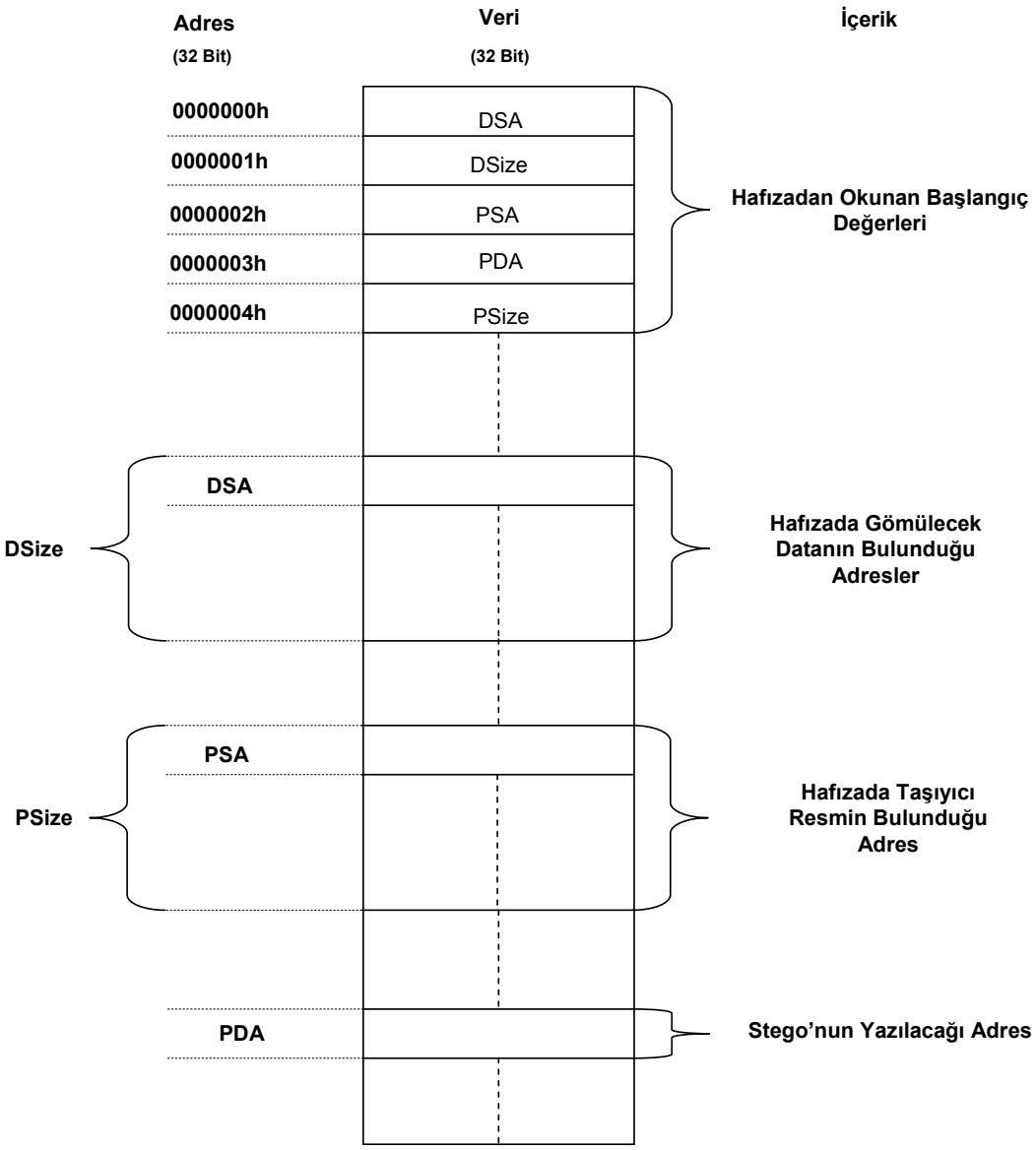
3.1. Veri Gömme Modülü DataPath ve Controller Blokları

3.1.1 DataPath bloğu

Şekil 3.4'te ayrıntılı blok diyagramı gösterilen `DataPath` bloğu birçok özel amaçlı elemandan oluşmuştur ve bunlar `Controller` bloğunun sinyalleri sayesinde kontrol edilirler.



Şekil 3.4. *LSBIN* Modülü DataPath Bloğu Şeması.



Şekil 3.5. *LSBIN* Modülü Hafıza Haritası.

Şekil 3.5'te ise veri gömme modülünün kullandığı hafıza haritası görülmektedir. Bu haritada ilk adresten itibaren gerekli bilgiler `DataPath` modülü içindeki tanımlanmış kayıtçılara alınır. Bu kayıtçıların isimleri de özel olarak tanımlanmıştır. Bu kayıtçılardan Data Source Adress (`DSA`) kayıtçısı gömülecek olan ham verinin hafızada bulunduğu yerin ilk adresini vermektedir. Bu kayıtçı arttırılabilir bir kayıtçı olarak tanımlanmıştır. Çünkü her bir gömülecek veriden sonra diğer gömülecek veriyi işaretlemek için adresi bir arttırmak gerekmektedir.

Diğer bir kayıtçı ise Data Size (`DSize`) için tanımlanmış kayıtçıdır. Bu kayıtçı, gömülmesi planlanan verinin kaç adres bölgesinde tutulduğunu bize

vermektedir. Hafızadan bu bilgi alınır ve her bir adresteki veri gömüldükten sonra içeriği bir azaltılır. İçeriği sıfır oluncaya kadar gömme işlemi devam eder.

Picture Source Address (*PSA*) kayıtçısı, taşıyıcı olarak kullanılacak resim verisinin hafızada tutulduğu yerin ilk adresini bize vermektedir. Bu kayıtçıda özel olarak arttırılabilir kayıtçılardan biridir.

Picture Destination Address (*PDA*) kayıtçısı taşıyıcı resme gizlenecek olan veri gömüldükten sonra kaydedilmeye hangi adresten başlanacağı bilgisi tutulmaktadır. İçeriği arttırılabilir bir kayıtçıdır.

Picture Size (*PSize*) kayıtçısı ise resim verisinin hafızada kaç adreste tutulduğunu belirtmek için tanımlanmıştır ve içeriği azaltılabilir bir kayıtçıdır. İçeriği sıfır olduğunda resmin bittiği belirlenebilir

Tablo 3.1. Kayıtçıların Temel Özellikleri.

Kayıtçı Adı	**Arttırılabilir**	**Azaltılabilir**	**İçeriğine Göre Değişen Çıkış Sinyali**
DSA	+	-	-
DSize	-	+	**InfDSize**
PSA	+	-	-
PDA	+	-	-
PSize	-	+	**InfPSize**

Tablo 3.1'de gösterildiği gibi bazı kayıtçılar kullanım amaçları dolayısıyla içeriği arttırılabilir veya azaltılabilir olarak tanımlanmıştır. Gerekli görüldüğü durumlarda içeriğin artıp azalabilmesi için dışarıdan arttır ya da azalt sinyali uygulanmıştır. Bu sinyaller *Controller'*dan gelen sinyallerdir. Kayıtçıların içeriklerinin kontrolü için *Controller'*a bazı sinyallerle içeriklerinin durumu yollanmıştır.

Bu kayıtçıların yanı sıra Temporary Register (*TMPReg*), Orginal Picture Data (*OPD*) ve Program Counter (*PC*) kayıtçıları kullanılmıştır. Bu kayıtçılardan *TMPReg* gömülecek olan veri bir seferde resmin bir verisine gömülemediği, resmin 11 piksel değeri için *DataPath*'de tutulması gerektiğinden dolayı kullanılmıştır.

PC kayıtçısı; genel amaçlı bir hafıza adres kontrol kayıtçısıdır ve sadece program başlayıp başlangıç bilgileri hafızadan okunana kadar kullanılan ve gerektiğinde birer birer arttırılabilen bir kayıtçıdır. Hafızadan gerekli bilgiler alındıktan sonra bu kayıtçıya ihtiyaç duyulmamaktadır.

OPD kayıtçısının tanımlanma sebebi; şayet bir veri gömme işlemi biterde resim verisi bitmez ise yeni oluşturulacak resim eksik kalacağı için bu eksik kalan kısımları resim verilerini hafızadan okuyup hiç değişiklik yapmadan olduğu gibi hedef adrese yazmaya devam etmek için bir tampon kayıtçı gibi kullanılmaktır.

Bunların yanı sıra gerekli iç bağlantılar adres ve hafızaya yazılacak verilerin seçimleri için uygun sayıda giriş ve çıkışlara sahip *Controller*'dan gelen sinyallere göre hareket eden çoğullayıcılar (Multiplexer *Mux*)'lar kullanılmıştır.

Tasarımımızda, gömülmesi planlanan veri için eşlik biti üreten ve bu bilgiyi de resme gömmemizi sağlayan bir Parity Generator (*PGen*) modülü tanımlanmıştır. Bu modül sayesinde veri ayrıştırılırken ayrıştırılan orijinal verinin doğruluğu kontrol edilmiş olup daha güvenli bir şekilde geri alma işlemi uygulanmış olmaktadır.

Ayrıca her kayıtçının içeriğinin sıfırlanabilmesi ve girişindeki verinin içeriğine eklenebilmesi için kullanılmış olan *Load* ve *Reset* sinyalleri mevcuttur.

Clk sinyali; kayıtçıların ihtiyaç duydukları saat sinyalidir ve her kayıtçı için ortaktır.

MuxA ise *PDA*, *PSA*, *DSA*, *PC* içindeki verileri kullanarak bunlardan bir tanesini hafıza adresi olarak belirlemek için kullanmaktır.

MuxD'nin görevi; *DSize* yada o an *DataIn* de hangi veri varsa o veriyi *TMPReg*'e ve *PGen*'e yüklemektir.

MuxMain'nin görevi; *OPD* yada *Mux1*, *Mux2*, *Mux3*'ten gelen verilerin uygun bir şekilde *DataOut*'a yönlendirilmesini sağlamaktır.

Mux1, *Mux2*, *Mux3* çoğullayıcılarının görevi; *TMPReg*'den ve *PGen*'den gelen verileri uygun sırası ile *MuxMain*'e yönlendirmektir.

Mux'ların nasıl davranacağı *Controller*'den gelen kontrol sinyalleri vasıtası ile yapılmaktadır.

3.1.2. Controller bloğu

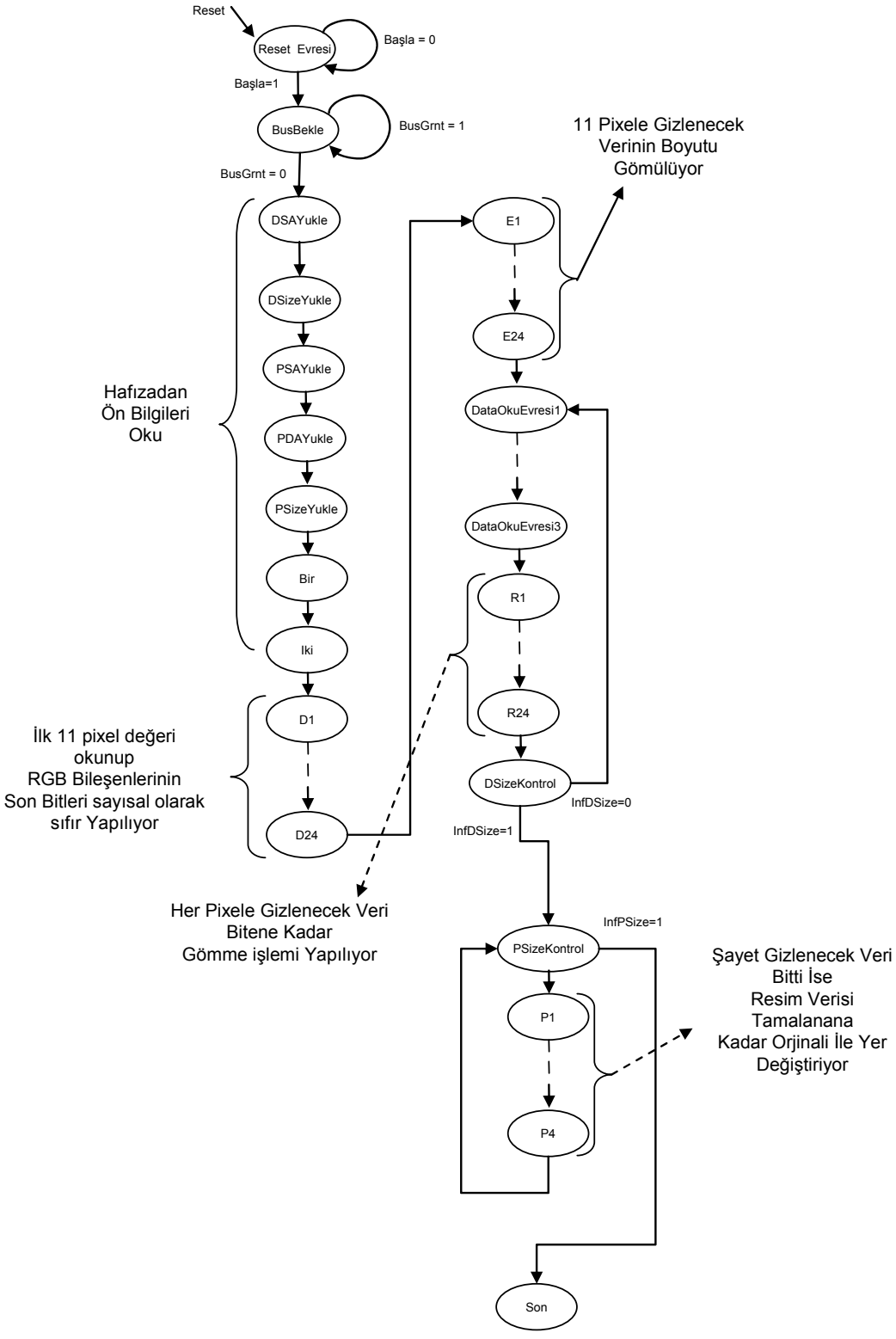
Bu blok Şekil 3.6'da *durum* akış diyagramı gösterildiği gibi toplam 111 *durum*'dan (state) oluşan bir Sonlu Durum Makinesi (Finite State Machine-FSM) olarak tasarlanmıştır.

Öncelikle sistem *Reset* evresinde iken gerekli kayıtçılar ve sistemin durumu ilk hallerine getirilerek işe başlamaya hazır durumda bekletilmektedir. *Başla* sinyali *Controller*'a verilmediği sürece bu evrede sistem bizden başlama sinyalini bekleyecektir. *Reset* evresine tekrar dönebilinmesi için *Reset* sinyalinin lojik olarak '1' yapılması yeterli olacaktır.

Başla sinyali kullanıcı tarafından '1' yapıldıktan sonra işlemler başlar ve evre bir sonraki aşama olan *BusBekle* evresine geçer.

Bu evrede çipin hafızaya erişebilmesi ve hafıza veri yolunun çipin kullanımına tahsis edilebilmesi için hafızaya *BusReq* (veri yolunu talep) sinyali yollanır. *BusReq* sinyali yollandıktan sonra *BusGrnt* sinyalinin '0' olması beklenir ki hafızanın bizim talebimiz sonucu bize tahsis edildiği anlaşılsın.

Bu evreden bir sonraki evreye ancak *BusGrnt* sinyalinin '0' olması ile geçilebilir.



Şekil 3.6. Veri Gömme Modülünün Akış Diyagramı.

Bir sonraki evre `DSAYukle` evresidir. Bu evrede hafızadan `DSA` bilgisinin okunabilmesi için gerekli sinyaller `DataPath` modülüne yollanır.

Bu gerekli bilgilerin hafızadan okunma işlemi *İki* isimli evreye kadar devam etmektedir.

Sonraki *D1'den D24'e* kadar olan durumlar; resme veri gömüldüğünü karşı tarafın anlaması için Resmin ilk 11 pikseline ait *RGB* bileşenlerinin son bitlerinin değerleri '0' yapılması için geçen durumlardır.

Bu durumlardan sonra resme gizlenecek olan mesajın toplam olarak kaç adreste tutulduğunu karşı tarafın anlaması için sonraki gelen 11 piksele *DSize'da* tutulan 32 bitlik veri gömülür. Bu adres boyutunun gömülme işlemi için *E1'den E24* e kadar olan durumlar geçilmektedir.

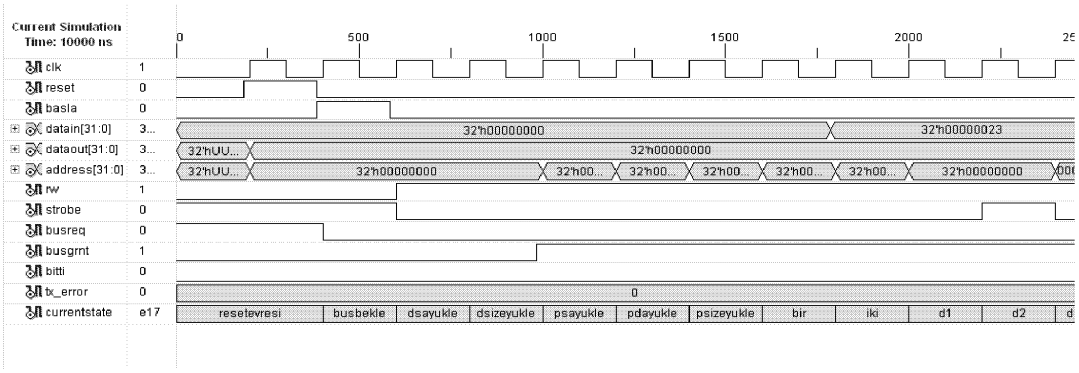
Bir *DataOkuEvresi1*-3 durumlarında gizlenecek olan mesajın başlangıç adresinden ilk verisini okuma işlemleri yapılmaktadır.

Gizlenecek olan verinin ilk verisi okunduktan sonra resim verisinden sırayla 11 piksele ait değerler okunarak her okunan 32 bitlik değerin 8-16-24 numaralı bitleri ile mesajın sırayla 3'er bitlik verileri ile yer değiştirilerek gömme işlemi yapılır. Bu işlem *R1'den R24'e* kadar olan durumlarda gerçekleştirilmektedir. Bu durumlar Gizlenecek olan veri bitene kadar döngüsel olarak devam etmektedir.

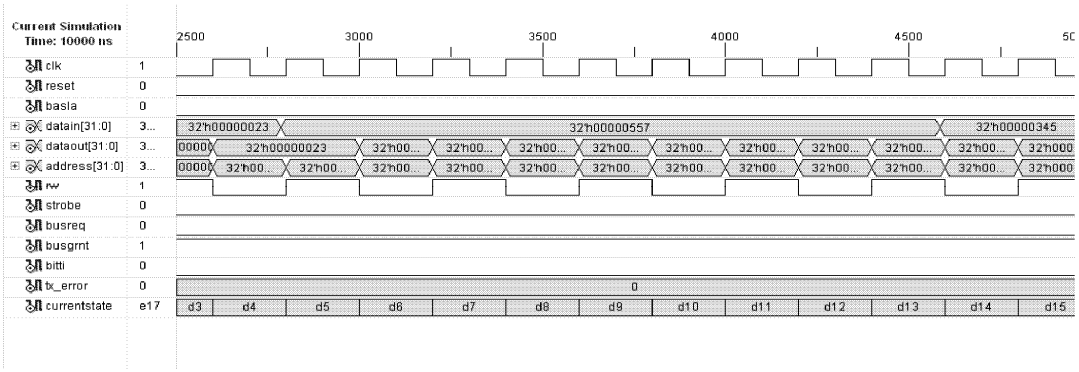
Sonraki durumda gizlenecek olan veri bitmişse; resim verisinin bitip bitmediğine bakılır. Şayet resim verisi de bitmişse *Son* duruma geçilir. Resim verisi bitmemişse yeni oluşturulmuş resim verisi orijinal resim verisi ile tamamlanarak işlem sona erdirilir. Resim verisinin orijinal resim verileri ile doldurulma işlemi *PSizeKontrol* ile *P4* durumları arasında gerçekleşmektedir.

Controller'a gelen *InfDSize* ve *InfPSize* geri besleme sinyalleri durumlar arasında geçiş yaparken kontrol amaçlı kullanılan sinyallerdir.

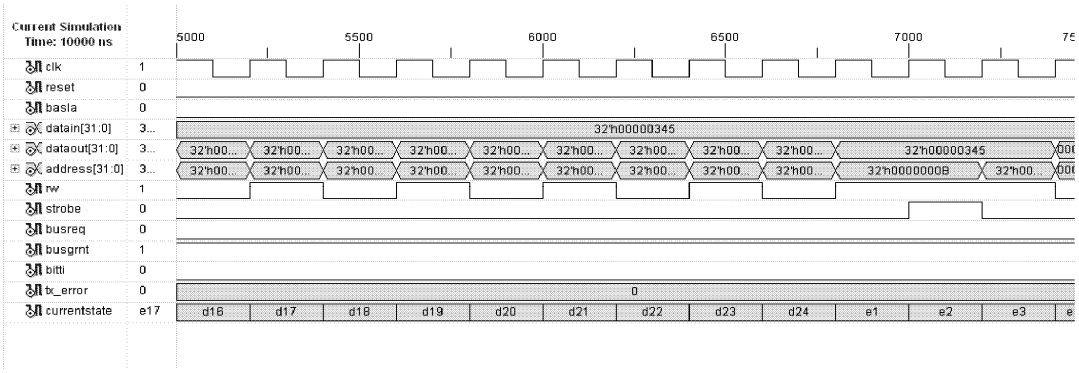
Şekil 3.7, Şekil 3.8 ve Şekil 3.9'da, modülümüzün gerçek verilerle elde edilmiş simülasyon sonuçlarının bir kısmı gösterilmiştir.



Şekil 3.7. Veri Gömme Modülü Simülasyon Sonuçları (a)



Şekil 3.8. Veri Gömme Modülü Simülasyon Sonuçları (b)



Şekil 3.9. Veri Gömme Modülü Simülasyon Sonuçları (c)

3.2. Veri Ayrıştırma Modülü DataPath ve Controller Blokları

3.2.1. DataPath bloğu

LSBOUT modülünün *DataPath* bloğunun ayrıntılı şeması Şekil 3.10'da detaylı bir şekilde gösterilmektedir. Bu blokta öncelikle *DataBusIn'*den gelecek olan 32 bitlik verinin 8-16-24. bitlerini alabileceğimiz 11 adet 3'er bitlik kayıtçılarımız mevcuttur. Bu kayıtçılar gömme işleminde resmin orijinal verisi ile yer değiştiren aslı mesajımızın bit mertebesinde ham halini sırasıyla geri almamız için kullanılmaktadır.

Bu 3'er bitlik 11 kayıtçımızın yüklenme sırasını, Controller'dan gelen sinyaller aracılığıyla yapan *KodCozucu* belirlemektedir. *KodCozucu* 4 bitlik bir girişi ve 11 adet çıkışı olan bir *Decoder* olarak çalışmaktadır. Bu 11 çıkış uygun kayıtçılara dağıtılmıştır.

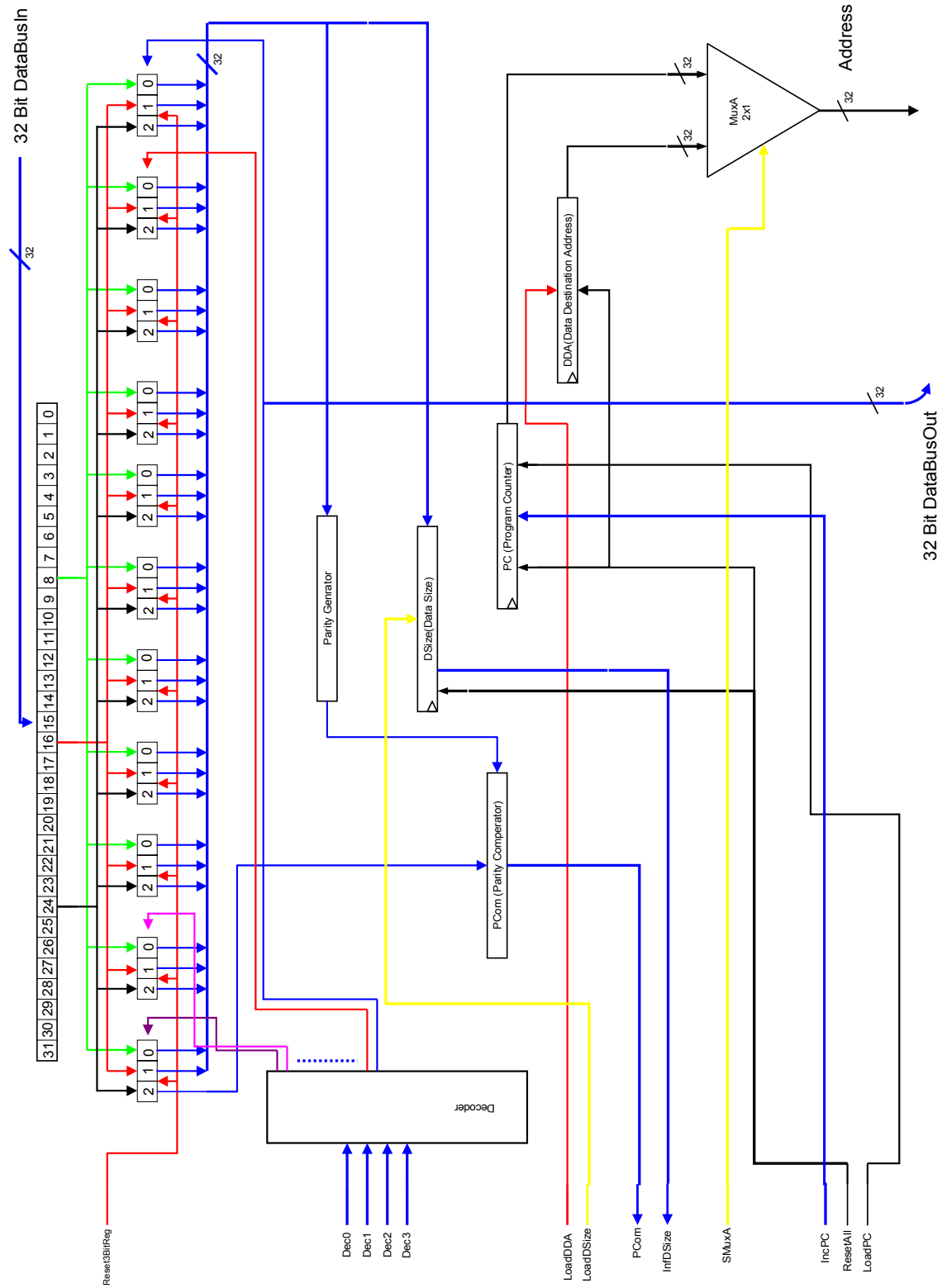
LSBOUT modülümüzün bir Parity Generator (*ParityGen*) modülü de mevcuttur. Bu modül girişindeki veri için eşlik biti üretmektedir.

Diğer bir modülümüz olan *PCom* modülü ise gömme esnasında üretilen değerlik biti ile ayrıştırma işlemi esnasındaki elde edilen veri için yeniden üretilen *ParityGen*'in ürettiği değerlik bitinin karşılaştırılması için tasarlanmıştır.

Data Destination Address (*DDA*) kayıtçımız, ayrıştırılacak verinin hedef bilgisayarda hangi hafıza bölgesine yazılacağı ile ilgili hafıza adres bilgisini tutmaktır.

Program Counter (*PC)* modülümüz ise işlemin akışı esnasında hangi adreste kalındığını hafızada tutmak için kullanılmaktadır.

Data Size (*DSize*) kayıtçımız ise gömülen verinin boyutunu geri aldığımızda bu bilgiyi saklayabilmek için tanımladığımız bir kayıtçıdır. Bu kayıtçının içeriği her bir ayrışan veri için bir azaltılır ve bu azaltma işlemi bu kayıtçının içeriği 0 olduğu zaman gömülmüş olan mesaj ayrıştırıldı anlamına gelene kadar devam eder. Bu kayıtçı içeriği '0' olduğunda Information Data Size *(InfDSize*) sinyalini 1 yapar. Bunun sebebi *Controller'*ın durumdan haberdar olması ve ayrıştırma işlemini bitirmesi içindir.



Şekil 3.10. *LSBOUT* Modülü DataPath Bloğu Şeması.

Modülümüzde bir adet 32 bitlik 2x1 *MuxA* çoğullayıcımız mevcuttur. Bu çoğullayıcımız hedef adres bilgisini seçmek için kullanılmaktadır. Girişinde *PC'nin* çıkışı yada *DDA*'nın çıkışı mevcuttur. Select Multiplexer A (*SMuxA*) sinyali vasıtasıyla bu girişlerden birini hedef adres olarak *AddressBusOut*'a yönlendirir.

3.2.2. Controller Bloğu

LSBOUT modülümüzün Controller bloğunun algoritmasının akış diyagramı Şekil 3.11'de gösterilmiştir. Veri ayrıştırma işlemi gerçekleştirilirken işlemin başlaması için gerekli olan hafıza adresi Data Destination Address *(DDA*) kullanıcı tarafından Şekil 3.12'de gösterildiği gibi hafıza bölgesinin ilk adresine yazılması gerekmektedir.

Şekil 3.11'de gösterilen akış şemasının ilk durumu *ResetEvresi*'dir. Bu evrede *DataPath* içinde kullanılan bütün kayıtçıların içeriği sıfırlanmıştır. Bu evreden ayrıştırma işleminin başlayacağı *BusBekle* evresine geçilebilmesi için kullanıcı tarafından Başla sinyalinin lojik olarak '1' yapılması gerekmektedir.

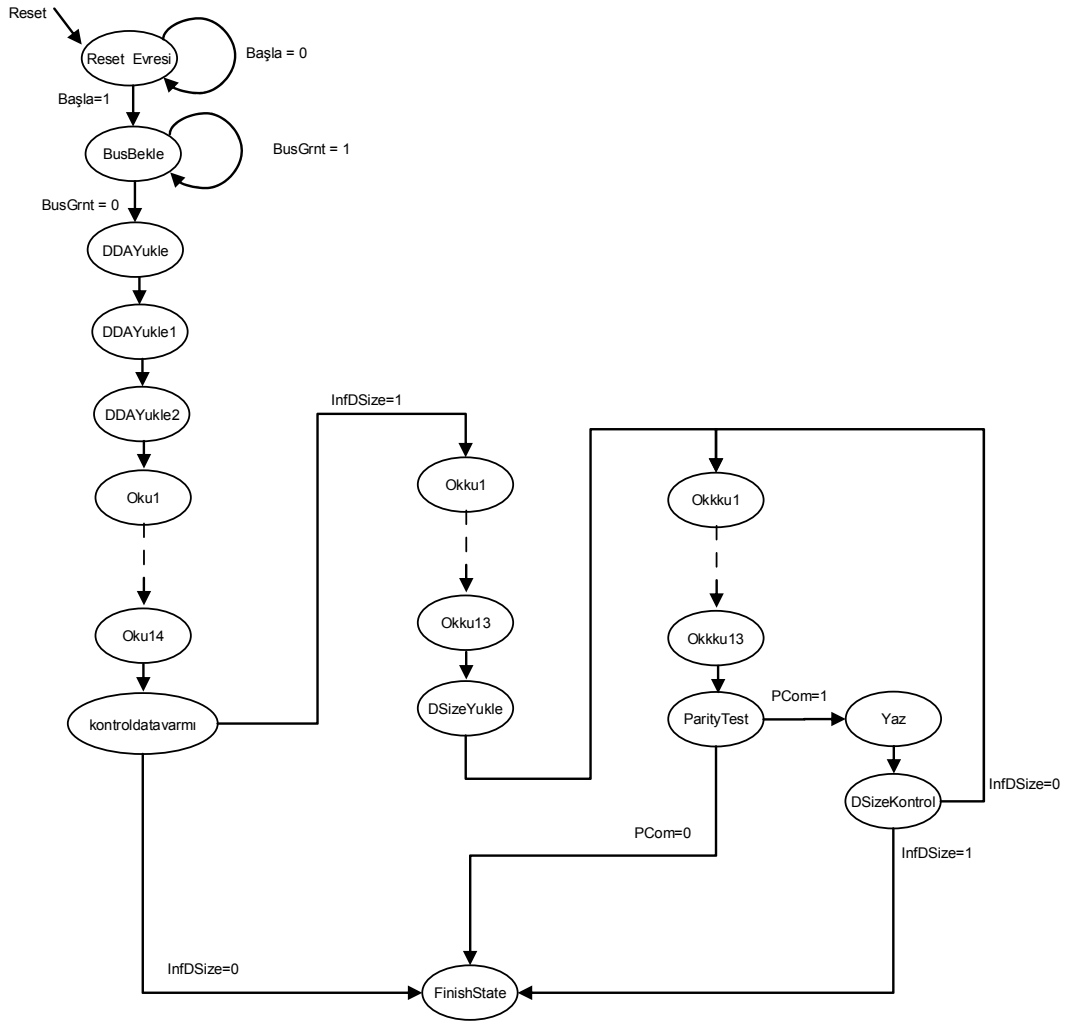
BusBekle evresinde çipin hafızaya erişebilesi ve hafıza veri yolunun çipin kullanımı için tahsis edilmesi için hafızaya *BusReq* (Veri yolunu talep) sinyali yollanır. *BusReq* sinyali yollandıktan sonra *BusGrnt* sinyalinin '0' olması beklenir ki hafızanın bizim talebimiz sonucu bize tahsis edildiği anlaşılsın.

Bu evreden bir sonraki evreye ancak *BusGrnt* sinyalinin '0' olması ile geçilebilir.

BusGrnt sinyali '0' olduktan sonra *DDAYukle* evresine geçilir. Bu evrede mesajın yazılacağı verinin bulunduğu ilk adres bölgesinden okuma işlemi yapılır. Bu işlem *DDAYukle2* evresine kadar devam eder. Okunacak bilgi *DDA* kayıtçısına yazılır.

Bu evreden sonra *Oku1*-*Oku14* evrelerinde, 11 adres bölgesinden resim bilgisinin içine gömülmüş olan veri ayrıştırılır. Ayrıştırılan veri *kontroldatavarmı* evresinde kontrol edilir. Ayrıştırılan 33 bitlik veri tamamen lojik olarak '0''lardan oluşuyorsa o resimde gizlenmiş mesaj var demektir. Bu kontrol evresinde *InfDSize* sinyalinin değerine bakılır, çünkü bu ilk 33 bitlik veri

DSize kayıtçısına yüklenmiştir. Şayet bu değer ‘1’ ise ayrıştırma işlemine *Okku1* evresi ile devam edilir.

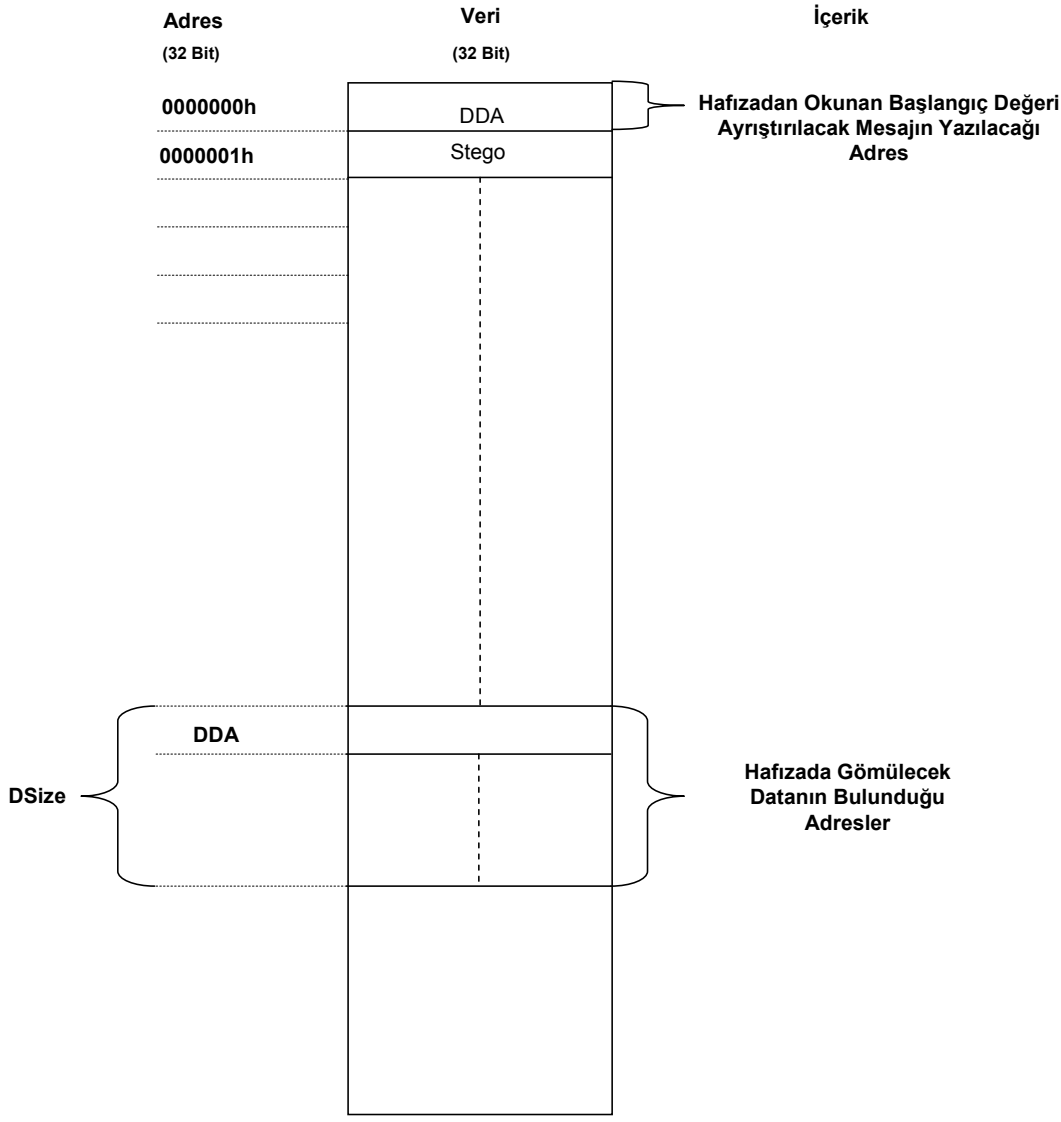


Şekil 3.11. Veri Ayrıştırma Modülünün Akış Diyagramı

Okku1 ile *Okku13* evrelerinde hafızadan gömülmüş olan mesajın boyut bilgisi beklenir ve *DSizeYukle* evresinde ise gömülmüş olan mesajın boyutu *DSize* kayıtçısına yüklenir.

Bu evrelerden sonra *Okkku1* ile *Okkku13* evrelerinde artık gömülmüş olan mesaj verilerinin okunması başlanmıştır. Her *Okku13* evresinden sonra *ParityTest* evrelerinde *PCom* değerine bakılır şayet bu değer ‘1’ ise *Yaz* evresine geçilir ve bu evrede hedef adrese ayrıştırılan mesaj verisi yazılır. Bu evrelerden sonra *DSizeKontrol* evresinde *InfDSize* sinyaline bakılır ve bu

sinyalin değeri ‘0’ ise *DSize* kayıtçısının içeriği ‘0’ değildir buda bize mesaj verisinin bitmediğini göstermektedir ve *Okkku1* evresinden algoritma devam eder. Bu döngü *DSize* kayıtçısının içeriği ‘0’ ve *InfDSize* sinyalinin değeri ‘1’ oluncaya kadar devam eder.

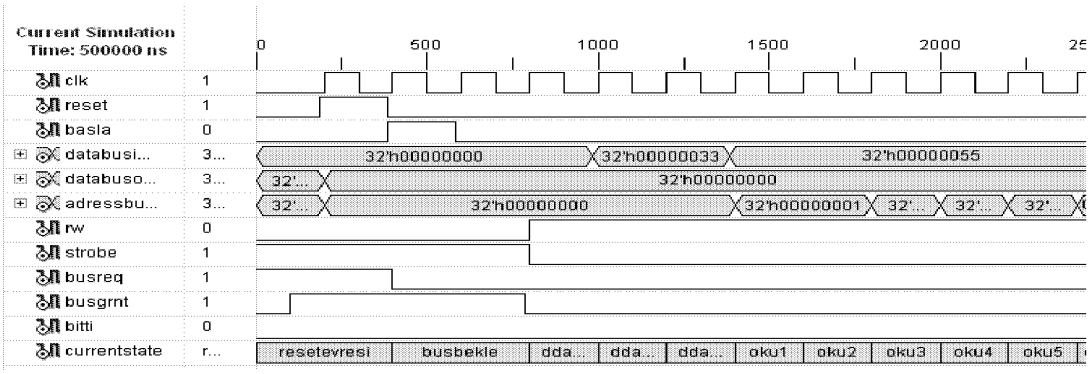


Şekil 3.12. Veri Ayrıştıran Modülün Hafıza Haritası

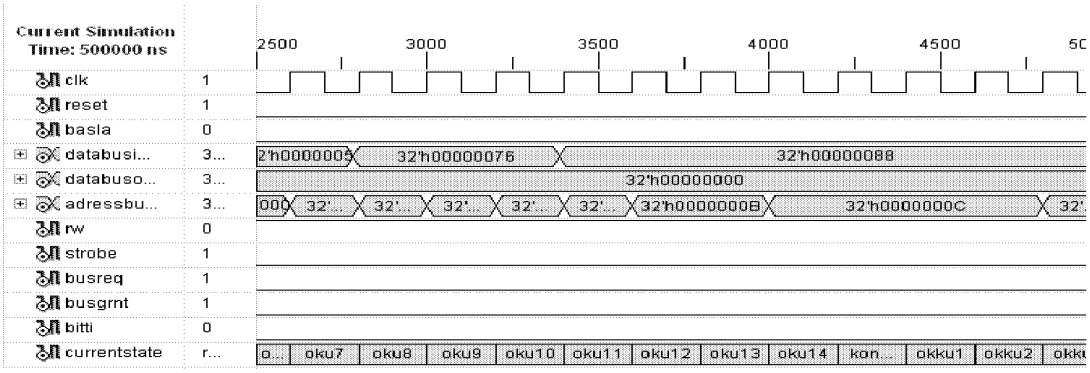
FinishState evresine üç durumda geçilebilir; İlk olarak resimde veri olmadığı anlaşılırsa, ikinci olarak ayrıştırılacak olan verinin bittiği tespit edilirse, son olarak ayrıştırılan mesaj verisi ile gömülen mesaj verisinin aynı olmadığı durumlarda bu evreye geçilebilir. Gömülmüş olan veri ile ayrıştırılan verinin aynı olmadığını verileri *ParityTest* evresinden geçirerek anlayabiliyoruz.

`FinishState` evresinde kullanıcıya `Bitti` sinyali gönderilir ve ayrıştırma işlemi biter.

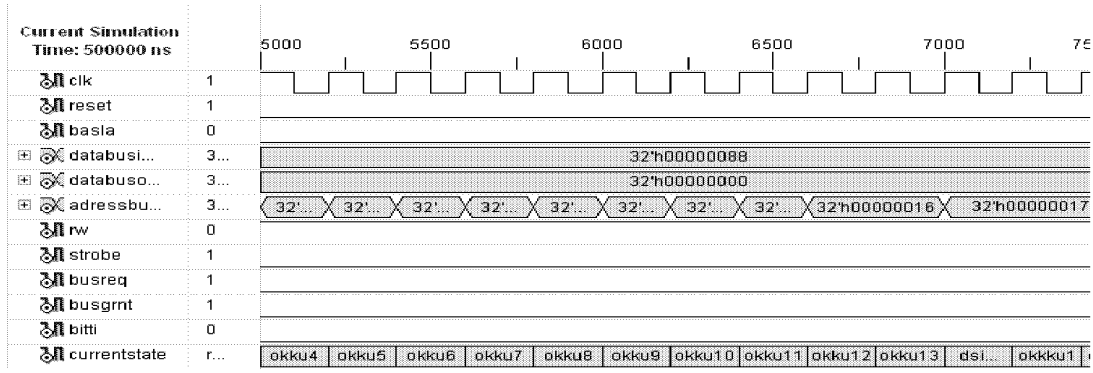
Ayrıştırma işlemine ait bazı simülasyon verileri Şekil 3.13-15'de gösterilmiştir.



Şekil 3.13. Veri Ayrıştırma Modülü Simülasyon Sonuçları (a)



Şekil 3.14. Veri Ayrıştırma Modülü Simülasyon Sonuçları (b)



Şekil 3.15. Veri Ayrıştırma Modülü Simülasyon Sonuçları (c)

4. MODÜLLERİN PERFORMANS SONUÇLARI

4.1 Veri Gömen Modülün Performans Bilgileri

Tasarlanmış olan Veri Gömme modülü, farklı model FPGA çipleri için sentezlenmiştir. FPGA çip istatistikleri ve modülün maksimum saat frekansları saptanmıştır. Modülün veri işleme süresinin anlaşılabilmesi için ISE programında uygun verilerle simülasyon [43] yapılmıştır.

Veri Gömme modülümüz Xilinx firmasının ürettiği *Virtex 5*, *Virtex 4* ve *Virtex 2* çipleri için ayrı ayrı sentezlenmiştir. Her çip için Tablo 4.1 de verilmiş olan istatistikler elde edilmiştir.

Tablo 4.1 Veri Gömme Modülünün FPGA Çip İstatistikleri.

FPGA Çip Türü	Slice Reg. Sayısı / %	LUTs Sayısı / %	Slice FFs Sayısı / %	Bonded IOBs Sayısı / %
Virtex 5	347/1	398/2	436/2	104/28
Virtex 4	279/4	463/3	348/2	104/43
Virtex 2	264/8	432/7	355/5	104/60
Spartan2	268/34	432/28	359/23	104/59

Sentez sonucu modüllerin *slice reg* ve *LUTs* sayıları göz önüne alındığında çiplerimize bu modüllerden çok sayıda sığabileceği görülmektedir. Bu nedenle bu modülümüz için en uygun çipin Spartan2 çipimizin olduğu açıkça görülmektedir.

Modülümüzün hız kazancının ölçülebilmesi için aynı algoritmaya sahip bir C++ programı Tablo 4.2'de konfigürasyon bilgileri verilmiş olan farklı bilgisayarlar üzerinde 100 hafızalık gizli mesaj verisinin gömülmesiyle ölçülmüştür. Elde edilen sonuçlar Tablo 4.3'de verilmiştir.

Tablo 4.2.Deneylerde Kullanılan Bilgisayarların Özellikleri .

PC Adı	CPU Hızı (GHz)	CPU Cache Bellek (KB)	RAM Hafıza Boyutu (MB)	Hafıza Tipi	CPU Türü	FSB BUS Hızı (MHz)	BUS Boyutu (Bit)
PC-1	1.73	2000	1024	DDR2	Intel Pent.	533	32
PC-2	2.00	2000	1536	DDR2	Intel C2Duo	667	64
PC-3	2.00	1000	2000	DDR2	AMDx2	667	64

Deneylerde farklı resim boyutları için sonuçlar elde edilmiştir. Her bir resme 100 adres boyutuna sahip bilgi gömülmüştür. Elde edilen sonuçlar Tablo 4.3'te gösterilmiştir.

Tablo 4.3.Bilgisayarların Farklı Boyutlarda Resimlere Veri Gömme Süreleri.

Eklenen Veri Boyutu	Görüntü Boyutu	PC-1 (μs)	PC-2 (μs)	PC-3 (μs)
100	**100x100**	537,612	188,615	170,296
100	**512x512**	663627,156	208988,160	180574,479
100	**1000x1000**	2567406,258	818609,900	705188,836

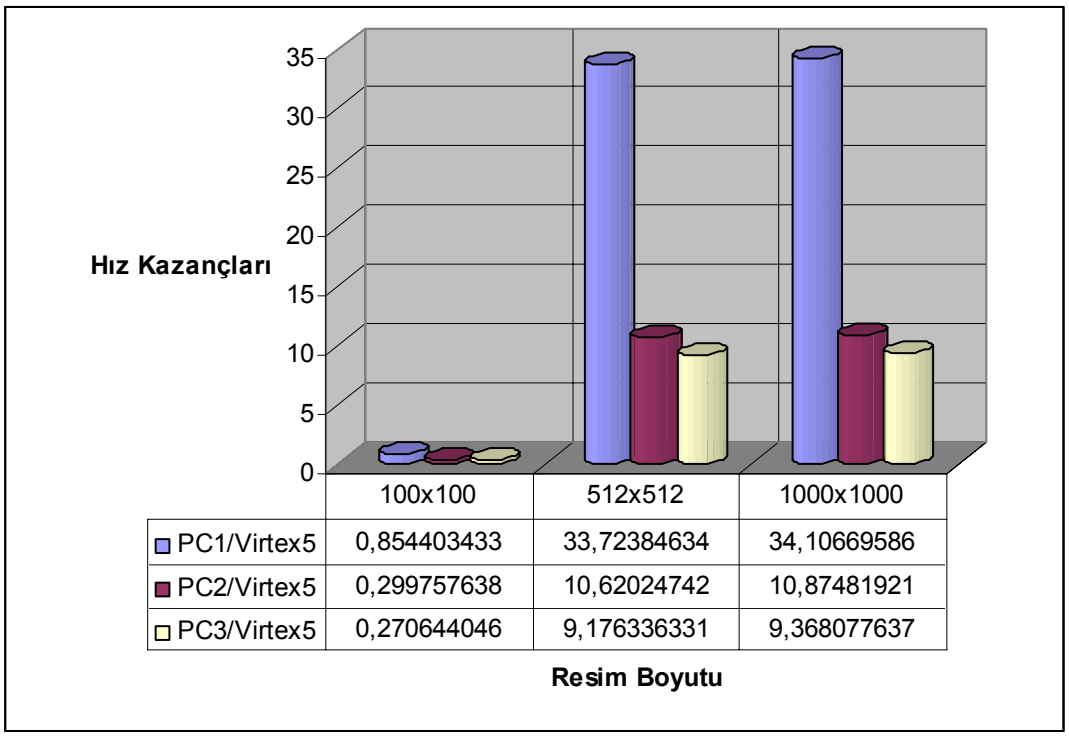
Veri gömme modülümüz ise Virtex 5 çipi için en çok $3{,}014 \times 10^{-9}$ saniyelik saat frekansında çalışabileceği sentezleme sonucu elde edilmiştir.

Aynı işlemin bizim algoritmamızla yapıldığında Tablo 4.4'de gösterildiği kadar durumdan geçilmesi gerekmektedir ve aynı tabloda bu durumlar için gerekli süreler verilmiştir.

Tablo 4.4.FPGA Çipinin Veri Gömme Süresi.

Eklenen Veri Boyutu	Görüntü Boyutu	Adım Sayısı	Süresi(μs)
100	**100x100**	225357	629,225
100	**512x512**	6528957	19678,276
100	**1000x1000**	24975357	75275,725

Bu süreler ve bu algoritma için özel olarak yazılmış olan yazılımımızın üretmiş olduğu süreler Şekil 4.1'de hız kazancı olarak karşılaştırılmıştır.



Şekil 4.1. Veri Gömme İşleminin Farklı Bilgisayarlara Göre Hız Kazançları.

Hız kazancı 1'in altındaki değerlerde yazılım donanımdan daha hızlıdır. 1'in üstündeki değerlerde ise donanımımız yazılımımıza göre hızlı olduğu değerlerdir. Bu değerlere göre hız kazancımız resim boyutu arttıkça artmaktadır.

4.2 Veri Ayrıştıran Modülün Performans Bilgileri

Tasarlanmış olan Veri Ayrıştırma modülü, farklı model FPGA çipleri için sentezlenmiştir. FPGA çip istatistikleri ve modülün maksimum saat frekansları saptanmıştır. Modülün veri işleme süresinin anlaşılabilmesi için ISE [43] programında uygun verilerle simülasyon yapılmıştır.

Veri Ayrıştırma modülümüz Xilinx firmasının ürettiği *Spartan3 Virtex 5, Virtex 4* ve *Virtex 2* çipleri için ayrı ayrı sentezlenmiştir. Her çip için Tablo 4.5 de verilmiş olan istatistikler elde edilmiştir.

Tablo 4.5 Veri Ayrıştırma Modülünün FPGA Çip İstatistikleri.

FPGA Çip Türü	Slice Reg. Sayısı / %	LUTs Sayısı / %	Slice FFs Sayısı / %	Bonded IOBs Sayısı / %
Virtex 5	135/1	191/1	207/1	104/28
Virtex 4	131/2	248/2	190/1	104/43
Virtex 2	121/3	206/3	182/2	104/60
Spartan3	124/16	206/13	185/12	104/83

Sentez sonucu modüllerin *slice reg* ve *LUTs* sayıları göz önüne alındığında çiplerimize bu modüllerden çok sayıda sığabileceği görülmektedir. Bu nedenle bu modül boyutu için en uygun çip *Spartan3* çipidir.

Modülün hız kazancının ölçülebilmesi için, modülle aynı algoritmayı kullanan bir C++ programı yazılmıştır. 100 hafızalık gizli mesaj verisi için farklı boyutlardaki resimlerden geri ayrıştırma işlemi yapılmıştır. Konfigürasyon bilgileri Tablo 4.2'de verilmiş olan farklı bilgisayarlar üzerinde çalışma zamanları ölçülerek Tablo 4.6'daki sonuçlar elde edilmiştir..

Veri ayrıştırma modülümüz ise Virtex 5 çipi için en çok $2{,}983 \times 10^{-9}$ saniyelik saat frekansında çalışabileceği sentezleme sonucu elde edilmiştir.

Aynı işlemin modülümüz tarafından işlenme adımı ve süresi tablo 4.7'de gösterildiği gibidir.

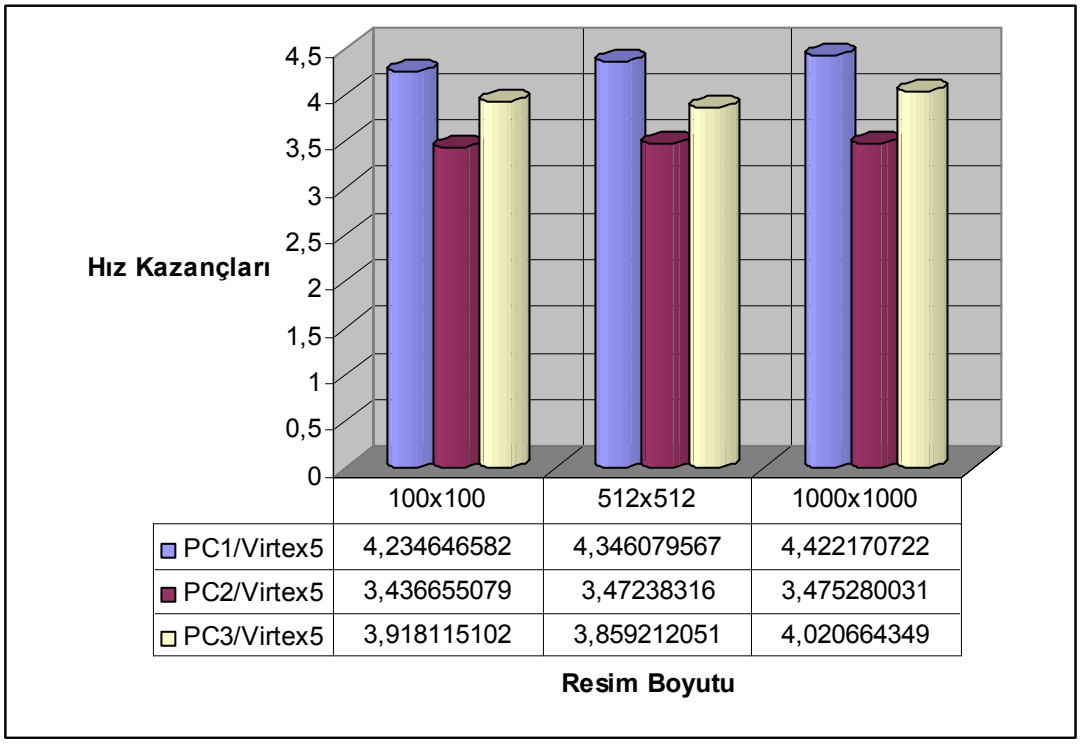
Tablo 4.6.Bilgisayarların Farklı Boyutlarda Resimlerden Veri Ayrıştırma Süreleri .

Eklenen Veri Boyutu	Görüntü Boyutu	PC-1 (μs)	PC-2 (μs)	PC-3 (μs)
100	**100x100**	21,927	17,795	20,288
100	**512x512**	22,504	17,980	19,983
100	**1000x1000**	22,898	17,995	20,819

Tablo 4.7.FPGA Çipinin Veri Ayrıştırma Süresi.

Eklenen Veri Boyutu	Görüntü Boyutu	Adım Sayısı	Süresi(μs)
100	**100x100**	1736	5,178
100	**512x512**	1736	5,178
100	**1000x1000**	1736	5,178

Farklı boyutlardaki resimler dahi olsa 100 hafıza adresi boyutundaki gizli mesajın ayrıştırılması aynı sürede olacaktır. Bunun nedeni ise veri ayrıştırma işlemi gizli mesaj verisi ayrıştırıldıktan sonra bitmesidir. Bu bilgiler ışığında bilgisayar üzerinde çalışan programa nazaran veri ayrıştırma modülümüzün hız kazancı Şekil 4.2'de verilmiştir.



Şekil 4.2. Veri Ayrıştırma İşleminin Farklı Bilgisayarlara Göre Hız Kazançları.

Şekil 4.2'de gösterildiği gibi modülümüz 100 hafıza adresine ait bilgiyi aynı işlemi gerçekleştiren bir yazılıma göre 3,5- 4,4 kat daha hızlı yapabilmektedir.

5. SONUÇ VE GELECEKTE YAPILABİLCEK ÇALIŞMALAR.

Bu çalışmada günümüz teknolojisinin iletişim için kullandığı yöntemlerin, güvenlik açıklarından dolayı oluşabilecek tehditlere karşı korunabilmesi ve buna yönelik uygulanabilecek metotlardan bir tanesi için donanım modülleri tasarlanmıştır.

Bilgisayar ağları içinde dolaşan verilere her noktadan ulaşabilmek mümkün olduğu için bu verileri güvenli bir şekilde iletmekte bir o kadar zordur. Ancak iletilecek olan veri dikkat çekmezse saldırı riski de o kadar az olacaktır. Bu nedenle saldırıya uğramasını istemediğimiz verilerimizi ya çok karmaşık şifrelerle şifrelemeli ya da bu tez boyunca bahsettiğimiz metotlar ile gizleyerek iletmeliyiz.

Karmaşık şifreler kullanıldığında iletilen mesajın geri elde edilmesi, şifreleme işlemi esnasında kullanılan karmaşık algoritmaya bağlı olarak zaman açısından verimsiz olacağından çok ideal bir yöntem değildir. Ancak Steganografi kullanılan yöntemler şifreli iletişime nazaran daha az dikkat çekmekte ve gönderilen mesaj daha hızlı bir sürede alıcı tarafta tekrar oluşturulabilmektedir.

Şayet resim steganografisi işlemi yazılımla yapılmak istenseydi, bütün aşamalardaki veriler yazılımın üzerinde çalıştığı işletim sisteminin bütün açıklarına karşı korumasız olacaktı. Ancak bu işlemi donanımla gerçekleştirdiğimiz için yazılımımızdan doğabilecek bütün güvenlik açıklarına karşı da önlem almış olmaktayız.

Modüllerimiz FPGA simülatöründe test edilmiştir. Modüllerimiz üzerinde çalışan algoritmanın görsel olarak başarısını göz önüne serebilmek için Matlab programının görsel ara yüzü kullanılarak program yazılmış ve sonuçları görsel olarak sunulmuştur.

Sonuç olarak; bu çalışmada resimler üzerinde uygulanan Steganografi sanatının LSB metodu başarıyla gerçekleştirilmiştir. Yazılıma oranla hız açısından; veri gömme modülünde %35, veri ayrıştırma modülünde %4,5 daha iyi bir performans sergilemiştir.

Her iki modül birlikte bir FPGA çipine sığabilmektedir. Bu nedenle iletişim içinde olan iki bilgisayar üzerinde aynı FGPA platformunda hem mesaj alma hem de gönderme işlemi için kullanılması çok uygundur.

Donanım modülleri FPGA çipleri için tasarlanmıştır. Günümüzde bir FPGA çipinin saat frekansı yaklaşık olarak 500 MHz'dir. Bu modüller ASIC tasarlandığında çalışma frekansları daha da artacağı düşünülürse bu işlemin donanım modüllerince yapılması daha iyi sonuçlar sunacaktır.

Gelecekte FPGA platformlarının daha da hızlandırılacağı ve yaygınlaşacağı düşünülürse, bu tip çalışmaların artık tamamen yazılımdan bağımsız olarak tasarlanıp hayata geçirileceği düşünülebilir.

5.1. Gelecekte Yapılabilecek Çalışmalar

Gelecekte bu çalışmaya ek olarak yapılabilecek çalışmalar içinde, akan resim dosyalarına (frame) veri gömme ve ayrıştırma işlemi gerçekleştirilebilinir.

Veri gömme ve ayrıştırma esnasında güvenlik bir adım daha ileri götürülerek bir Stego key kullanılabilinir.

Resim haricinde ses verileri için de aynı metodu kullanan modüller tasarlanabilinir.

Bir bitlik alan yerine her pikselin RGB bileşenlerinin 2'şer bitlik alanları veri gömmek için kullanılabilinir.

Sıkıştırılmış resim verileri üzerinde bu metod için modüller tasarlanabilinir.

Modüllerimizin boyutları da düşünülerek bir tek FPGA çipine kompleks bir FPGA modülü tasarlanabilir ve bu çip içine farklı tekniklerle veri gömme ve ayrıştırma işlemleri için modüller tasarlanabilir.

KAYNAKÇA

[1] Canbek, G., Sağıroğlu, S., "*Bilgi ve Bilgisayar Güvenliği Casus Yazılımlar ve Korunma Yöntemleri*", Grafiker, Ankara, 1-50 (2006).

[2] Sağıroğlu, S., Alkan, M., "*Her Yönüyle Elektronik İmza (eİmza)*", Grafiker,Ankara,3,5,33 (2005).

[3] Caldwell, 2nd Lt. J., "*Steganography*", CROSSTALK The Journal of Defense Software Engineering, 25-27 (2003).

[4] Öcal, F. "*Güvenli İletişim İçin FPGA Kullanarak Şifreleme Sistemi Tasarımı Ve Gerçekleştirilmesi*." Gazi Üniversitesi, Yüksek Lisans Tezi,Ankara-Türkiye.2006.

[5] Hamblen J. O., Furman M. D., "*Rapid Prototyping of Digital System*", Second Edition, Kluwer Academic Publishers**,** Boston America,163-178 (2003).

[6] Parnell,K.,Meta,N.,Programmable Logic Design Quick Start Hand Book,Xilink, Chapter1,2003.

[7] Sahin, İ. *"A Compilation Tool for Automated Mapping of Algorithms onto FPGA Based Custom Computing Machines"*. NC State University, Doktora Tezi, Raleigh-USA. 2002.

[8] Sırmaçek, B."*FPGA ile Mobil Robot İçin Öğrenme Algoritması Modellenmesi*",Yıldız Teknik Üniversitesi, Yüksek Lisans Tezi, İstanbul, 2007.

[9] Koyuncu, İ. *"A Matrix Multiplication Engine for Graphic Systems Designed tor run on FPGA Devices"*. Düzce Üniversitesi, Yüksek Lisans Tezi Düzce-Türkiye. 2008.

[10] Stallings W., "*Cryptography and network security Principles and Practices*" Third edition Prentice hall**,** Washinton DC, 26-89 (2003).

[11] Spartan-3 Starter Kid Board User Guide, Xilinx , America, 4-56 (2004)

[12] Anonim : http://www.ntvmsnbc.com/news/74525.asp

[13] Anonim: http://www.altera.com/products/devices/dev-index.jsp. (Kasım2007)

[14] Şahin, İ. ve Gloster, C.S. *"FPGA Tabanlı CCM'ler İçin Genel Amaçlı Bir Arayüz"*. Bilimde Modern Yöntemler Sempozyumu, Kocaeli. 2005.

[15] Anonim: 2008. http://www.hitechglobal.com/

[16]Anonim:http://www.bilesim.com.tr/tr/index.nsf?lf=/tr/leftbaryayincilik.html&rf = http://www.bilesim.com.tr/mistoportal/showmakale.nsf?xd=2296.xml.

[17] Anonim: http://www.xilinx.com/support/documentation/data_sheets.2007.

[18] Anonim: 2004. http://www.bilesim.com/.

[19] Anonim: http://tr.wikipedia.org/wiki/Renk#cite_note-0

[20] Anonim: http://tr.wikipedia.org/wiki/RGB.

[21] Anonim: http://www.mathworks.com/matlabcentral/fxfiles/14142/1/showPixel Values.png.

[22] Murray A.H., Burchfiled R.W (eds.), "*The Oxford English Dictionary: Being a Corrected Re-issue*", Oxford, England: Clarendon Pres, 1933.

[23] Anonim: Cummins, J., Diskin, P., Lau, S., Parlett, R., " *Steganography and Digital Watermarking*", http://www.cs.bham.ac.uk /~mdr/teaching/ modules03 /security/students/SS5/Steganography.pdf

[24] Adli, A.; Nakao, Z., "*Three steganography algorithms for MIDI files*" Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference , Guangzhou, China, 4 : 2401- 2404 (2005).

[25] Gopalan, K., "*Audio steganography using bit modificatio,*" , 2003 InternationalConference on Multimedia and Expo, Baltimore, Maryland, 629-632 (2003).

[26] Niimi, M., Noda, H., Kawaguchi, E., Eason, R.O., "*High capacity and secure digital steganography to palette-based images*" Image Processing 2002. Proceedings 2002 International Conference , Rochester, New York, USA, 2: 917-920 (2002).

[27] Sağıroğlu, S., Tunckanat, M., "*A Secure Internet Communication Tool*", Turkish Journal of Telecommunications, 1(1):40-46 (2002).

[28] Shahreza, S., "*Stealth steganography in SMS*" Wireless and Optical Communications Networks, 2006 IFIP International Conference, Bangalore, 5- (2006).

[29] Sui, X.G., Luo, H., "*A new steganography method based on hypertext*" Radio Science Conference, 2004. Proceedings. 2004 Asia-Pacific, Qingdao, China, 181-184 (2004).

[30] Tseng, H.W., Chang, C.C., "*Steganography using JPEG-compressed images*" Computer and Information Technology, 2004. CIT '04. The Fourth International Conference , Wuhan, China ,12- 17 (2004).

[31] Swanson, M.D., Zhu, B., Tewfik, A.H., "*Robust Data Hiding For Images*", In 7th Digital Signal Processing Workshop (DSP 96), Loen Norway, 37-40 (1996).

[32] Anonim http://www.gulum.net/kartlari-resimleri/manzara-resimleri/images/manz araresmi6.jpg.

[33] Anonim: El-Khalil, R., Keromytis, A.D., "*Hydan: Hiding Information in Program Binaries*" http://www1.cs.columbia.edu/~angelos/Papers/hydan.pdf

[34] Bender, W., Gruhl, D., Morimoto, N., and Lu, A. "*Techniques for data hiding*". IBM Syst. J., 35(3&4):313-336 (1996).

[35] Anonim: " Veri Gizle" www.verigizle.com

[36] Noda, H., Spaulding, J., Shirazi, M.N., Kawaguchi, E., "*Application of bitplane decomposition steganography to JPEG2000 encoded images*" Signal Processing Letters, IEEE 9(12):410-413 (2002).

[37] Brisbane, G., Safavi-Naini, R., Ogunbona, P., "*High-capacity steganography using a shared colour palette*," Vision, Image and Signal Processing, IEE Proceedings, 152: 787- 792 (2005).

[38] Anonim: http://e-bergi.com/2009/Mart/Veri-Gizleme-Bilimi.

[39] AKLEYLEK S., NURİYEV U., "*Steganografi ve Steganografinin Yeni Bir Uygulaması*".Signal Processing and Communications Applications Conference, 2005. Proceedings of the IEEE 13th Page(s):64 – 67, 2005.

[40] Lee, Y.K., Chen, L.H., "*High capacity image steganographic model*" Vision, Image and Signal Processing, IEE Proceedings- , 147(3):288-294 (2000).

[41] Mahmoud M.I., M. D. M. I., Deyab S., and H. Elfouly F. "*Wavelet Data Hiding using Achterbahn-128 on FPGA Technology*", UbiCC Journal - Special Issue of IKE'07 Conference , IKE'07 - Special Issue , 2008

[42] Santi P. Maity, Ayan Banerjee, and Malay K. Kundu,"*An Image-in-Image communication scheme and VLSI implementation using FPGA"*,India Annual Conference, 2004. Proceedings of the IEEE INDICON 2004.

[43] Anonim: 2007. http:// www.xilinx.com/.

EKLER

Ek-A:Görsel Amaçlı Test İçin Yazılan Matlab Gui Program Kodları

Bu program görsel olarak yapılan gömme işleminin taşıyıcı olarak kullanılan resimde ne kadar az değişikliğe sebep olduğunu göstermek amaçlı yapılmıştır.

Bu kodlar Matlab programının GUI alt programında yapılmıştır.

```
function varargout = resimload(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @resimload_OpeningFcn, ...
                   'gui_OutputFcn',  @resimload_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end


if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end

function resimload_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','off');
set(handles.pushbutton6,'enable','off');
set(handles.pushbutton7,'enable','off');
set(handles.pushbutton5,'enable','off');
set(handles.pushbutton4,'enable','off');

function varargout = resimload_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

```
function dosyaac_Callback(hObject, eventdata, handles)
[f,rep]=uigetfile('*.bmp');
OriginalImage=imread([rep,f]);
asilresim=OriginalImage;
setappdata(handles.resimload,'asilresim',asilresim);
OriginalImage=imresize(OriginalImage,[300 300]);
setappdata(handles.resimload,'OriginalImage',OriginalImage);
axes(handles.axes1);
imshow(OriginalImage);
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','on');
set(handles.pushbutton5,'enable','on');
set(handles.pushbutton6,'enable','on');
set(handles.pushbutton4,'enable','on');
set(handles.pushbutton1,'enable','off');

function dosyakaydet_Callback(hObject, eventdata, handles)
resim2=getappdata(handles.resimload,'gomulmusresim');
[filename, pathname] = uiputfile('*.bmp','Resmi Kaydet');
if isequal(filename,0) | isequal(pathname,0)
   disp('User selected Cancel')
else
   disp(['User selected',fullfile(pathname,filename)])
end

filename=[pathname,'\',filename];
imwrite(resim2,filename);

function Untitled_1_Callback(hObject, eventdata, handles)

function OriginalImage_Callback(hObject, eventdata, handles)

function pushbutton1_Callback(hObject, eventdata, handles)

resim=getappdata(handles.resimload,'OriginalImage');
a=size(resim);
i=a(1);
j=a(2);
sifir=zeros(i,j);
r=resim;
r(:,:,2)=sifir;
r(:,:,3)=sifir;
g=resim;
g(:,:,1)=sifir;
g(:,:,3)=sifir;
b=resim;
b(:,:,1)=sifir;
b(:,:,2)=sifir;
```

```
function pushbutton2_Callback(hObject, eventdata, handles)

resim=getappdata(handles.resimload,'OriginalImage');
a=size(resim);
i=a(1);
j=a(2);
sifir=zeros(i,j);
r=resim;
r(:,:,2)=sifir;
r(:,:,3)=sifir;
g=resim;
g(:,:,1)=sifir;
g(:,:,3)=sifir;
b=resim;
b(:,:,1)=sifir;
b(:,:,2)=sifir;
grayscale(r,g,b,resim)

function pushbutton3_Callback(hObject, eventdata, handles)
close(handles.resimload)

function edit1_Callback(hObject, eventdata, handles)

function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

function pushbutton4_Callback(hObject, eventdata, handles)

tic
set(handles.pushbutton6,'enable','on');
set(handles.pushbutton7,'enable','on');
set(handles.pushbutton5,'enable','on');
resim=getappdata(handles.resimload,'asilresim')
resim1=resim;
karakter=get(handles.edit1,'string')
karakter=['ğ',karakter]
ikilik=abs(karakter)
ikilik=dec2bin(ikilik);
[harfsayisi,sutun]=size(ikilik)
harfsayisi=harfsayisi-1
dizi='000000000';
ikilik(1,:)=dizi
```

```
if ne(harfsayisi,1)
        for i=1:1:harfsayisi-1
                dizi=[dizi;'000000000']
        end
end

%karakterleri diziye çevirdim
m=1
for z=2:1:harfsayisi+1
        for t=1:1:9
                g(m)=ikilik(z,t);
                m=m+1;
        end
end

%karakterleri diziye çevirdim
n=m-1;
m=1;
mm=1;
[x,y,boyut]=size(resim1)
n=n/9;%her karakter 9 bit olduğu için kaç karakter olduğu öğreniliyor
nn=dec2bin(n);
%resimde veri olduğuna dair 12-13-14 üncü pixellerin son bitlerine o değeri atanacak
i=1
for j=12:1:14
        for k=1:1:3
                b=resim1(i,j,k);
                b=uint16(b);
                b=[300,b];
                b=dec2bin(b);
                b(2,9)=('0');
                b=bin2dec(b);
                resim1(i,j,k)=b(2);
        end
end

%33 bitlik bir binary değişkene karakter uzunluğu kodlanıyor
bc=('00000000000000000000000000000000')
inf=length(nn);
inf1=length(bc);
z=inf
for i=inf1:-1:1
        if z>=1
                bc(i)=nn(z)
                z=z-1
        end
end
%33 bitlik bir binary değişkene karakter uzunluğu kodlanıyor
```

```
%%veri boyutu gömme ilk 11 pixele
i=1
m=33
for j=1:1:11
        for k=1:1:3
                a=resim1(i,j,k);
                a=uint16(a)
                a=[300,a]
                a=dec2bin(a);
                a(2,9)=bc(m);
                b=a(2,:)
                b=bin2dec(b);
                resim1(i,j,k)=b;
                m=m-1
        end
end

%%veri boyutu gömme
%%veri gömme
m=1
n=n*9;
for i=1:1:x
        for j=1:1:y
                for k=1:1:3
                        sss=j>14|i>1;
                if m~=n+1&sss==1
                        a=resim1(i,j,k);
                        a=uint16(a);
                        a=[300,a]
                        a=dec2bin(a);
                        a(2,9)=g(m);
                        b=a(2,:);
                        b=bin2dec(b)
                        resim1(i,j,k)=b;
                        m=m+1;
                end
                end
        end
end

%veri gömüldü
%resim1 set edildi

setappdata(handles.resimload,'gomulmusresim',resim1);
axes(handles.axes2);
imshow(resim1);
msgbox('Veri Başarıyla Gizlendi')
```

```
toc
zaman=toc
%%
function pushbutton5_Callback(hObject, eventdata, handles)

resim=getappdata(handles.resimload,'asilresim');

a=size(resim);
i=a(1);
j=a(2);
k=a(3);
boyut=i*j*k;
set(handles.edit4,'string',i);
set(handles.edit5,'string',j);
boyut=boyut-2;
bit=str2num(get(handles.bitsayisi,'string'));
bit=uint32(bit);
if bit>8
        msgbox('Tekrar Giriniz Yanlış Rakam Girdiniz','Uyarı');
elseif bit<=8
        boyut=boyut/8;
        boyut=uint32(boyut);
        boyut=boyut*bit;
        boyut=num2str(boyut);
        set(handles.karaktersayisi,'string',boyut);
end

function karaktersayisi_Callback(hObject, eventdata, handles)

function karaktersayisi_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

function bitsayisi_Callback(hObject, eventdata, handles)

function bitsayisi_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

function edit4_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

function edit5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

function edit6_Callback(hObject, eventdata, handles)

function edit6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

function resimload_CreateFcn(hObject, eventdata, handles)

function pushbutton6_Callback(hObject, eventdata, handles)

resim=getappdata(handles.resimload,'asilresim')
a=resim;

c=('00000000000000000000000000000000');
z=1
i=1
for j=12:1:14
        for k=1:1:3
                b=a(i,j,k);
                b=uint16(b);
                b=[300,b];
                b=dec2bin(b);
                c(z)=b(2,9)
                z=z+1
        end
end
c=bin2dec(c)
if c==0
%%boyut geri al
        m=33
```

```
        byt=('00000000000000000000000000000000')
        i=1
for j=1:1:11
        for k=1:1:3
                b=a(i,j,k);
                b=uint16(b);
                b=[300,b];
                b=dec2bin(b);
                byt(m)=b(2,9);
                m=m-1
        end
end
%%boyut geri alindi
[x,y,boyut]=size(a)
byt=bin2dec(byt);%karakter sayısı
bytbit=9*byt;%karakter bit sayisi
m=1
for i=1:1:x
        for j=1:1:y
                for k=1:1:3
                        sss=j>14|i>1;
                if m~=bytbit+1&sss==1
                        b=a(i,j,k);
                        b=uint16(b);
                        b=[300,b];
                        b=dec2bin(b);
                        kar(m)=b(2,9);
                        m=m+1
                end
                end
        end
nd

m=1
[q,w]=size(kar)
for i=1:1:byt
        for j=1:1:9
        if w~=m-1
                dizi(i,j)=kar(m)
                m=m+1
        end
        end
end
dizi=bin2dec(dizi);
dizi=char(dizi)

s=[dizi(1)];
```

```
for i=2:1:size(dizi)
        s=[s,dizi(i)]
end
set(handles.edit6,'string',s);
xx=[s,' Karkterleri Gömülmüş']
msgbox(xx,'Karakterleri Gömülmüş')
else
        msgbox('Bu Resimde Veri Yoktur','Uyarı')
end

function pushbutton7_Callback(hObject, eventdata, handles)

resim=getappdata(handles.resimload,'asilresim');
resim2=getappdata(handles.resimload,'gomulmusresim');
figure;
subplot(1,2,1)
imshow(resim);
subplot(1,2,2);
imshow(resim2);
```

Ek-B: Bilgisayar İşlem Süresi Hesaplanması İçin Hazırlanan Program Kodları

Aşağıda C++ kodları verilen program deneylerde kullanılan PC'lerin veri işleme sürelerini belirlemek amacıyla oluşturulmuştur.

Veri Gömme Süresini Bulmak İçin Kullanılan C++ Program Kodu:

```
#include<stdio.h>
#include<fstream>
#include<string.h>
#include<math.h>
#include<iostream>
#include <time.h>
#include <iomanip>
using namespace std;

#define CpuId __asm __emit 0fh __asm __emit 0a2h
#define RdtSc __asm __emit 0fh __asm __emit 031h

unsigned *GetCycles()
{
        unsigned *cycles;
        cycles = new unsigned[2];
```

```
	unsigned cycles_high=0, cycles_low=0;
	__asm
	{
		pushad
		CpuId
		RdtSc
		mov	cycles_high, edx
		mov	cycles_low, eax
		popad
	}

	cycles[0] = cycles_high;
	cycles[1] = cycles_low;
	return cycles;
}


double GetMsec (unsigned *Cycle1, unsigned *Cycle2, int CPU)
{
	unsigned __int64 temp_cycles1=0, temp_cycles2=0;
	__int64 total_cycles=0;

	temp_cycles1 = ((unsigned __int64)Cycle1[0] << 32) | Cycle1[1];
	temp_cycles2 = ((unsigned __int64)Cycle2[0] << 32) | Cycle2[1];

	total_cycles = temp_cycles2 - temp_cycles1;

	return double(total_cycles)/double(CPU);
}



int main()
{
	unsigned	*T0,*T1;
	T0 = new unsigned[2];
	T1 = new unsigned[2];
	const int CPU =2000;
	const int Boyut = 500;

	unsigned int veri[100],ss,kk,a=0,c;
	unsigned int **resimdata = new unsigned int *[100];
	int x,y,d;

	for (int i=0; i<100 ; i++)
		resimdata [i] = new unsigned int [100];
```

```
        ofstream resimveriler("resim.dat");//resim datası oluşturuldu
        for(x=0;x<=99;x++)
        {
                for(y=0;y<=99;y++)
                {
        ss=rand();
        resimveriler <<ss<<endl;}
        }
        resimveriler.close();

        ofstream dataveriler("data.dat");//gömülecek data oluşturuldu
        for(x=0;x<=99;x++)
        {
        kk=rand();
        dataveriler <<kk<<endl;
        }
        dataveriler.close();
        ifstream pixeller ("resim.dat");
for(d=0;d<=99;d++)
                                        for(c=0;c<=99;c++)

                                                pixeller >> resimdata[d][c];

pixeller.close();
ifstream veriler ("data.dat");
        for(a=0;a<=99;a++)
        veriler >> veri[a];
veriler.close();


//şimdi veri dizisinde 100 adet veri var ve resimdata dizisinde 10000 adet veri var

        T0 = GetCycles();
        x=0;
        int r=1,g=2,b=4,red,green,blue,toplam=0,z;


                for(int i=0;i<=99;i++)
                {
                        for( int j=0;j<=99;j++)
                        {

                                for(int k=1;k<=10;k++)
                                {
                                red=r&veri[x];
                                green=g&veri[x];
                                blue=b&veri[x];
                                red=red<<8;
```

```
                    green=green<<15;
                    blue=blue<<22;
                    toplam=red+green+blue;

                    resimdata[i][j]=toplam&resimdata[i][j];
                    veri[x]=veri[x]>>3;
                    j=j+1;
                    z=j;
                    y=i;
                    }

                    x=x+1;

                    if(x==99)
                    break;
                }
                if(x==99)
                break;

        }
                for (int i=y;i<=99;i++)
                    for(int j=z;j<=99;j++)
                    {
                            pixeller >> resimdata[i][j];
                    }


            T1 = GetCycles();

    cout << setiosflags(ios::showpoint|ios::fixed);
    cout << setprecision(10);
    cout << "\nExecution Time = " << setw(12) <<
GetMsec(T0,T1,CPU)/1000000 << " sec.\n";
    cout << "Execution Time = " << setw(12) << GetMsec(T0,T1,CPU)/1000 <<
" msec.\n";
    cout << "Execution Time = " << setw(12) << GetMsec(T0,T1,CPU) << "
usec.\n";

    cin.get();
    cin.get();
}
```

Veri Ayrıştırma Süresini Bulmak İçin Kullanılan C++ Program Kodu:

```
#include<stdio.h>
#include<fstream>
#include<string.h>
```

```
#include<math.h>
#include<iostream>
#include <time.h>
#include <iomanip>
using namespace std;

#define CpuId __asm __emit 0fh __asm __emit 0a2h
#define RdtSc __asm __emit 0fh __asm __emit 031h

unsigned *GetCycles()
{
	unsigned *cycles;
	cycles = new unsigned[2];

	unsigned cycles_high=0, cycles_low=0;
	__asm
	{
		pushad
		CpuId
		RdtSc
		mov	cycles_high, edx
		mov	cycles_low, eax
		popad
	}

	cycles[0] = cycles_high;
	cycles[1] = cycles_low;
	return cycles;
}
double GetMsec (unsigned *Cycle1, unsigned *Cycle2, int CPU)
{
	unsigned __int64 temp_cycles1=0, temp_cycles2=0;
	__int64 total_cycles=0;

	temp_cycles1 = ((unsigned __int64)Cycle1[0] << 32) | Cycle1[1];
	temp_cycles2 = ((unsigned __int64)Cycle2[0] << 32) | Cycle2[1];

	total_cycles = temp_cycles2 - temp_cycles1;

	return double(total_cycles)/double(CPU);
}

int main()
{

	unsigned	*T0,*T1;
	T0 = new unsigned[2];
	T1 = new unsigned[2];
```

```
	const int CPU =2000;
	const int Boyut = 500;

	unsigned int **resimdata = new unsigned int *[100];
	unsigned int veri[100],i,x,y,j;

	for (i=0; i<100 ; i++)
		resimdata [i] = new unsigned int [100];

	ofstream resimveriler("resim.dat");//resim datası oluşturuldu
	for(x=0;x<=9999;x++)
	{
	i=rand();
	resimveriler <<i<<endl;
	}
	resimveriler.close();

	ifstream pixeller ("resim.dat");
	for(x=0;x<=99;x++)
	{
		for(y=0;y<=99;y=y+1)
		{
		pixeller >> resimdata[x][y];
		}
	}
pixeller.close();
//resimdata dizisinde 10000 adet veri var
	T0 = GetCycles();
x=0;
	int r,g,kk,b,toplam=0,l,m,n,maske=0,tmp=0;
	r=pow(2,24);//red için
	g=pow(2,16);//green için
	b=pow(2,8);//blue için
	maske=r+g+b;//maske oluşturuldu

	for(i=0;i<=99;i++)
		{
			for(j=0;j<=99;j++)
			{
				for(kk=1;kk<=10;kk++)
				{
				tmp=maske&resimdata[i][j];
				l=tmp>>8;
				m=tmp>>15;
				n=tmp>>22;
				veri[x]=veri[x]+l+m+n;
				if (kk!=10)
				veri[x]=veri[x]<<3;
```

```
                    }
                    x=x+1;
                    if(x==99)
                    break;
                }
                if(x==99)
                break;

            }
            T1 = GetCycles();

        cout << setiosflags(ios::showpoint|ios::fixed);
        cout << setprecision(10);
        cout << "\nExecution Time = " << setw(12) <<
GetMsec(T0,T1,CPU)/1000000 << " sec.\n";
        cout << "Execution Time = " << setw(12) << GetMsec(T0,T1,CPU)/1000 <<
" msec.\n";
        cout << "Execution Time = " << setw(12) << GetMsec(T0,T1,CPU) << "
usec.\n";

        cin.get();
        cin.get();
}
```